



KUBE-DEV

v5

webofmars

the devops companion





TOUR DE TABLE



ACCUEIL

- **Objectifs pédagogiques**
 - Comprendre les principes de l'orchestration de conteneurs
 - Utiliser et mettre en oeuvre Kubernetes
 - Déployer et packager des applications conteneurisées
 - Orchester et gérer une application de conteneurs
- **Niveau requis**
 - Avoir de bonnes compétences sur un système (Linux)
 - Connaître les technologies de conteneurs (Docker)
- **Public concerné**
 - Architectes, administrateurs, développeurs...



Agenda - Jour 1

- 09h00 : Accueil et introduction
- 09h30 - 12h30 : Rappels et fondamentaux
- 12h30 - 13h30 : ----- Pause Déjeuner -----
- 13h30 - 14h00 : Quizz
- 14h00 - 17h00 : Container as a Service (CaaS) et orchestration

Agenda - Jour 2

- 09h00 - 09h30 : Questions de la veille
- 09h30 - 12h30 : Kubernetes : mise en oeuvre
- 12h30 - 13h30 : ----- Pause Déjeuner -----
- 13h30 - 14h00 : Quizz
- 14h00 - 15h00 : Kubernetes : mise en oeuvre (suite)

Agenda - Jour 3

- 09h00 - 09h30 : Questions de la veille
- 09h30 - 12h30 : Déployer des applications d'entreprise
- 12h30 - 13h30 : ----- Pause Déjeuner -----
- 13h30 - 14h00 : Quizz
- 14h00 - 16h00 : Travaux Pratiques
- 16h00 - 17h00 : Concepts avancés

Conventions



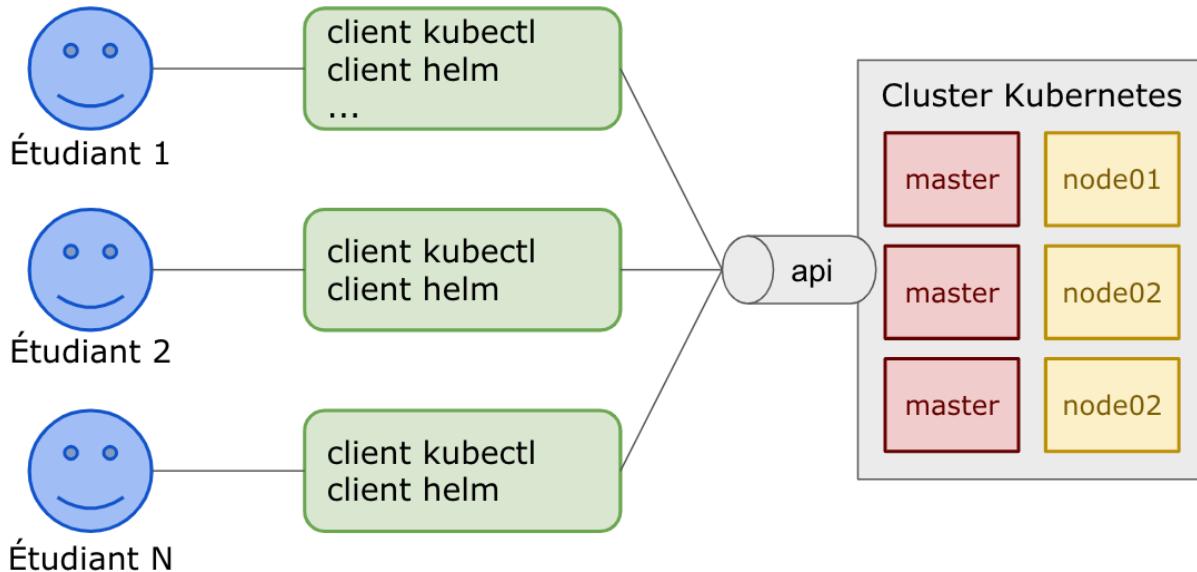
```
# commande à taper dans votre terminal
```



Ceci est une mise en garde

```
# yaml file
kind: File
metdata:
    name: « yaml-file »
```

Architecture du cluster de TPs



Pré-requis Poste de Travail

- CPU 64 bits, minima 4GO RAM
 - Options de virtualisation activée dans le bios
 - Accès au bios non restreint
 - Compte administrateur non restreint
 - Windows 10 Professional
 - Clé USB non restreinte
- Logiciels génériques
 - Putty / Puttygen / Putty agent
 - Editeur de code (VisualStudio Code / Notepad++ / Sublime Text)
 - Logiciel Typora (Markdown)
 - Virtualbox 6.1
- Flux réseaux
 - Pas de VPN entreprise actif
 - Accès http / https (sites: store.docker.com / docs.docker.com)
 - Docker pull push (http/https vers docker.io)
 - Accès SSH vers DigitalOcean
 - Accès HTTP / HTTPS vers IPs DigitalOcean
 - Accès au site kahoot.it (via app mobile ou navigateur sur PC)

Pré-requis Poste de Travail

- **Logiciels spécifiques kubernetes**
 - kubectl (1 version d'écart par rapport au cluster) - <https://kutt.it/kubectl>
 - helm 3 - <https://kutt.it/helm-3>
 - krew - <https://kutt.it/krew>
 - Plugins krew intéressants
 - ns / ctx
 - custom-cols
- Sous windows il est possible d'utiliser WSL2 et de se reporter aux instructions Linux
 - <https://kutt.it/wsl2>



CRÉATION DE NOTRE CLUSTER

Création de notre cluster kubernetes

- Nous allons créer **un** cluster kubernetes pour l'ensemble du groupe
 - sur une offre managée
 - digital ocean
 - **avant-dernière** version disponible (ex: 1.30)
 - 1 seul master ou HA (au choix)
 - 1 pool avec minimum 3 noeuds
 - autoscaling activé ou désactivé (au choix)
 - ajout de modules complémentaires standards
 - metrics-server



Création de notre cluster kubernetes

- Une fois le cluster prêt
 - on récupère le fichier kubeconfig depuis l'interface
 - on ajoute les modules complémentaires (addons) suivants :
 - Kubernetes Metrics Server
 - Nginx ingress controller
 - Cert-manager
 - Kubernetes Monitoring Stack
 - OpenEBS NFS Provisioner



Autres VMs

- Par ailleurs nous allons créer 1 autre VM
- form-access qui permet à tout le monde d'utiliser les mêmes outils
 - kubectl
 - des plugins krew
 - helm
 - etc...
- Cette VM est située chez le même cloud-provider pour des raisons de performances mais est complètement **indépendante** du cluster kubernetes



Instructions 1/2

- Attendre **la fin** de la création du cluster et de la VM d'accès
- Configurer votre client ssh pour utiliser les **clés fournies par le formateur**
- **Accéder au poste d'admin distant**
 - ssh user-0x@form-access.wmars.party
 - où x est attribué par le formateur
- **Recopier les sources des TP sur le serveur**
 - Sur la machine form-access
 - cd ~/TPs && tar -zxvf /TPs/_TPs.tgz



Instructions 2/2

- **Configurer le client kubernetes**
 - Sur la machine form-access
 - cp /TPs/_kubeconfig ~/.kube/config
 - chmod 600 ~/.kube/config
 - kubectl version
 - kubectl cluster-info
- **Création de son propre namespace**
 - Sur la machine form-access
 - kubectl create namespace <mon-prénom>
 - kubectl config set-context --current --namespace <mon-prénom>
 - kubectl config get-contexts
 - kubectl get pods



Instructions 3/3

- Ce cluster est là pour que vous **expérimentiez**
- N'hésitez pas à vous "**amuser**" avec
- Testez ce que vous voulez, notamment les **commandes affichées dans les slides**
- Il nous servira de base pour la suite des **Travaux Pratiques**

FONDAMENTAUX



- **Rappel des concepts du Cloud**
- **Rappels sur les conteneurs**
- **Le concept d'orchestrateurs de conteneurs**
- **Présentation de kubernetes**

RAPPEL DES CONCEPTS DU CLOUD

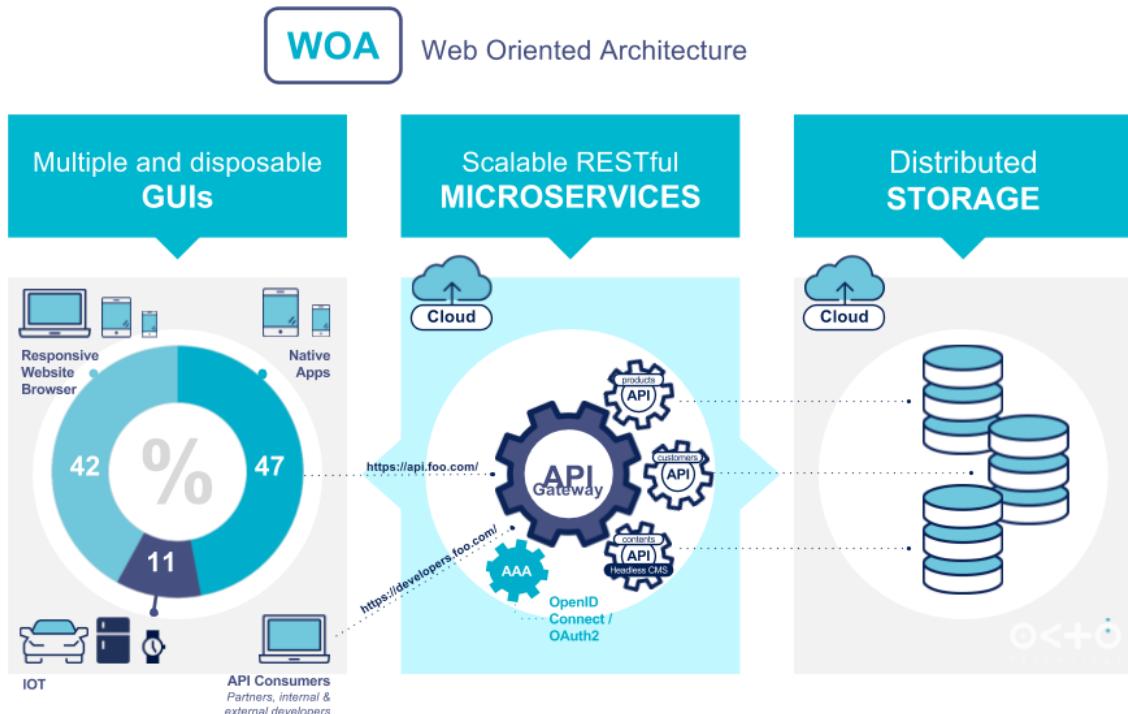


RAPPEL SUR LES APIS



- « En informatique, une interface de programmation applicative ou API pour « Application Programming Interface » désigne une interface structurée permettant d'offrir des services à un autre logiciel »





- Au lieu d'utiliser le XML pour faire effectuer une demande, le REST utilise une simple requête http sur une URL.
- Pour effectuer des actions particulières, le REST peut utiliser des méthodes HTTP spécifiques
- Et gérer des codes de retour précis, les codes d'erreur HTTP

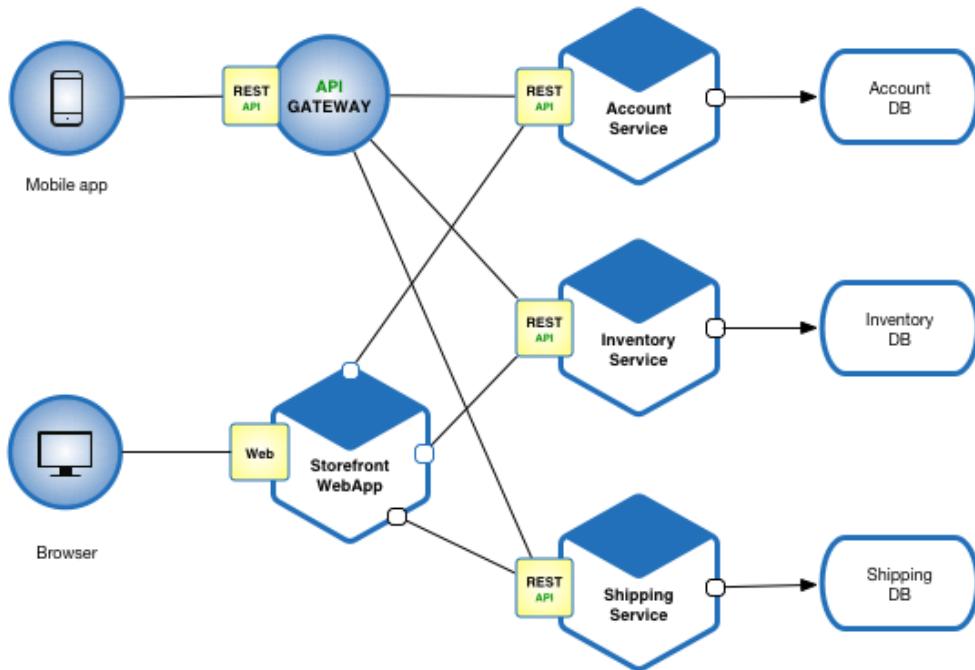
RAPPEL SUR LES MICRO-SERVICES



L'architecture MicroServices

- L'architecture en micro-services (MSA) désigne une conception logicielle basée sur les architectures orientées service (SOA) en opposition aux architectures dite monolithique (qui sont fait d'une seule pierre)
- L'idée de base est de résoudre les problèmes de complexité qui jouent sur
 - L'évolutivité
 - La fiabilité
 - La scalabilité
- Les micro-services doivent donc
 - Être de petite taille
 - Utiliser un protocole de communication très léger (HTTP)

L'Architecture MicroServices



- Qui les utilisent ?



NETFLIX



Google

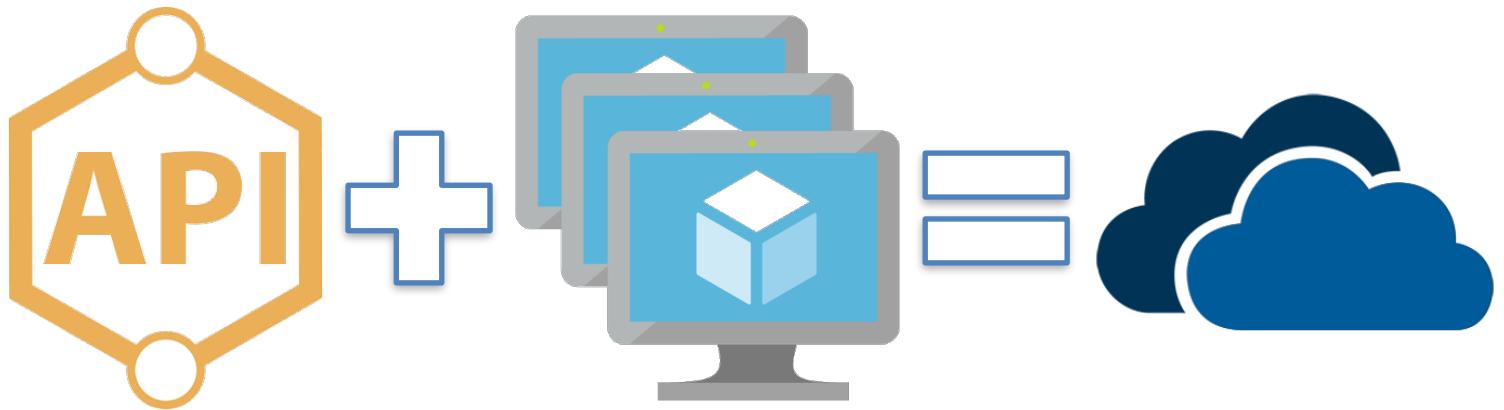


CLOUD COMPUTING

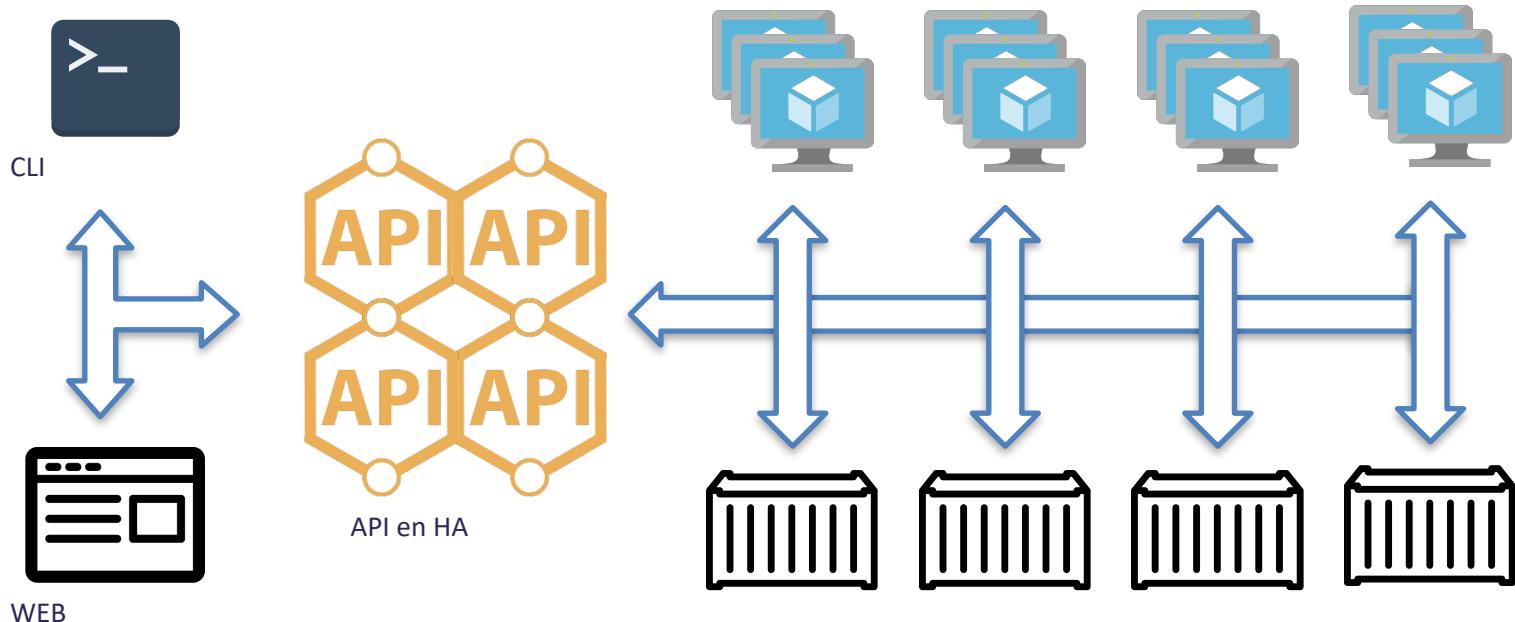


- L'utilisation massive des API, de la virtualisation et des containers a vu naître l'émergence de différents clouds publics ou privés. Il est important de bien maîtriser les différentes fonctionnalités des clouds (IaaS, PaaS et SaaS) afin d'optimiser son architecture et donc ses coûts.

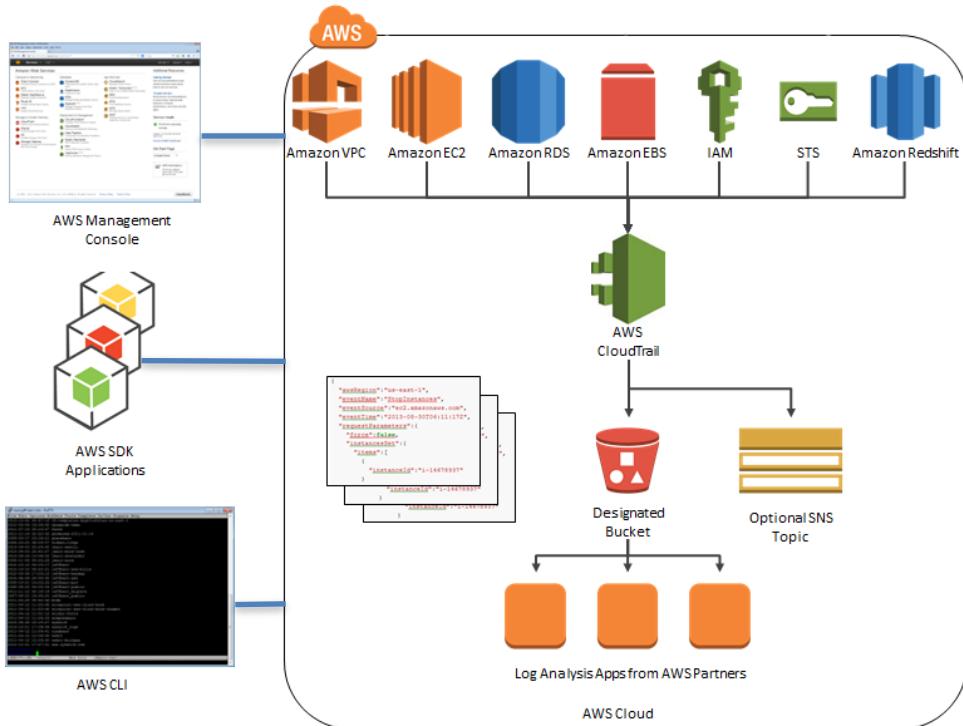
Définition du Cloud



Définition du Cloud



Définition du Cloud



Infrastructure as a Service

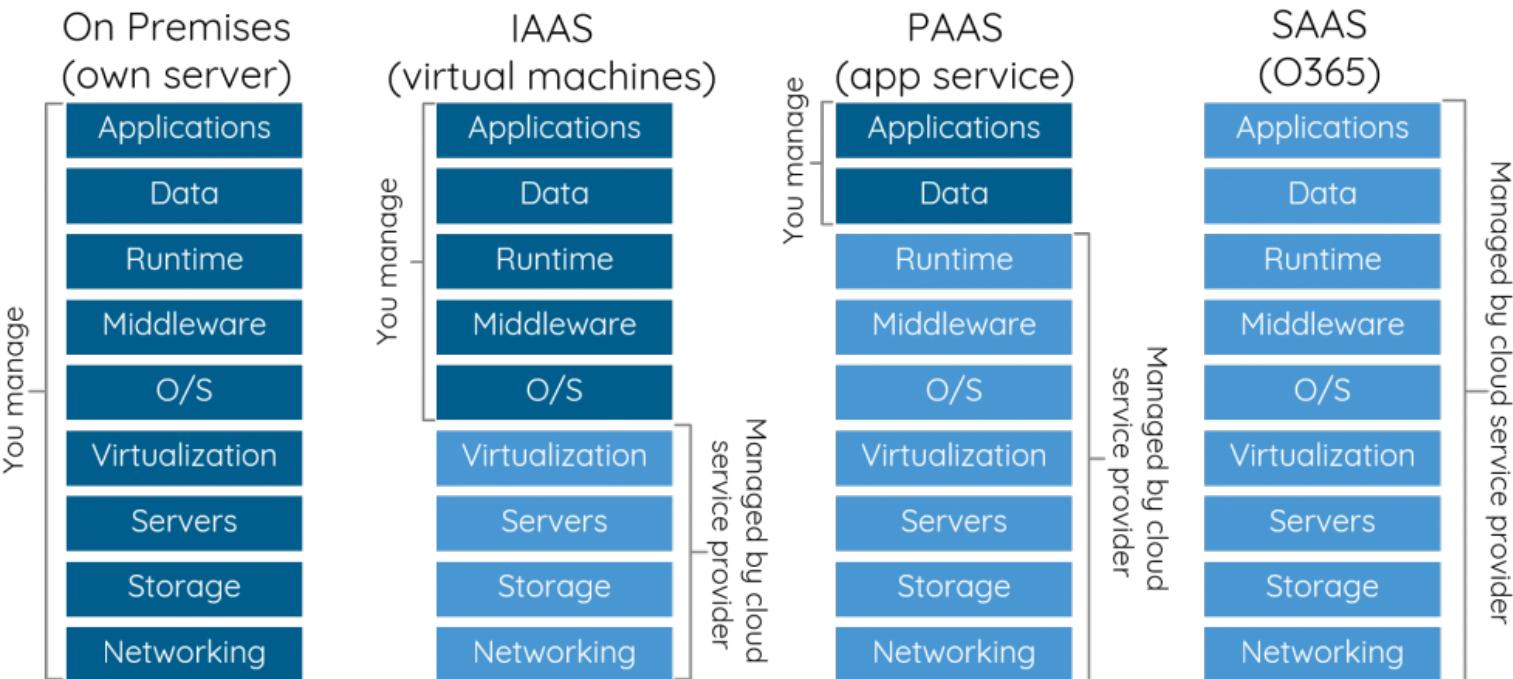
- Ce modèle fournit l'infrastructure nécessaire à l'exécution des applications, mais l'approche du cloud computing permet d'offrir un modèle du type paiement à l'usage ainsi que la possibilité de redimensionner son instance en fonction du besoin actuel.

- Dans le modèle SaaS, le client n'achète pas un logiciel, mais paye un droit d'utilisation. Ce droit d'utilisation peut être facturé grâce à un abonnement, selon l'utilisation du client, ou même gratuitement (souvent avec des fonctionnalités limitées ou une utilisation commerciale des données).

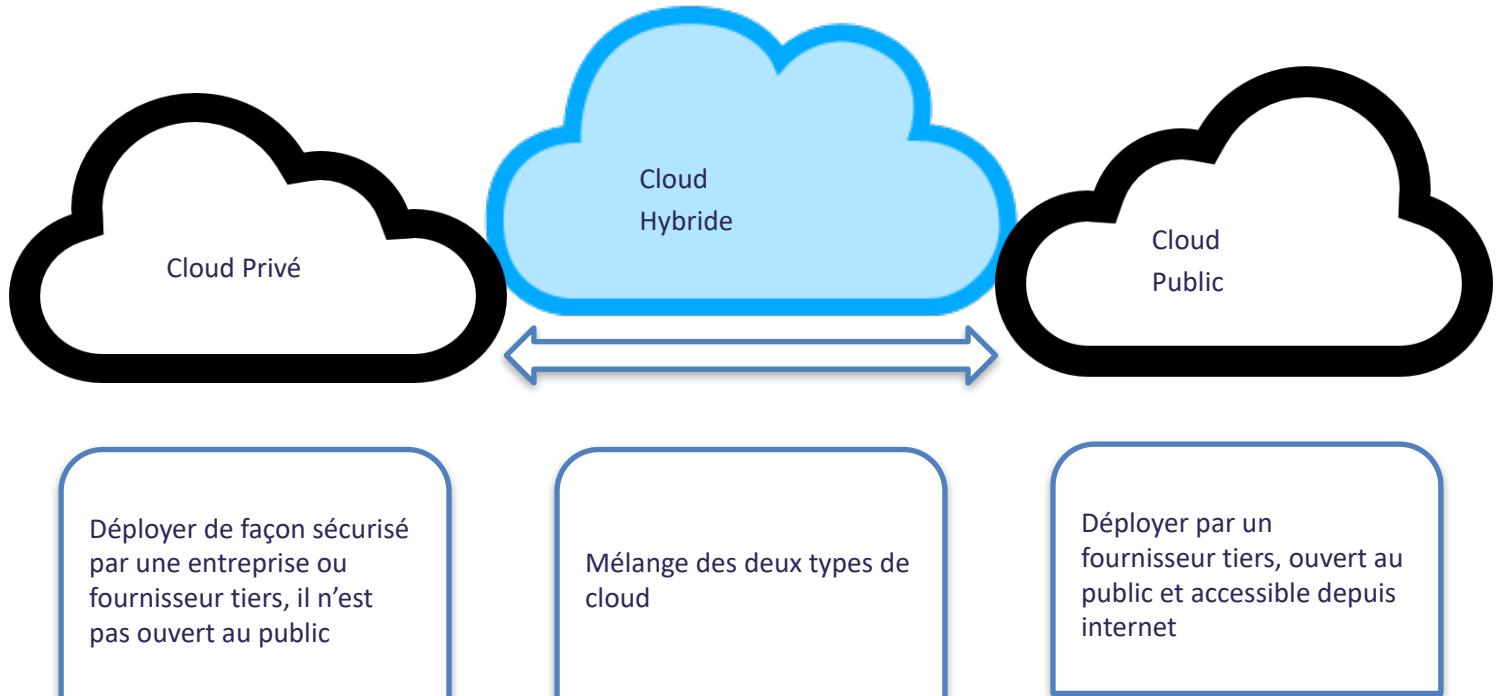
Platform as a Service

- Le Platform as a Service peut être vu comme une variante du Software as a Service dans lequel l'environnement de développement serait proposé en tant que service. Les développeurs utilisent les outils du fournisseur pour créer leurs propres applications, ils peuvent souvent créer des applications web sans avoir à installer le moindre outil sur leur poste de travail.

IaaS, PaaS et SaaS



Clouds public et privé



RAPPELS SUR LES CONTENEURS



LE BESOIN INITIAL

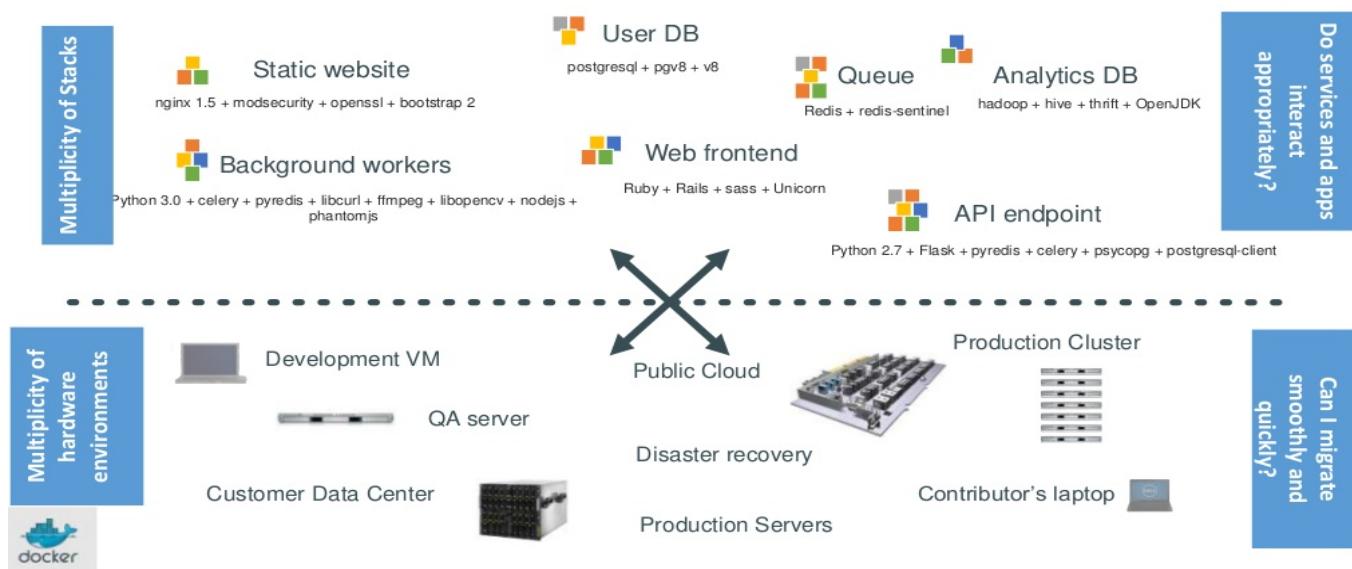


- « Docker est un logiciel libre qui automatise le déploiement d'applications dans des conteneurs logiciels » (wikipedia)
- « Docker est un outil qui permet d'empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur »



Présentation du concept de conteneur Linux

The Challenge



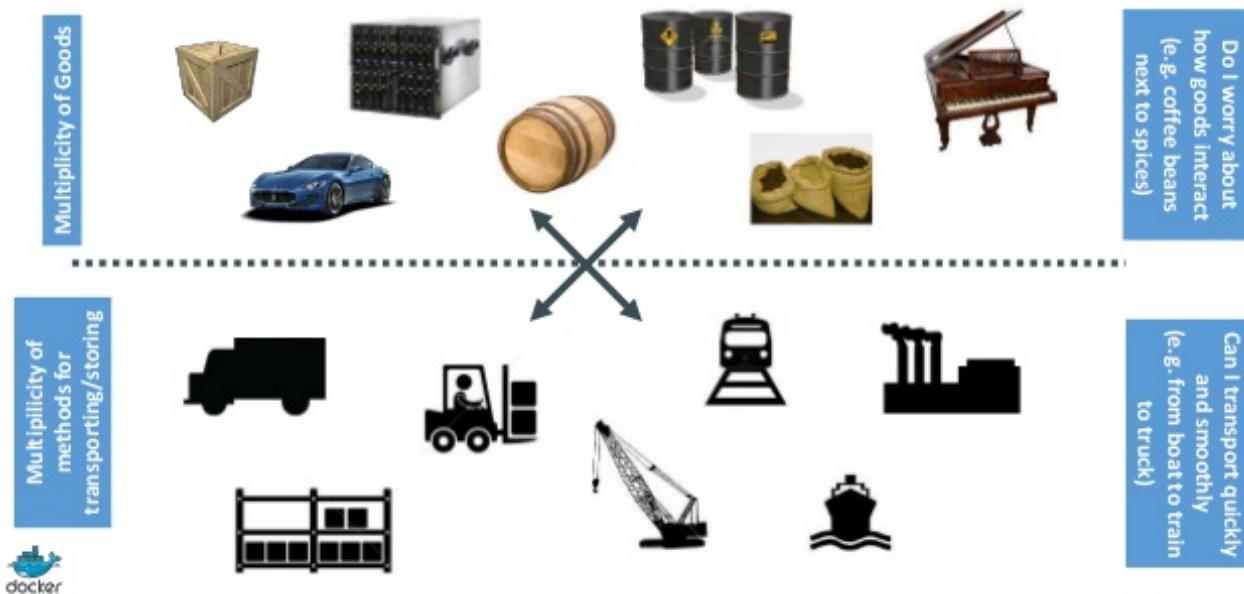
Présentation du concept de conteneur Linux

The Matrix of Hell

| | | | | | | | | |
|---|---|---|---|---|--|---|---|---|
|  | Static website | ? | ? | ? | ? | ? | ? | ? |
|  | Web frontend | ? | ? | ? | ? | ? | ? | ? |
|  | Background workers | ? | ? | ? | ? | ? | ? | ? |
|  | User DB | ? | ? | ? | ? | ? | ? | ? |
|  | Analytics DB | ? | ? | ? | ? | ? | ? | ? |
|  | Queue | ? | ? | ? | ? | ? | ? | ? |
| | Development VM | QA Server | Single Prod Server | Onsite Cluster | Public Cloud | Contributor's laptop | Customer Servers | |
|  |  |  |  |  |  |  |  | |

Présentation du concept de conteneur Linux

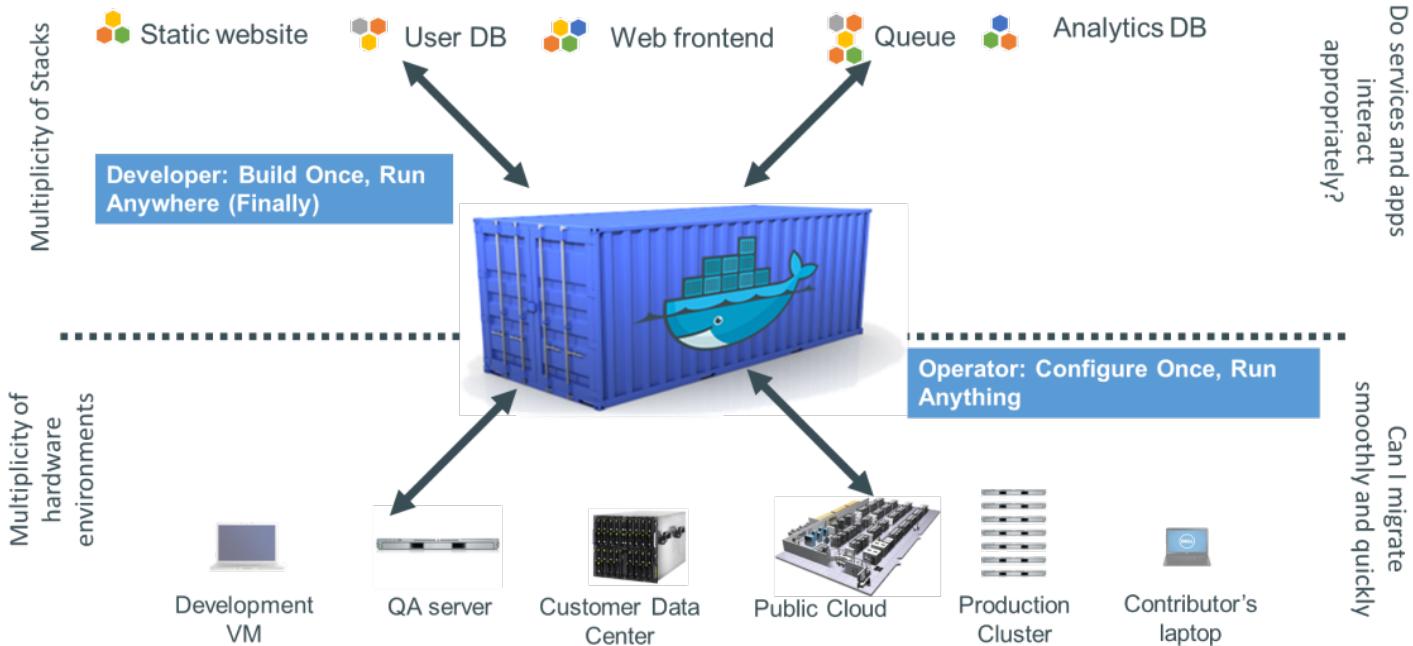
Cargo Transport Pre-1960



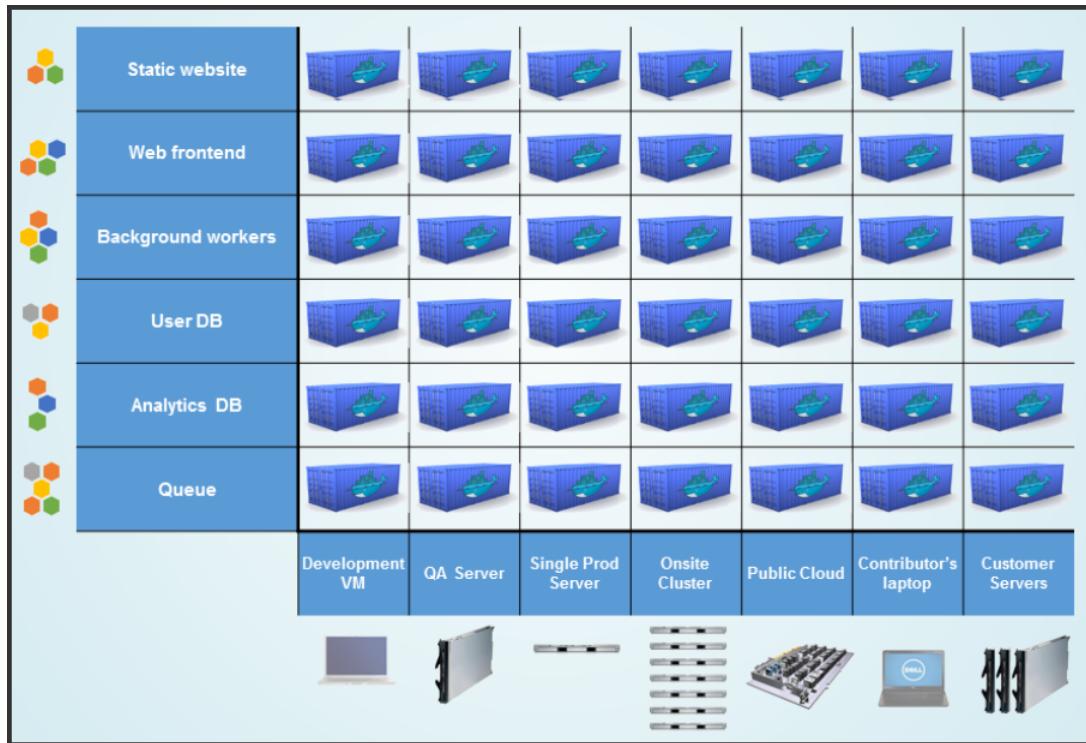
Présentation du concept de conteneur Linux



Présentation du concept de conteneur Linux



Présentation du concept de conteneur Linux



Présentation du concept de conteneur Linux

- Transformer ça ...



L'HISTORIQUE DES CONTENEURS



Les conteneurs

- 1979 : Apparition du chroot dans Unix 7
- 1991 : Le terme de Jail apparaît dans BSD
- 2003 : Premier Paas et SaaS basé sur le jail
- 2008 : Première version de LXC, première apparition du Container
- **2013 : Création de Docker**
- **2015 : Kubernetes**

- **En quelques dates**
 - 2012 : Fondation de dotcloud
 - Mars 2013 : Présentation à la PyCon de Santa Clara
 - Septembre 2013 : Partenariat avec Red Hat pour la solution OpenShift
 - Novembre 2014 : Intégration de Docker chez AWS EC2.
 - Mai 2016 : principaux contributeurs Cisco, Google, IBM, Microsoft et Red Hat
 - 2017 : Plus de 13 milliards de téléchargement sur l'année
 - 2018-19 : Docker penne à percer plus loin que le poste de dev
 - 2019 : Docker revend sa partie « services » à Mirantis

TECHNOLOGIES DES CONTENEURS DOCKER



NOTIONS D'ISOLATION - CGROUPS



Control Groups

- Fonctionnalité du noyau Linux pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.).
 - Limitation des ressources
 - Priorisation
 - Comptabilisation
 - Contrôle

NOTIONS D'ISOLATION - NAMESPACES



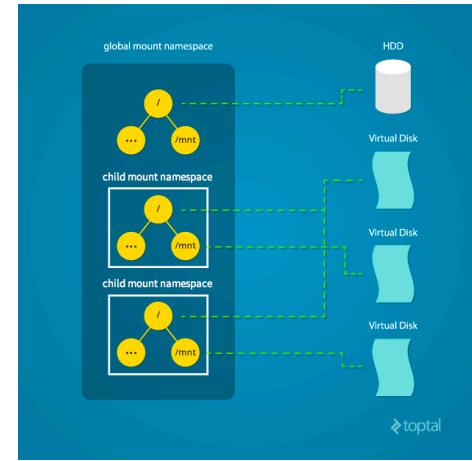
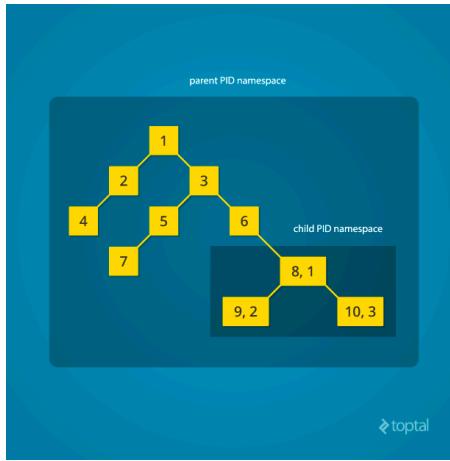
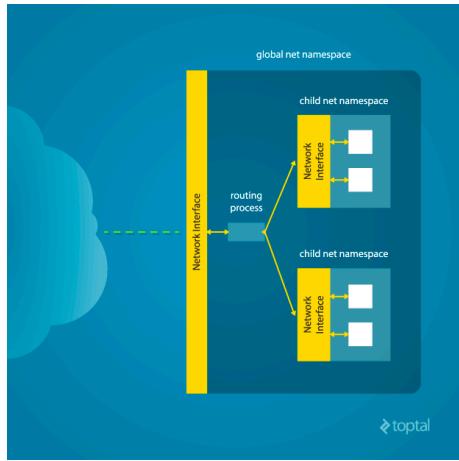
Namespaces

- Les espaces de noms sont une fonctionnalité du noyau Linux qui partitionne les ressources du noyau de sorte qu'un ensemble de processus voit un ensemble de ressources, tandis qu'un autre ensemble de processus voit un ensemble de ressources différent.

| | Mount namespace | Mount Points |
|----------------------------|--------------------------|---|
| Linux 2.6.19 - 29 Nov 2006 | UTS namespace | Hostname |
| | IPC namespace | Interprocess communication |
| Linux 2.6.24 - 24 Jan 2008 | PID namespace | Processes in different PID namespace can have the same PID |
| | Network namespace | Network devices, IP addresses, routing tables, iptables entries |
| Linux 3.8 - 18 Feb 2013 | User namespace | Root privileges for operations inside a user namespace, but unprivileged outside the namespace. Number of Linux filesystems are not yet user-namespace aware. |



Namespaces

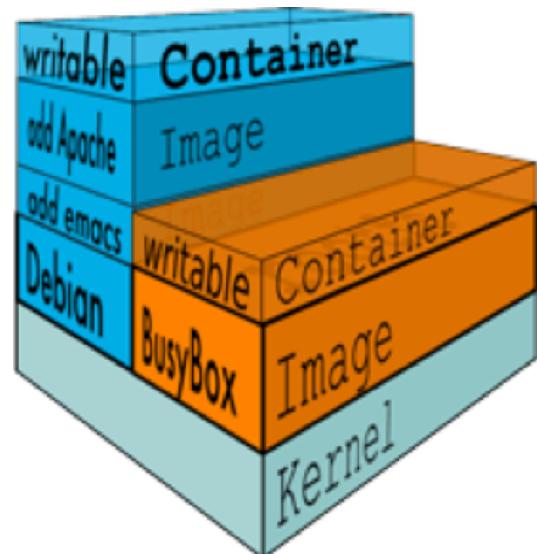


FILESYSTEMS



Layers

- Docker implémente un système de filesystem en "couches"
- Elles sont partagées entre les containers sur un même host
- Les couches d'une image sont en lecture seule
- Une couche vide en écriture est montée par dessus lors de l'instanciation d'un container
- 1 instruction dans le Dockerfile = 1 couche



LES CONTENEURS PAR RAPPORT À LA VIRTUALISATION



LES DIFFÉRENTS TYPES DE VIRTUALISATION



Les différents types de virtualisation

- « La virtualisation consiste à exécuter sur une machine hôte dans un environnement isolé des systèmes d'exploitation — on parle alors de virtualisation système — ou des applications — on parle alors de virtualisation applicative. » (wikipedia)

Les différents types de virtualisation

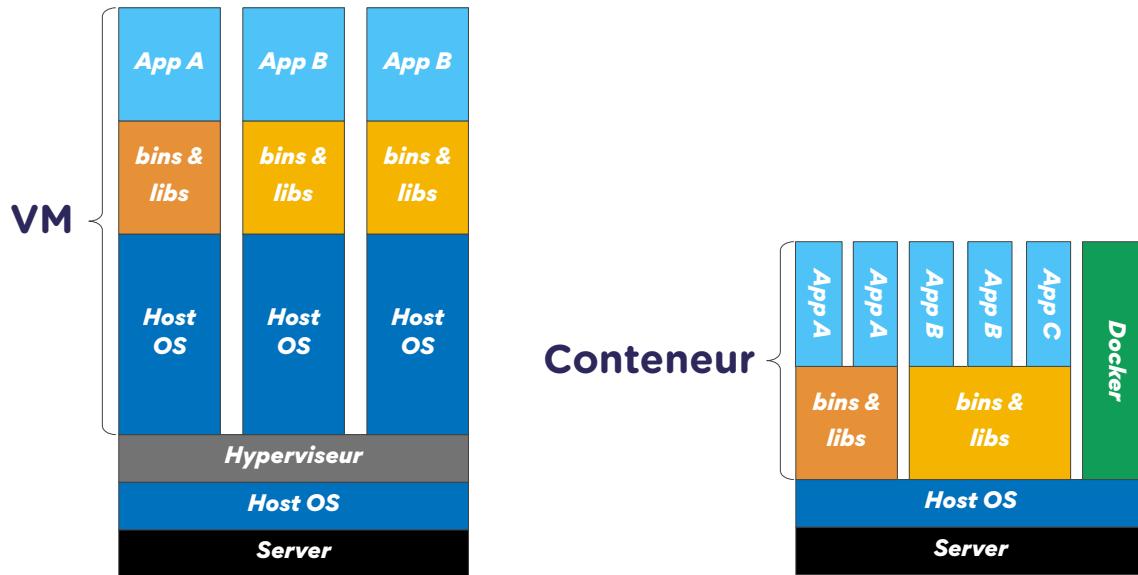
- 4 types de virtualisation différents
 - Système
 - Type I: Execution en direct sur le matériel (VMWare ESXi / Xen ...)
 - Type II: Exécution via un OS hôte (Virtualbox, VMWare player ...)
 - Applicative
 - Noyau isolé en espace utilisateur
 - Logiciel d'isolation basés sur des fonctions noyau
- Les containers (et Docker) se placent dans la dernière catégorie: La virtualisation applicative de type Isolation
- Le paradigme de la virtualisation est remonté du niveau de l'OS à l'applicatif



MACHINES VIRTUELLES VS CONTAINERS



Machines Virtuelles vs Conteneurs



LES LIMITES DES CONTENEURS



Les limites

- Moins forte isolation (mais bonne quand même)
 - du réseau
 - des process
- Impossible d'émuler un autre OS
- Applications GUI ne sont pas portables dans des conteneurs (pour l'instant)
 - moins impactant sous linux
- 1 seul process
- Orienté "application" plutôt que "système"



CAS D'USAGE DES CONTENEURS

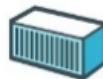


Cas d'usage des conteneurs

Le but ultime de Docker est de minimiser le temps et les infrastructures entre le développement, les tests, le déploiement et l'utilisation en production.



Build



Ship



Run

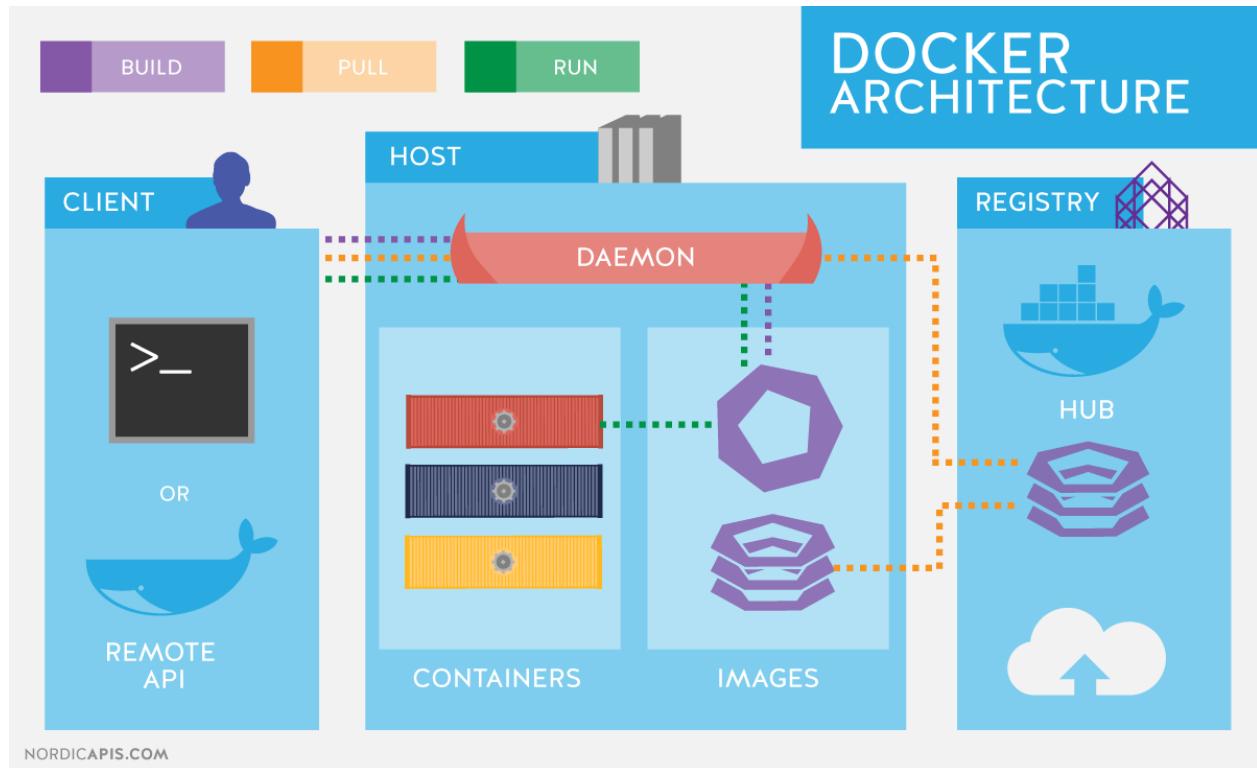
Cas d'usage des conteneurs

- Hébergement traditionnels
- CI/CD
- Micro-Services
- Embarqué
- IOT
- Tout ?

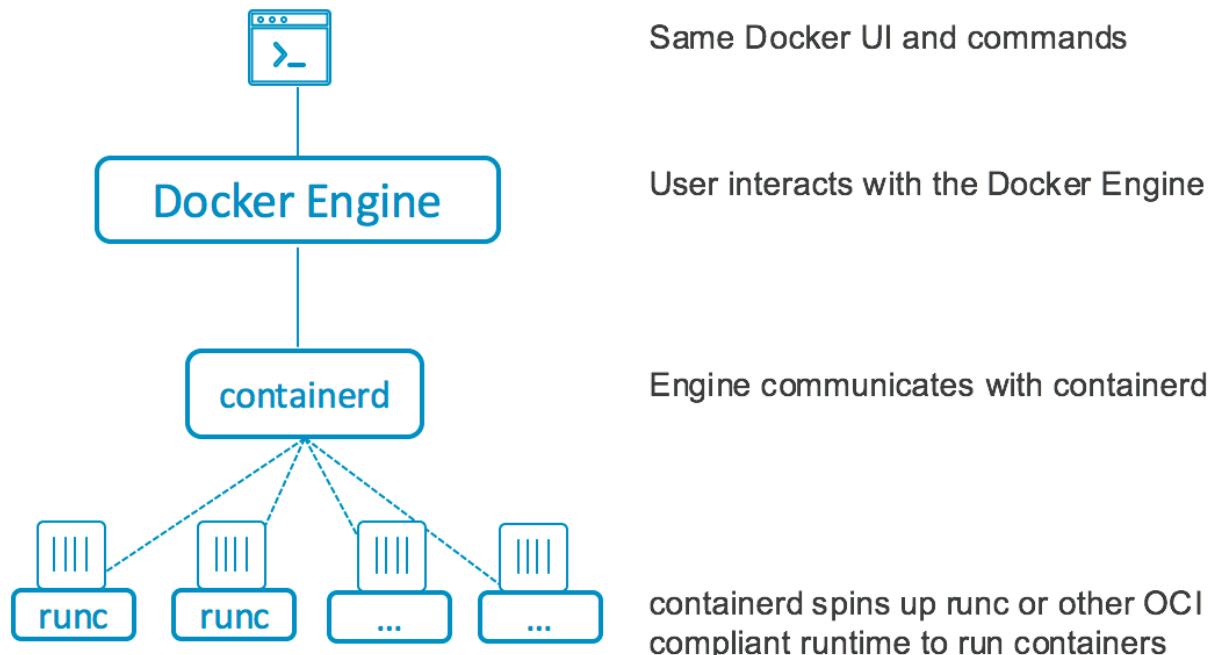
COMPOSANTS D'UNE INFRASTRUCTURE DE CONTENEURS



Architecture Docker

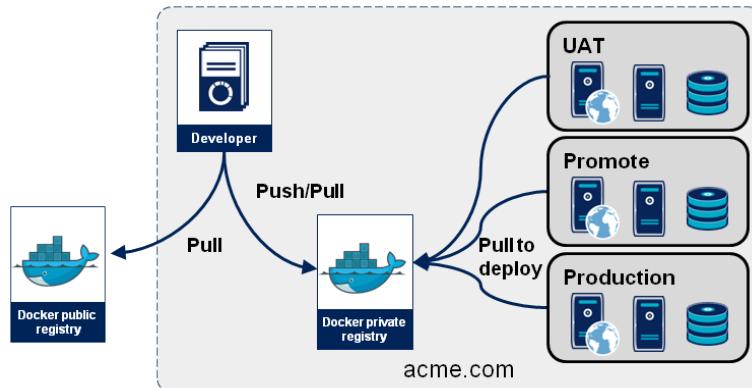


Architecture Docker



Registry

- Entrepôt où l'on stocke les images en attendant de les utiliser
- Elle est publique (docker hub) ou privée (harbor, etc...)



LES ORCHESTRATEURS DE CONTENEURS



Les orchestrateurs

- “Processus automatique d'organisation, de coordination, et de gestion de systèmes informatiques complexes.
- Ils définissent les politiques et les niveaux de service grâce à des flux de travail automatisés.”

Les orchestrateurs



kubernetes

Les orchestrateurs

- Se chargent de gérer l'aspect multi-hosts des déploiements de conteneurs tout en fournissant un package de services
- Dans la terminologie conteneurs on parle de cluster dès que l'on commence à avoir plusieurs nodes qui exécutent une workload donnée
- Se pose alors des problèmes de gestion du multi-hosts
 - Volumes partagés
 - Co-scheduling de containers (sur le même host)
 - Equilibrage de charge (CPU / RAM)
 - Service discovery (comment trouver tous les containers d'un même service)
 - Plan d'adressage commun
 - Routage des requêtes entrantes
 - Routage des requêtes internes



INTRODUCTION À KUBERNETES

Présentation de Kubernetes

- Historique
- Google et Kubernetes
- Linux Foundation et CNCF
- Les autres contributeurs

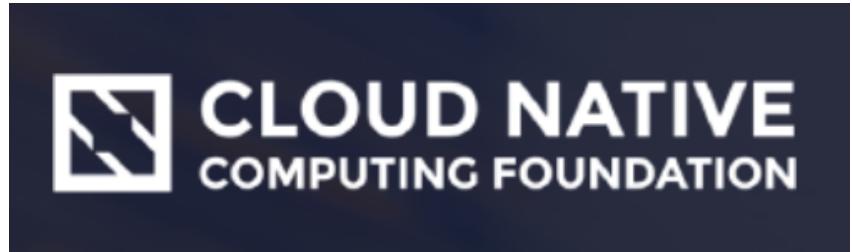
- Un orchestrateur pour piloter les architectures micro-services
 - Applicatifs
 - Serveurs
 - Routage
 - Certificats
 - Stockage
 - Dimensionnement
 - A peu près ce que vous voulez ...

HISTORIQUE



- k-u.b.e.r.n.e.t.e-s : 8 lettres entre le 'k' et le 's' : K8S
- En grec ancien (oui !) = Timonier
- Développé en interne par google
 - Borg puis Omega
- Opensourcé en 2014
- Récupéré par la CNCF
- Diplômé en Avril 2018 !
- Adoption rapide et massive

-
- Cloud Native Computing Foundation
 - “La mission de la fondation est de créer et de favoriser l’adoption d’un nouveau paradigme informatique optimisé pour les environnements de systèmes distribués modernes capables de se dimensionner jusqu'à des dizaines de milliers de nœuds multi-tenant et résiliants”



- Pré-Requis
 - Distribuer sous forme de conteneurs
 - Gestion dynamique de la configuration
 - Orienté micro services
- Rôles
 - Intendance des projets
 - Faire grossir et évoluer l'écosystème
 - Rendre la technologie accessible
 - Promouvoir la technologie

CNCF Graduated Projects (25)

| | | | | | | | | | |
|---|--|--|--|---|---|---|---|--|--|
|  argo Argo Cloud Native Computing Foundation (CNCF) ★ 14,145 Funding: \$3M |  cilium Cilium Cloud Native Computing Foundation (CNCF) ★ 16,516 Funding: \$3M |  containerd containerd Cloud Native Computing Foundation (CNCF) ★ 14,857 Funding: \$3M |  CoreDNS CoreDNS Cloud Native Computing Foundation (CNCF) ★ 11,071 Funding: \$3M |  cri-o cri-o Cloud Native Computing Foundation (CNCF) ★ 4,754 Funding: \$3M |  envoy Envoy Cloud Native Computing Foundation (CNCF) ★ 22,759 Funding: \$3M |  etcd etcd Cloud Native Computing Foundation (CNCF) ★ 44,537 Funding: \$3M |  fluentd Fluentd Cloud Native Computing Foundation (CNCF) ★ 12,199 Funding: \$3M |  flux Flex Cloud Native Computing Foundation (CNCF) ★ 5,300 Funding: \$3M |  HARBOR Harbor Cloud Native Computing Foundation (CNCF) ★ 20,890 Funding: \$3M |
|  HELM HELM Cloud Native Computing Foundation (CNCF) ★ 24,952 Funding: \$3M |  Istio Istio Cloud Native Computing Foundation (CNCF) ★ 33,694 Funding: \$3M |  JAEGER Jaeger Cloud Native Computing Foundation (CNCF) ★ 18,318 Funding: \$3M |  KEDA KEDA Cloud Native Computing Foundation (CNCF) ★ 4,831 Funding: \$3M |  KEDA (Genetics) KEDA (Genetics) Cloud Native Computing Foundation (CNCF) ★ 6,821 Funding: \$3M |  kubernetes Kubernetes Cloud Native Computing Foundation (CNCF) ★ 101,881 Funding: \$3M |  LINKERD Linkerd Cloud Native Computing Foundation (CNCF) ★ 9,881 Funding: \$3M |  Open Policy Agent (OPA) Open Policy Agent (OPA) Cloud Native Computing Foundation (CNCF) ★ 8,496 Funding: \$3M |  Prometheus Prometheus Cloud Native Computing Foundation (CNCF) ★ 49,987 Funding: \$3M |  ROOK Rook Cloud Native Computing Foundation (CNCF) ★ 11,830 Funding: \$3M |
|  spiffe spiffe Cloud Native Computing Foundation (CNCF) ★ 1,211 Funding: \$3M |  SPIRE SPIRE Cloud Native Computing Foundation (CNCF) ★ 1,392 Funding: \$3M |  TUF The Update Framework (TUF) Cloud Native Computing Foundation (CNCF) ★ 1,541 Funding: \$3M |  KV KV Cloud Native Computing Foundation (CNCF) ★ 13,617 Funding: \$3M |  Vitess Vitess Cloud Native Computing Foundation (CNCF) ★ 16,897 Funding: \$3M | | | | | |



- Open Containers Initiative
- Crée sous la Linux Fondation
- But : Créer un standard Open Source concernant la manière de "runner" et le format des conteneurs et images
- Non lié à des produits Non lié à des COE (Container Orchestration Engine)
- runC a été donné par Docker à l'OCI comme implémentations de base



Cycle de release et support



Kubernetes n'a pas de version LTS à ce jour

Une nouvelle version mineure de Kubernetes tous les 3 mois (soit 4 / an)



Seules 3 versions mineures officiellement supportées en même temps

Il est nécessaire de mettre à jour le cluster Kubernetes avant que la version ne soit plus supportée. A ce jour les fonctionnalités dites "stable" sont toujours rétro-compatibles lors d'une mise à jour mineure.

GOOGLE ET KUBERNETES





kubernetes.io



donne
K8S
en
2015



pour
former



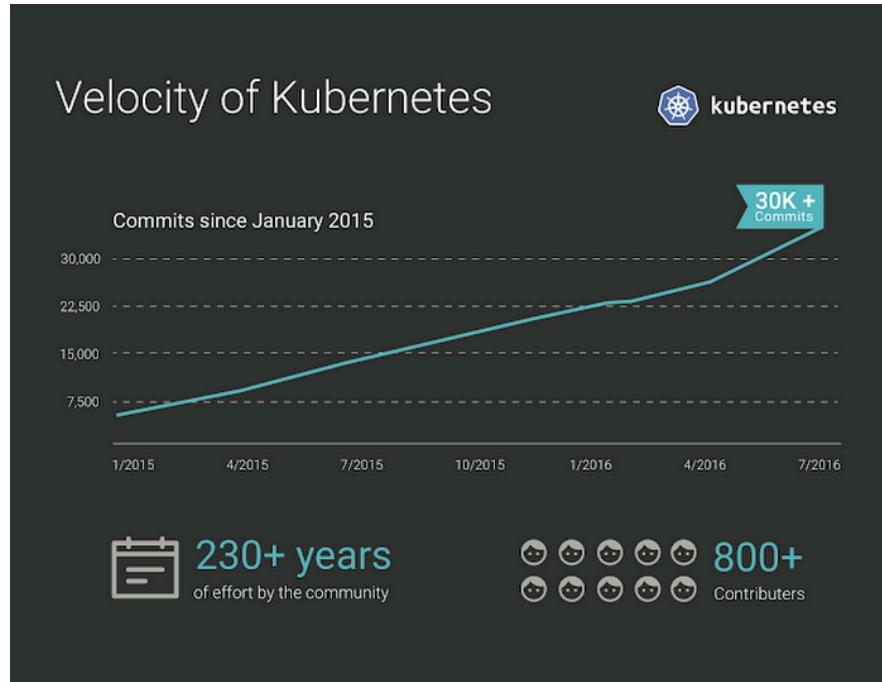
CLOUD NATIVE
COMPUTING
FOUNDATION

Google et Kubernetes

- Borg puis Omega utilisés pour faire tourner les principaux services Google
 - Search
 - Gmail
 - YouTube
 - Docs
 - Etc ...
- En 2015 google reverse une partie du code source de Borg à la communauté
- L'engouement est rapide et fulgurant (notamment à partir de 2017)



Google et Kubernetes



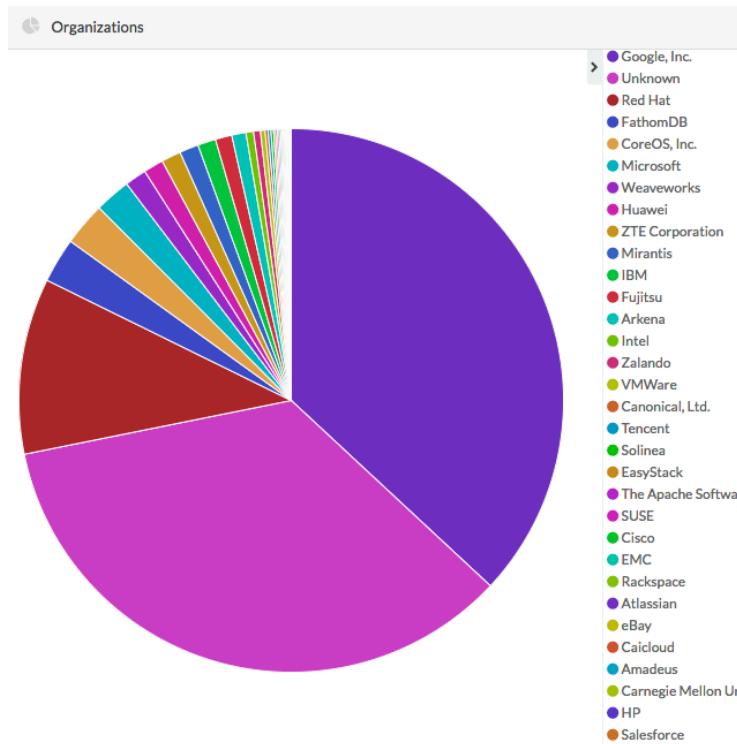
LES AUTRES CONTRIBUTEURS : RED HAT...



Les autres contributeurs

- En 2014 Microsoft, Docker, RedHat et IBM rejoignent la communauté Kubernetes (cette alliance sera la base de la CNCF)
- La majorités des contributions ne sont pas reliés à des sociétés en particulier
- Parmi les sociétés les plus actives
 - RedHat
 - Meteor
 - CoreOS
 - Microsoft
 - Weaveworks
 - Huawei

Les autres contributeurs



ARCHITECTURE DE KUBERNETES



Composants

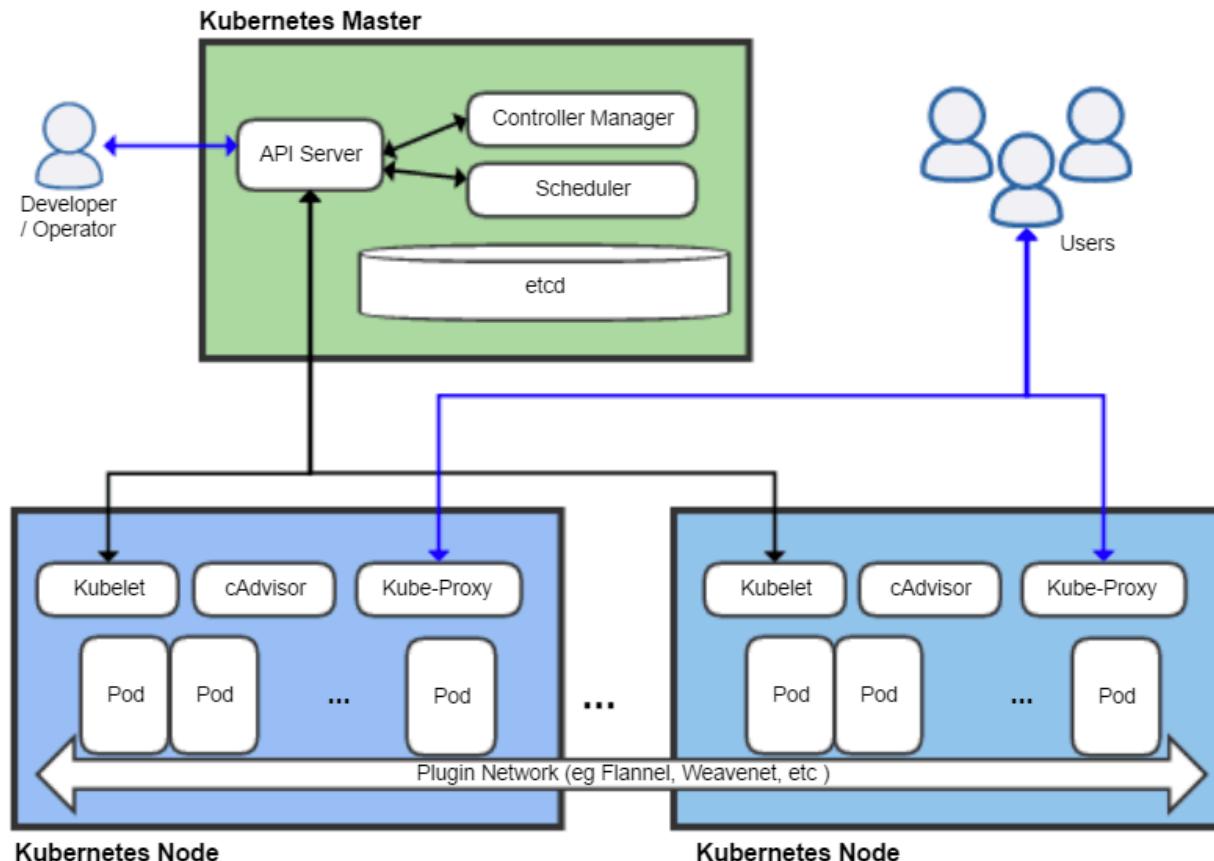
- Kubernetes est écrit en Go, compilé statiquement.
- Un ensemble de binaires sans dépendance
- Faciles à conteneuriser et à packager
- Peut se déployer uniquement avec des conteneurs sans dépendance d'OS

Composants 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



Composants control-plane



- 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

- 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

- 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.)

- Responsable de la publication de services
- Utilise iptables (par défaut)
- Route les paquets à destination des PODs et réalise le load balancing TCP/UDP

KUBE-CONTROLLER-MANAGER

- Boucle infinie qui contrôle l'état du cluster
- Effectue des opérations pour atteindre un état donné
- De base dans Kubernetes:
 - replication controller
 - endpoints controller
 - namespace controller
 - serviceAccounts controller
 - etc

- Transforme les ordres issues de l'API-server en actions auprès du runtime de conteneurs
- Présent sur tous les noeuds du cluster

- kubectl : Ligne de commande permettant de piloter un cluster Kubernetes

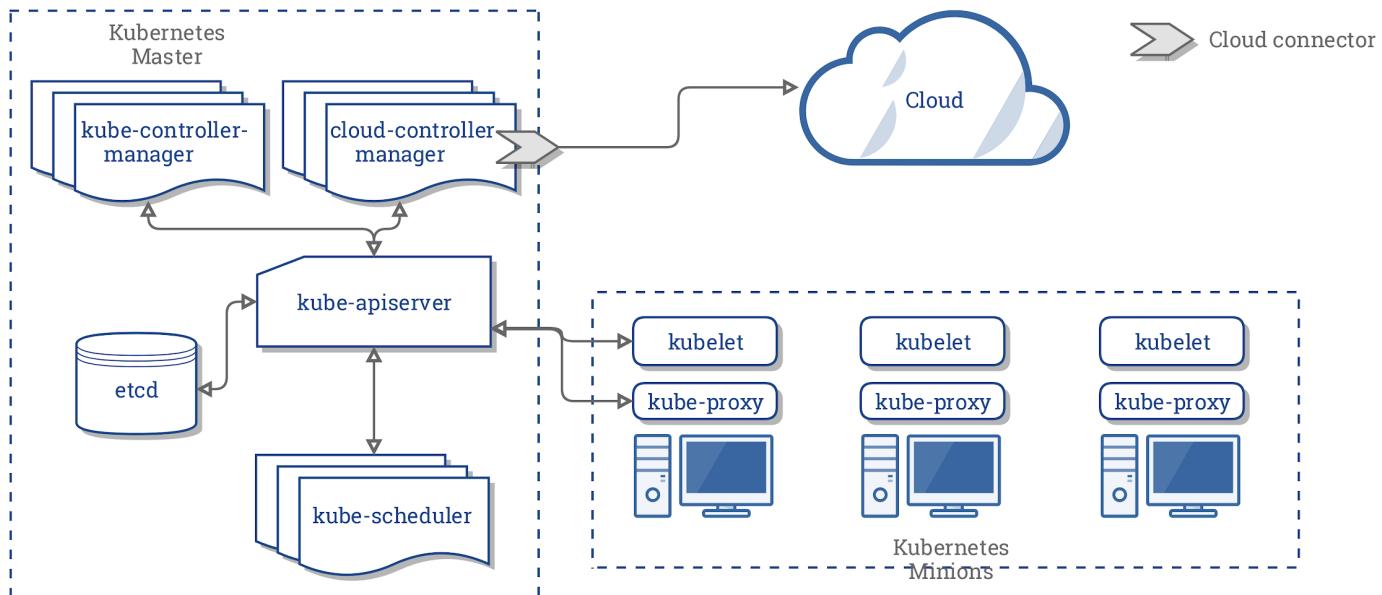
- Kubernetes n'implémente pas de solution réseau par défaut, mais s'appuie sur des solutions tierces qui implémentent les fonctionnalités suivantes:
 - Chaque pods reçoit sa propre adresse IP
 - Les pods peuvent communiquer directement sans NAT
- Exemples
 - Canal
 - Flannel
 - Calico
 - Cilium
 - etc ...

INTÉGRATION CLOUD



- Kubernetes propose différentes intégrations avec des vendeurs de cloud
 - Publics
 - Privés
 - Hybrides

Cloud Controller Manager



Ressources

- Load Balancers ← Services
 - Disques Bloc ← Persistent Volumes
 - Nodes ← Nodes
-
- Node controller
 - Volume controller
 - Route controller
 - Service controller

INSTALLATION ET CONFIGURATION



CHOISIR SON MODÈLE D'INSTALLATION



Choisir son modèle d'installation

- Il existe différentes façons d'installer kubernetes
- Techniquement il suffit
 - remplir les dossiers de configuration
 - de démarrer les binaires fournis
- Solutions
 - En local => pour tester et développer
 - On-Premises => Pour garder le contrôle sur son infrastructure
 - Cloud => Pour externaliser la gestion de k8s

- Minikube => L'outil officiel
- KIND => Kubernetes IN Docker
- Docker desktop
- Rancher Desktop
- k3s / k3d

On-Premise

- kubeadm => l'outil officiel
- kubespray => Basé sur ansible
- k3s
- k0s
- VMWare tanzu
- Rancher

Services managés des Cloud Providers

- EKS => AWS
- GKE => GCP
- AKS => Azure
- De nombreux autres
 - DOKS => Digital Ocean
 - OVH Kubernetes => OVH
 - Capsule => Scaleway
- En plus de profiter d'une externalisation du management du service, une des forces des services des cloud-providers sont l'intégration avec l'écosystème

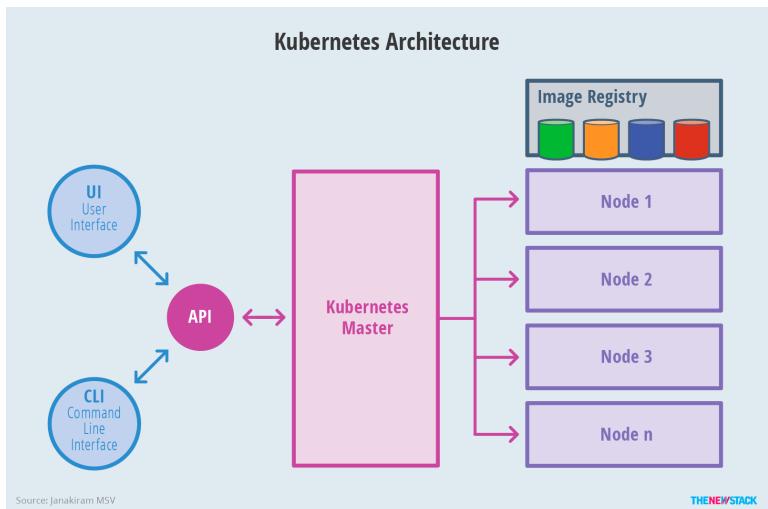


DASHBOARD, CLI ET API



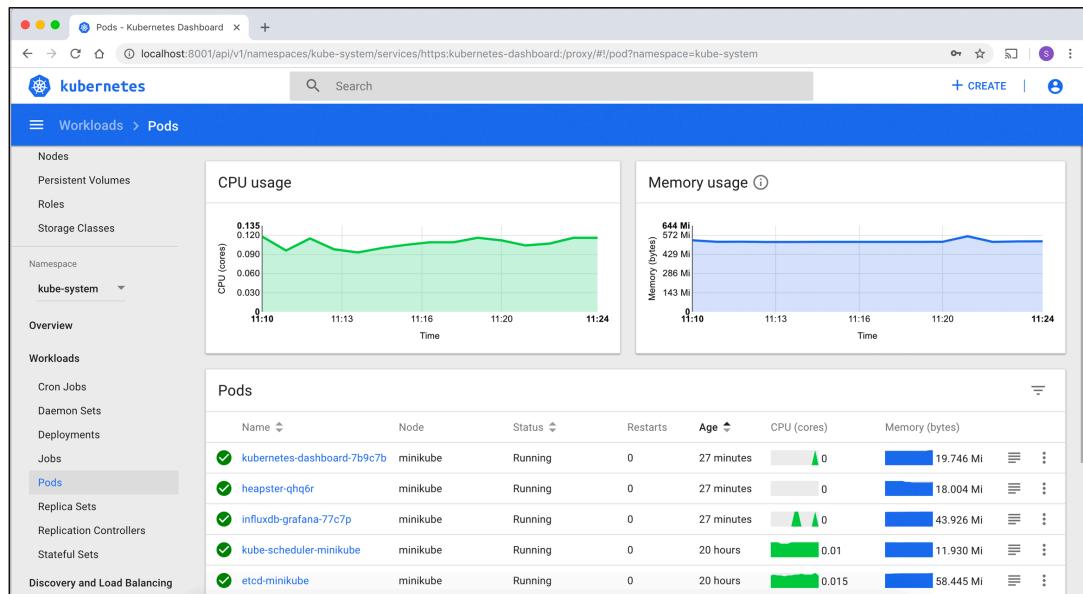
Dashboard, CLI et API

- Interfaces avec kubernetes
 - CLI : kubectl
 - Web : Dashboard
 - API: Permet d'interagir à distance en HTTP



Dashboard

- Nécessite un outil de collecte de métriques (metrics-server)
- Ne jamais l'exposer sur internet
- L'affichage dépend des droits du compte utilisé



- Kubectl, le client par défaut pour kubernetes

```
kubectl get nodes  
kubectl get pods -l app=nginx  
kubectl run -it web --image=nginx
```

- Tous les outils utilisent l'API
 - Kubectl
 - Dashboard
 - Rancher
 - Openshift
 - Gitlab
 - Etc ...
- On peut la requêter soi même
 - curl -H "Authorization: Bearer \$TOKEN" -kv https://localhost:56027/api/

CONCEPTS DE BASE



CONTEXTES



Contextes

- kubectl est configuré par défaut via le fichier `~/.kube/config` (sans extension)
- Peut être surchargé temporairement par la variable d'environnement **KUBECONFIG**
- Ce fichier peut contenir plusieurs **contextes**
- **Un contexte = une entrée de configuration qui agit sur**
 - le cluster ciblé
 - l'utilisateur sélectionné
 - le namespace où l'on travaille
- On peut changer de contexte via une commande

```
apiVersion: v1
kind: Config
clusters:
- name: kind
  cluster:
    server: https://localhost:50323
    certificate-authority-data: >-
      LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1i
    insecure-skip-tls-verify: false
users:
- name: kubernetes-admin
  user:
    client-certificate-data: >-
      LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1i
    client-key-data: >-
      LS0tLS1CRUdJTiBSU0EgUFJJVFKFURSBLRVktLS0t
contexts: []
preferences: {}
current-context: kubernetes-admin@kind
```



- Afficher les contextes disponibles

```
>_
```

```
kubectl config get-contexts
```

ou

```
kubectl ctx
```

- Sélectionner un contexte

```
>_
```

```
kubectl config use-context moncontext
```

ou

```
kubectl ctx moncontext
```

NAMESPACES



Namespaces

- Les namespaces sont des espaces permettant de gérer des clusters virtuels au sein d'un cluster commun
- Par défaut il existe dans un cluster 3 namespaces :
 - default => Environnement utilisateur par défaut
 - kube-public => Environnement accessible depuis les autres namespaces
 - kube-system => Environnement dédié aux applications système



Namespaces

- Les namespaces servent
 - à la segmentation des droits des utilisateurs (via RBAC)
 - à l'isolation de charges de travail (via les Network Policies)
 - à la réutilisation de noms de ressources
 - 2 namespaces différents peuvent avoir un pod nommé "monpod"
 - De « poignées » pour cibler certains objets
 - Security Policies
 - Quotas
 - Etc ...



Namespace

- Afficher les namespaces présents dans le cluster

```
>_
```

```
kubectl get ns
```

ou

```
kubectl ns
```

- Création d'un namespace

```
>_
```

```
kubectl create ns monns
```

- Configurer le contexte actuel pour utiliser ce namespace par défaut

```
>_
```

```
kubectl config set-context --current --namespace=monns
```

ou

```
kubectl ns monns
```

DÉMARRER SES PREMIERS CONTENEURS



Création d'un pod

Créer un simple Pod nginx nommé « monpod »

```
>_
```

```
kubectl run monpod --image=nginx
```

Voir l'état du Pod nommé « monpod »

```
>_
```

```
kubectl get pod monpod
```

Voir l'état détaillé du Pod

```
>_
```

```
kubectl describe pod monpod
```

Voir le manifest correspondant une fois le pod lancé

```
>_
```

```
kubectl get pod monpod -o yaml
```

VISUALISER LES OBJETS KUBERNETES



Visualiser les Objets Kubernetes

- Tous les objets dans kubernetes vont pouvoir se requêter via une interface homogène
 - Le format est quasiment toujours le même
 - **kubectl <verb> <type-objet> <identifiant> <options>**
- L'affichage en revanche va dépendre du type d'objet ciblé

GÉRER LES OBJETS KUBERNETES



Visualiser les Objets Kubernetes

- Voir l'état d'un pod

```
>_
```

```
kubectl get pod monpod
```

- Voir l'état d'un pod avec des colonnes détaillées

```
>_
```

```
kubectl get pod monpod -o wide
```

Visualiser les Objets Kubernetes

- Voir le manifest correspondant à un pod

```
>_
```

```
kubectl get pod monpod -o yaml
```

- Obtenir de l'information détaillée sur un pod

```
>_
```

```
kubectl describe pod monpod
```

- Surveiller les changements d'un pod

```
>_
```

```
kubectl get pod monpod --watch
```

```
> k get pod --watch
NAME      READY   STATUS            RESTARTS   AGE
monpod   0/1     ContainerCreating   0          10s
monpod   1/1     Running           0          34s
```

CRÉATION MISE À JOUR



- Crée un objet



```
kubectl create configmap maconfig --from-literal secret=1234
```

- Mettre à jour un objet



```
kubectl edit configmap maconfig
```



```
kubectl patch configmap maconfig --patch '{"data": {"secret":
```

SUPPRESSION



- Supprimer un objet

```
>_
```

```
kubectl delete configmap maconfig
```

CLEANUP



Cleanup

- Supprimer un namespace ainsi que son contenu

```
>_
```

```
kubectl delete namespace mon-ns
```

- Supprimer toute les ressources d'un type donné

```
>_
```

```
kubectl delete pods --all
```

- Supprimer toutes les workloads

```
>_
```

```
kubectl delete all --all
```

TROUBLESHOOTING



Techniques de déboggage des pods

- Voir les logs dans un Pod

```
>_ kubectl logs -f appcache -c app
```



- Lancer une commande dans un conteneur d'un Pod

```
>_ kubectl exec -it --container=app appcache -- sh
```

- Lancer un pod éphémère à des fins de debug

```
>_ kubectl run -it --rm debug --image=busybox -- sh
```



Techniques de déboggage des pods

- Voir l'objet tel qu'il est persisté dans l'API

```
>_
```

```
kubectl get pod monpod -o yaml # ou json
```

- Voir l'état de l'objet détaillé

```
>_
```

```
kubectl describe pod monpod
```

- Surcharger l'entrypoint utilisé pour pouvoir debugger

```
>_
```

```
kubectl run debug --image buggy --command /bin/sleep - 1d
```

LABELS ET ANNOTATIONS



LABELS



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
- Crètent un lien entre différents objets kubernetes
 - Exemple un service vers des pods
 - Exemple un replicaSet et ses pods enfants
 - Exemple un type de noeud et sa workload

| | |
|--------|--|
| Labels | app.kubernetes.io/instance=echo-server |
| | app.kubernetes.io/name=echo-server |
| | pod-template-hash=788c5c8ff8 |



- Appliquer un label

```
>_
```

```
kubectl label pod monpod foo=bar
```

- Supprimer un label

```
>_
```

```
kubectl label pod monpod foo-
```



Pods: Utilisation des labels

- **Voir les labels d'un job**

```
>_
```

```
kubectl get pod --show-labels
```

- **Voir les pods qui partagent ce label**

```
>_
```

```
kubectl get pod -l foo=bar
```

LABEL SELECTOR



Label selector

- Sert à certains objets à trouver les objets avec lesquels ils interagissent:
 - Ses enfants
 - Ses relations
 - Ses pairs
 - Un pool de noeuds spécifiques
- C'est un mécanisme utilisé intensivement en interne par kubernetes
- Supporte des expressions complexes

| | |
|----------|--|
| Selector | app.kubernetes.io/instance=echo-server |
| | app.kubernetes.io/name=echo-server |

Label selector

```
< ... >
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-rs
  template:
    metadata:
      labels:
        app: demo-rs
  spec:
    containers:
    - name: web
      image: nginx:latest
```



Label selector

```
< ... >
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [primary, slave]}
    - {key: environment, operator: NotIn, values: [dev]}
< ... >
```



ANNOTATIONS



Annotations

- Ce sont des metadata arbitraires attachés aux objets kubernetes
- Ces annotations peuvent être utilisées par d'autres composants ou par des logiciels tiers extérieurs
- Les labels servent à identifier les objets
- Les annotations à donner du contexte aux objets

| Annotations | deployment.kubernetes.io/revision=1 |
|-------------|--|
| | meta.helm.sh/release-name#echo-server |
| | meta.helm.sh/release-namespace#echo-server |



- Annoter un objet



```
kubectl annotate pod monpod com.webofmars=teamA
```

- Retirer l'annotation d'un objet



```
kubectl annotate pod monpod com.webofmars-
```

Annotations

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    name: myapp
  annotations:
    com.webofmars.team: 'teamA'
spec:
  containers:
  - name: webofmars
    image: nginx
```



Les principaux objets de l'API

Basic resources

Pod / Containers

Controllers

Deployment

DaemonSet

StatefulSet

Exposition

Service

Ingress Controller

Ingress Rule

Storage resource

Volume

Volume Claim

Storage Class

Jobs

Job

CronJob

Security

Namespace

Service Account

Cluster Role

...

Configuration

ConfigMap

Secret

Custom resources

Custom Resource Definition



WORKLOADS



PODS



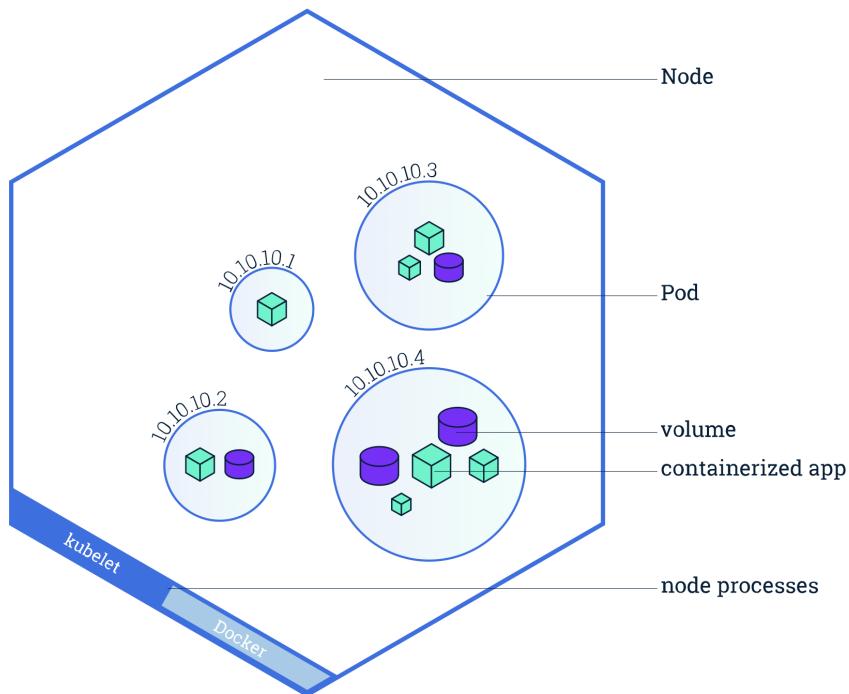
- Ensemble logique composé de un ou plusieurs conteneurs
- Les conteneurs d'un pod fonctionnent ensemble (instanciation et destruction) et sont orchestrés sur un même hôte
- Les conteneurs partagent certaines spécifications du POD
 - La stack IP (network namespace)
 - Inter-process communication (IPC namespace)
 - Volumes (mount namespace)
- C'est la plus petite unité orchestrable dans Kubernetes

Pods

Le pod est la plus petite entité qui peut être orchestrée sur un cluster Kubernetes

Un pod contient des conteneurs et des volumes qui partagent le même espace réseau

Deux conteneurs d'un même pod peuvent donc se joindre via l'adresse 127.0.0.1 et peuvent partager des fichiers via un volume



- En plus de leur spécifications les containers vont pouvoir individuellement définir certains paramètres
 - Ressources utilisées (CPU et mémoire)
 - Requests = Réservation au démarrage
 - Limits = Maximum que le conteneur peut utiliser
 - Sondes de vie
 - Readiness Probe
 - Exécutée régulièrement pendant le démarrage jusqu'à réussite
 - Liveness Probe
 - Exécutée régulièrement après le démarrage afin de s'assurer que le conteneur est toujours en bonne santé
 - En cas d'échec le conteneur est redémarré automatiquement



CRÉER UN POD ET UN POD MANIFEST



Créer un pod et un pod manifest

- **Créer un pod**

```
>_
```

```
kubectl run --image=nginx pod-un
```

Créer un pod et un pod manifest

- **Créer un pod**

```
>_
```

```
kubectl apply -f mon-pod.yaml
```

```
# mon-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: monpod
spec:
  containers:
    - name: web
      image: nginx
```

GÉRER LES PODS (*LISTER, SUPPRIMER...*)



Gérer les pods (*lister, supprimer...*)

- Lister les pod



```
kubectl get pods
```

- Voir un pod donné



```
kubectl get pod mon-pod
```

- Lister les pods avec filtrage



```
kubectl get pods -l foo=bar
```

ACCÉDER À UN POD



Accéder à un pod

- Accéder à un pod - via port forward

```
>_
```

```
kubectl port-forward pod/monpod 8080:80
```

- Accéder à un pod - logs

```
>_
```

```
kubectl logs -f monpod
```

- Accéder à un pod - via exec

```
>_
```

```
kubectl exec -it monpod -- sh
```

- Accéder à un pod - copie de fichiers

```
>_
```

```
kubectl cp ./data.tgz monpod:/data
```

Récupérer les logs des pod

- Affiche les derniers logs du container “logs” d'un pod (dans le cas d'un pod multi containers)

```
>_
```

```
kubectl logs -c logs monpod
```

- Affiche les 20 derniers logs et suivants du pod

```
>_
```

```
kubectl logs -f --tail=20 pod monpod
```

PODS: NOTIONS SUPPLÉMENTAIRES



HEALTHCHECKS



- **Liveness Probe**
 - Exécutée dès le démarrage du pod
 - Jusqu'à qu'elle réussisse
 - But: Vérifie si le container est « vivant » et le redémarrer si besoin
 - exemple: "curl <http://localhost:80/health>"
- **Liveness Probe**
 - Exécutée après démarrage du pod
 - But: Vérifie si le **container** est « vivant » et le **redémarrer** si besoin
 - exemple: "ps aux | grep nginx"
- **Readiness Probe**
 - Exécuter au démarrage du pod
 - Vérifie si le container est prêt à recevoir du trafic supplémentaire
 - exemple: "curl <http://localhost:80/healthz>"

Healthchecks

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: myapp
      image: nginx:latest
      livenessProbe:
        httpGet:
          path: /
          port: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: myapp
      image: nginx:latest
      readinessProbe:
        tcpSocket:
          port: 80
```



RESSOURCES



- Les requests et limits vont s'exprimer en unités fixes
 - CPU: milicores
 - Memory: Octets ou Octets internationaux
- On utilise les 2 conjointement (requests et limits)
- Influe sur la QoS d'un pod

Requests

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: myapp
    image: nginx
    resources:
      requests:
        memory: "128Mi"
        cpu: "500m"
```



Limits

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: myapp
    image: nginx
    resources:
      limits:
        memory: "256Mi"
        cpu: "1000m"
```



ENVIRONNEMENT



Configuration de l'environnement

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
    - name: myapp
      image: nginx
      env:
        - name: DATABASE_HOST
          value: 192.168.7.3
```



INITCONTAINERS



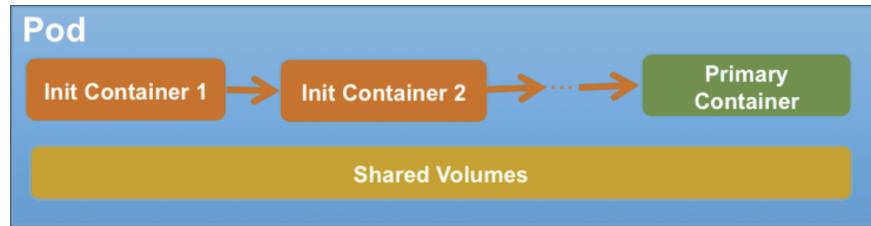
les conteneurs d'initialisation : initContainers

- Les initContainers sont des conteneurs qui vont s'exécuter dans un Pod avant que les conteneurs principaux ne démarrent
- Ils peuvent servir à :
 - attendre la disponibilité d'une ressource
 - préparer des fichiers dans un volume
 - migrer des données
 - etc.
- Globalement, ils sont utiles pour découpler les phases de préparation des phases de run



les conteneurs d'initialisation : initContainers

- Lancés avant le démarrage du/des containers principaux
- Suive un ordre précis de démarrage (successif)
- Doivent “réussir” (exitCode 0)



les conteneurs d'initialisation : initContainers

```
# mon-app.yaml
apiVersion: v1
kind: Pod
metadata:
  name: monapp
  labels:
    run: monapp
spec:
  containers:
    - name: monapp
      image: nginx
  initContainers:
    - name: init-wait
      image: busybox
      args:
        - "sleep"
        - "10"
```



CONFIGMAP ET SECRETS



ConfigMaps

- Listing de valeurs de configuration externes à l'application
- Permet de découpler les containers et leur configuration au runtime
- On peut y stocker :
 - Des enregistrements clés-valeurs
 - Des fichiers de configuration entier
- Attention limite de taille à 1 Mo (etcd)
- Une même configMap peut être utilisée / partagée par plusieurs pods



- Créer une configMap



```
kubectl create configmap maconfig --from-literal user=marcel
```

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: maconfig
data:
  user: 'marcel'
```

- Idem que configMap mais les valeurs sont encodées en base64
- Pas exposés par get ou describe directement
- C'est là l'intérêt des secrets
- Au delà il faut l'encoder soi même ou utiliser des outils comme Hashicorp Vault
- 3 types
 - Literral (classique)
 - Certificat
 - Docker Registry

- **Créer un secret**

>_

```
kubectl create secret generic credentials \
--from-literal password=1234
```

Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: credentials
data:
  password: MTIzNA==
```

Utilisation des configMaps et secrets

- Les configMaps et secrets sont utilisés pour
 - importer des variables d'environnements dans un pod
 - monter des fichiers de configurations dans un pod
 - éventuellement importer des scripts dans un pod

Injecter une valeur

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: nginx
    env:
      - name: USERNAME
        valueFrom:
          configMapKeyRef:
            name: maconfig
            key: username
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: credentials
            key: password
```



Injecter toutes les valeurs

```
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
  namespace: default
spec:
  containers:
    - name: mon-conteneur
      image: nginx
      envFrom:
        - configMapRef:
            name: maconfig
        - secretRef:
            name: credentials
  restartPolicy: Never
```



En tant que volume

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: myapp
      image: nginx
      volumeMounts:
        - name: config
          mountPath: /config
        - name: secret
          mountPath: /credentials
  volumes:
    - name: config
      configMap:
        name: ma-config
    - name: secret
      secret:
        secretName: credentials
```

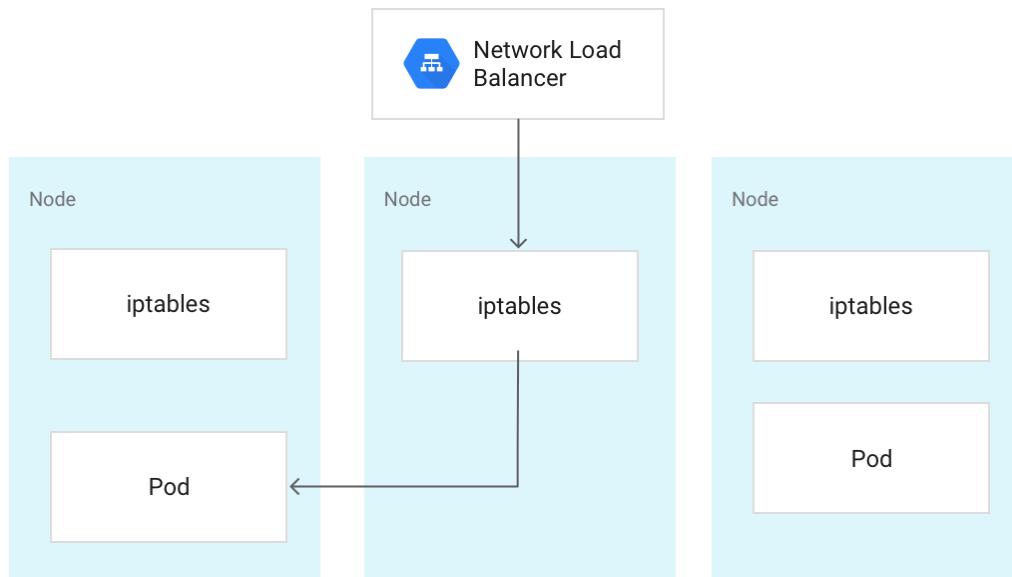


SERVICES



Vue d'ensemble

- Kubernetes offre en interne un système de load-balancing basé sur les objets de type Service
- Basé (par défaut) sur iptables



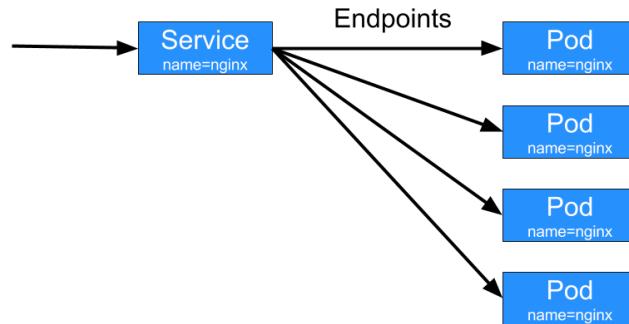
Exposer une application via les services

- ClusterIP : pour un accès interne
- NodePort: pour binder un port de l'host
- LoadBalancer: Utilise les LBs du cloud provider (selon le cas)
- ExternalName: pour faire des alias vers des éléments externes (ex: DB)
- Headless:
 - Pas de ClusterIP
 - Pas géré par kube-proxy (i.e. pas de LB)
 - Expose les replicas dans le DNS



Endpoints

- En réalité kubernetes maintient une liste dynamique des pods qui offrent un « service » donné. On parle alors de « endpoints ».
- Endpoint = une IP + un port
- Service -> Endpoint -> Pod
- Endpoint = Objet kubernetes manipulable (mais rarement touché par un humain)



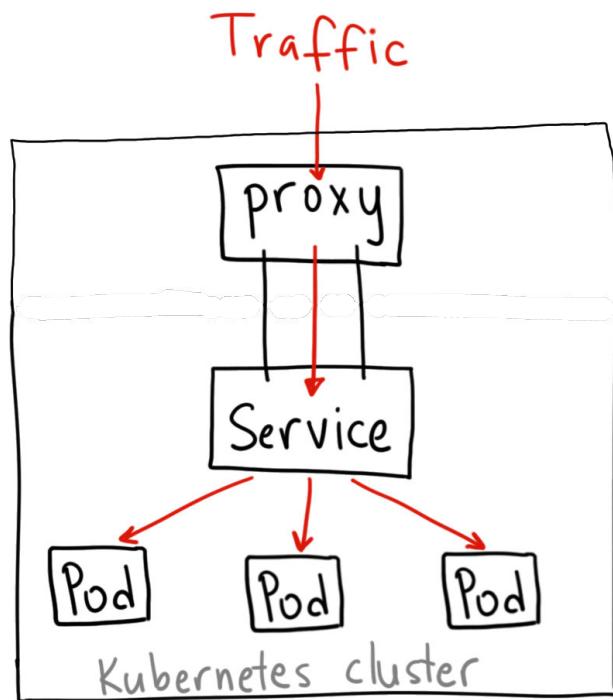
Kube-proxy

- Tourne sur tous les nodes du cluster
- Fournit une IP virtuelle pour les services (sauf headless)
 - Utilise un autre objet les endpoints
- Traduit les différents « chemins » en règles iptables de NAT
- Il est possible de remplacer iptables par IPVS

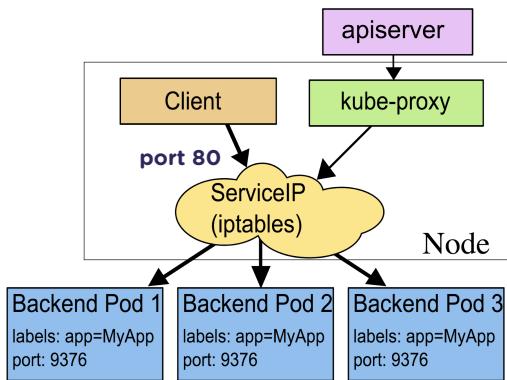
- N.B: Le range IP des services est différent de celui des pods (10.43.x.x et 10.44.x.x par exemple)



Service ClusterIP

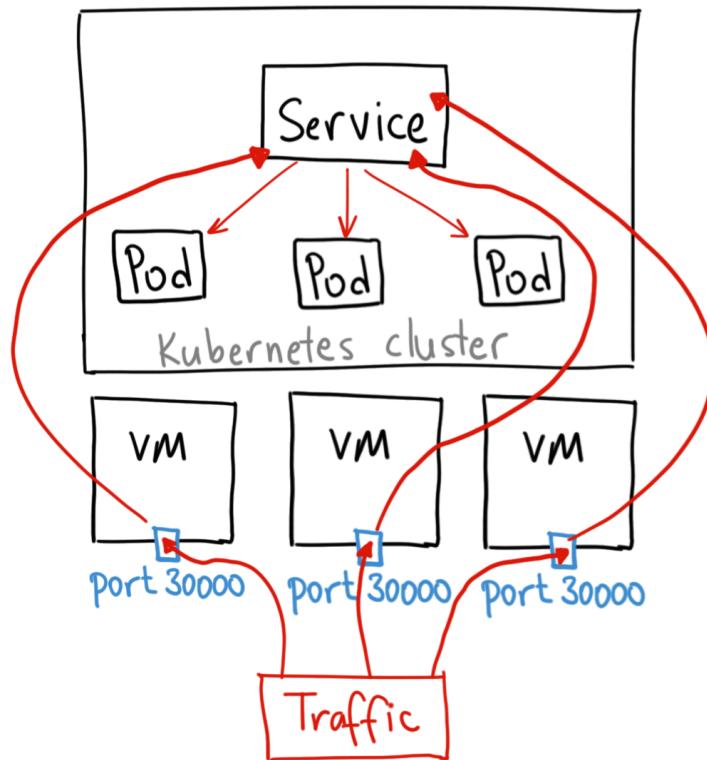


Service et DNS / ClusterIP

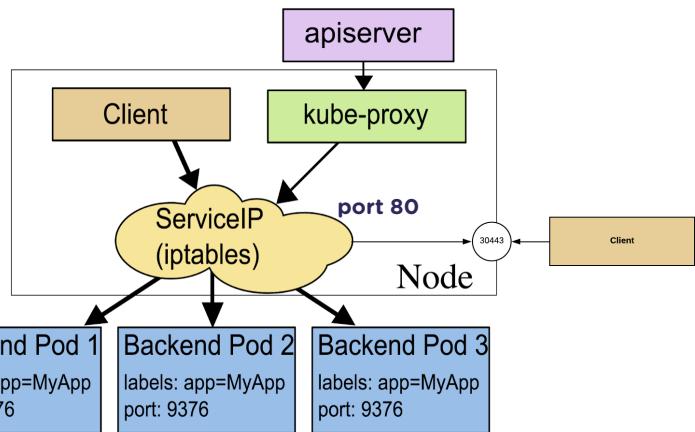


```
kind: Service
apiVersion: v1
metadata:
  name: myapp
spec:
  selector:
    app: MyApp
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Service NodePort

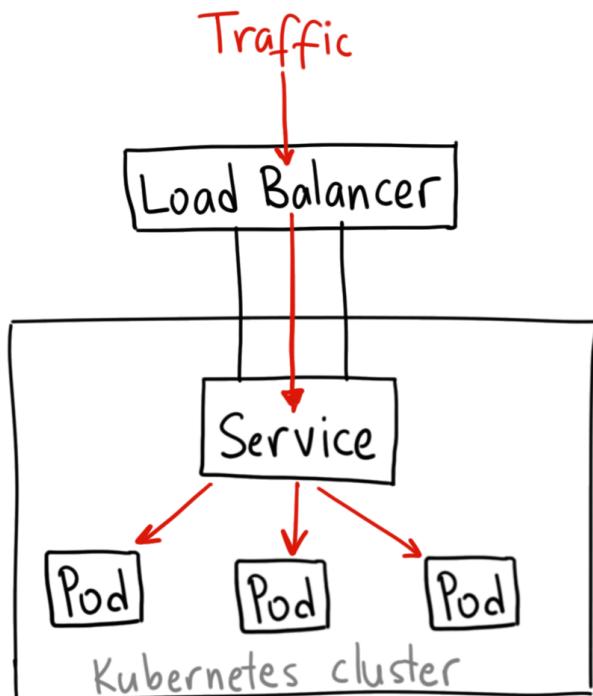


Service NodePort

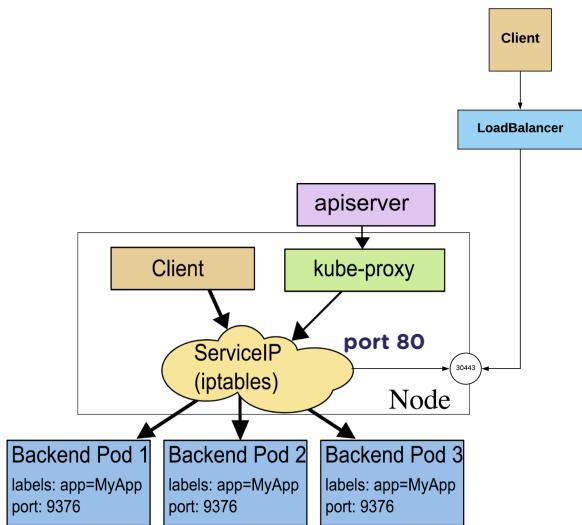


```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
      nodePort: 30443
```

Service LoadBalancer

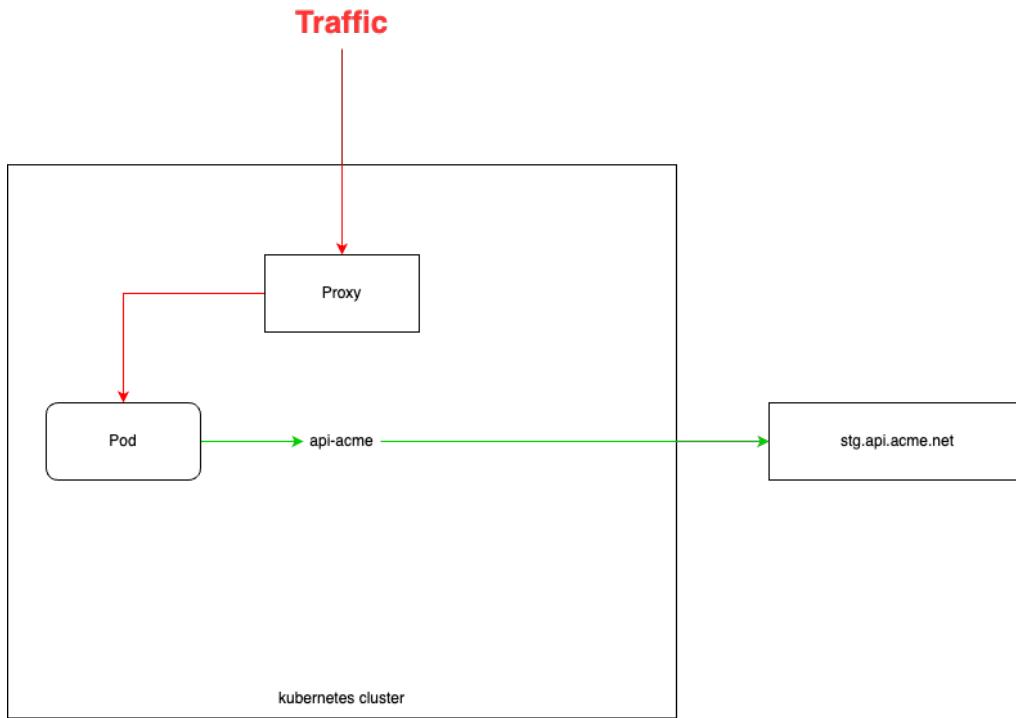


Service LoadBalancer

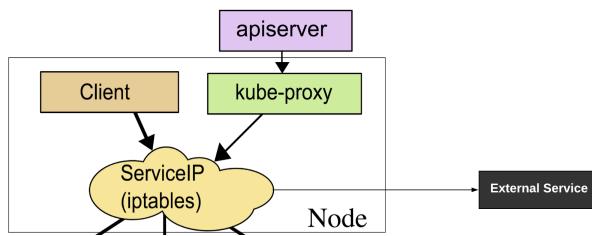


```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
      nodePort: 30443
```

Service externalName



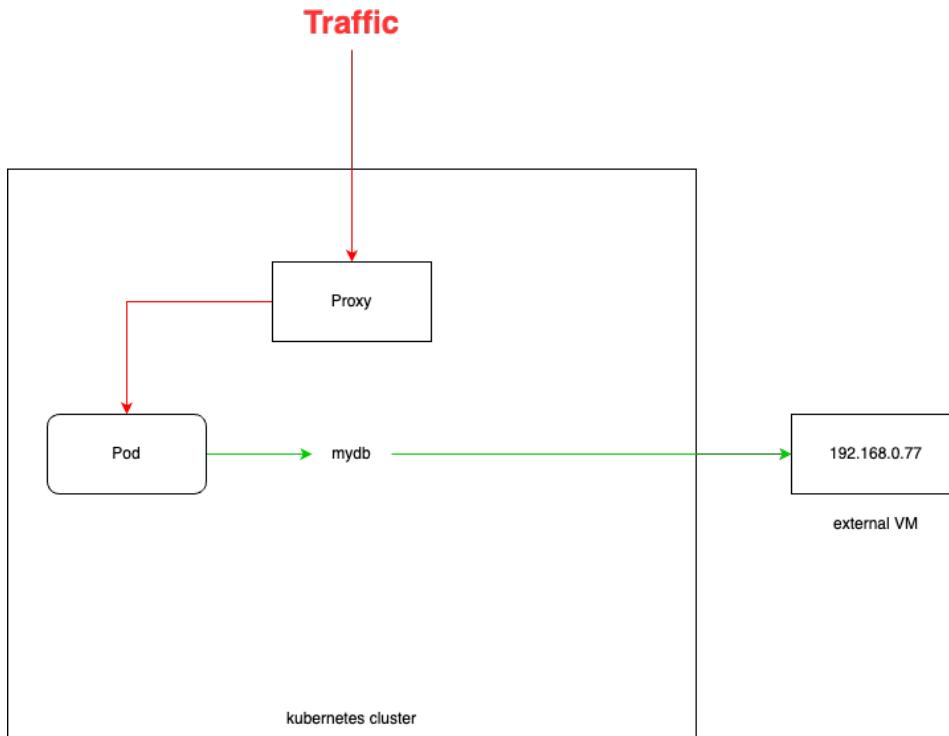
Service ExternalName



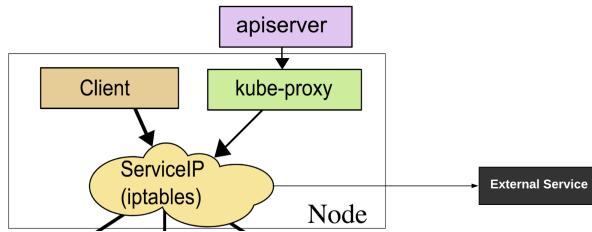
```
kind: Service
apiVersion: v1
metadata:
  name: api-acme
spec:
  selector:
    app: MyApp
  type: ExternalName
  externalName: app.acme.com
```



service externalIPs

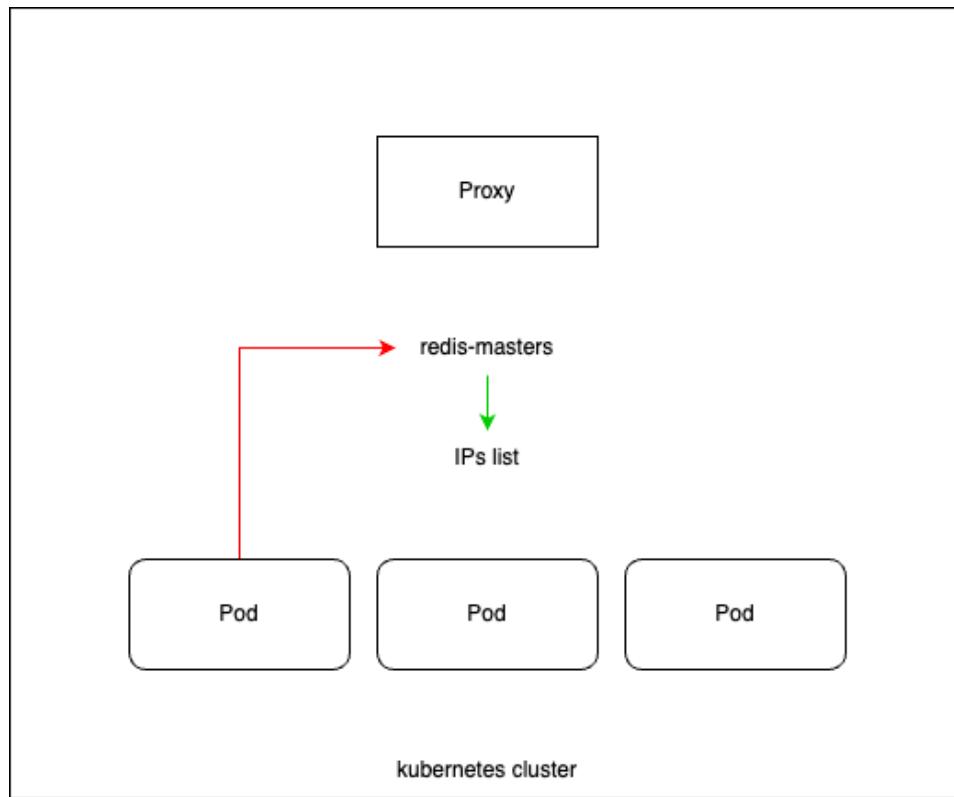


Service externalIPs

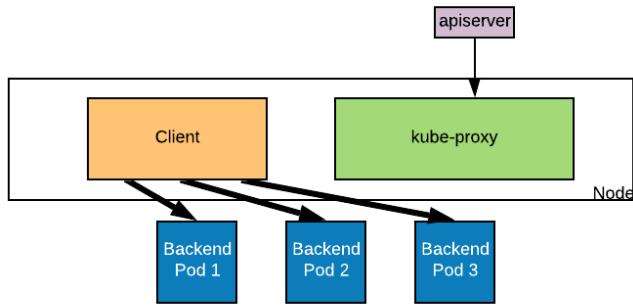


```
kind: Service
apiVersion: v1
metadata:
  name: my-db
spec:
  selector:
    app: MyApp
  type: ExternalName
  externalIPs: 192.168.0.77
```

Service headless



Service Headless



```
kind: Service
apiVersion: v1
metadata:
  name: redis-masters
spec:
  selector:
    app: MyApp
  type: ClusterIP
  ClusterIP: None
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```



- Dans un cluster kubernetes, le service discovery est fait via le service DNS interne
 - Un pod DNS autoscaler est déployé par défaut sur tout nouveau cluster
 - Un ou plusieurs pods DNS sont ensuite déployés dans le cluster
 - Le nombre de répliques va évoluer en fonction de la charge du service
- Les noms des services internes sont résolus par le DNS où transmises à un serveur de strate supérieure (forwarder)

Services & DNS

- 2 types d'enregistrement DNS
 - IN A : ClusterIP / Headless
 - CNAME: LoadBalancer / ExternalName / ExternalIPs

Service DNS

```
# au sein du même namespace  
nslookup myapp
```

```
Name:      myapp  
Address 1: 10.101.23.117 myapp.default.svc.cluster.local
```

- **Format FQDN:**

- my-svc.my-namespace.svc.cluster.local



VOLUMES

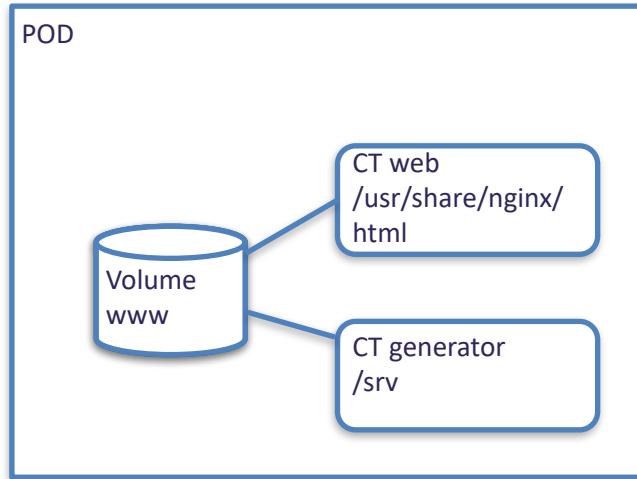


- Les volumes sont des espaces de travail pour les conteneurs
- Lorsqu'un conteneur d'un Pod est en échec, Kubernetes va tuer le conteneur et le recréer à l'intérieur même du Pod. Tout ce qui était stockés dans le conteneur est alors perdu. Les volumes permettent de conserver la donnée au sein du Pod entre plusieurs instances d'un conteneur
- Grâce aux volumes, plusieurs conteneurs peuvent partager de la donnée au sein d'un même Pod
- Il en existe plusieurs types chacun utilisant un driver et/ou un protocole différent



Exemple de pod avec volume

Exemple d'utilisation d'un volume au sein d'un pod



Pod avec volume

```
apiVersion: v1
kind: Pod
metadata:
  name: horloge
spec:
  containers:
    - image: nginx
      name: web
      volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
    - image: demo/generator
      name: generator
      volumeMounts:
        - name: www
          mountPath: /srv
  volumes:
    - name: www
      emptyDir: {}
```



Persistent Volume

- Le cycle de vie d'un volume simple est lié au cycle de vie du pod qui l'utilise
- Il est donc nécessaire d'avoir une abstraction qui permet de les décorrélérer et donc de fournir un stockage réellement persistant à l'application
- Il existe trois modes d'accès qui caractérisent les PersistentVolumes :
 - `ReadWriteOnce` => ne peut être monté que par un seul Pod en lecture/écriture
 - `ReadOnlyMany` => peut être monté par plusieurs Pods mais en lecture seule
 - `ReadWriteMany` => peut être monté par plusieurs Pods en lecture/écriture



Le mode d'accès dépend de la technologie que vous utilisez en stockage

Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Mi
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-path
  local:
    path: >-
      /var/lib/rancher/k3s/storage/_test
# ...
```



Persistent Volume Claim

- Utiliser un PersistentVolume directement dans un Pod est une mauvaise pratique. Cela implique que le développeur qui écrit le manifest doit connaître à l'avance les détails techniques du stockage qui sera utilisé et qui peut dépendre de l'environnement.
- Les PersistentVolumeClaim sont une abstraction qui permet à l'administrateur du cluster de choisir le stockage utilisé pour répondre à la demande de Volume du manifest du développeur.
- Lorsqu'un Pod est schedulé avec un PersistentVolumeClaim, le scheduler va lui attribuer le plus petit PersistentVolume répondant aux contraintes demandées dans le PVC (access mode et quantité de stockage minimum).
- Soit il existe un volume correspondant aux contraintes qu'il est possible de rattacher au pod, soit celui-ci sera créé dynamiquement



Persistent Volume Claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-pvc
spec:
  storageClass: rapide
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```



Utilisation de Persistent Volume Claim

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: myapp
      image: nginx
      resources: {}
      volumeMounts:
        - name: data
          mountPath: /data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: test-pvc
```



StorageClass (pour les OPS)

- Permet de définir les classes de stockage (STO tiers)
- Des modèles type de PVs
- Plusieurs critères
 - Durée de vie : infinie ou temps d'exécution
 - Prix
 - Localisation
 - Rapidité
 - Résilience
 - Autres ...
- Très lié à l'environnement d'execution (fatalement)
 - Cloud Provider
 - Solution de cloud privé
 - Parc des solutions de stockage déployées

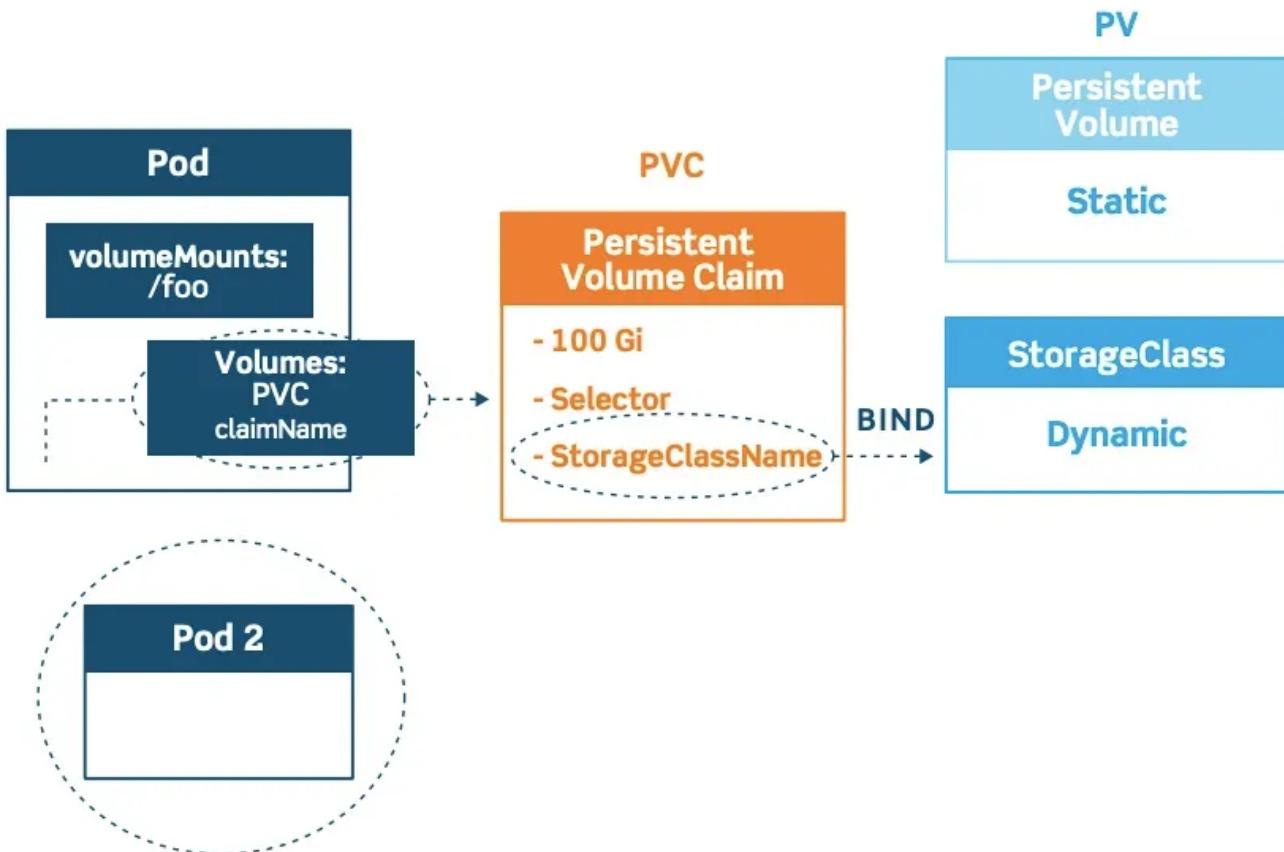


StorageClass (pour les OPS)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-path
provisioner: rancher.io/local-path
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```



Vue d'ensemble



INGRESS RULE

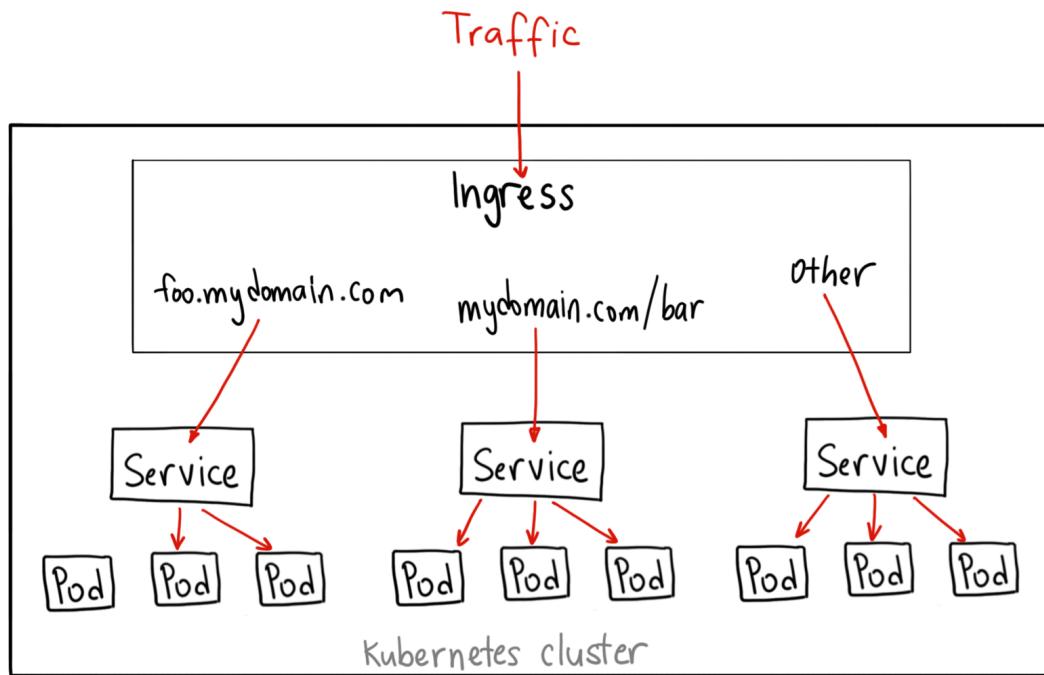


Ingress Rule

- Les objets Ingresses permettent d'exposer un service http à l'extérieur d'un cluster Kubernetes de manière plus fine que les Load Balancers ou NodePort
 - Vhosts routing
 - URI routing
 - Terminaisons TLS
 - Metrics
 - etc ...
- Prend la forme de règles déclaratives et d'un contrôleur
- Pas de contrôleur par défaut



Ingress Rule



Ingress Controller

- Les Ingress Rules sont surveillées par un controller nommé ingress controller
- Cet ingress controller va adapter sa configuration à chaque ajout/changement/suppression d'Ingress afin d'en respecter les règles
- Cela permet notamment de n'avoir qu'un seul service exposé publiquement, celui concernant l'Ingress Controller, et de lui laisser la charge de router en interne vers les services.
- Permet également de centraliser des problématiques courantes dans un seul composant pour en décharger les applications
 - Metrics
 - SSL
 - Sécurité
 - etc ...

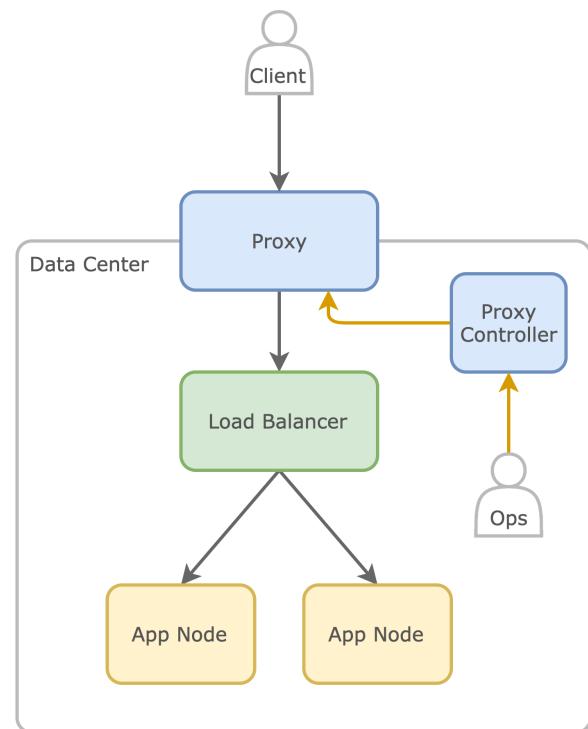


Ingress Controller

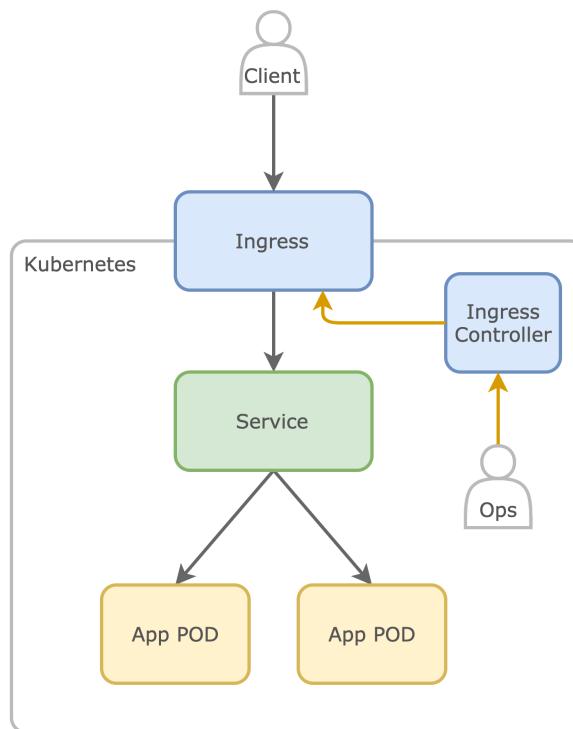
- Nginx Controller
- Traefik
- HAproxy
- Linkerd
- Contour
- API Six

Ingress Controller

Standard



Kubernetes



Ingress Rule

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo
spec:
  rules:
  - host: demo.example.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: myapp
            port:
              number: 80
    tls:
    - hosts:
      - demo.example.com
      secretName: demo-cert
```



KUBERNETES : DÉPLOYER DES APPLICATIONS D'ENTREPRISE



ReplicaSets

- ReplicaSet et pods
- Créer un ReplicaSet
- Identifier un ReplicaSet
- Rechercher un ensemble de pods pour un ReplicaSet
- Mettre à l'échelle les ReplicaSets

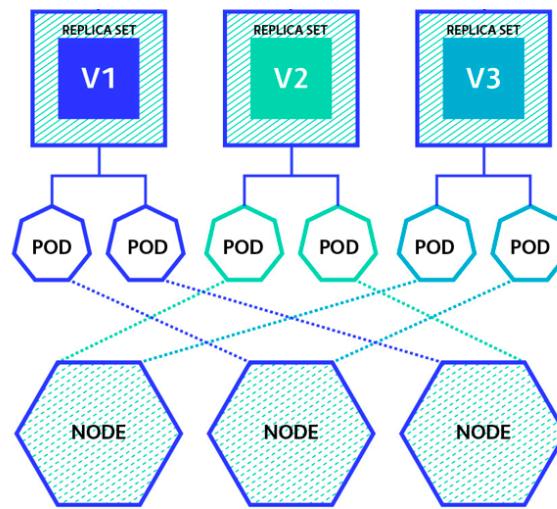
REPLICASETS ET PODS



ReplicaSets

- Abstraction qui est en charge
 - de créer un pool homogène de pods
 - d'en avoir exactement le bon nombre à tout moment
 - d'ordonner les actions sur ces pods

ReplicaSets



CRÉER UN REPLICA SET



ReplicaSets

ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: web
  labels:
    app: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: web:latest
```



METTRE À L'ÉCHELLE LES REPLICA SETS



Mettre à l'échelle les Replica Sets

- Redimensionner un ReplicaSet

```
>_
```

```
kubectl scale replicaset --replicas 5 my-rs
```

```
> kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
my-rs-wcsk9 1/1     Running   0          11m
my-rs-4pbng 1/1     Running   0          11m
my-rs-lmvhz 1/1     Running   0          6s
my-rs-c64xl 1/1     Running   0          6s
my-rs-x8l7g  1/1     Running   0          6s
```

Deployments

- Deployments, ReplicaSets et Pods
- Mettre à l'échelle des déploiements
- Stratégies de déploiement
 - Recreate
 - Rollingupdate
 - Rollout
- Nommage des Deployments, ReplicaSets et Pods



DEPLOYMENTS, REPLICASETS ET PODS

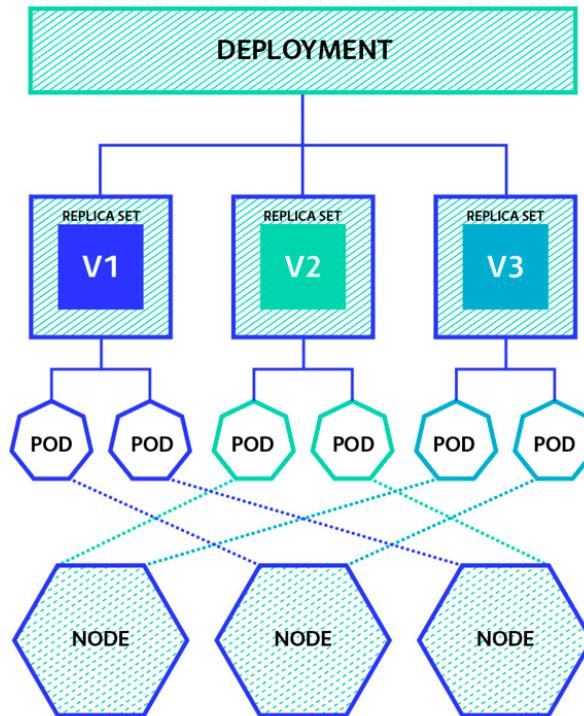


Deployments

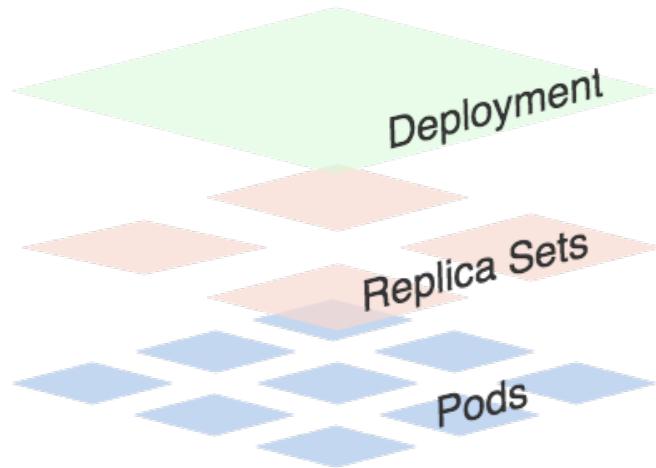
- Se chargent de
 - Piloter les ReplicasSets
 - Remplacer une spec de ReplicaSets par une autre de manière transparente
 - Contrôler et ordonner les actions sur les Pods
- Définissent
 - 1 template de création de Pods
 - 1 nombre désiré de Replicas
 - 1 label de sélection pour les identifier



Deployments



Deployments / ReplicaSets / Pods



Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deploy
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - name: test
          image: alpine:edge
```



- Créer un deployment



```
kubectl create deployment --image=nginx \
--replicas 5 my-deploy
```

| NAME | READY | STATUS | RESTARTS | AGE |
|----------------------------|-------|---------|----------|-----|
| my-deploy-8448c488b5-v757t | 1/1 | Running | 0 | 36s |
| my-deploy-8448c488b5-b85pc | 1/1 | Running | 0 | 36s |
| my-deploy-8448c488b5-tv9hw | 1/1 | Running | 0 | 36s |
| my-deploy-8448c488b5-hn27r | 1/1 | Running | 0 | 36s |
| my-deploy-8448c488b5-fhzhg | 1/1 | Running | 0 | 36s |

METTRE À L'ÉCHELLE DES DÉPLOIEMENTS



Mettre à l'échelle des déploiements

- Redimensionner un deployment

```
>_
```

```
kubectl scale deployment --replicas 5 my-deploy
```

| NAME | READY | STATUS | RESTARTS | AGE |
|----------------------------|-------|-------------------|----------|-------|
| my-deploy-8448c488b5-v757t | 1/1 | Running | 0 | 3m48s |
| my-deploy-8448c488b5-b85pc | 1/1 | Running | 0 | 3m48s |
| my-deploy-8448c488b5-4769s | 0/1 | ContainerCreating | 0 | 3s |
| my-deploy-8448c488b5-w8v6p | 0/1 | ContainerCreating | 0 | 3s |
| my-deploy-8448c488b5-lxwhm | 0/1 | ContainerCreating | 0 | 3s |

STRATÉGIES DE DÉPLOIEMENT



Stratégies de déploiement

- Les Deployment vont permettre de mettre à jour les applications sans coupures ou tout du moins de manière sécurisée
- Pour cela kubernetes offre plusieurs stratégies et paramètres
- Recreate
 - Les pods sont stoppés avant que les nouveaux soient démarrés
- Rollingupdate
 - Les pods sont arrêtés lot par lot et un nombre équivalent est démarré



Stratégies de déploiement: Recreate

Déploiement avec stratégie Recreate

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-deploy
spec:
  replicas: 3
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
      - name: demo
        image: webofmars/go-whoami
```

Stratégies de déploiement: RollingUpdate

Déploiement avec stratégie Rolling Update

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-deploy
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
      - name: demo
        image: webofmars/go-whoami
```



Stratégies de déploiement: Rollout

- Les déploiements vont également permettre une chose essentielle: sécuriser les rollbacks
- Les commandes 'kubectl rollout' permettent de gérer l'historique des révisions d'un Deployment et donc des ReplicasSets

Stratégies de déploiement: Rollout

- Afficher les revisions



```
kubectl rollout history deploy my-deploy
```

```
> k rollout history deploy my-deploy
deployment.apps/my-deploy
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
```

- Initier un rollback



```
kubectl rollout undo deployment my-deploy
```

```
> k custom-cols -o images pods
NAME                      IMAGE        STATUS
my-deploy-65ff58b875-nzwbg  nginx:1.16  Running
my-deploy-65ff58b875-tcwlf  nginx:1.16  Running
my-deploy-65ff58b875-rkvl5  nginx:1.16  Running
my-deploy-5f645cc6cd-2l9x8  nginx:1.15  Running
```

Stratégies de déploiement: Rollout

- Initier un rollback à une version spécifique



```
kubectl rollout undo --to-revision 4 deployment \
my-deploy
```

| NAME | IMAGE | STATUS |
|----------------------------|------------|---------|
| my-deploy-d8579d958-x8z96 | nginx:1.19 | Running |
| my-deploy-d8579d958-pkfrk | nginx:1.19 | Running |
| my-deploy-65ff58b875-2ml6x | nginx:1.16 | Running |
| my-deploy-65ff58b875-vfxn5 | nginx:1.16 | Pending |
| my-deploy-d8579d958-p5pmw | nginx:1.19 | Pending |

NOMMAGE DES REPLICAS ET DEPLOYMENTS



Nommage des ReplicaSets

- Deployments -> ReplicaSet -> Pods
- Les objets enfants d'une abstraction de plus haut niveau vont avoir des noms qui découlent du nom de l'objet parent
- Exemple
 - Deployment: myapp
 - ReplicaSet: myapp-xxxxx
 - Pods: myapp-xxxxx-yyyyy



AUTRES CONTRÔLEURS



STATEFUL SET



- Gère le déploiement et la mise à l'échelle des Pods tout en garantissant leur **ordre et persistance**
- **Point clés**
 - Ordre de création, mise à jour et suppression des Pods respecté
 - Associe chaque Pod à un identifiant stable
 - Stockage persistant avec des volumes spécifiques à chaque Pod
 - Utilisé pour les applications stateful (ex: bases de données, clusters Kafka)
- **But**
 - Prise en charge de la récupération automatique de l'état après redémarrage

StatefulSet

StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysts
spec:
  selector:
    matchLabels:
      app: myapp
  serviceName: myheadlessservice
  replicas: 2
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx:latest
```



DAEMONSET



- Assure qu'un Pod tourne sur chaque nœud du cluster.
- **Points Clés**
 - Exécute un Pod par nœud
 - Automatiquement adapté aux nouveaux nœuds ajoutés au cluster
 - Peut être configuré pour tourner uniquement sur certains types de nœuds (grâce à des sélecteurs)
- **But**
 - Très utile pour les tâches de gestion d'infrastructure (ex: agents, collecteurs)

DaemonSet

DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: myds
  labels:
    app: myapp
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx:latest
```



HELM (V3)



- HELM est un outil permettant d'utiliser un catalogue d'applications packagées pour Kubernetes
- Ce catalogue est maintenu par la communauté, et propose une implémentation possible de chaque application disponible. Il y a plusieurs dépôts, dont stable, incubator et inception. Les éditeurs de logiciels peuvent aussi proposer leur propre dépôt
- HELM intègre nativement une notion de template qui permet de personnaliser chaque déploiement d'application en fonction du cluster, de l'environnement et du souhait de la personne qui déploie l'application



artifacthub.io

- Création d'une release



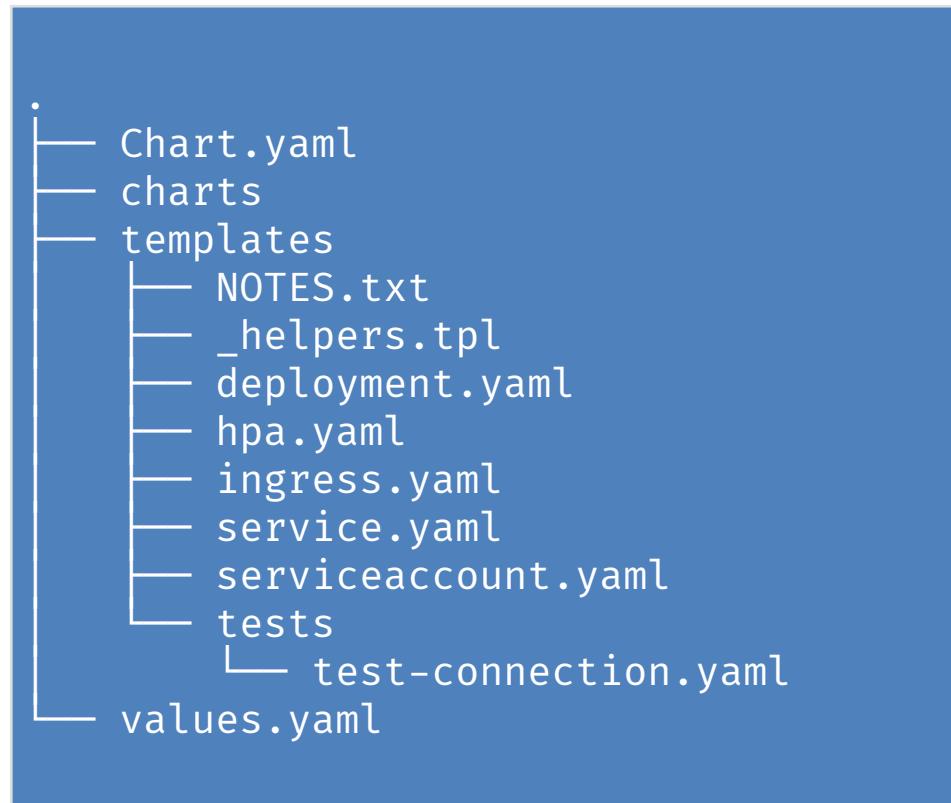
```
helm install test ingress-nginx \
--namespace ingress-nginx --version 1.41.1
```

- Upgrade d'une release



```
helm upgrade test ingress-nginx \
--namespace ingress-nginx --version 1.41.2
```

Helm (v3) - Structure d'un chart



Helm (v3) - templates

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "app22.fullname" . }}
  labels:
    app: {{ template "app22.name" . }}
    chart: {{ template "app22.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ template "app22.name" . }}
      release: {{ .Release.Name }}
template:
  metadata:
    labels:
      app: {{ template "app22.name" . }}
      release: {{ .Release.Name }}
spec:
  containers:
    - name: {{ .Chart.Name }}
      image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
      imagePullPolicy: {{ .Values.image.pullPolicy }}
```



- Paramétrage d'une release



```
helm install test ingress-nginx \
--namespace ingress-nginx \
--version 1.41.1 \
-f my-values.yaml \      # P2
-f dev-values.yaml \     # P1
--set database=192.168.0.2 # P0
```

- Crédation d'un chart



```
helm create test
```

CONCEPTS AVANCÉS

SCALING AUTOMATISÉ



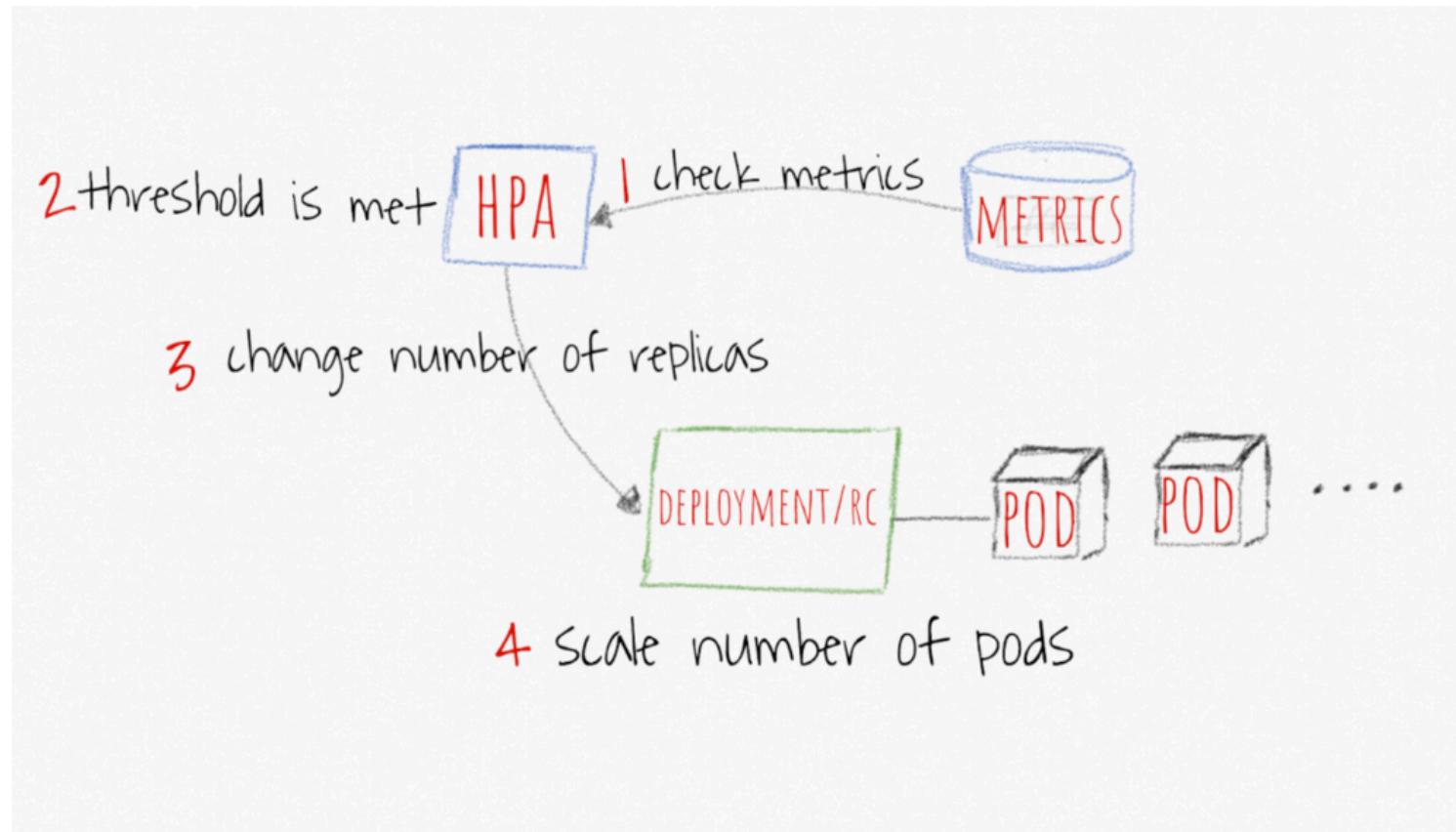
Scaling automatisé - horizontal

- Il existe une ressource nommée HorizontalPodAutoscaler qui permet de laisser Kubernetes gérer automatiquement le nombre de Pod en se basant (par défaut) sur la consommation CPU des Pods.
 - Les HPA nécessitent de déployer un monitoring metrics
-
- **Activer l'autoscaling horizontal**



```
kubectl autoscale deployment my-deploy --min=1 --  
max=10
```

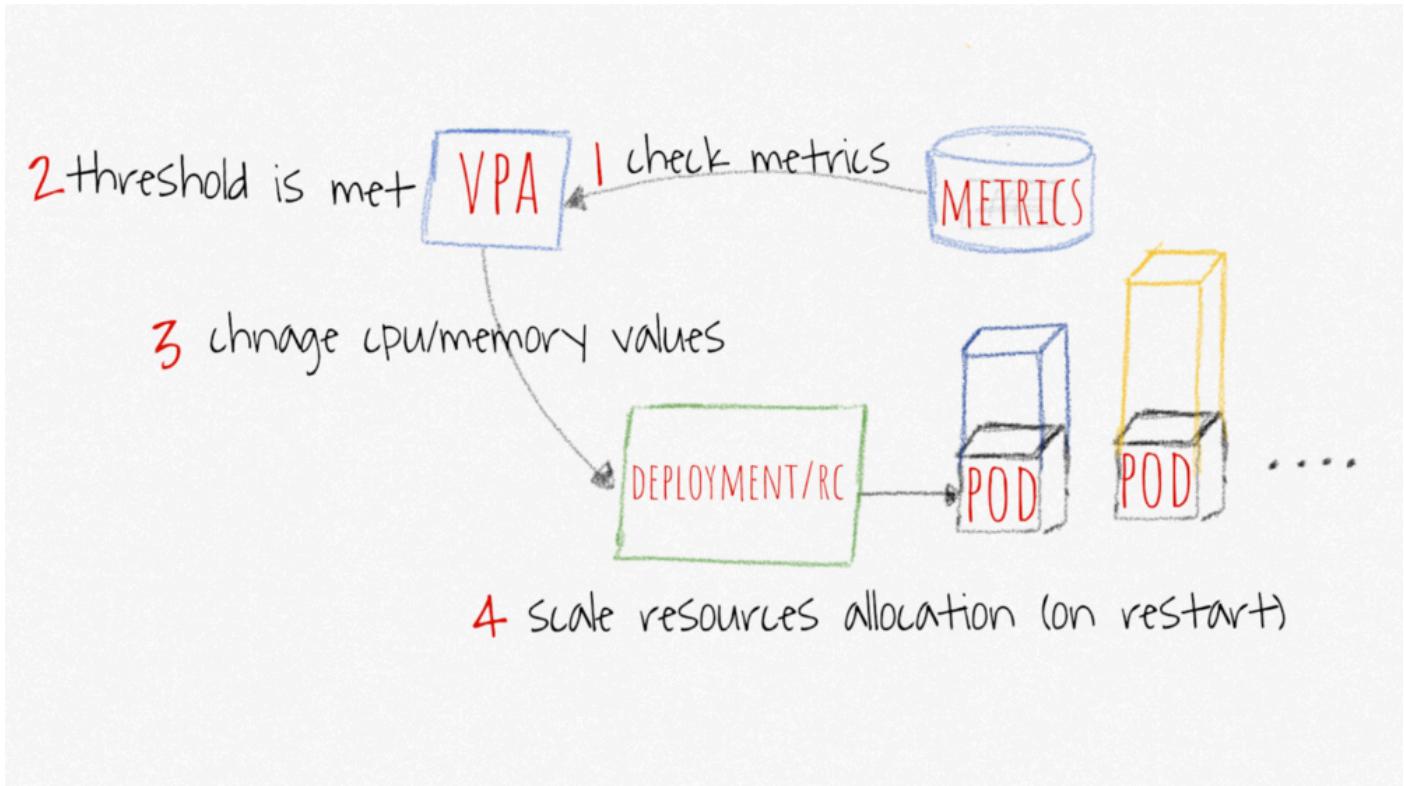
Scaling automatisé - Horizontal



Scaling automatisé - vertical

- Il existe une ressource nommée VerticalPodAutoscaler (VPA) qui permet de laisser Kubernetes gérer automatiquement les ressources utilisables par un modèle de pod.
- Pas compatible avec le HPA
- Les VPA nécessitent de déployer un monitoring metrics
- S'installe à part (contrairement au HPA)

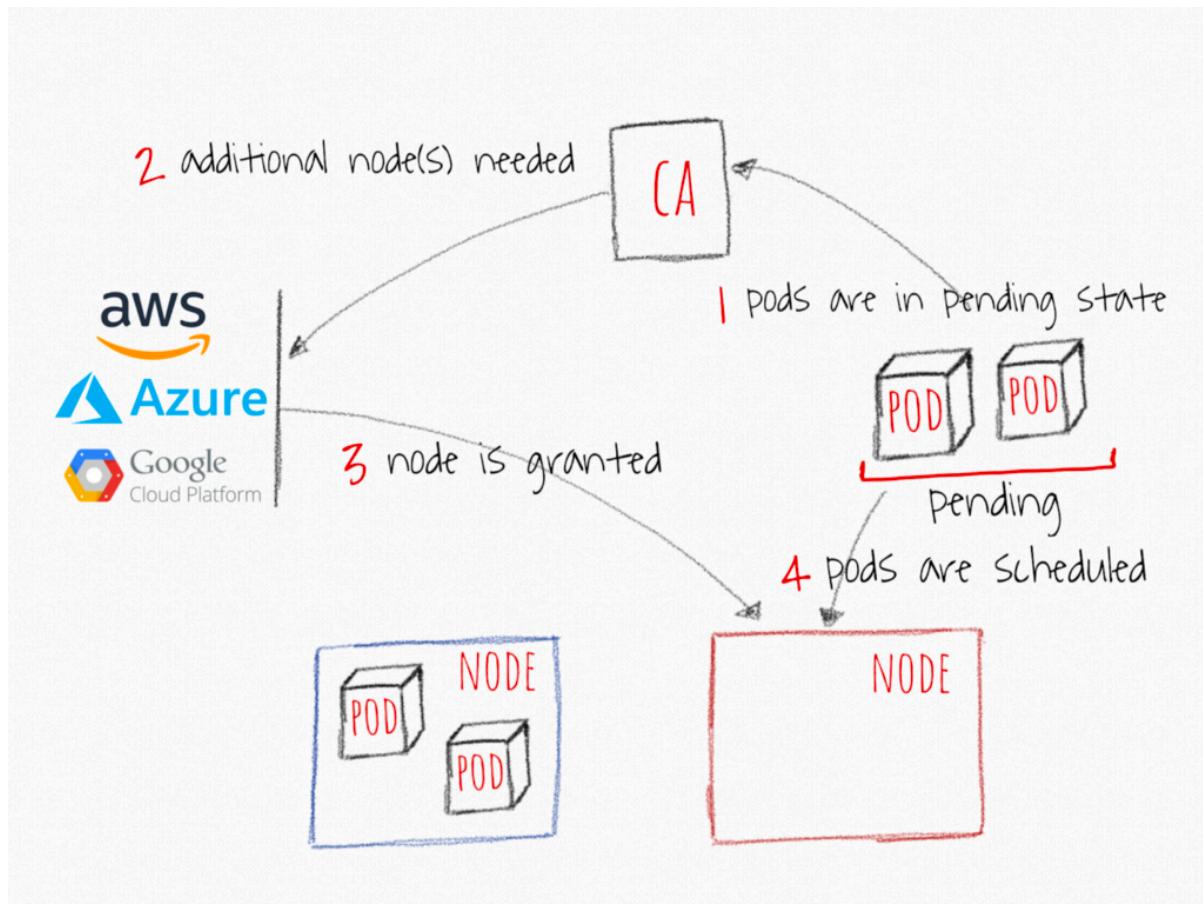
Scaling automatisé - Vertical



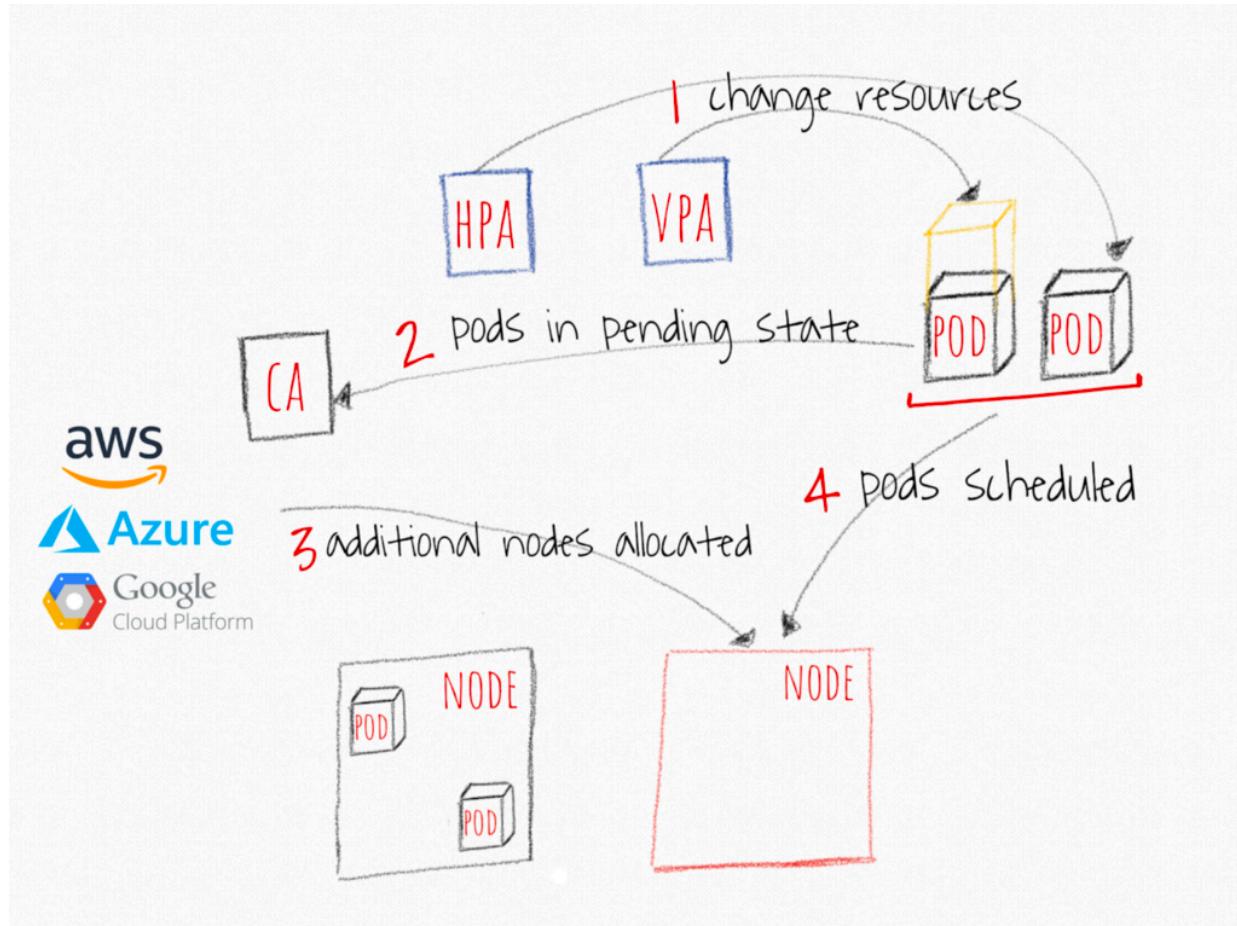
Scaling automatisé - Cluster Autoscaler

- Va ajouter et retirer des noeuds en fonction de
 - Nombre de pods bloqués en status "Pending"
 - Ressources utilisées sur chacun des noeuds
- Nécessite de déployer un monitoring metrics

Scaling automatisé - Cluster Autoscaler



Scaling automatisé - Vue d'ensemble



CONTRÔLE FIN DES WORKLOADS



NodeSelector

- Permet de cibler un ou des nodes lors du déploiement d'un pod
- Il est nécessaire d'ajouter des labels sur les noeuds au préalable
- Ou d'utiliser ceux par défaut:
 - [kubernetes.io/hostname](#)
 - [failure-domain.beta.kubernetes.io/zone](#)
 - [failure-domain.beta.kubernetes.io/region](#)
 - [beta.kubernetes.io/instance-type](#)
 - [kubernetes.io/os](#)
 - [kubernetes.io/arch](#)



NodeSelector

- labeliser un ou des noeuds

>_

```
kubectl label node node-1 foo=bar
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    foo: bar
```



Affinity et Anti-Affinity

- Evolution du NodeSelector
 - Plus riche
 - Permet d'imaginer des patterns de déploiements complexes
- 2 types de scheduling
 - requiredDuringSchedulingIgnoredDuringExecution
 - preferredDuringSchedulingIgnoredDuringExecution
- Affinité et anti-affinités entre
 - 1 pod et des nodes
 - entre pods

Affinity et Anti-Affinity: Nodes

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: topology.kubernetes.io/zone
                operator: In
                values:
                  - eu-west-1a
                  - eu-west-1b
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions: topology.kubernetes.io/instance-type
            - key:
              operator: In
              values:
                - r2.xlarge
  containers:
    - name: with-node-affinity
      image: k8s.gcr.io/pause:2.0
```

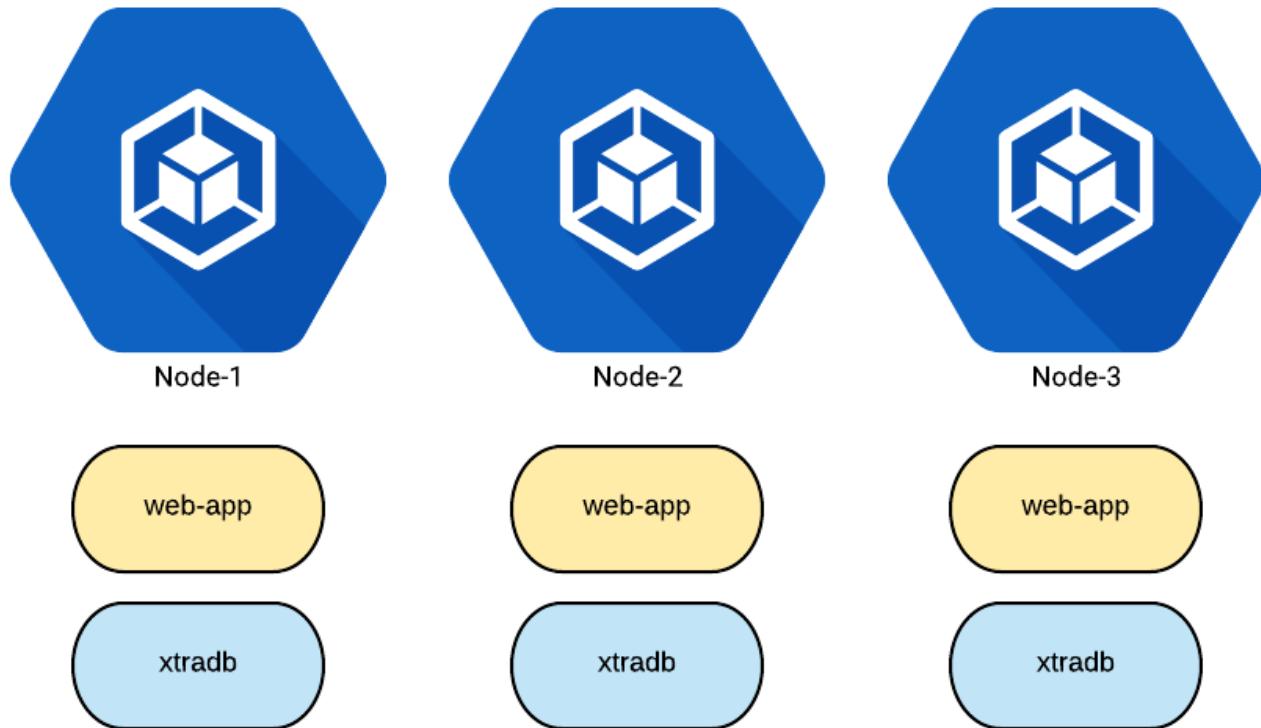


Affinity et Anti-Affinity: Inter Pods

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
  labels:
    app: web
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - cache
      topologyKey: kubernetes.io/hostname
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - web
      topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```



Affinity et Anti-Affinity: Inter Pods



Taints & Tolerations

- Taints : Influe sur la disposition des pods sur des noeuds
 - PreferNoSchedule / NoSchedule / NoExecute
- Tolérations : Permet à certains pods de “supporter” des teintes pour modifier l’effet des teintes
- Eviction de pods selon des teintes prédéfinies
 - node.kubernetes.io/not-ready
 - node.kubernetes.io/unreachable
 - node.kubernetes.io/out-of-disk
 - node.kubernetes.io/memory-pressure
 - node.kubernetes.io/disk-pressure
 - node.kubernetes.io/network-unavailable
 - node.kubernetes.io/unschedulable
 - node.cloudprovider.kubernetes.io/uninitialized



- teinter un ou des noeuds

>_

```
kubectl taint node node-1 foo=bar:NoSchedule
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  ...
tolerations:
- key: "foo"
  operator: "Equal"
  value: "bar"
  effect: "NoSchedule"
```

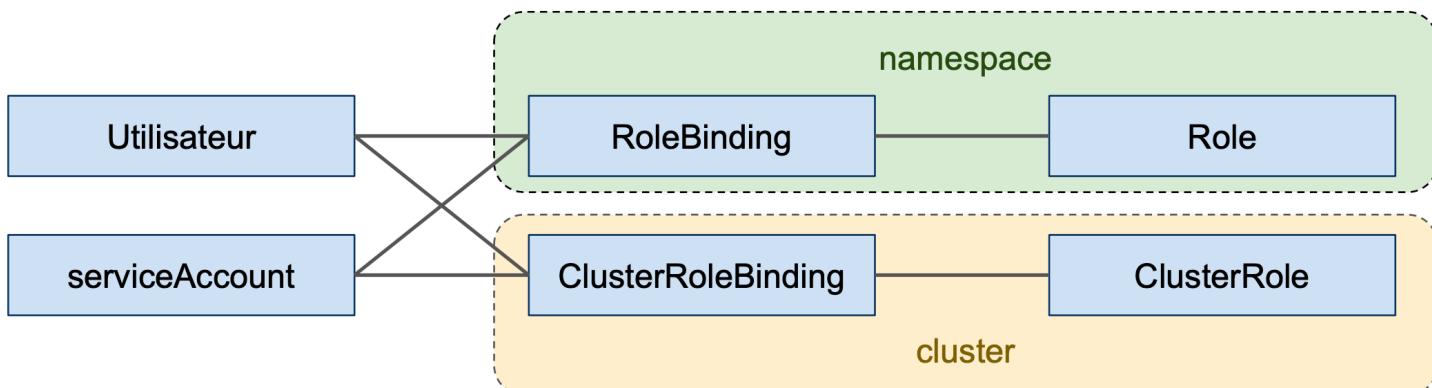


SÉCURITÉ



Sécurité: authentification & RBAC

- L'authentification est effectuée grâce à des certificats SSL ou des tokens. Chaque cluster contient sa propre PKI (infrastructure de clé publique) et utilise des certificats auto-signés pour assurer l'authentification du client et du serveur.
- RBAC (Role Based Access Control) est un mécanisme qui propose un contrôle d'accès au cluster basé sur des rôles affectés aux utilisateurs. Les rôles peuvent être au niveau d'un namespace, ou global au niveau du cluster complet.



Sécurité: authentification & RBAC

- En place depuis la v1.8
- Limite les droits des apps et users

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```



Sécurité: authentication & RBAC

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
  - kind: User
    name: jane
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```



Sécurité: authentication & RBAC

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  < ... >
  serviceAccountName: default
  automountServiceAccountToken: false
  < ... >
```

- Déployer une application Web d'entreprise
 - basée sur un service, une base de données, une base in-memory (optionnelle)
- Configurer les composants
- Créer le service Kubernetes pour cette application
- Déployer le service applicatif
- Gérer le cluster

- Application CMS codée en PHP
 - Socle Lamp (Linux / Apache / PHP / MySQL)
 - Le contenu des pages en BDD (MySQL ou MariaDB)
 - Les assets sous format de fichiers dans le répertoire wp-content
- Wordpress
 - 50% des sites dans le monde !
 - Un écosystème riche (plugins / thèmes / Intégrations etc...)

CONCLUSIONS
