



# KUBERNETES

-

## CONCEPTS & ADMINISTRATION



© ILKI 2018

[pvillard@ilki.fr](mailto:pvillard@ilki.fr)



## MODULE 1

---

# INTRODUCTION ET HISTORIQUE DE KUBERNETES



© ILKI 2018



# PRÉSENTATION DE KUBERNETES



© ILKI 2018



# RAPPEL DOCKER

Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

Kernel

Hardware +  
hypervisor

Container

Application code

Dependencies

Kernel +  
Container runtime

Hardware



# ÉMERGENCE DU CAAS

- **Développement initial Docker**

- Focalisé sur le format des conteneurs, les fonctionnalités du *Runtime*, l'écriture de fichier de construction des images, la gestion « locale » du réseau et du stockage...
- Pensé mono-hôte à l'origine
- Aspect mono-hôte inadapté aux environnements d'entreprise et de cloud

- **Évolution vers le clustering et l'orchestration**

- Apparition de la notion de *Container As A Service*
- Développement de fonctionnalités permettant de manipuler des conteneurs sur plusieurs hôtes
- Émergence des acteurs de l'orchestration
  - Kubernetes, Docker Swarm, Rancher Cattle, Hashicorp Nomad, CoreOs Fleet, Mesos Marathon



# HISTORIQUE

- **Origine**

- Issu de solutions d'orchestration internes Google
- Orchestrateurs Borg puis Omega
- Retours d'expérience de plus de 10 ans
- Projet en Opensource par Google fin 2014



**kubernetes**

- **Dénomination K8S**

- Kubernetes est un mot grec signifiant « timonier » ou « pilote »
- Pourquoi K8S ? K/ubernete(8lettres)/S

- **LINUX FOUNDATION**

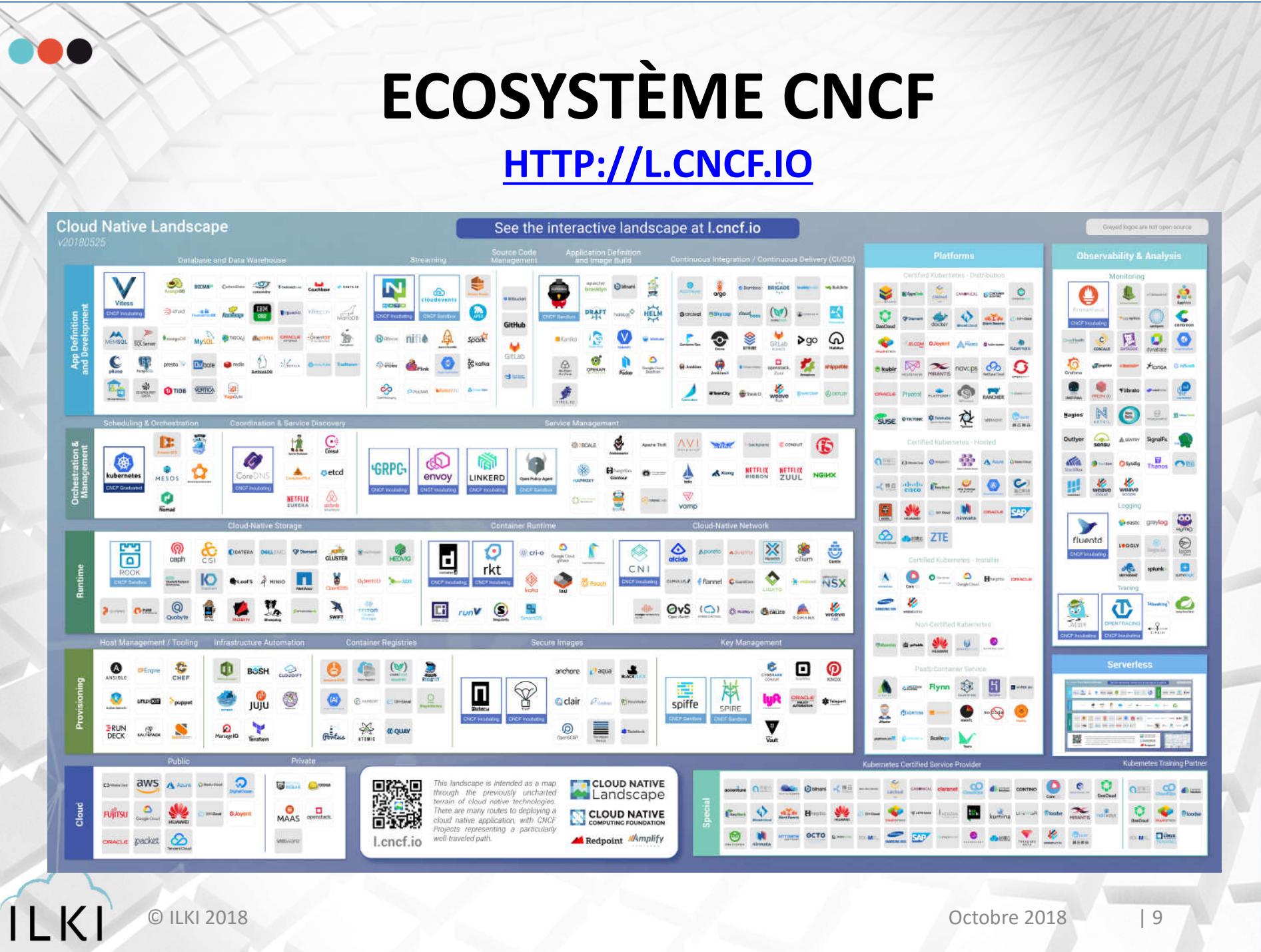
- Projet à l'origine de la création de la CNCF au sein de la Linux Foundation
- Cloud Native Computing Foundation



# CNCF

- **Cloud Native computing Foundation**
  - <http://www.cncf.io>
- **Créé par la *Linux foundation* en 2015**
  - En réalité, démarche initiée par Google et Docker
- **Objectifs**
  - Créer les conditions d'adoption des technologies de conteneurs pour éviter l'effet de verrouillage et augmenter la portabilité
  - Soutenir et sponsoriser des projets au sein de l'écosystème des technologies de conteneurs
- **Adhérents**
  - Docker, CoreOs, AWS, Microsoft, Cisco, Google, Alibaba, VMware, Dell, Fujistu, Redhat, IBM...







# K8S LEADERSHIP

- **Kubernetes - leader du marché ?**
  - Contributeurs principaux : Google 45% et RedHat 19%
  - Docker supporte Kubernetes depuis fin 2017
  - Les offres de PaaS se basent désormais en quasi-totalité sur Kubernetes
- **Projet Mature**
  - 1<sup>er</sup> projet de la CNCF « graduated » en mars 2018



Kubernetes  
Orchestration



Prometheus  
Monitoring



Envoy  
Network Proxy



CoreDNS  
Service Discovery



containerd  
Container Runtime



Fluentd  
Logging



# OBJECTIFS DE KUBERNETES



© ILKI 2018



# ORCHESTRATEUR

- **Objectifs**

- Gérer un ensemble d'instances exécutant des conteneurs
- Gérer des ensembles de conteneurs s'exécutant sur ces instances
- Automatiser les services d'infrastructure de l'environnement de conteneurs
  - Réseau, volumes de stockage, DNS, DHCP, health check...
- Garantir la disponibilité des conteneurs sur l'infrastructure en fonction de règles et de filtres
- Intégrer des fonctionnalités identifiées *PaaS* – auto-scalabilité, centralisation des logs, reporting...

- **Fonctionnalités**

- Clustering d'instances exécutant des conteneurs
- Fonctionnalités d'orchestration :
  - Plan de contrôle
  - Scheduler
  - Auto-scalabilité
  - Supervision (healthcheck des instances, des conteneurs...)
  - Gestion des réplicas de conteneurs
  - .....



# DÉFINITION - CLUSTER

- Control plan = master
- Data plan = plusieurs nodes

cluster

master

node

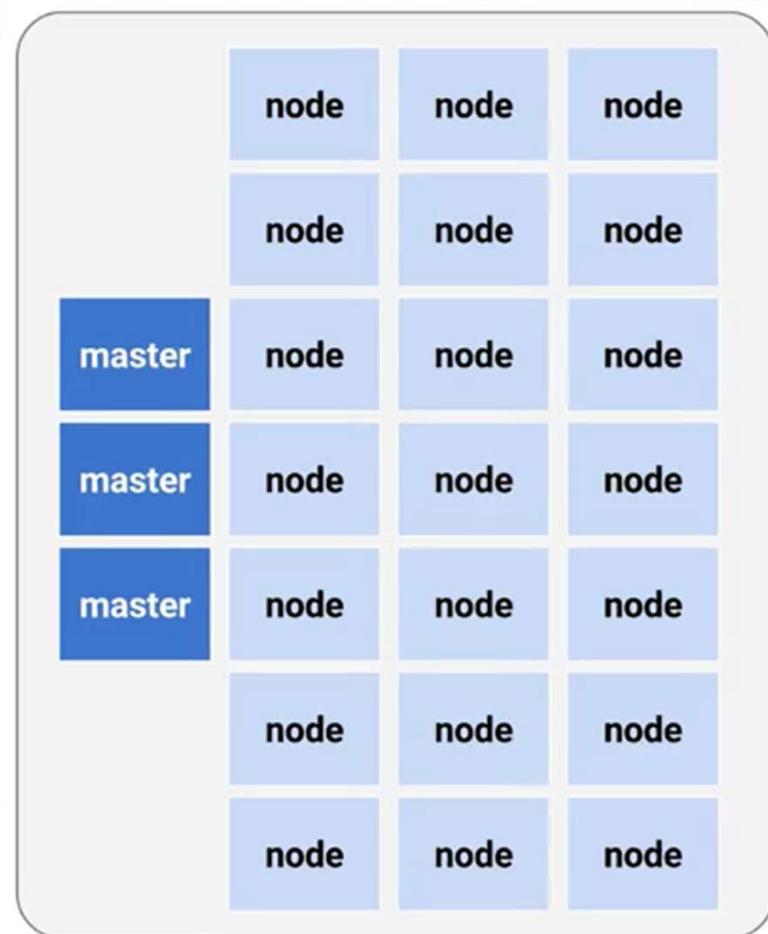
node

node



# CLUSTER K8S – MASSIVE SCALING

- **Control plan**
  - Plusieurs masters
  - Haute disponibilité du plan de contrôle
- **Data plan**
  - Pool de nodes
- **K8S est massivement scalable**
  - Supporte jusqu'à 5.000 nœuds
- **Entreprises en production**
  - Uber, Bloomberg, the NY Times, blablacar, Cdiscount Ebay...





# LIMITES V1.17

- **5.000 nodes**
- **150.000 pods**
- **300.000 containers**
- **100 pods/node**



# FONCTIONNALITÉS DE KUBERNETES



© ILKI 2018



# DÉFINITION ET FONCTIONNALITÉS

- **Qu'est-ce que Kubernetes ?**

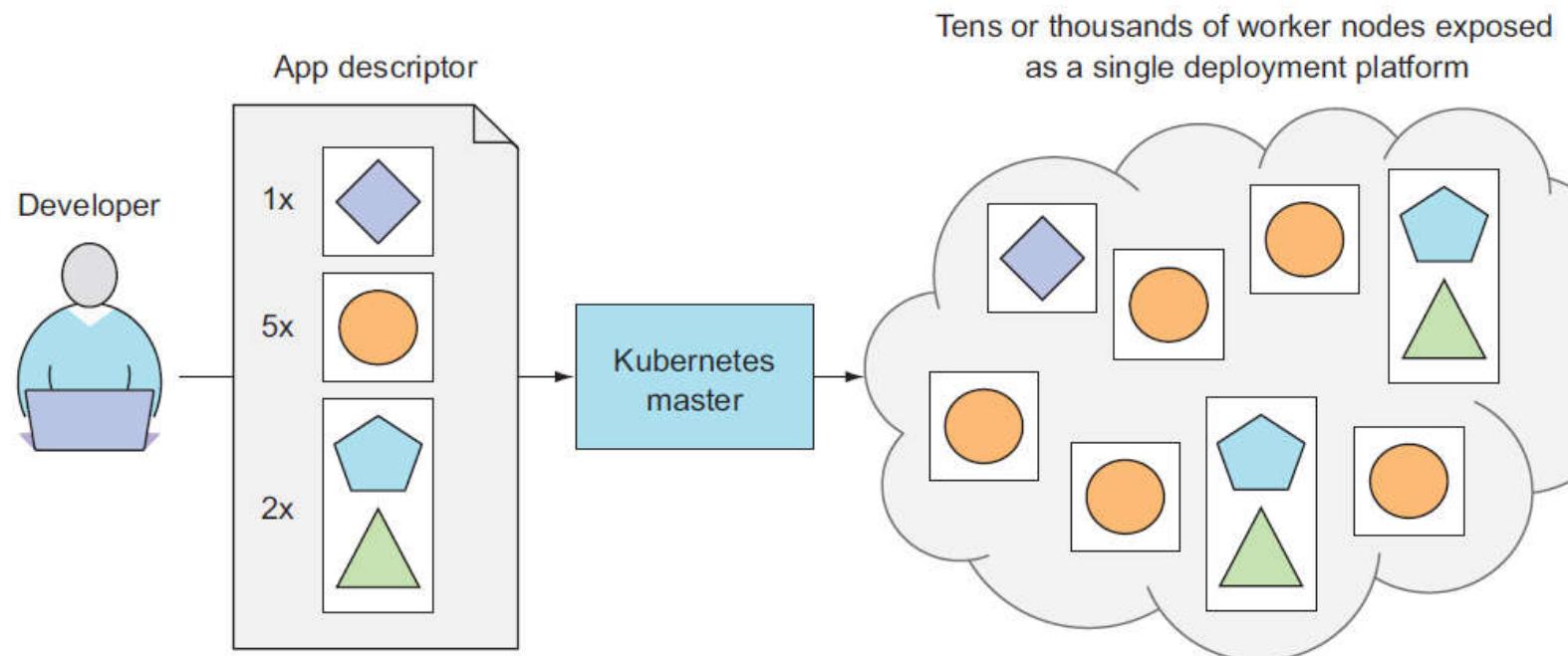
- Logiciel qui permet de déployer et de gérer à grandes échelles des applications containerisées ou micro-services
- Abstraction des couches sous jacentes par la technologie de conteneur
- Simplification du développement, du déploiement, de la gestion des différents environnements par la nature des conteneurs

- **Fonctionnalités**

- Gestion complète de cluster d'hôtes Docker (ou compatible OCI)
- Répartition de charge et auto-scalabilité
- Gestion du réseau à plat
- Fonctionnalités avancées de gestion des volumes
- Gestion de cluster distribués sur de multiple datacenters
- Mises à jour des hôtes/conteneurs intégrées (*rolling update*)



# FONCTIONNEMENT KUBERNETES





# DISTRIBUTIONS KUBERNETES



© ILKI 2018



# CHOIX DE LA DISTRIBUTION KUBERNETES

## Kubernetes Distributions

Community Supported	Vendor Distro (no value add)	Vendor Distro (with value add)	App Platforms / PaaS
kubernetes/kops kubernetes minikube	APPRENDA CANONICAL juju-solutions/bundle-canonical-kubernetes Diamanti ElasticKube Giant Swarm Google Container Engine HYPER.SH JETSTACK KUBE2GO RANCHER SUPRGIANT SUSE texncloud.com	Caicloud fabric8 fission samsung-cnct/k2 pivotal.io/kubo LIVEVIEWER vmware/photon-controller PLATFORM9 qstack BlueJeans STRATORGALE TECTONIC by CoreOS Telekube weavecloud	APPUIO IBM Bluemix FusionStage getup HASURA KeT KONTENA LAST.BACKEND OPENSHIFT RED HAT OPENSHIFT Container Platform RED HAT OPENSHIFT Dedicated origin



# MODULE 2

-

# ARCHITECTURE K8S



© ILKI 2018



# LES COMPOSANTS KUBERNETES

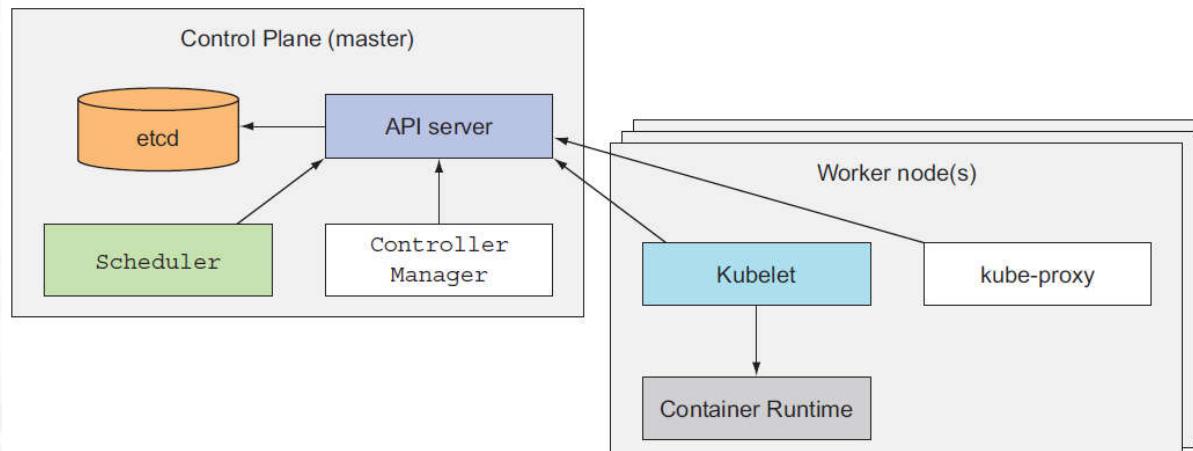


© ILKI 2018



# CLUSTER KUBERNETES

- D'un point de vue matériel, un cluster Kubernetes est composé de plusieurs nœuds
- Deux types de nœuds :
  - **Master** : nœud hébergeant le *Control Plane* de Kubernetes qui manage l'ensemble du système
  - **Worker** : nœud exécutant les applications déployées - « nodes »





# COMPOSANTS KUBERNETES

- **Les composants du *Control plane***

- **Kubernetes API Server**
  - Composant avec lequel les utilisateurs et les autres composants communiquent
- **Scheduler**
  - Composant qui ordonne le placement des containers applicatifs (affecte un nœud worker sur lequel déployer l'application)
- **Controller Manager**
  - Composant réalisant toutes les tâches de niveau cluster comme la réPLICATION des containers, le suivi des nodes, la gestion des pannes, le respect des stratégies ...
- **ETCD**
  - Stockage clef/valeur fiable et distribué permettant de stocker de façon persistante l'état et la configuration du cluster



# COMPOSANTS KUBERNETES

- **Les composants des nœuds worker**
  - Container runtime
    - Docker, rkt ou autre (compatible OCI)
  - Kubelet
    - Composant qui communique avec le serveur API et gère les conteneurs s'exécutant sur le nœud
  - **Kubernetes Service Proxy (kube-proxy)**
    - Composant permettant de répartir le trafic réseau entre les containers.



# COMMENT CES COMPOSANTS COMMUNIQUENT ?

- **Les composants systèmes Kubernetes ne communiquent que par le biais du serveur API**
  - Ils ne discutent pas directement entre eux
- **Le serveur API est le seul composant qui communique avec l'etcd**
  - Tous les autres ne discutent pas avec l'etcd directement
  - Ils modifient l'état du cluster en discutant avec le serveur API
- **Les connexions entre le serveur API et les autres composants sont quasiment toujours initiées par les composants**
  - Dans certains cas, le serveur API se connecte au Kubelet (récupération des logs, connexion à un container, port forwarding..)
  - Modèle PUB/SUB

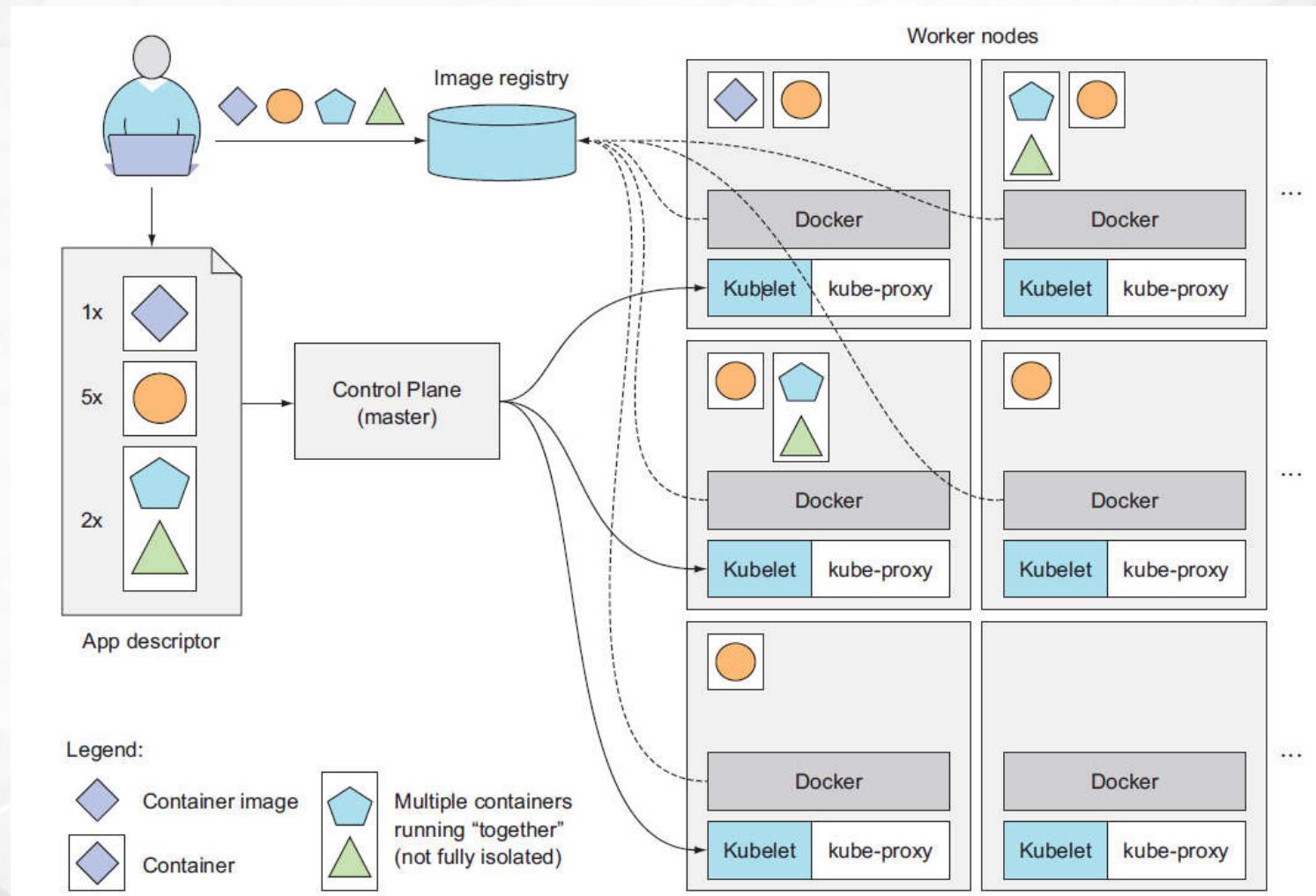


# COMMENT CES COMPOSANTS S'EXÉCUTENT ?

- **Les composants du « contrôle plane » peuvent être séparés**
  - Plusieurs instances peuvent également être lancées pour la haute disponibilité
- **Les composants du « contrôle plane » et le kube-proxy**
  - Peuvent être déployés sur le système directement ou en containers
  - Déploiement en containers = hosted-mode
- **Kubelet**
  - Est le seul composant qui s'exécute toujours sur le système, jamais en container
  - C'est lui qui est en charge de lancer les autres composants en conteneurs !
  - Ce qui veut dire que si l'on souhaite exécuter le « Control Plane » en tant que conteneurs, Kubelet sera également déployé sur les nœuds dit master



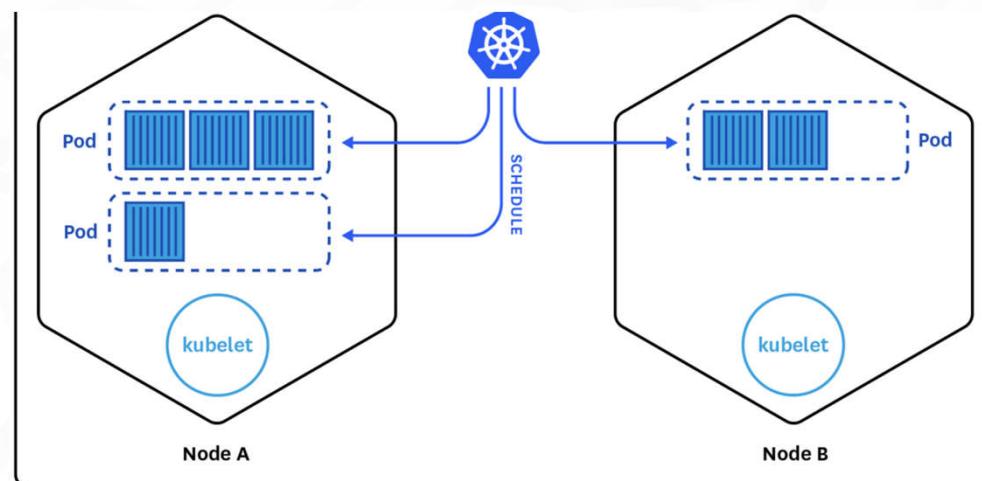
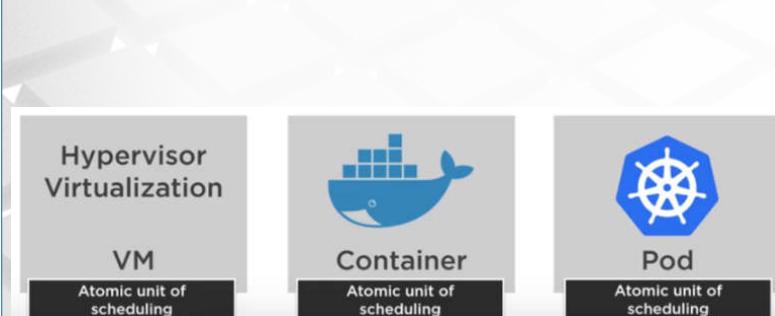
# VUE GLOBALE DU DÉPLOIEMENT D'UNE APPLICATION





# CONTAINER VS POD

- **Le pod est La plus petite ressource de Kubernetes**
  - C'est l'unité de base de l'ordonnancement
- **Un pod est un groupement de conteneurs**
  - A minima, un pod peut contenir un unique conteneur
  - Les conteneurs ne sont pas des ressources K8S





# ETCD



© ILKI 2018



# ETCD

- **Rappel : Qu'est-ce que l'etcd ?**

- Stockage fiable et distribué permettant de stocker de façon persistante l'état et la configuration du cluster
- Développé par CoreOS (Redhat depuis 2018)
- C'est le seul endroit où Kubernetes stocke l'état du cluster et l'ensemble des métadonnées

- **Caractéristiques**

- Stockage de type « key-value »
- Stockage distribué = possibilité d'exécuter plusieurs instances pour améliorer les performances et la disponibilité

- **Le serveur API est le seul composant qui communique directement avec l'etcd**

- Tous les autres composants lisent et écrivent dans l'etcd de manière indirecte (par le biais du serveur API)



# SERVEUR API



© ILKI 2018

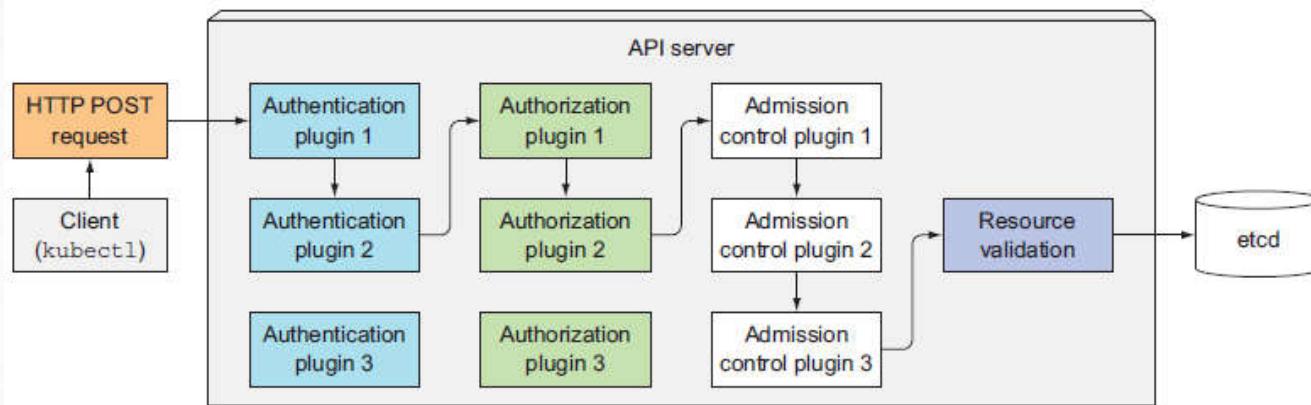


# SERVEUR API

- **Composant central de l'architecture Kubernetes**
  - Utilisé par tous les composants et par les administrateurs
- **interface CRUD (Create, Read, Update, Delete)**
  - Permet d'effectuer des requêtes et de modifier l'état du cluster via une API RESTful
  - **Il stocke cet état dans etcd !**
- **Il permet également de :**
  - Gérer l'OCC
  - Valider la bonne configuration des objets
  - Notifier les composants et clients des changements sur le ressources sur un modèle PUB/SUB



# COMMENT EFFECTUE-T-IL LA VALIDATION DES REQUÊTES ?

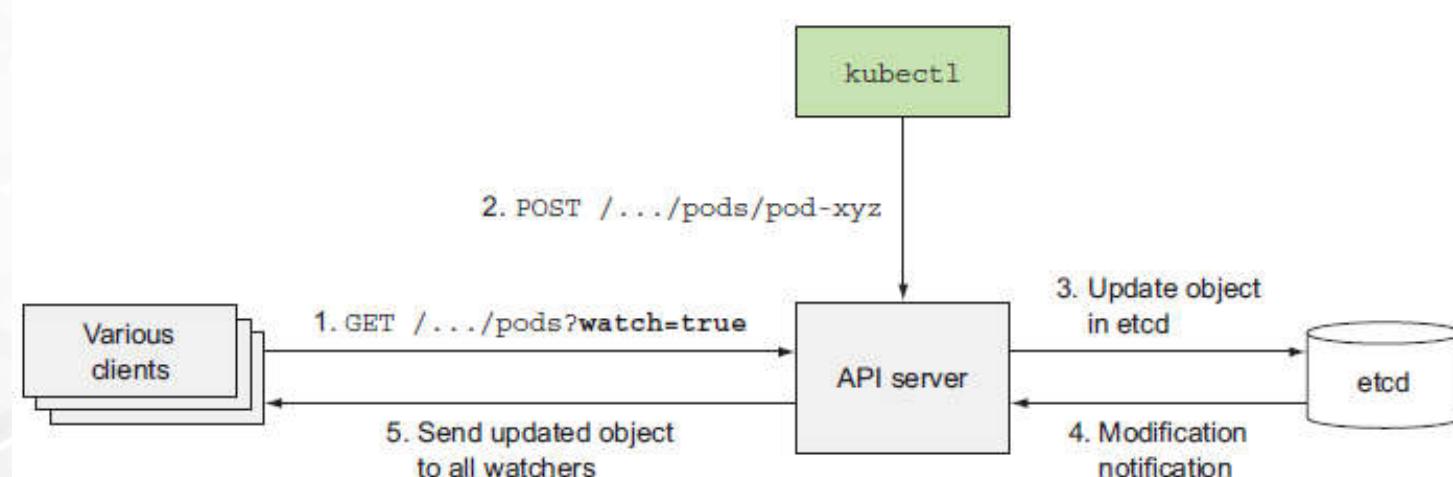


1. **Authentification**
    - Extraction du username, user ID et group par le biais du certificat client ou de l'en-tête HTTP
  2. **Autorisation**
  3. **Admission : Validation ou modification de la ressource**
  4. **Stockage dans l'etcd**
- 
- **Note**
    - Si la requête consiste à simplement lire de la donnée, celle-ci ne passe pas par l'étape d'admission



# COMMENT LES CLIENTS SONT NOTIFIÉS DES CHANGEMENTS ?

- Rappel : le serveur API ne modifie pas les objets
- Serveur API « watch system »
  - Les clients ouvrent une connexion HTTP
  - Ils reçoivent un flux de modifications sur l'objet surveillé





# SCHEDULER



© ILKI 2018

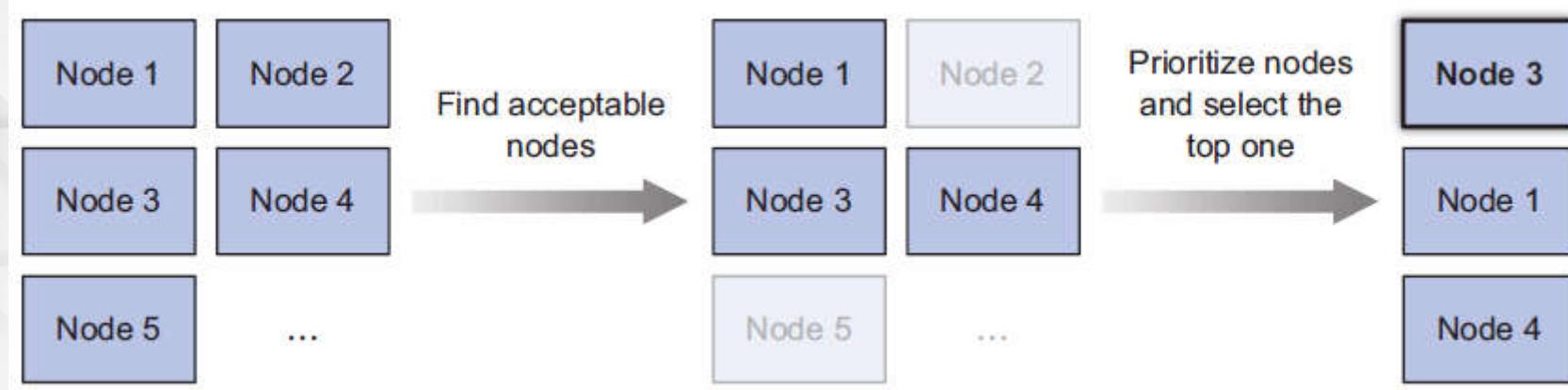


# SCHEDULER

- **Que fait Le scheduler ?**
  - Il détermine sur quel nœud vont être déployés les pods
- **Comment fait-il ?**
  - Lorsqu'une demande de création ou de modification de pod est nécessaire, le scheduler, en mode « watch », prend en compte la demande
  - Il affecte ensuite un nœud à chaque nouveau pod si ce n'est pas encore fait
    - **Note :** il n'informe pas le nœud, il ne fait que modifier la définition du pod via le serveur API dans l'ETCD
    - Le serveur API notifie le composant Kubelet
    - Le Kubelet crée le pod
- **Le scheduler doit donc connaître l'état du cluster à tout moment.**
- **Deux possibilités :**
  - Demander au serveur API à chaque fois qu'un pod doit être ordonné (impact sur les performances)
  - Conserver les informations sur le cluster en cache (meilleures performances)



# COMMENT LE SCHEDULER FAIT POUR CHOISIR LA BONNE NODE ?



- **Etape 1**
  - Filtre la liste des nodes pour obtenir une liste adaptée
- **Etape 2**
  - Priorise les nœuds obtenus précédemment pour choisir le plus adéquat
  - Si plusieurs nœuds obtiennent le même score, un système de Round Robin est utilisé
- **Une fois le nœud déterminé, il modifie les spécifications du pod**



# COMMENT FONCTIONNE LE CHOIX DE LA NODE ?

- **Passage de chaque node par des fonctions de prédicat et de priorisation**
- **Ces fonctions vont vérifier plusieurs choses telles que :**
  - Est-ce que le nœud possède le matériel spécifié ?
  - Est-ce que le nœud possède encore assez de ressources physiques (RAM, disque...) ?
  - Si le pod nécessite d'être bindé sur un port particulier, est-ce que ce port est disponible sur le nœud ?
  - Existe-t-il des règles d'affinités/anti-affinités ?



# CONTROLLER MANAGER



© ILKI 2018



# CONTROLLER MANAGER

- **Rappel**

- Le serveur API a pour unique rôle de stocker les ressources dans ETCD et notifier les composants des changements effectués
- Le Scheduler ne fait qu'assigner un nœud à un pod
- Il faut donc d'autres composants pour assurer que l'état actuel du système est identique à l'état désiré

- **rôle du controller manager**

- S'assurer que l'état désiré est valide et agir dans le cas contraire
- En réalité, c'est une combinaison de multiples contrôleurs
- Chaque contrôleur est un processus unique



# LES CONTRÔLEURS

- **liste non exhaustive des contrôleurs présents :**
  - Replication Manager
  - ReplicaSet, DaemonSet, and Job controllers
  - Deployment controller
  - StatefulSet controller
  - Node controller
  - Service controller
  - Endpoints controller
  - Namespace controller
  - PersistentVolume controller
  - ....



# LES CONTRÔLEURS

- **Tous les contrôleurs utilisent le mécanisme de watch pour attendre les changements transmis par le serveur API**
  - Et effectuent une action en conséquence
- **En général**
  - Les contrôleurs sont dans une boucle de réconciliation : est-ce que l'état actuel correspond à l'état désiré ?
  - Ils mettent ensuite à jour le nouvel état dans ETCD
  - Ils effectuent également des opérations de « re-listing » régulières
- **Les contrôleurs ne discutent jamais directement entre eux !**



# KUBELET



© ILKI 2018



# KUBELET

- **Rappel**

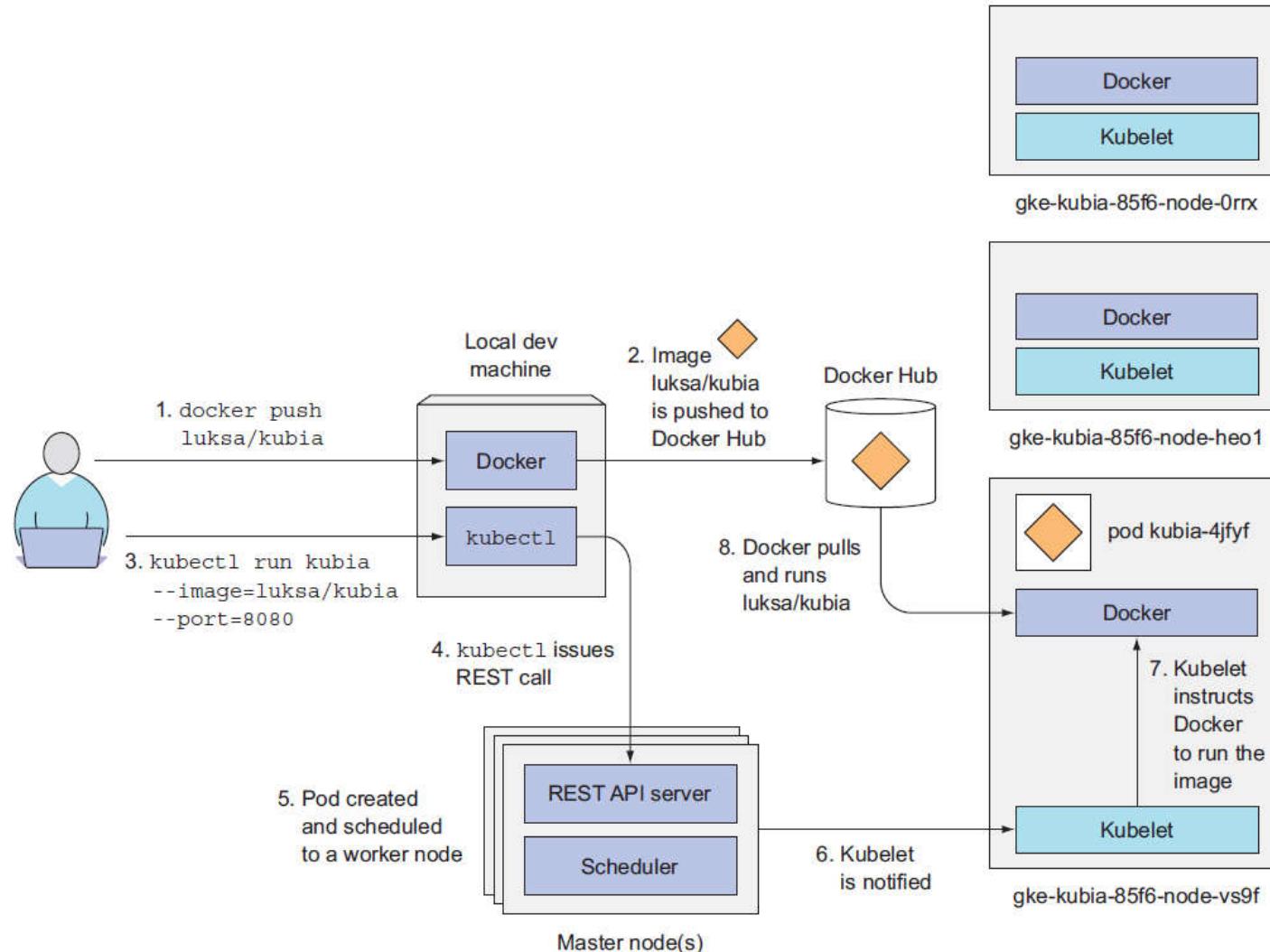
- Contrairement aux composants du **Control Plane**, Kubelet s'exécute à la fois sur les nœuds dits « master » et « worker »

- **Kubelet est responsable de tout ce qui est exécuté sur le nœud**

- Son rôle initial est d'enregistrer le nœud en créant une ressource « node » auprès du serveur API
- Ensuite, il surveille en permanence les informations provenant du serveur API, dans le cas où des conteneurs devraient s'exécuter sur sa node
- Il s'adresse ensuite au CRI « container runtime interface » pour démarrer le conteneur
- Il surveille l'état de santé des conteneurs et envoie régulièrement leur état, les évènements associés et la consommation des ressources, au serveur API
- Enfin, il est responsable du cycle de vie des containers sur sa node



# EXÉCUTION DES PODS





# KUBERNETES SERVICE PROXY



© ILKI 2018



# KUBE-PROXY

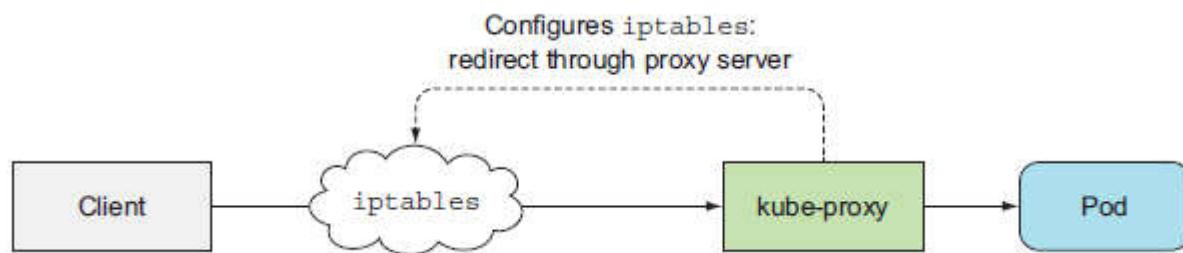
- Comme pour le kubelet, chaque NODE exécute le « kube-proxy »
- Rôle :
  - S'assurer que les clients puissent se connecter aux services définis par le biais de l'API
    - Adresse IP et port
  - Lorsque le service est fourni par plus d'un seul pod, il est également en charge de la répartition de charge



# KUBE-PROXY

- **Proxy-mode: userspace**

- Implémentation d'origine du kube-proxy
- Il utilise un process permettant d'accepter les connexions et de les renvoyer vers les pods
- Pour cela, il configure des règles ***iptables*** pour rediriger les connexions vers le serveur proxy
- Répartition de charge basée sur du round-robin
- Impact sur les performances

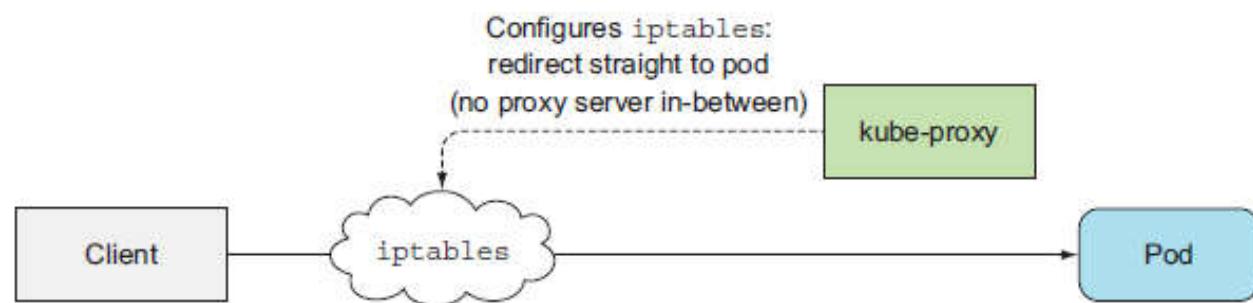




# KUBE-PROXY

- **Proxy-mode: iptables & IPVS**

- Désormais, mode par défaut de kube-proxy
- Il configure des règles *iptables* pour rediriger les connexions directement vers les pods
- La répartition de charge est faite de manière aléatoire
- Iptables Meilleures performances si peu de services et pods. LB aléatoire
- IPVS – Très bonne performances fixe. LB configurable (RR, LC,... )





# PAUSE CONTAINER

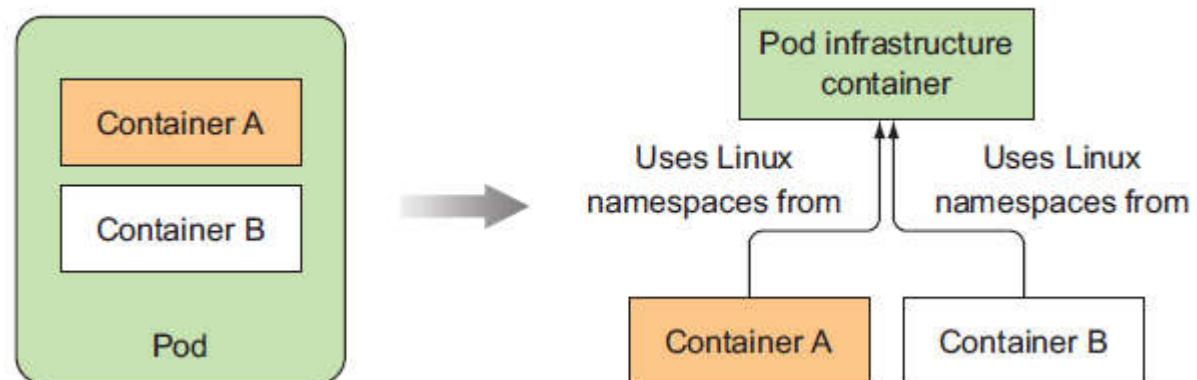


© ILKI 2018



# PAUSE CONTAINER

- Ce container permet de faire fonctionner tous les containers d'un pod ensemble
  - Gère les communication réseau et les namespaces
  - Un seul container infrastructure « pause container » par pod
  - Crée au démarrage d'un pod, supprimé à sa suppression





# LE FONCTIONNEMENT DU RÉSEAU

-

# LES MODÈLES ET PLUGINS RÉSEAU



© ILKI 2018



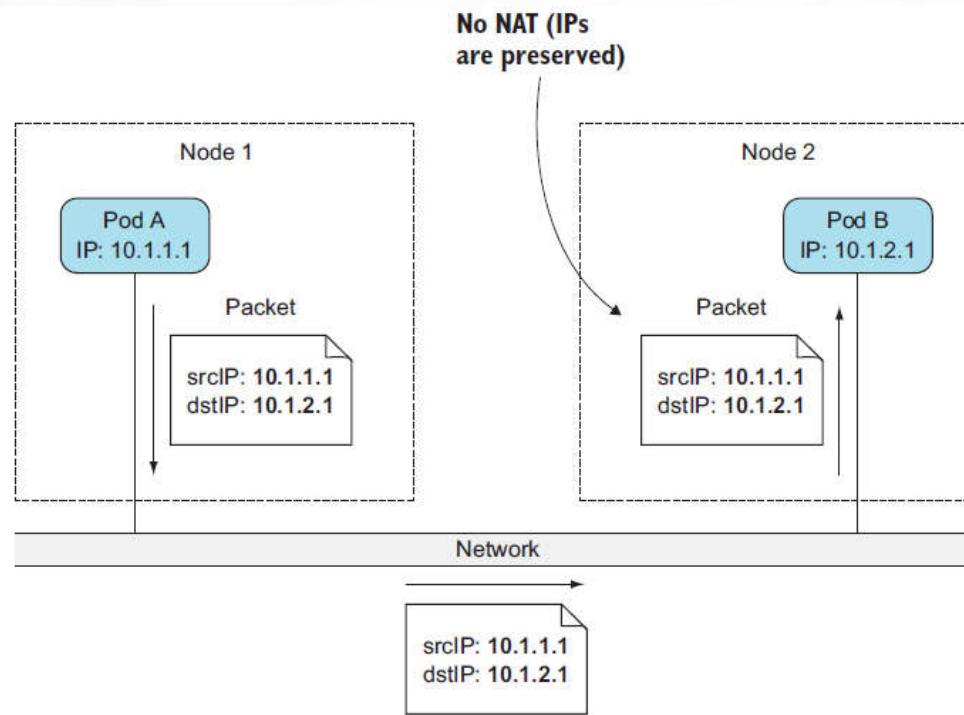
# MODÈLE RÉSEAU K8S

- **Kubernetes impose aux réseaux de suivre les règles suivantes :**
  - Tous les pods peuvent communiquer entre eux sans NAT
  - Les nodes peuvent communiquer avec les pods (et inversement) sans NAT
  - L'IP vue par le container est la même que l'IP vue par les composants externes
- **Deux types de mise en place :**
  - Utilisation du réseau K8S par défaut
  - Utilisation de CNI et des plugins



# RÉSEAU INTER-POD

- Chaque pod possède sa propre adresse IP
  - Il peut communiquer avec tous les autres pods par le biais d'un réseau à plat et sans utiliser de NAT
  - Il peut communiquer avec les autres pods qu'ils s'exécutent ou non sur la même node





# RAPPEL : LE RÉSEAU

- **Le réseau peut être défini de deux façons différentes : l'underlay et l'overlay.**
  - Il est possible avec Kubernetes d'utiliser les deux.
- **L'underlay**
  - Réseau physique, matériel composé des switchs, routeurs...
- **L'overlay**
  - Réseau virtuel composé de vlan, veth (interface virtuelle) et VxLAN
  - Encapsulation de l'architecture réseau matériel
  - Création de tunnels entre les hôtes



# CONTAINER NETWORK INTERFACE (CNI)

- **CNI – container network interface.**
  - Spécifications et librairies écrites en GO
  - Le but est de faciliter l'intégration de plugins réseau
  - Il gère la mise en place de l'interface dans un namespace et sa configuration sur la node (connexion avec un bridge, gestion des routes)
- **plugin réseau**
  - Automatise les configurations réseaux.
  - Création et gestion du réseau facilité.



# CNI ET PLUGINS RÉSEAU

- **Multitude de plugins**

- Flannel
- Calico
- Weave Net
- Canal
- Romana
- Kopeio-vxlan
- ...





# FLANNEL



- Développé par CoreOS
- CNI le plus populaire
- Facile à installer et configurer => service « flanneld » sur chaque nœuds du cluster K8S
- Flannel peut utiliser l'ETCD pour stocker son état.
- VxLAN (Encapsulation UDP)
- Les containers d'un même hôte communiquent via le réseau « bridge » de docker.
- Les containers d'hôtes différents utilisent le tunnel VxLAN en UDP pour communiquer entre eux.



# CALICO



- Projet CNI populaire.
- Plus complexe à mettre en place que Flannel
- Calico utilise le protocole de routage BGP pour permettre les communications entre les containers => pas d'encapsulation L3 => meilleures performances que Flannel.
- Le fait de ne pas encapsuler les paquets L3 via VxLAN permet également de rendre plus simple le troubleshooting du réseau.
- Calico dispose de politiques de gestion du réseau plus poussés que ses concurrents et offre une excellente synergie avec Istio (Service Mesh)
- Il existe une version avec support de Calico (payant)



# CANAL



- Projet CNI avec pour objectif initial d'apporter les fonctionnalités d'encapsulation de Flannel tout en supportant les politiques de gestion du réseau de Calico.
- Ce projet n'existe plus
  - Durant son développement, les projets Flannel et Calico se sont standardisés, et ils peuvent maintenant coexister ensemble => Cette cohabitation à le nom de Canal, même si le projet Canal originel n'existe plus.



# STRATÉGIES RÉSEAU

- **Une stratégie réseau définit comment les pods sont autorisés à communiquer entre eux**
  - Utilisation des labels afin de sélectionner les pods et de définir les règles
  - Les stratégies réseau sont implémentées par le plugin réseau
- **Par défaut**
  - Si aucune stratégie n'existe tout le trafic entrant et sortant est autorisé au sein d'un même *namespace* Kubernetes
- **Stratégies disponibles par défaut**
  - Deny all ingress traffic
  - Allow all ingress traffic
  - Deny all egress traffic
  - Allow all egress traffic
  - Deny all ingress traffic and all egress traffic



# LES DIFFÉRENTS MODES DE DÉPLOIEMENT DE K8S



© ILKI 2018



# LES MODES DE DÉPLOIEMENT

- **Kubernetes peut fonctionner sur différentes plateformes**
  - Sur un PC, sur des serveurs, chez un cloud provider..
- **Choix entre une solution complètement managée ou customisée ?**
- **Les possibilités**
  - Solutions installées sur une machine en local
  - Solutions hébergées
  - Solutions Cloud clés en main
  - Solutions on-premise clés en main
  - Solutions custom



# LES MODES DE DÉPLOIEMENT

Type	Commentaires	Exemple
<b>Solutions installée sur une machine en local</b>	<ul style="list-style-type: none"><li>Meilleur moyen pour commencer avec Kubernetes</li><li>Utilisation pour du développement ou des tests</li></ul>	Minikube, Kubeadm-dind...
<b>Solutions hébergées</b>	<ul style="list-style-type: none"><li>Complètement managée</li><li>Garantie de fonctionnement</li></ul>	AppsCode.com, Amazon EKS, Azure AKS, Google GKE, OpenShift, Platform9, StackPoint.io ...
<b>Solutions Cloud clés en main</b>	<ul style="list-style-type: none"><li>Support et développement de la communauté</li><li>Flexibilité des cloud providers</li><li>Gestion du control plan</li></ul>	AWS EC2, Alibaba CCS, Azure ACS, Google GCE
<b>Solutions on-premise clés en main</b>	<ul style="list-style-type: none"><li>Déploiement sur votre cloud privé</li><li>Nécessite des ressources internes</li></ul>	Agile Stacks, APPUiO, Pivotal Container Service, Rancher 2.0, SUSE CaaS Platform
<b>Solutions custom</b>	<ul style="list-style-type: none"><li>Liberté de configuration</li><li>Nécessite de l'expertise</li></ul>	From Scratch



# MINIKUBE ET KUBEADM

- **Minikube**

- Kubernetes single-node
- Déployé dans une VM sur votre PC
- Outil recommandé pour un déploiement en local
- L'installation est complètement automatisée

- **Kubeadm-DIND**

- Kubernetes multi-node
- Le seul prérequis est la présence d'un daemon Docker
- Utilisation de la technique « docker-in-docker » (DIND)

- **L'utilisation de minikube est recommandée**

- Notamment, car l'outil est plus mature et moins dépendant à l'environnement local



# GKE VS AKS VS EKS

	GKE	AKS	EKS
<b>Offering</b>	GA	GA	GA
<b>K8s version</b>	1.10	1.9	1.10
<b>Multi AZ</b>	Yes	Partial	Yes
<b>Upgrades</b>	Auto / On-demand	On-demand	Unclear
<b>Auto-scale</b>	Yes	Self-deployed	Self-deployed
<b>RBAC</b>	Yes	Yes	Yes
<b>Network Policy</b>	Yes	Self-deployed	Self-deployed
<b>GPU support</b>	Yes	Yes	Yes
<b>Persistent Volumes</b>	Block	Block and CIFS	Block
<b>Load Balancer</b>	Yes	Yes	Yes
<b>Mgmt via (vendor) CLI</b>	Complete	Complete	Minimal



# GOOGLE GKE

- Google utilise des containers depuis plus de 10ans
- Intégration avec les fonctionnalités de GCP :
  - Load Balancing utilisé pour GCE
  - Pools de nœuds
  - Auto-scaling des nodes.
  - Mise à jour automatique pour les nœuds du cluster
  - Réparation automatique des nœuds afin de maintenir la santé et la disponibilité
  - Logging et Monitoring avec Stackdriver
  - Google Cloud Build pour construire des images
  - Google Container Registry pour stocker ces images
- Mise à jour automatique des masters lorsque la nouvelle version est stable



# ATELIER 1

-

## DÉPLOIEMENT KUBERNETES



© ILKI 2018



# MODULE 3

-  
**ADMINISTRATION**



© ILKI 2018



# OUTILS D'ADMINISTRATION

- **2 possibilités**
  - Interface graphique (dashboard)
  - Ligne de commandes
- **L'interface graphique n'est pas déployée par défaut**
  - La ligne de commande est donc privilégiée pour démarrer
- **Kubectl peut être déployé sur toutes les plateformes**
  - Il ne fait qu'envoyer des requêtes REST au serveur API
- **Dans le cas de l'administration de plusieurs clusters**
  - Il est préférable d'utiliser kubectl



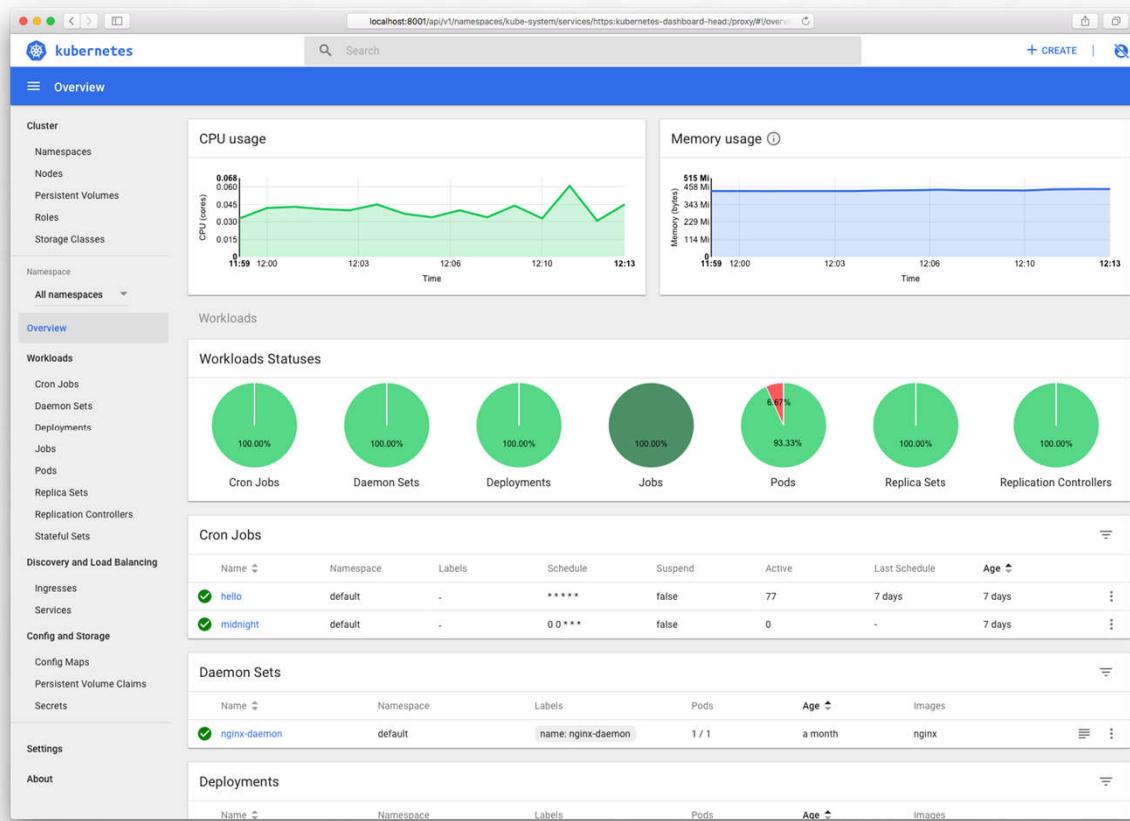
# KUBECTL

- **Kubectl permet de dialoguer avec le serveur API**
  - Afficher, créer, supprimer, mettre à jour les ressources :
    - **Get** – afficher la liste des éléments d'une ressource.
    - **Delete** – supprimer une ressource.
    - **Create** – créer une ressource.
    - **Apply** – mettre à jour une ressource
    - **Describe** – afficher les détails d'une ressource.
  - Afficher les informations relative à notre cluster, fichier de configuration :
    - # kubectl cluster-info
    - # kubectl config view
    - ...



# KUBERNETES DASHBOARD

- Interface web





# ATELIER 2

-

# ADMINISTRATION



© ILKI 2018



# MODULE 4

-  
**LES RESSOURCES**



© ILKI 2018



# NAMESPACES

- **Kubernetes supporte la création de multiples clusters virtuels**
  - Ils sont appelés **namespaces**
- **Quand utiliser plusieurs namespaces ?**
  - Partage des ressources entre différentes équipes/projets/environnements
  - À noter qu'ils permettent de définir un périmètre pour les noms utilisés
    - Les noms de ressource doivent être uniques dans un namespace donné
- **Kubernetes s'initialise avec 3 namespaces**
  - **Default** – namespace par défaut pour les objets sans namespace défini
  - **Kube-system** – namespace pour les objets créés par le système Kubernetes
  - **Kube-public** – namespace créé automatiquement et visible par tous les utilisateurs (y compris ceux qui ne sont pas authentifiés).

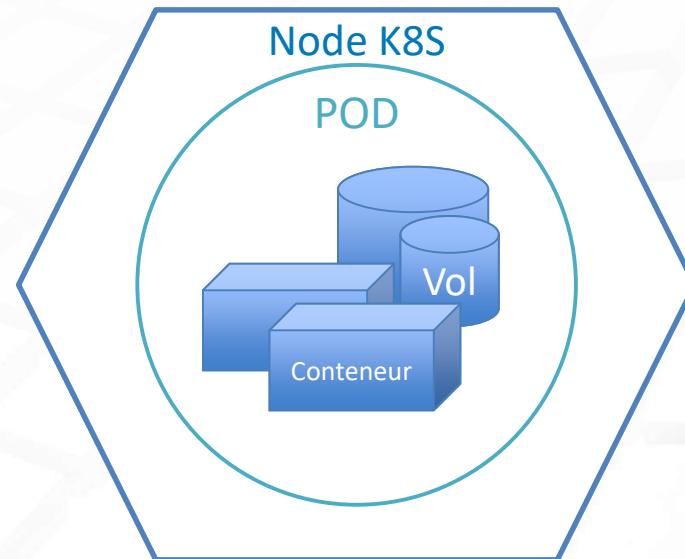
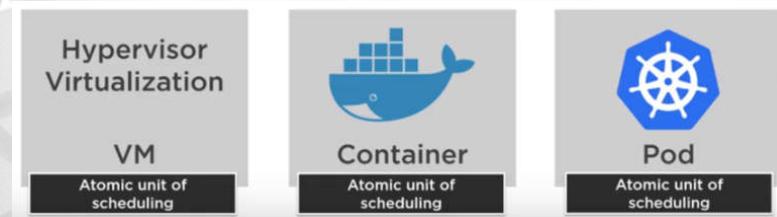
```
$ kubectl get namespaces
NAME        STATUS   AGE
default     Active   1d
kube-system Active   1d
kube-public Active   1d
```



# POD

- **Un pod est un groupe d'un ou plusieurs containers**

- Ces containers fonctionnent toujours sur le même nœud et dans le même namespaces Linux
- Réseau/stockage partagés
- Comme une machine virtuelle, chaque pod possède son adresse IP, hostname, process ...





# ATELIER 3

-

## NAMESPACES & PODS

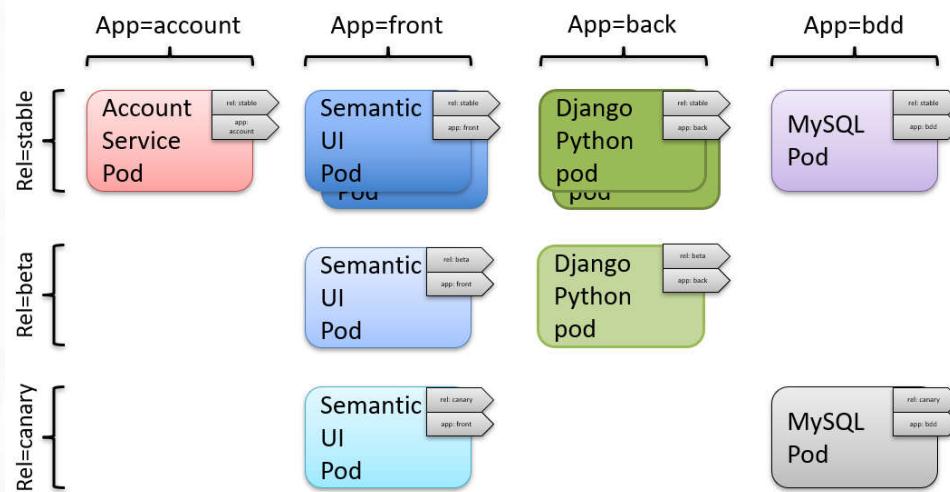


© ILKI 2018



# LABELS

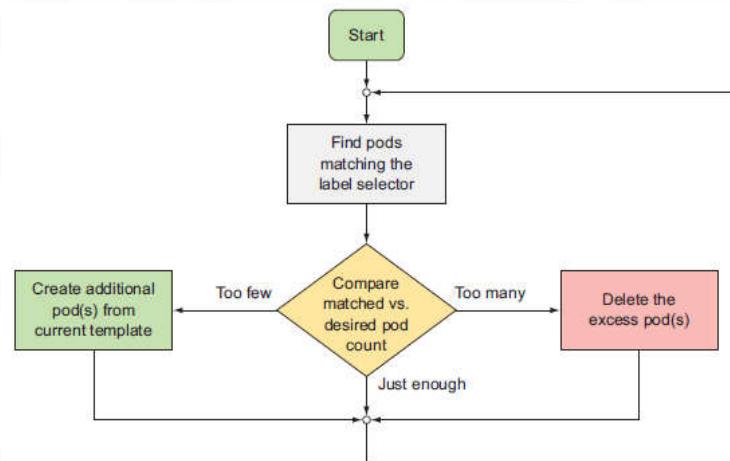
- Les Labels sont utilisés par Kubernetes pour organiser l'ensemble des ressources
  - Fonctionnement clé-valeur
  - Ils sont ensuite utilisés lorsque vous sélectionnez des ressources via les *Labels selectors*
  - Une ressource peut avoir plus d'un seul label
  - Les labels peuvent être fixés à la création ou après





# REPLICATIONCONTROLLERS

- Ressource qui s'assure que ses pods sont toujours en fonctionnement
  - Si le pod disparait pour une quelconque raison, il demandera la création d'un nouveau pod
- Généralement il gère plusieurs copies du pod (répliques)
  - Il surveille donc constamment la liste des pods en fonctionnement pour s'assurer qu'elle contient le nombre désiré





# REPLICATIONCONTROLLERS VS REPLICASETS

- **ReplicatSets = ReplicationControllers nouvelle génération**
  - Fin de vie des ReplicationControllers
- **Les deux ressources ont le même comportement général**
- **ReplicationControllers label selector**
  - Permet uniquement de sélectionner les pods qui possèdent un label spécifique (clé=valeur)
- **ReplicaSets label selector**
  - Permet également de sélectionner les pods qui ne contiennent pas un label, ou qui ont un label qui contient une clé spécifique (peu importe la valeur).



# DAEMONSETS

- **Les ressources précédentes permettent de faire fonctionner un certain nombre de pods**
  - Peu importe où dans le cluster
  - Sur les nœuds selon un label défini
- **Et dans le cas où il doit y avoir un pod sur chaque nœud ?**
  - Comme par exemple un collecteur de métriques/logs
- **Il est possible d'utiliser la ressource DaemonSet**
  - Elle n'a pas de notion de nombre de répliques désirés
  - Elle vérifie juste qu'un pod fonctionne sur chaque nœud



# ATELIER 5

-

## REPLICASETS



© ILKI 2018



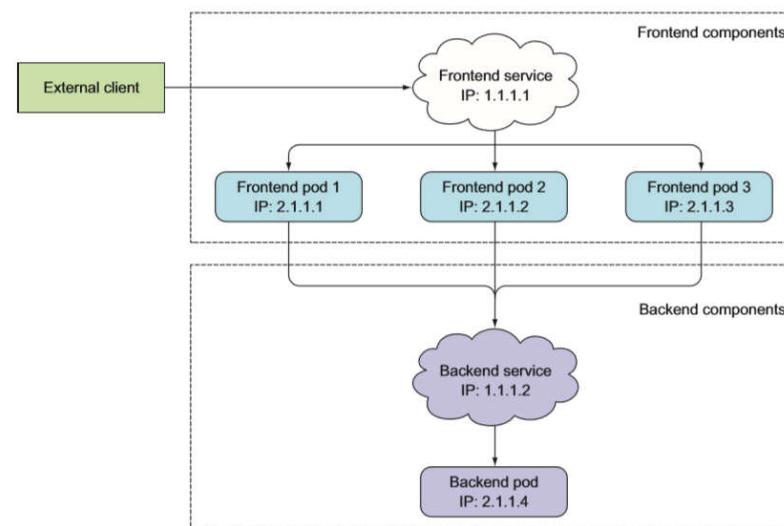
# SERVICES

- **Constat**

- Les pods sont éphémères
- Kubernetes assigne une adresse IP à un pod une fois que celui-ci est ordonné et avant qu'il soit démarré
- Il est possible que plusieurs pods fournissent le même service (Répliques)

- **Pour pallier ces problèmes, Kubernetes met à disposition une ressource appelée Service**

- Utilisée pour créer un seul et constant point d'entrée pour un service fourni par un ou plusieurs pods



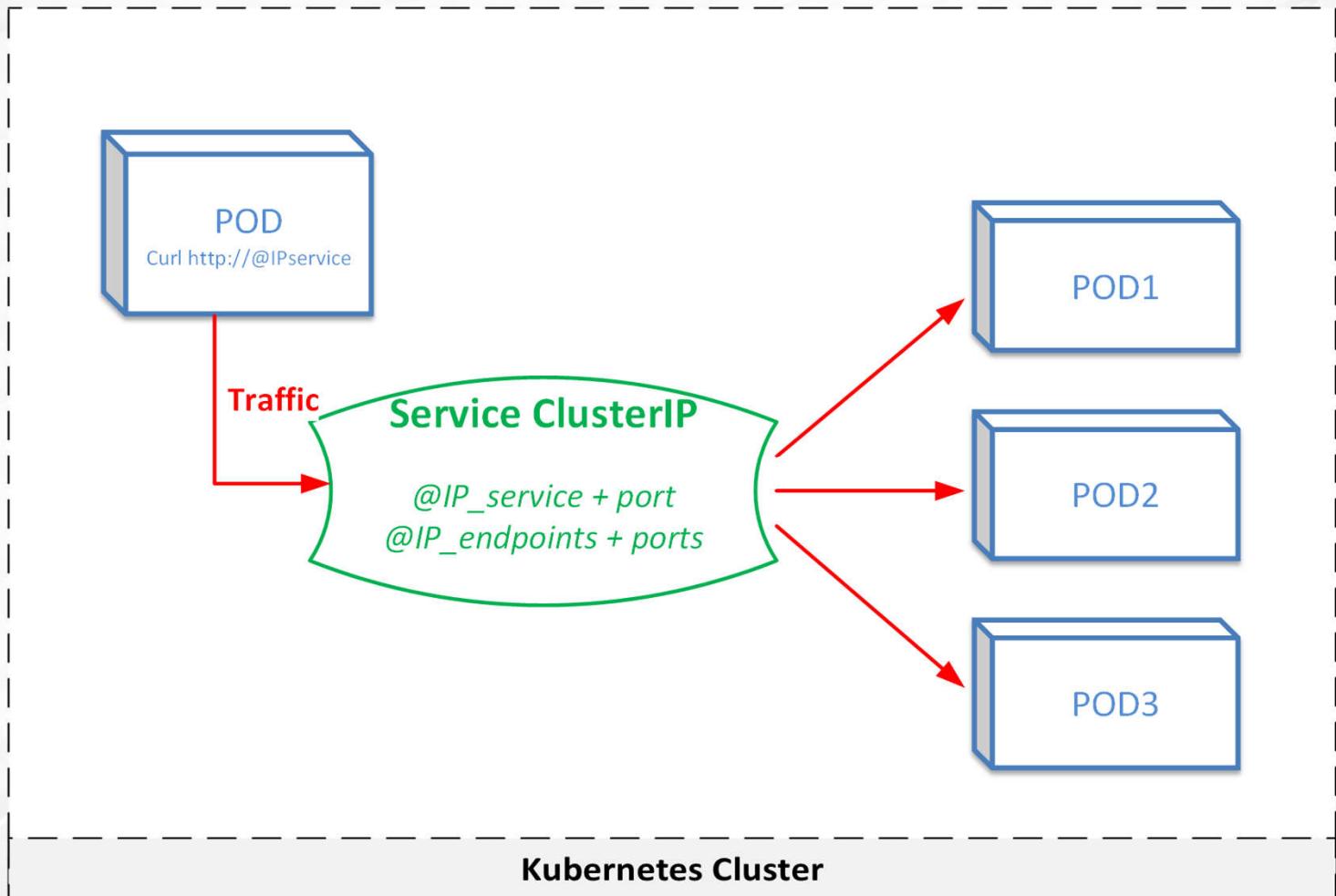


# SERVICES

- **3 types de services**
  - **ClusterIP**
    - Expose le service sur une adresse IP interne au cluster
  - **NodePort**
    - Expose le service sur l'IP de chaque nœud sur un port statique
  - **LoadBalancer**
    - Expose le service à l'extérieur à l'aide d'un load balancer (du fournisseur cloud ou physique ou virtuel).

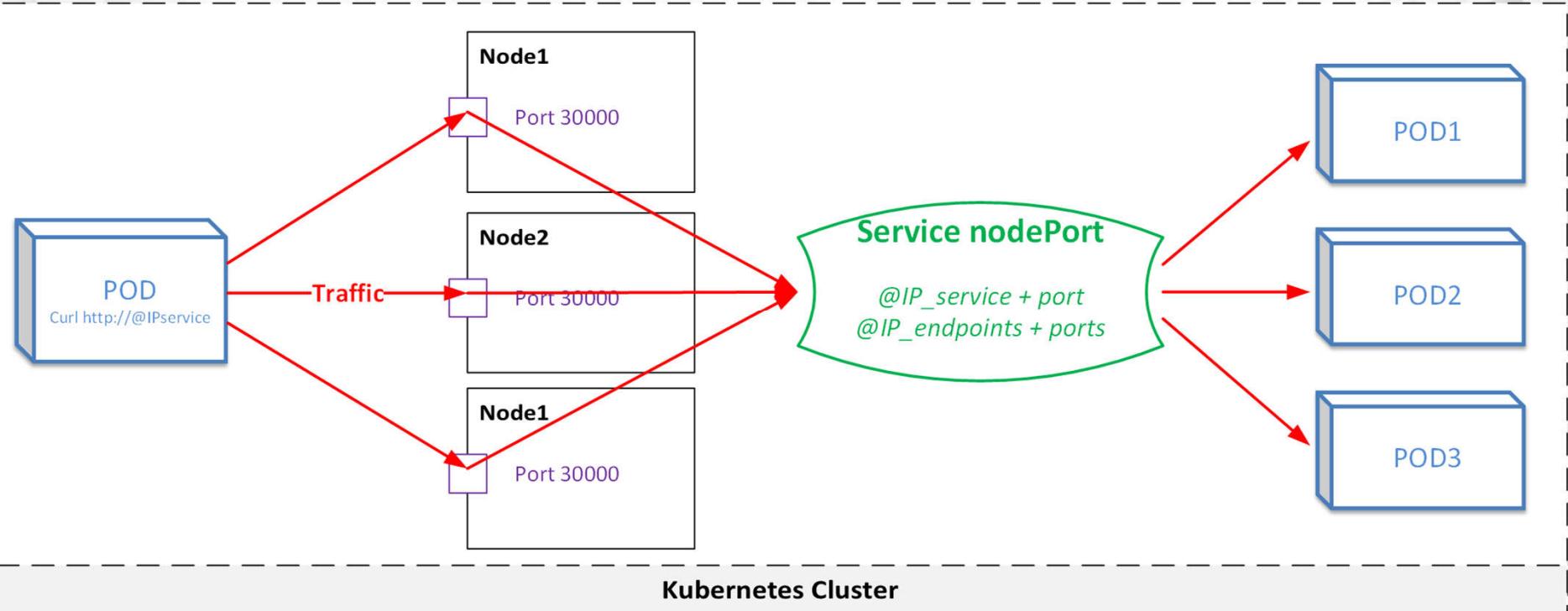


# SERVICES - CLUSTERIP



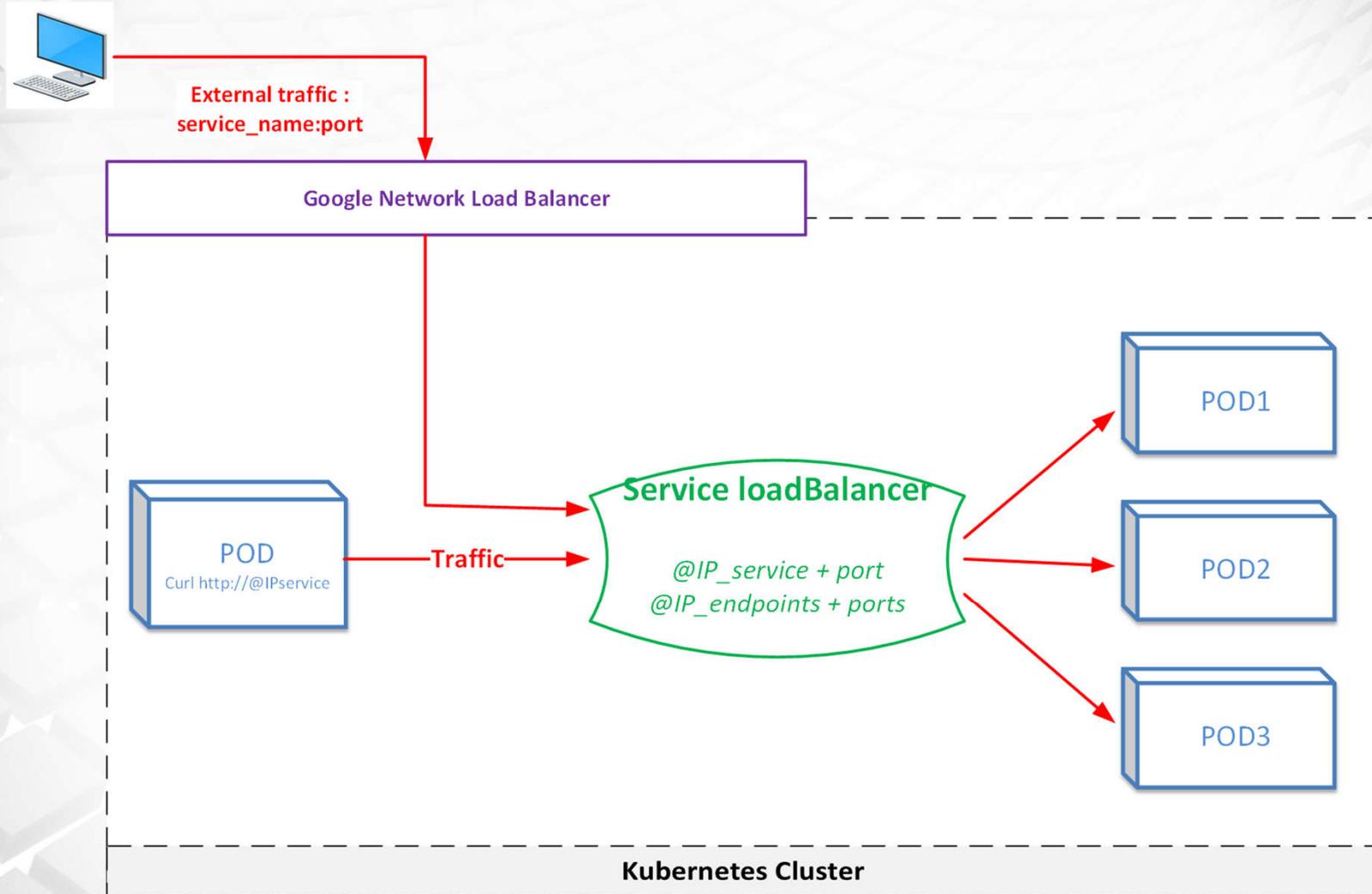


# SERVICES - NODEPORT





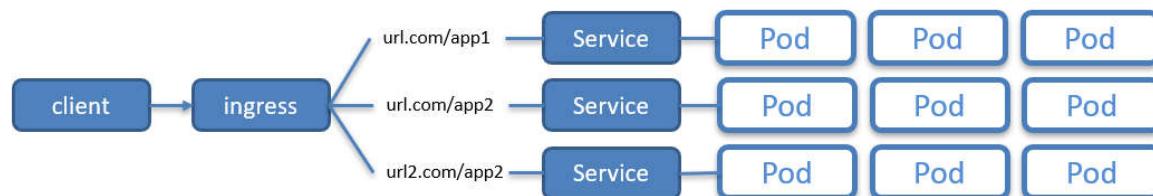
# SERVICES - LOADBALANCER





# INGRESS

- **Constat**
  - Chaque service de type Load Balancer a besoin de son propre Load Balancer en frontal
  - Chaque service de type Load Balancer a besoin d'une IP « publique »
- **La ressource Ingress n' a besoin que d'une seule IP Publique**
  - Agit comme un Load Balancer de niveau 7
  - C'est le type de ressource à privilégier lorsque le service à exposer est un service Web





# ATELIER 6

-

## SERVICES



© ILKI 2018



# DEPLOYMENTS

- **Un Deployment est une ressource « high-level » utilisée pour déployer des applications et les mettre à jour**
  - Elle permet de le faire de manière déclarative plutôt que de le faire par le biais d'un ReplicationController/ReplicaSet
- **Lorsqu'un Deployment est créé, *a minima* un ReplicaSet est créé**



- **Un Deployment est également composé :**
  - D'un Label selector
  - D'un nombre de répliques désiré
  - D'un modèle de pod



# ATELIER 7

-

## DEPLOYMENTS & ROLLING UPDATE



© ILKI 2018



# SANTÉ DES PODS

- **Kubelet est en charge de la santé des containers**
  - Si le process principal du container crash, Kubelet le redémarrera automatiquement
- **Que se passe-t-il si l'application fige sans que le processus crash ?**
  - Kubernetes peut vérifier périodiquement si un container est encore vivant
  - Il est possible de définir une sonde pour chaque container dans la description du pod



# SANTÉ DES PODS

- **3 méthodes pour vérifier l'état de santé d'un container**
  - **HTTP GET** – requête sur une @IP, un port et un chemin
    - Si la réponse est un code d'erreur, redémarrage
  - **TCP Socket** – ouverture d'une connexion TCP sur un port
    - Si la connexion n'est pas correctement établie, redémarrage
  - **Exec** – exécution d'une commande
    - Si l' « exit status code » est différent de 0, redémarrage
- **2 types de sondes**
  - Liveness probe
    - Si un container n'apparaît plus comme fonctionnel, il est tuer/redémarrer
  - Readiness probe
    - S'assure que le container est prêt avant de participer au service fourni



# MODULE 6

-

# GESTION DES VOLUMES



© ILKI 2018



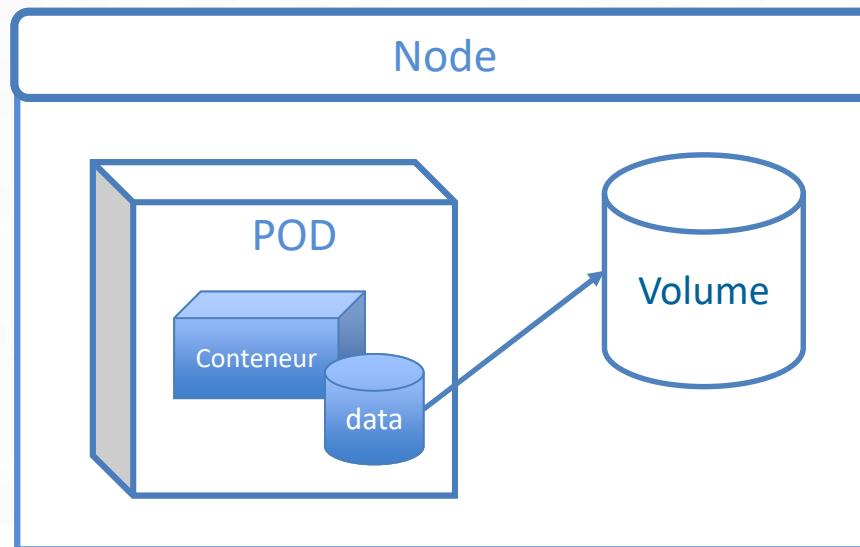
# VOLUMES

- **Problématique:**

- Comment conserver les données des pods alors que ceux-ci sont éphémères ?
- Comment partager des données entre pods ?

- **Solution:**

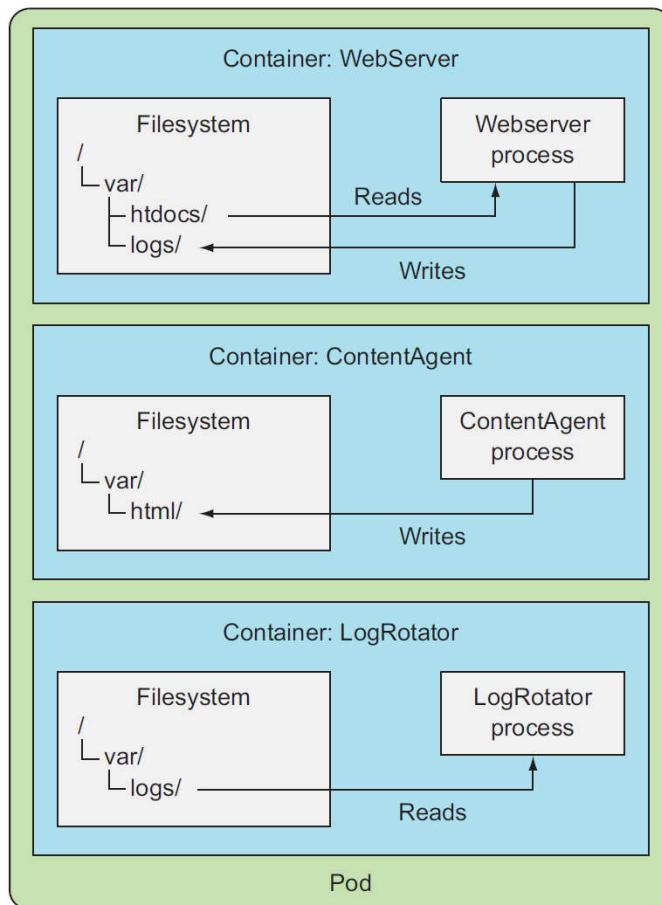
- Déplacer les données à l'extérieur des pods => **Volume**



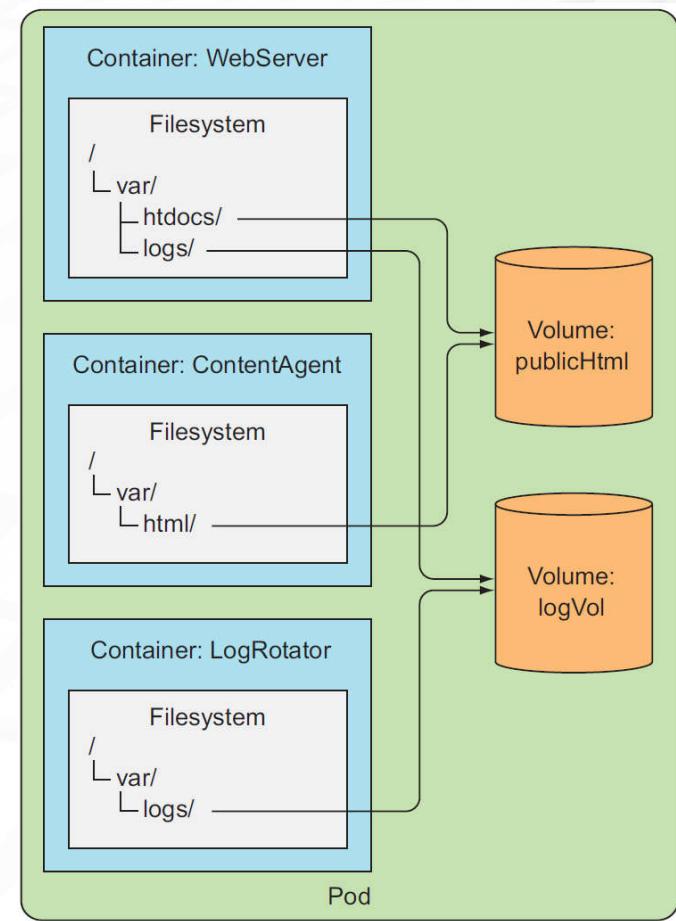


# VOLUMES - EXAMPLES

## Without shared storage



## WITH SHARED STORAGE





# TYPES DE VOLUMES

- **Un large panel de volumes disponibles**
  - Certains sont génériques, d'autres dépendent de la technologie de stockage sous-jacente
  - Différentes utilités pour différents besoins
- **Un pod peut utiliser différents types de volumes**
  - Gestion des accès pour les différents containers



# TYPES DE VOLUMES

- **Quelques exemples de volumes disponibles :**
  - **emptyDir** – simple dossier vide utilisé pour stocker des données transitoires
  - **hostPath** – utilisé pour monter des dossiers du filesystem de la node
  - **gitRepo** – volume initialisé en checkant le contenu d'un repository Git
  - **nfs** – partage NFS
  - **gcePersistentDisk, awsElasticBlockStore, azureDisk...** – Stockage cloud
  - **Cinder, cephfs, iscsi, flocker, glusterfs, quobyte, rbd, vsphereVolume, photonPersistentDisk, scaleIO...** – autre stockage réseau
  - **configMap, secret, downwardAPI** – les volumes Kubernetes
  - **persistentVolume Claim** – une manière de pré/dynamiquement provisionner du stockage persistant



# ATELIER 8

-

# VOLUMES BASIQUES

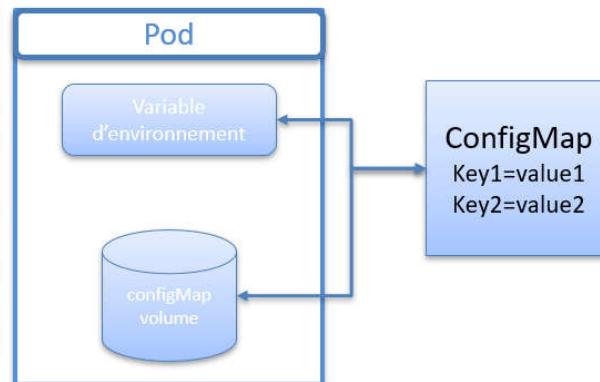


© ILKI 2018



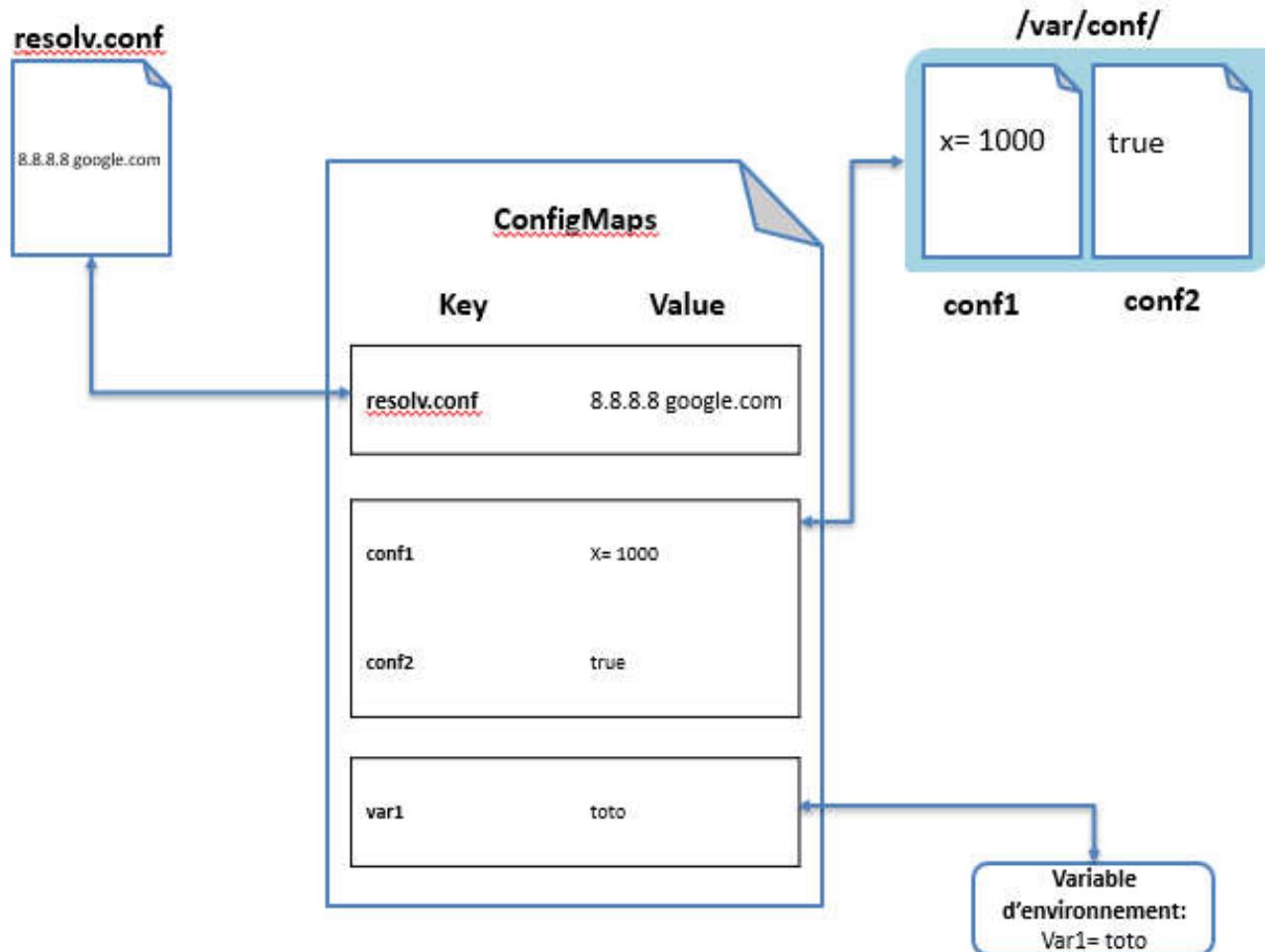
# CONFIGMAPS

- **Permet de centraliser des configurations**
  - Au format clé/valeur
  - Le contenu est passé au container soit sous forme de variables d'environnement ou de fichiers dans un volume
  - Utilisé pour les données non sensibles





# CONFIGMAPS





# SECRETS

- **Rôle identique aux configmaps**

- Mais utilisé pour stocker des informations sensibles
- Les Secrets sont toujours stockés en mémoire et non écrit sur les disques
- Depuis la version 1.7, ils sont stockés dans l'etcd de manière chiffrée
- Il est nécessaire de les déclarer avant la création du pod
- Ils peuvent être utilisés pour s'authentifier sur un registre lors d'un pull



# DOWNWARDAPI

- **Constat**

- Certaines métadonnées associées au pod ne sont créées qu'une fois le pod démarré
- Les informations de type **Label** et **Annotation** ont déjà été définies en amont

- **Il est donc possible de donner ces informations aux pods par le biais de la DownwardAPI**
  - Soit en tant que variables d'environnement
  - Soit en tant que fichier dans un volume





# DOWNWARDAPI

- **Les informations qu'il est possible de mettre à disposition**
  - Le nom du pod
  - L'adresse IP du pod
  - La namespace dans lequel le pod s'exécute
  - Le nom du Service Account que le pod utilise
  - Les réservations CPU et mémoire de chaque container
  - Les limites CPU et mémoire de chaque container
  - Les labels associés au pod
  - Les annotations associées au pod



# ATELIER 9

-

# CONFIGMAPS & SECRETS



© ILKI 2018



# ATELIER 10

-

# SIMPLE WEB



© ILKI 2018



# CONTAINER INIT

- **Rappel :**
  - Un pod peut contenir plusieurs containers faisant fonctionner une application
- **Il peut également contenir un ou plusieurs containers Init**
  - Les containers Init sont utilisés pour initialiser un pod et ils sont donc exécutés avant les autres containers du pod.
  - Ils sont exécutés séquentiellement et les containers principaux du pod ne démarreront uniquement lorsque la tâche du dernier container Init est terminée.
- **Un container Init n'apparaît pas dans la liste des containers applicatifs du pod.**



# JOBS ET CRONJOBS

- **Un Job crée un ou plusieurs pods et s'assure de la réussite d'une suite de tâches**
  - Une fois la/les tâche(s) complétée(s), les process associés ne sont pas relancés
  - Contrairement au ReplicationController, ReplicaSet et DaemonSet, qui exécutent des tâches en continu
    - Ces tâches ne sont jamais considérées comme complétées.
- **Kubernetes supporte également Les CronJobs**
  - Permet de planifier les jobs



# ATELIER 11

-

## VOLUMES GCP



© ILKI 2018



# PROBLÉMATIQUE

- **Idéalement, un développeur ne doit pas se soucier des détails techniques liés au stockage sous-jacent**
  - Tout ce qu'il doit savoir, c'est s'il veut ou non un volume de stockage persistant
- **Seulement, l'utilisation de certains types de volumes présentés nécessite de connaître des détails techniques**
  - Par exemple pour utiliser NFS, il faut *a minima* connaître l'IP du serveur et le point de montage

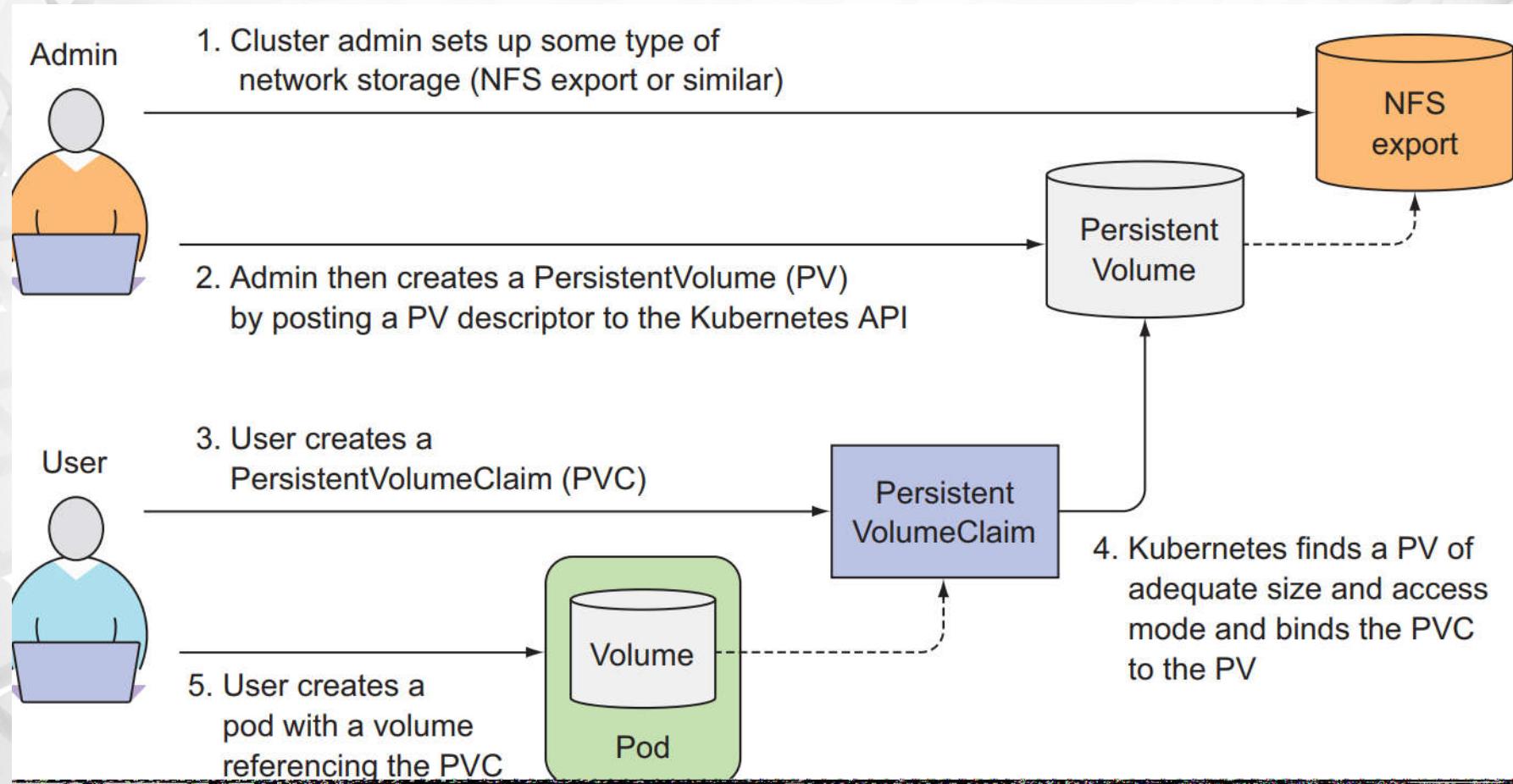


# PERSISTENTVOLUME & PERSISTENTVOLUMECLAIM

- L'administrateur du cluster Kubernetes peut donc utiliser des ressources PersistentVolume
  - L'administrateur crée donc un PersistentVolume et spécifie sa taille et les modes d'accès supportés
- Lorsqu'un utilisateur du cluster Kubernetes a besoin d'un stockage persistant, il commence par créer un manifest décrivant un PersistentVolumeClaim
  - Il spécifie dans le manifest, la taille minimum requise et le mode d'accès souhaité
  - Il le soumet ensuite au serveur API, Kubernetes s'occupe de trouver le PersistentVolume correspondant puis il associe le volume au claim
  - Les autres utilisateurs ne pourront pas utiliser le PersistentVolume utilisé tant qu'il ne sera pas libéré

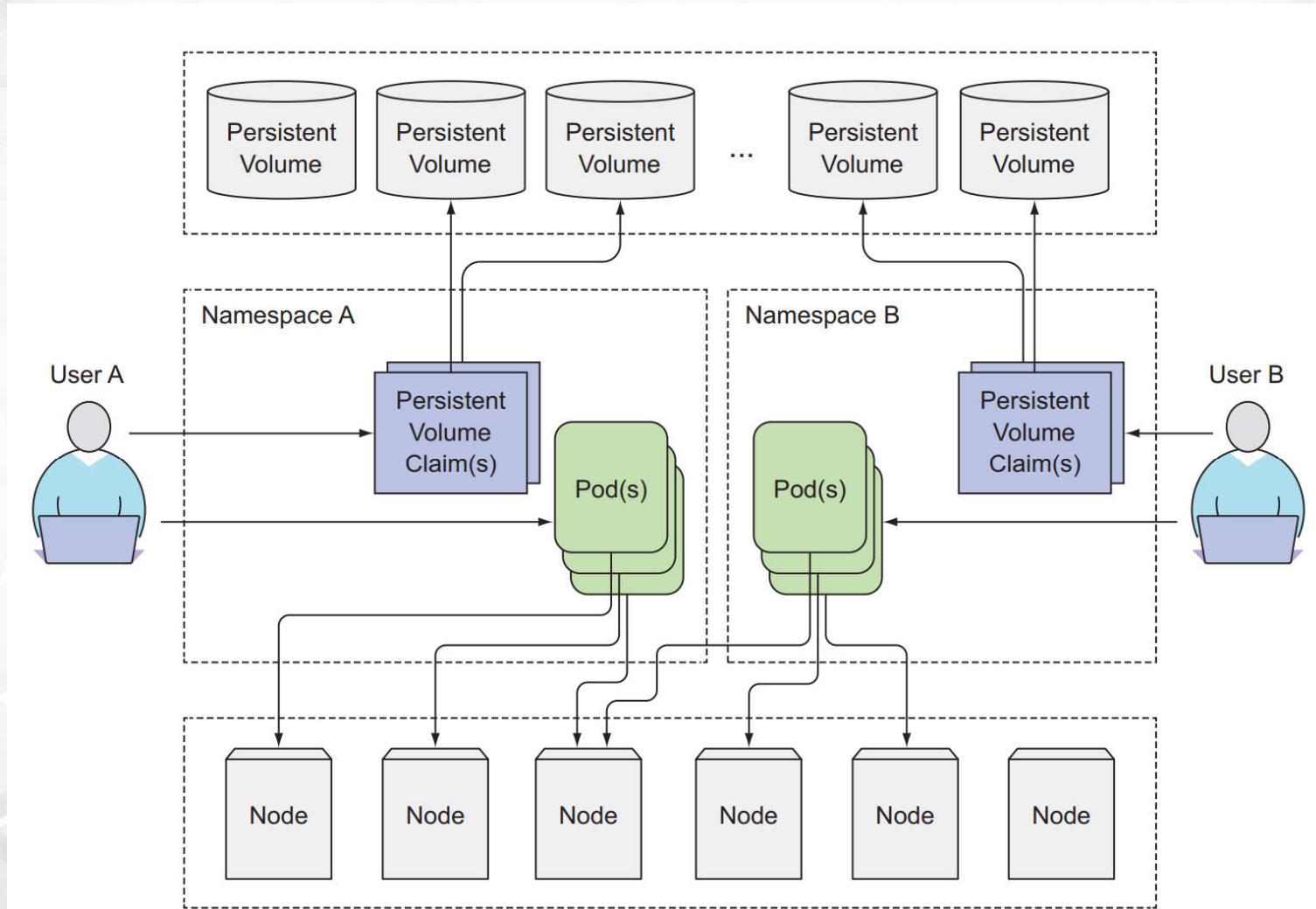


# PERSISTENTVOLUME & PERSISTENTVOLUMECLAIM





# PERSISTENTVOLUME & PERSISTENTVOLUMECLAIM



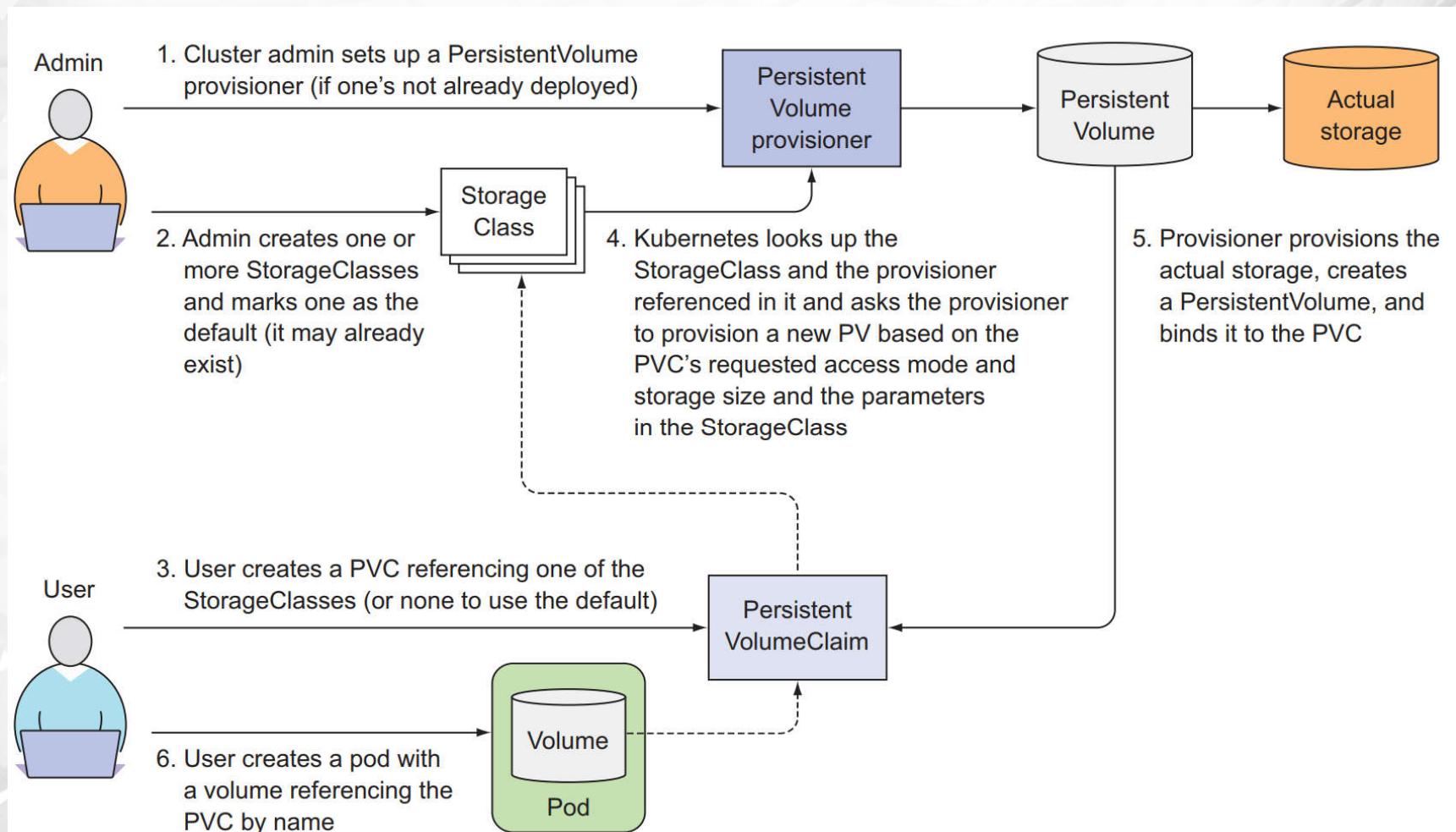


# STORAGECLASS

- **Et si l'administrateur n'avait pas besoin de provisionner le `persistentVolume` ?**
  - Il est possible qu'il déploie simplement un `PersistentVolume` Provisionner
  - Ensuite il définit un ou plusieurs objets de type `StorageClass` pour laisser la possibilité à l'utilisateur de choisir le type de volume souhaité
  - Il peut donc faire référence à une `StorageClass` dans le manifest de son `PersistentVolumeClaim`
    - Le provisioner prendra en compte les souhaits lors de la création du volume
- **Kubernetes inclus des provisionners pour la plupart des cloud providers**
  - Si Kubernetes est déployé en local, il faudra déployer un provisionner custom



# DYNAMIC PROVISIONNING OF PERSISTENT VOLUMES





# kubernetes

## QUESTIONS



© ILKI 2018