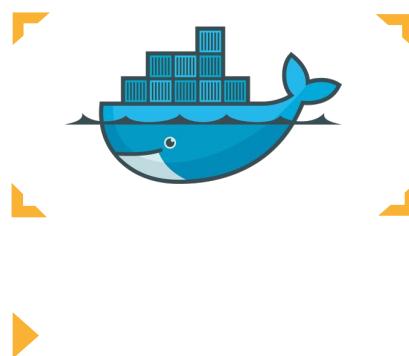


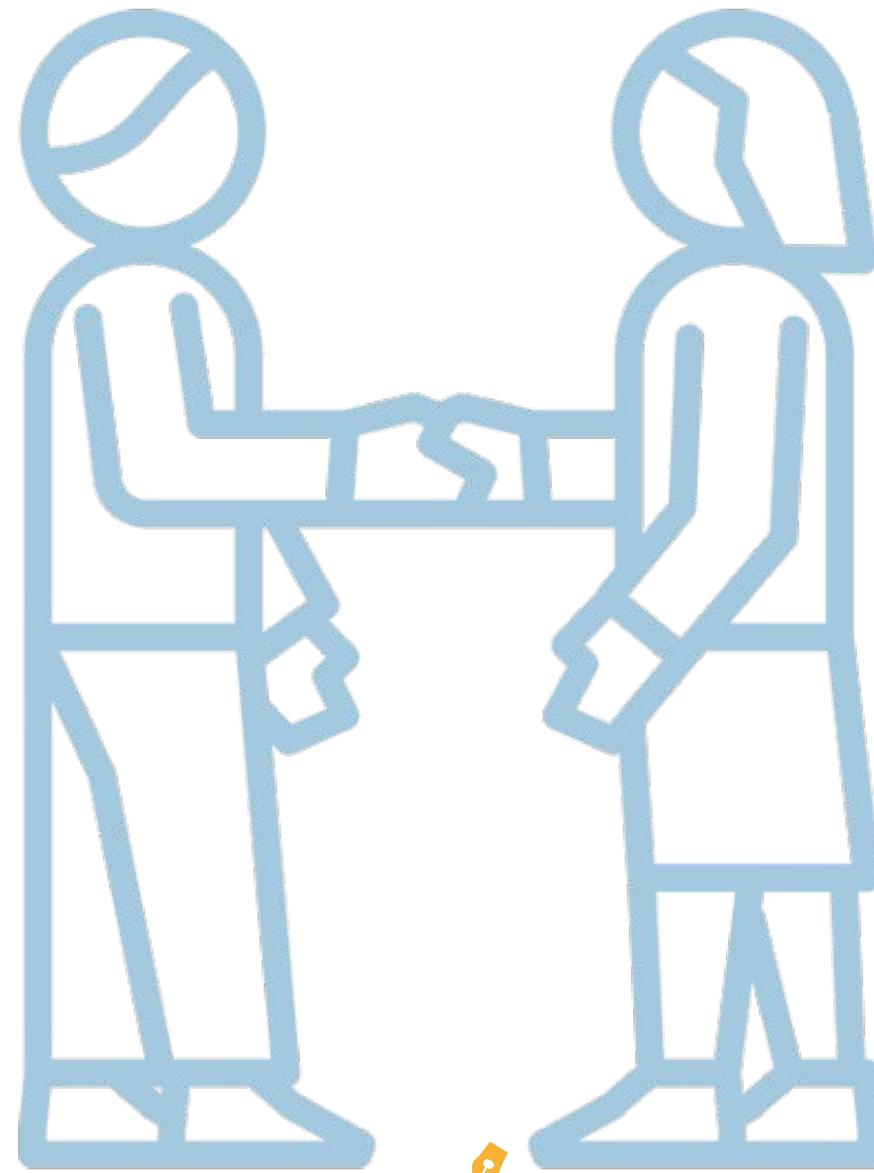


Formation Docker / Kubernetes Utilisateur

Utilisation de Docker et Kubernetes



Avant de commencer...



Les dix commandements de la formation

- ▶ Respecter les horaires
- ▶ Signer la feuille de temps
- ▶ Concentration
- ▶ Poser des questions
- ▶ Partager son expérience
- ▶ Donner son avis
- ▶ Revenir de la pause (et oui :))



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

AGENDA

Les bases de Docker

- ▷ **Introduction : l'avant Docker**
- ▷ **Qu'est ce que Docker**
- ▷ **Architecture et concepts Docker**

TP#1

Docker en pratique

- ▷ **Les images Docker**
- ▷ **Utilisation de Docker**
- ▷ **Les volumes**
- ▷ **Création d'images et registres**
- ▷ **Docker Compose**

TP#2

Les bases de Kubernetes

- ▷ **Introduction & historique de K8s**
- ▷ **Utilisation du client kubectl**

TP#3

Manipulation simple de Kubernetes

- ▷ **Concepts de base de Kubernetes**

Mettre son application en prod dans K8s

- ▷ **Secrets et Configmaps** TP#4
- ▷ **Liveness et Readiness**
- ▷ **Routes HTTP** TP#5
- ▷ **Maîtrise des capacités**
- ▷ **Monitoring applicatif** TP#6
- ▷ **Log Management** TP#7

Gestion des conteneurs à état

- ▷ **Les volumes, PV et PVC**
- ▷ **Les statefulsets**
- ▷ **CRD et opérateurs** TP#8

Le Continuous Delivery avec Kubernetes

- ▷ **Exemples de Continuous Integration**
- ▷ **Exemples de Continuous Deployment**

Conclusion et Take Away

Back to 2013 : Contexte d'arrivée de Docker

- ▶ De nombreuses problématiques liées aux applications
 - > la portabilité des applications
 - > la distribution des applications
 - > le besoin de décorrélérer applications et infrastructure
 - > la rationalisation des infrastructures
- ▶ La montée en puissance
 - > des solutions de PaaS
 - > de la philosophie DevOps



Back to 2013 : La problématique de la portabilité logicielle

Comment assurer le déploiement homogène
d'une application sur tous ses environnements ?



	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Jobs en Arrière Plan	?	?	?	?	?	?	?
Base de données	?	?	?	?	?	?	?
Analytics	?	?	?	?	?	?	?
Files de messages	?	?	?	?	?	?	?



Back to 2013 : La problématique de la distribution des applications

Comment distribuer un logiciel de façon simple et efficace ?

Les différentes méthodes de distribution logicielle



Binaire



Paquet



Installeur



Dépôts
de paquets



Virtual
Appliance



Application
Store



Back to 2013 : Rationalisation infra et cycle de vie applicatif

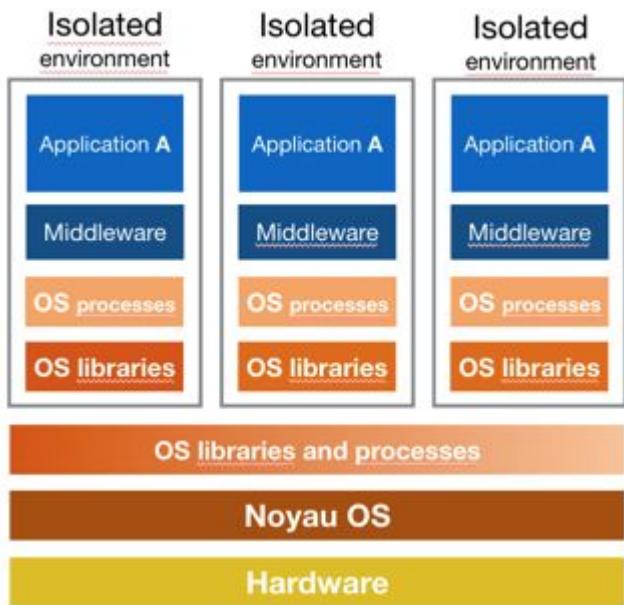
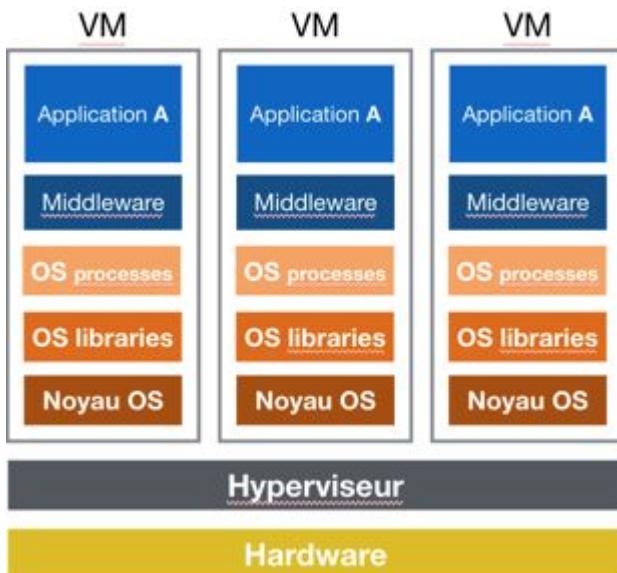
Comment optimiser l'utilisation des ressources ?
Comment décorreler application et infrastructure ?

Les 2 technologies de virtualisation des systèmes

Virtualisation



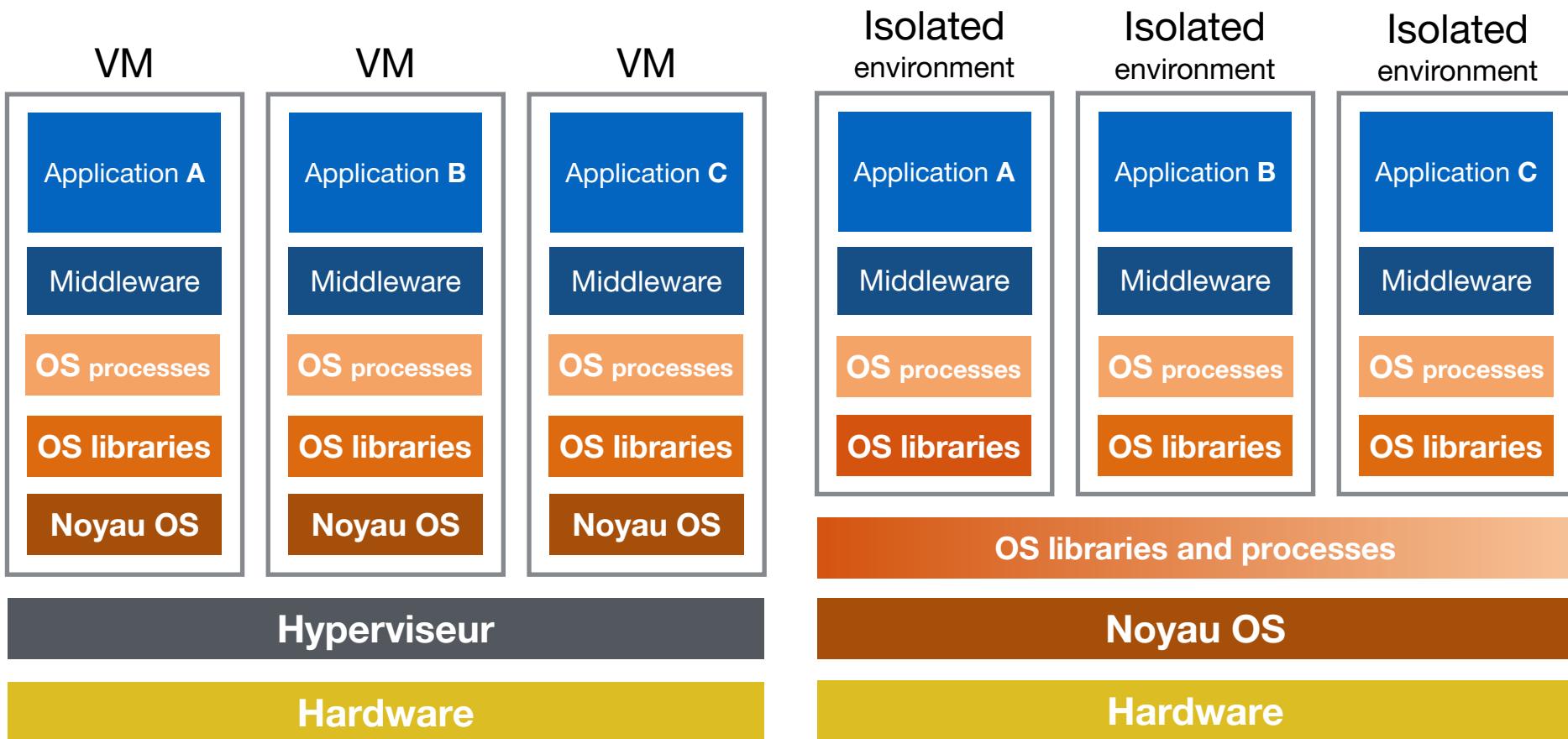
Isolation



Virtualisation



Isolation Système



Back to 2013 : Des technologies qui ne datent pas d'hier

Isolation



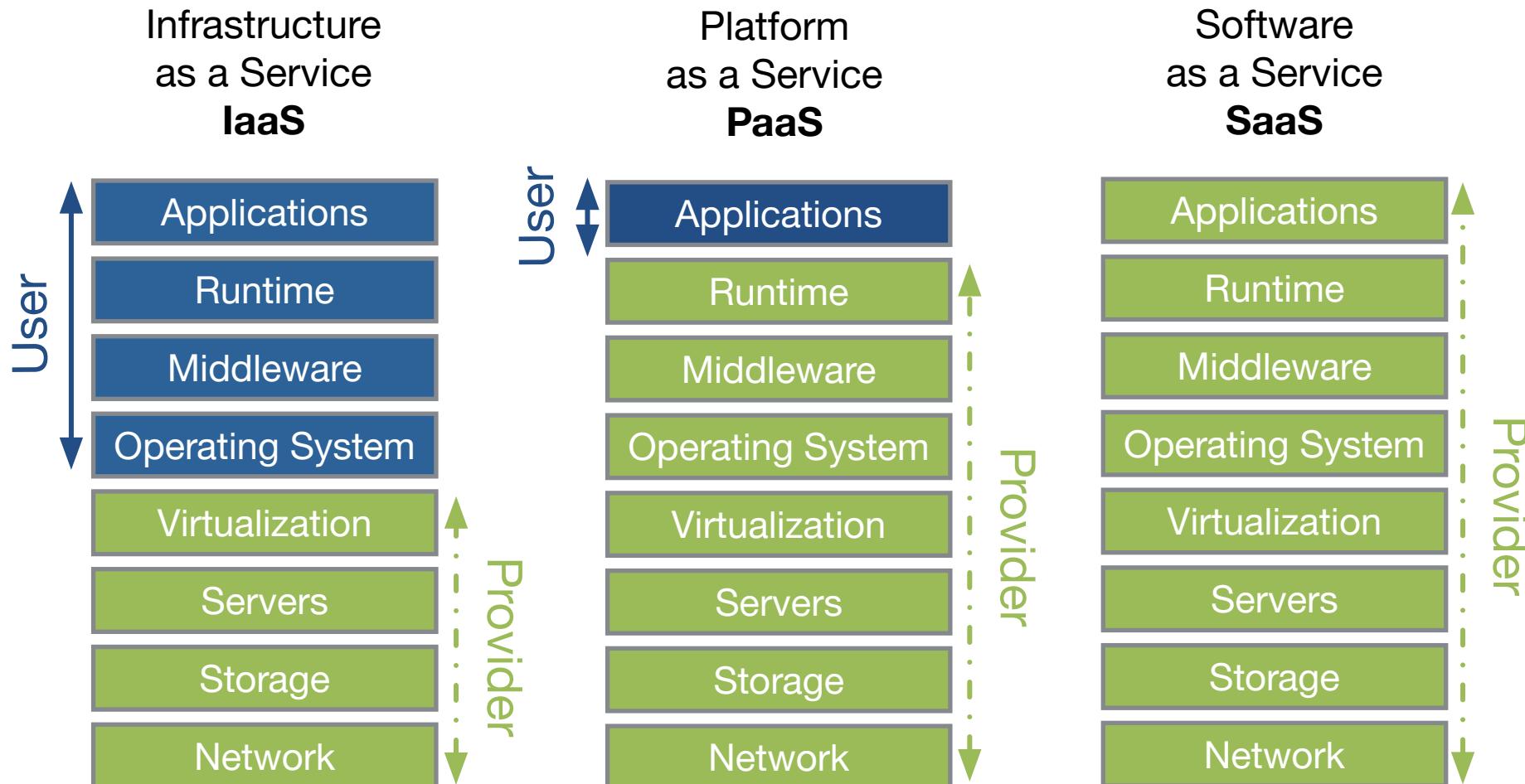
Virtualisation

IBM CP/CMS
IBM LPAR
Vmware ESX
Vmware Vmotion
Xen
KVM
Sun LDOM
Hyper V



Back to 2013 : Qu'est ce qu'un PaaS ?

Schéma des différents niveaux de services Cloud



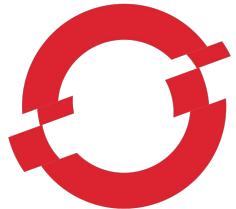
Back to 2013 : Les challenges du PaaS

- ▶ Les PaaS concentrent toutes les problématiques indiquées et doivent en plus
 - Déployer rapidement des nouvelles applications
 - Assurer une élasticité rapide
 - Isoler les applications entre elles
- ➔ **Besoin d'une sur-couche légère d'isolation et d'abstraction
... des conteneurs !!**



Back to 2013 : Le conteneur avant l'heure

Les principaux PaaS et leur technologie de conteneurs



OPENSHIFT

Openshift V2 gears



Warden containers



LXC containers



Cedar containers



La philosophie DevOps

« **DevOps** est un ensemble de pratiques qui visent à réduire le Time to Market et améliorer la Qualité en optimisant la coopération entre les **Développeurs** et la **Production** »



→ Un mouvement friand de technologies adressant à la fois les dev et les ops

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Docker (homonymie)

 Cette page d'*homonymie* répertorie les différents sujets et articles partageant un même nom.

Informatique

- **Docker Inc**, la compagnie qui développe la plateforme Docker.
- **Docker Engine**, le logiciel qui construit et fait tourner les conteneurs.
- **Docker**, le format de conteneur.
- **Docker CLI**, l'outil en ligne de commande pour piloter les conteneurs.
- **Docker Platform**, l'ensemble des logiciels de Docker Inc permettant de gérer les conteneurs.



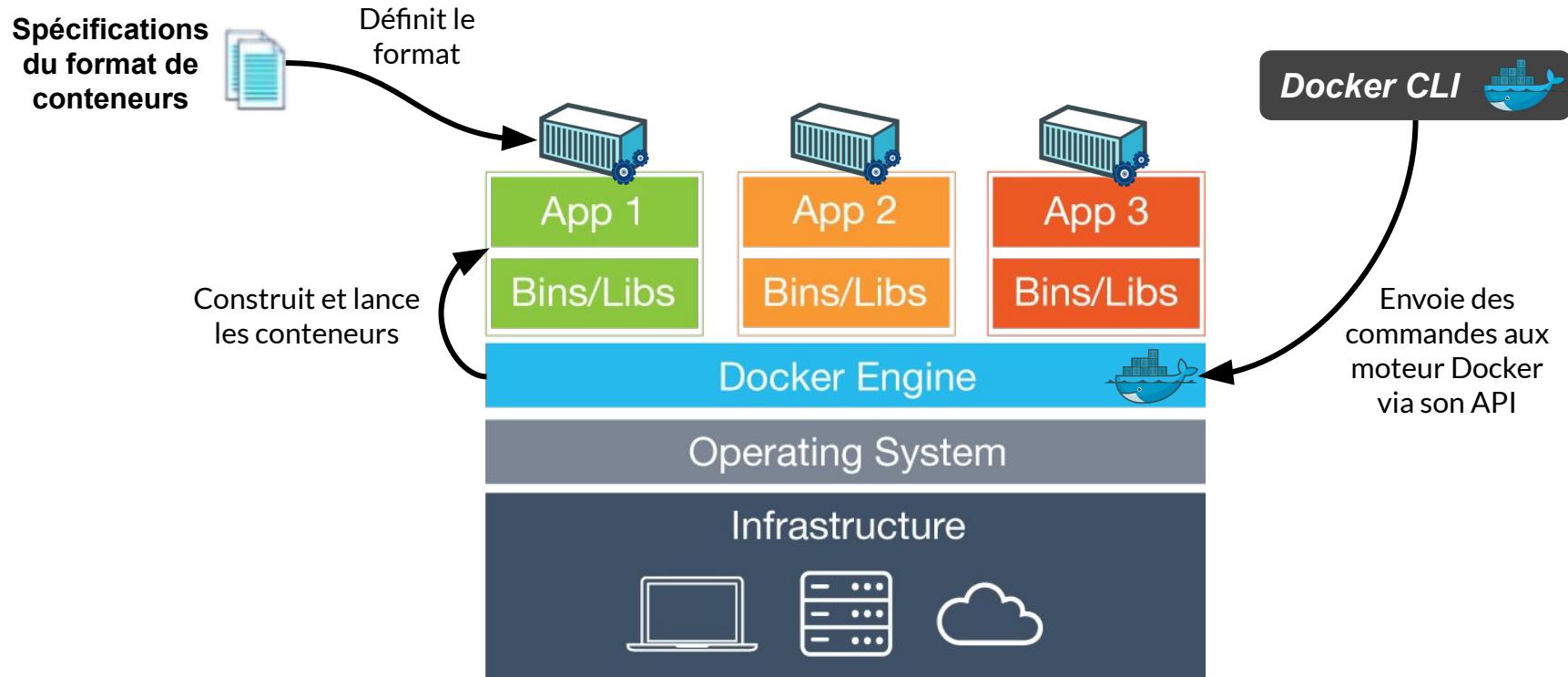
Une définition de Docker

« Une **technologie** permettant de **standardiser le packaging et l'opération des applications** »



Docker Engine, Conteneur Docker et Docker CLI

- ▶ L'ensemble de technologies **initialement appelé Docker**



- ▶ Des technologies **standardisées** au sein de l'Open Container Initiative (OCI)

Le conteneur Docker

de l'isolation système

...

à l'isolation applicative

Isolated environment

Isolated environment

Isolated environment

Application A

Application B

Application C

Middleware

Middleware

Middleware

OS processes

OS processes

OS processes

OS libraries

OS libraries

OS libraries

OS libraries and processes

Noyau OS

Hardware

Conteneur

Application A

Middleware

OS libraries

Conteneur

Application B

Middleware

OS libraries

Conteneur

Application C

Middleware

OS libraries

OS libraries and processes

Noyau OS

Hardware



La société Docker Inc

- ▶ Société créée initialement en 2008 à San Francisco sous le nom de DotCloud pour offrir un service de PaaS
- ▶ Renommée en Docker Inc fin 2013 pour se concentrer autour du projet Docker, puis revend la partie PaaS mi-2014
- ▶ Mène depuis 2014 une politique d'acquisition des solutions qui émergent de la communauté (orchard, kitematic socketplane, tutum...)
- ▶ Se positionne comme le leader du projet communautaire Docker de développement d'une plateforme ouverte pour les applications distribuées
- ▶ Docker Inc modifie son système de packaging du Docker Engine avec l'introduction de Moby en 2017
- ▶ En 2017, Docker Inc lance le Modernize Traditional Applications (MTA) en s'associant avec des éditeurs traditionnels
- ▶ En 2018, Docker EE 2.0 intègre désormais Kubernetes et rends possible le déploiement des stacks et des fichiers compose au travers de Swarm ou Kubernetes



La plateforme Docker packagée avec Moby

- ▷ Un ensemble complet d'outils de Docker Inc. permettant de gérer la construction, livraison et la création de conteneurs



Gestionnaire de Cluster Docker



Dépôt open-source de distribution d'images



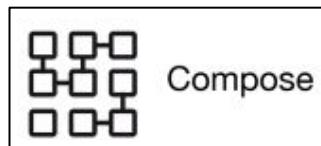
Dépôts d'image public sur Internet



Gestion de conteneurs sous Mac



Gestion de réseaux inter-conteneurs



Gestion d'applications multi-conteneurs



Dépôt privée de distribution d'images



Outils de provisionnement de noeuds Docker



Gestion du cycle de vie des containers



Comprendre Docker par ses caractéristiques

Des caractéristiques uniques

PO

PORTABLE



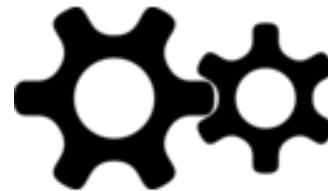
DI

DISPOSABLE



LI

LIVE



SO

SOCIAL



Portable



	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Jobs en Arrière Plan	?	?	?	?	?	?	?
Base de données	?	?	?	?	?	?	?
Analytics	?	?	?	?	?	?	?
Files de messages	?	?	?	?	?	?	?



Portable



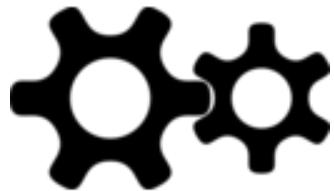
	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique							
Web frontend							
Jobs en Arrière Plan							
Base de données							
Analytics							
Files de messages							



Disposable



- ▶ Des images de conteneurs en lecture seule : la modification entraîne la création d'une nouvelle image
- ▶ Les modifications dans les conteneurs sont locales et temporaires
- ▶ La vie d'un conteneur est liée à l'application



- ▶ Les images de conteneurs sont versionnées et incrémentales
 - ▶ La configuration (port, processus, ...)
 - ▶ Le système de fichiers
- ▶ Un système de versionning similaire à Git
 - ▶ Gestion des diffs
 - ▶ Gestion des versions en arbre
 - ▶ Gestion des Tags

Social



- ▶ Système de dépôt d'images de conteneurs (registre)
 - Accessible depuis internet ou en interne
 - Recherche facile
- ▶ Des outils communautaires
 - Ouverts gratuitement
 - Système de vote sur les images de conteneurs
 - “Trusted images” et images de conteneurs “officielles”

Un mot sur le déroulement des TPs

La plupart des TPs sont à réaliser en autonomie

Régulièrement, des exercices et questions serviront de **point de rendez-vous** dans l'avancement des TPs

Questions

- Quelles sont les versions du client et du serveur ?
- Les versions sont-elles strictement identiques ?

Exercice

En utilisant le format de sortie `-o=custom-columns`, produire la liste des nœuds du cluster avec le format suivant :

NAME	ARCH	KERNEL
k8s-training-dkusr-m1	amd64	4.4.0-1061-aws
k8s-training-dkusr-n1	amd64	4.4.0-1061-aws
k8s-training-dkusr-n2	amd64	4.4.0-1061-aws
k8s-training-dkusr-n3	amd64	4.4.0-1061-aws



Un mot sur les environnements

Des machines ont été créées pour vous permettre de réaliser les TPs

- ▷ Elles ont été approvisionnées sur le *cloud*
- ▷ Elles seront purgées en fin de formation

The screenshot displays a web-based IDE interface for developing and deploying applications. It includes:

- Web Console:** Top left, shows a terminal window with a command-line interface (CLI) for managing Kubernetes resources. The command `ls` is run, showing files like `Dockerfile`, `app.py`, and configuration files for Docker and Kubernetes.
- Code Editor:** Middle right, shows an editor window displaying the content of the `app.py` file. The code uses Flask and Redis to handle requests.
- File Browser:** Bottom left, shows a file tree for the project directory. It includes sub-directories like `ubuntu` and `app.py` which contains the application logic.
- Terminal:** Bottom right, shows another terminal window running the command `kubectl get po` to list pods in the cluster.

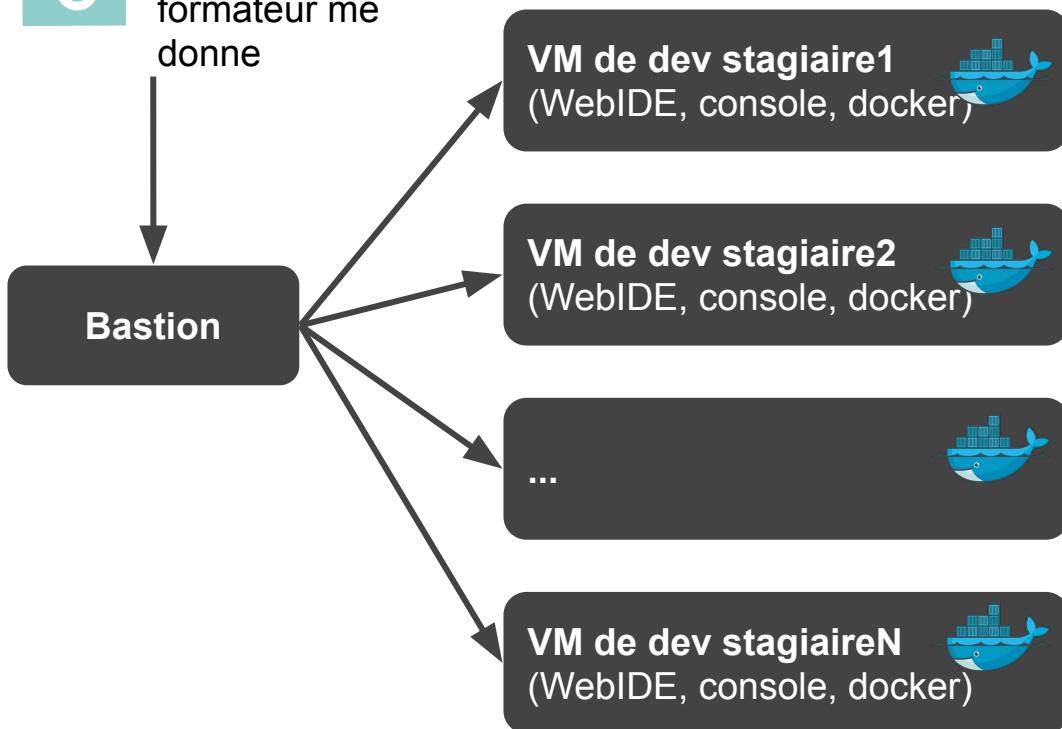
Yellow arrows point from the text labels to their corresponding parts in the interface:

- A yellow arrow points from the text "Web IDE et les TPs" to the file browser and code editor area.
- A yellow arrow points from the text "Web Console" to the top-left terminal window.

Un mot sur les environnements



J'accède à mon environnement avec l'@ IP que le formateur me donne



TP#1

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ **Architecture et concepts Docker**

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Les concepts de Docker

L'image



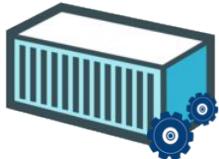
Une arborescence de fichiers contenant tous les éléments requis pour faire tourner une application

Le montage de répertoires



Un espace de stockage indépendant de l'image utilisable pour les données persistantes

Le conteneur



Une instantiation d'une image en cours d'exécution sur un système hôte

Le *Dockerfile*



Un fichier contenant les instructions permettant de construire une image

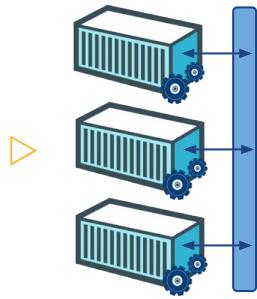
Le registre



Un service centralisé de stockage et distribution d'images

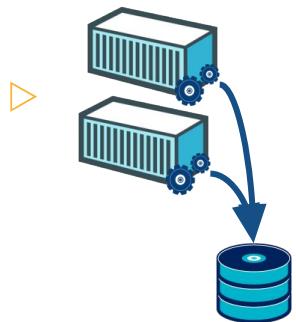


Les concepts de Docker (que vous ne verrez pas dans Kubernetes)



les networks (docker network)

Permet la communication de conteneurs dans un ou plusieurs réseaux sur une ou plusieurs machines hôtes

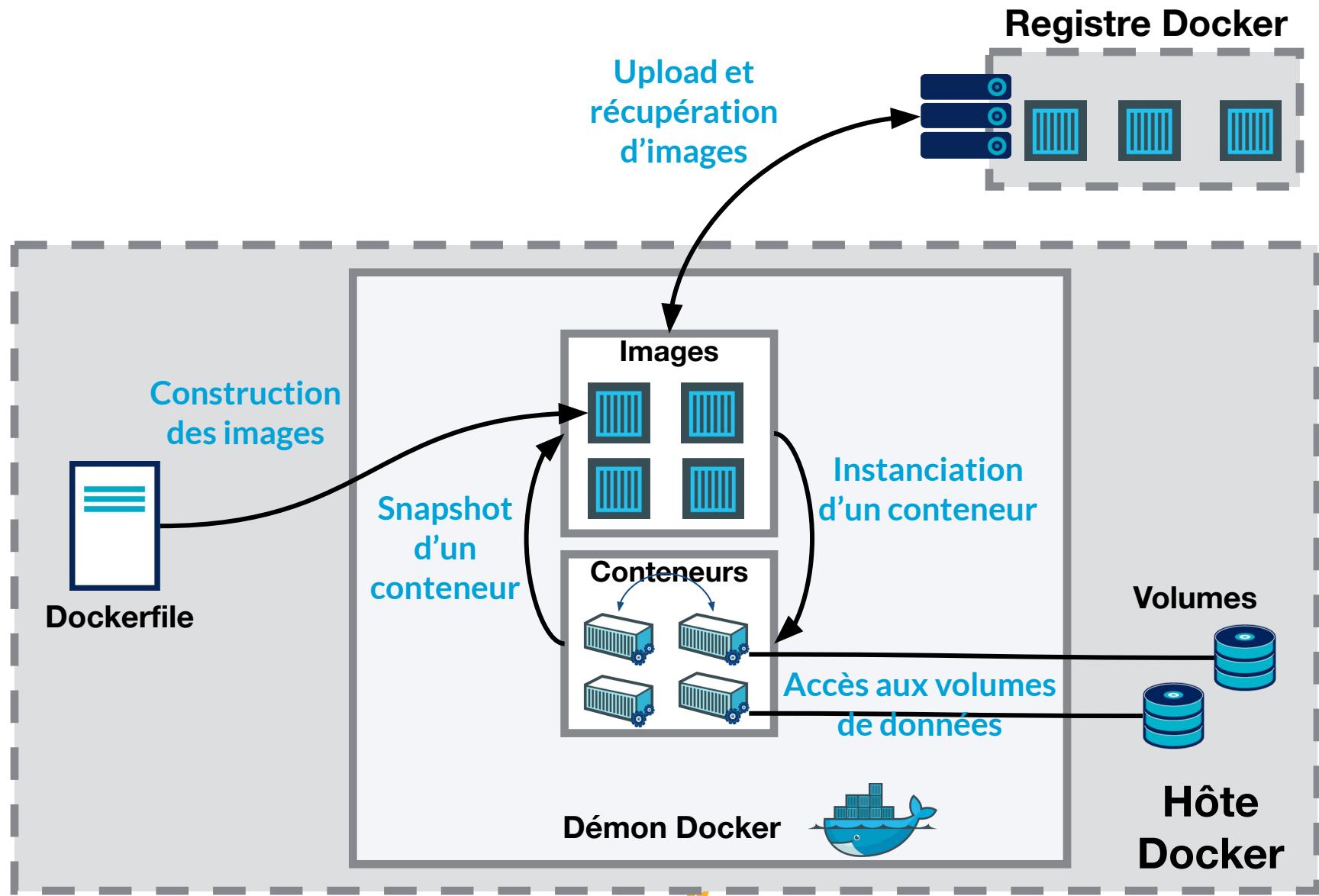


Les volumes (docker volume)

Un espace de stockage indépendant de l'image utilisable pour les données persistantes. Ils possèdent leur propre cycle de vie, sont créés à côté des conteneurs sur lesquels ils peuvent être attachés (et détachés).

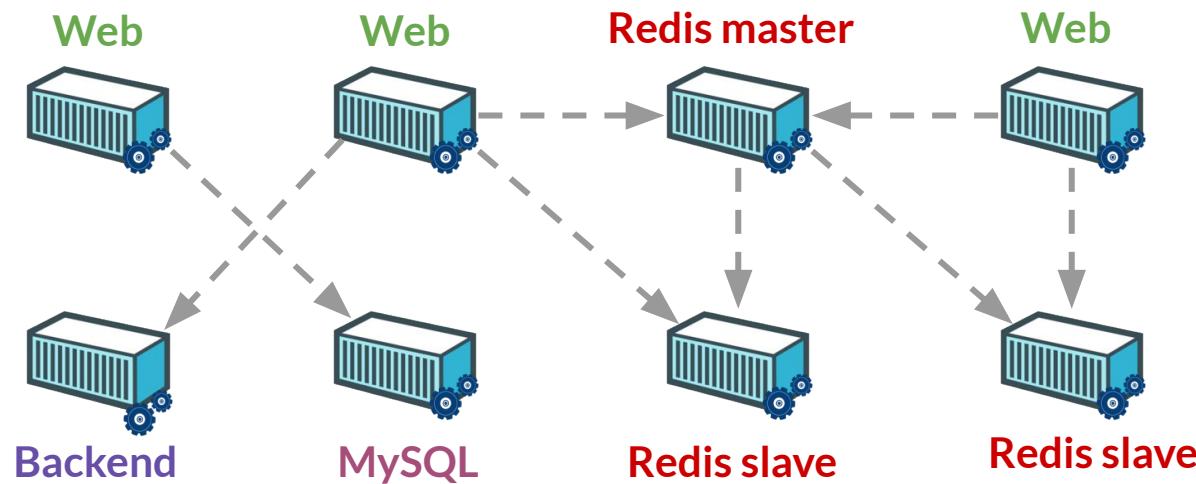


Les concepts de Docker



La philosophie Docker des conteneurs

- Un principe fort : **Un conteneur = 1 processus**
- **Une brique de base à composer dans une architecture rapidement multi-conteneurs**



- ➔ Le conteneur Docker **n'est pas une VM light**
- ➔ Il recentre l'infrastructure autour de l'application



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ **Les images Docker**
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Les images Docker

Une image Docker c'est

- ▶ **un système de fichiers auto-suffisant** contenant *a minima* librairies et binaires de base (libc, libresolv, bash...)
- ▶ **Un identifiant unique** assigné à l'image à sa création
- ▶ **Des métadonnées** pour préciser la façon d'instancier l'image
 - > le processus à exécuter à l'instanciation de l'image,
 - > les variables d'environnement à positionner
 - > l'utilisateur qui va lancer l'application
 - > la configuration réseau (ports exposés, réseau...),
 - > les volumes de données à connecter,
 - > ...

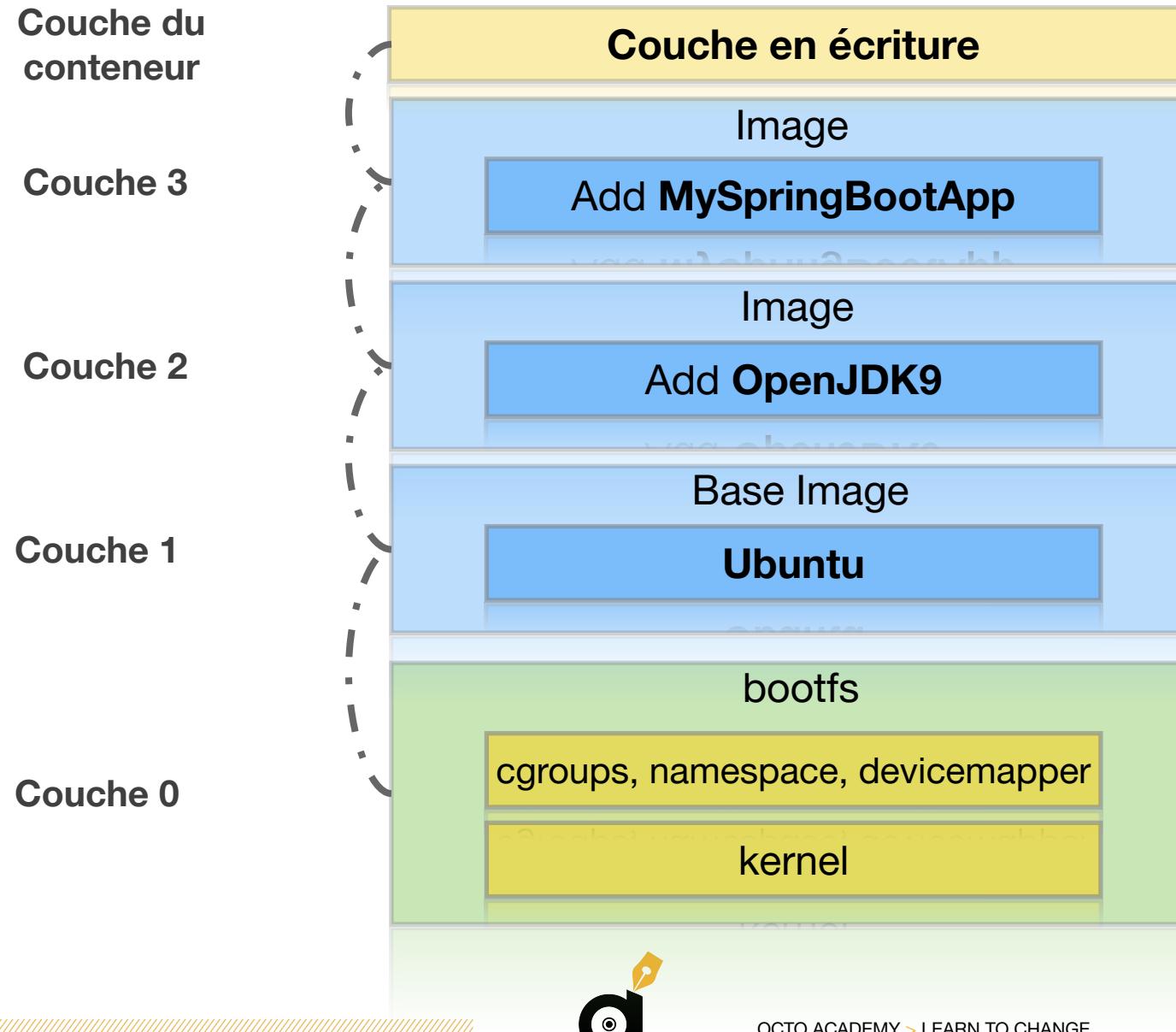


Le mille-feuille des images Docker

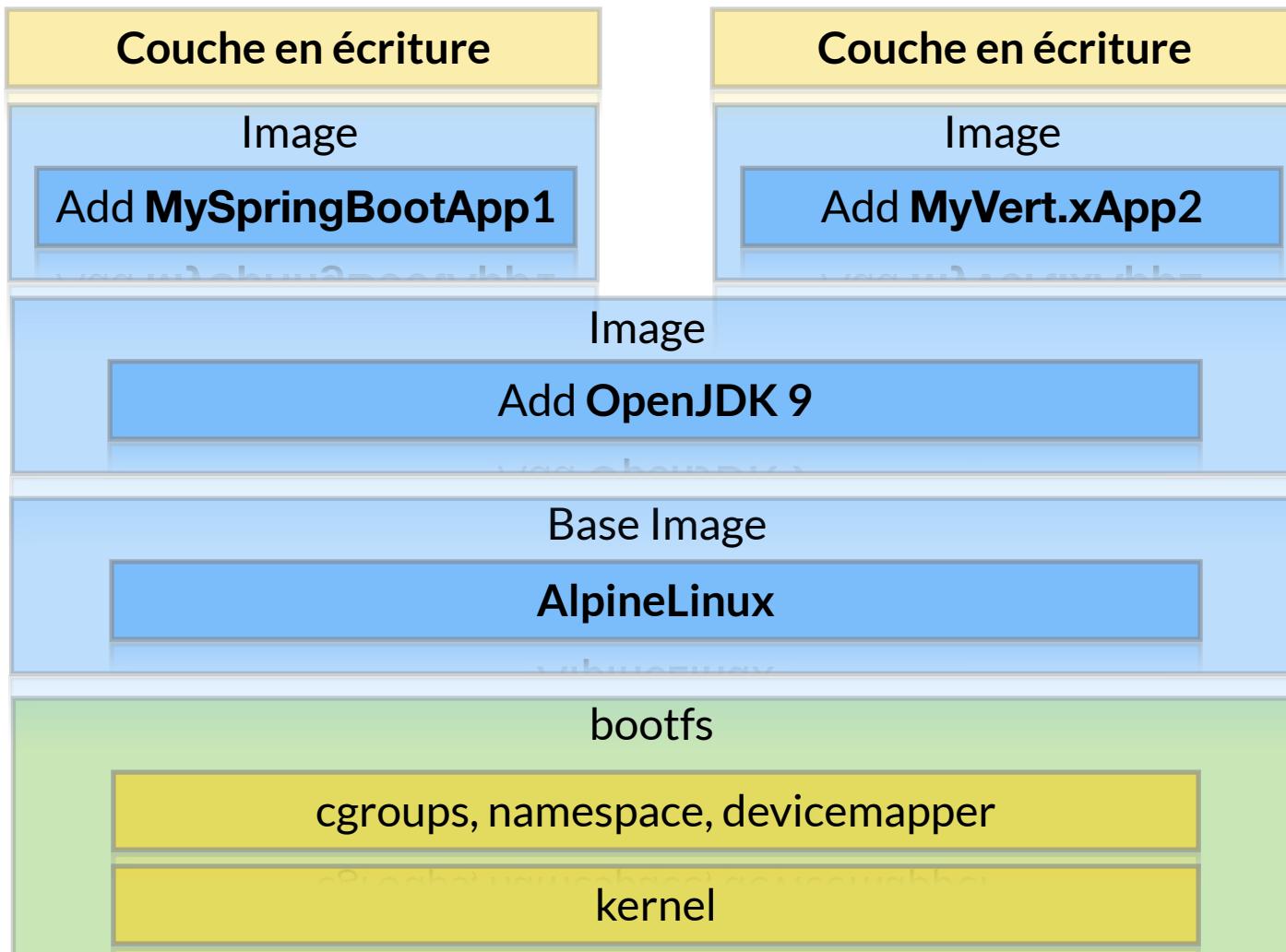
- ▷ Docker utilise un système de fichiers avec un **système de couches** pour les images de conteneurs
- ▷ **Principe**
 - Un ensemble de couches partagées en lecture seule
 - Unifiées par le système pour simuler un unique système de fichiers à plat pour le conteneur
- ▷ **Avantages**
 - Évite la perte d'espace avec 4 x 500Mo par OS Ubuntu dans des VMs
 - Permet la récupération et le démarrage rapide des conteneurs



Le mille-feuille des images Docker



Le mille-feuille des images Docker



L'essentiel des commandes

Lister les images locales

docker image ls

Télécharger une image à partir du Registre

docker image pull



docker image rm

Supprimer une image

docker image tag

Labelliser une image

docker image history

Lister les couches d'une image

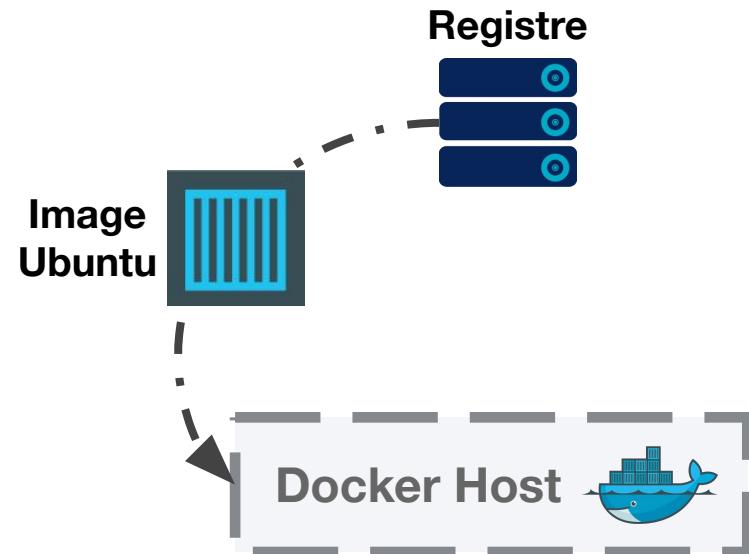


Commande Docker : `docker image pull`

Demande à Docker de récupérer localement une image de conteneurs sur un registre distant

Exemple de récupération de la dernière image d'ubuntu

```
$ docker image pull ubuntu
```



Syntaxe d'un nom d'image : [**<repository>/**]image[:<tag>]

où:

- *repository* indique l'url du dépôt d'images, si non précisé, va sur le dépôt par défaut (hub.docker.com)
- *image* indique le nom de l'image
- et *tag* identifie l'image de manière unique dans le dépôt et vaut *latest* par défaut



Commande Docker : `docker image ls`

Interroge le registre local à l'hôte Docker et affiche la liste des images présentes localement

```
$ docker image ls
```

Exemple de sortie de la commande `image`

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
<none>	<none>	5b7faa37374	2 hours ago	1.207 GB
ubuntu	latest	9aafc45696a	6 hours ago	546 MB
ubuntu	willy	9aafc45696a	6 hours ago	546 MB
tutum	mysql	9bb40134b3d	8 hours ago	1.041 GB

- Une image locale peut n'être associée à aucun repository ou tag
- **VIRTUAL SIZE** représente la somme de toutes les couches de l'image



Commande Docker : `docker image history`

Affiche toutes les couches d'une image et les commandes utilisées pour la créer

```
$ docker image history <image>
```

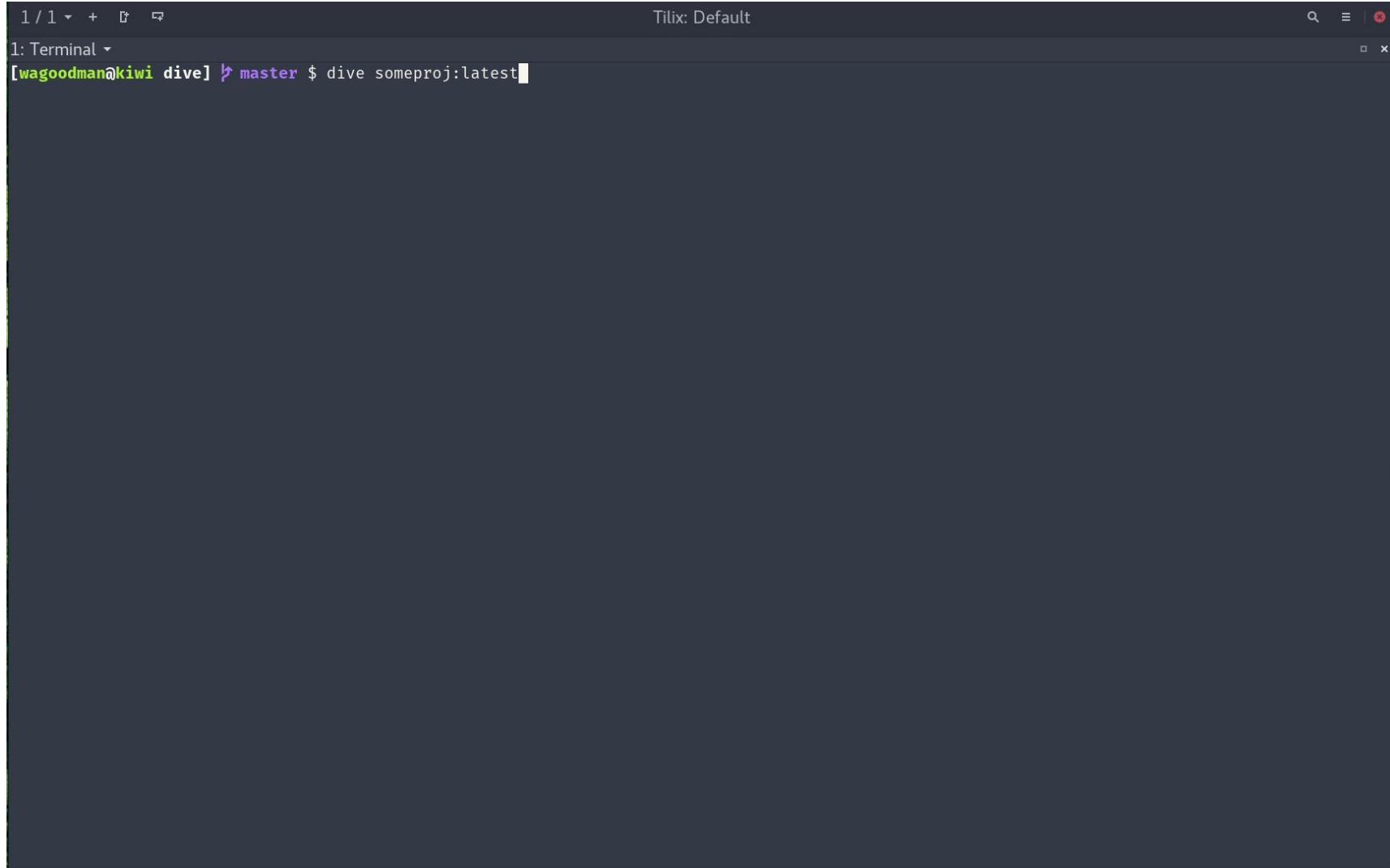
Exemple de sortie de la commande history

IMAGE	CREATED	CREATED BY	SIZE
3e23a5875458	8 days ago	/bin/sh -c #(nop) ENV LC_ALL=C.UTF-8	0 B
8578938dd170	8 days ago	/bin/sh -c dpkg-reconfigure locales && loc	1.245 MB
be51b77efb42	8 days ago	/bin/sh -c apt-get update && apt-get install	338.3 MB
4b137612be55	6 weeks ago	/bin/sh -c #(nop) ADD jessie.tar.xz in /	121 MB
750d58736b4b	6 weeks ago	/bin/sh -c #(nop) LABEL Jean Veplut	0 B

- ▶ Chaque commande donne lieu à la création d'une couche supplémentaire associée à un identifiant
- ▶ La commande de création et la taille réelle de chaque couche peut être inspectée facilement



Outils en + Dive: *docker image history* en mieux



```
1 / 1 + ⌂ ⌂  Tilix: Default
1: Terminal ▾
[wagoodman@kiwi dive] ↵ master $ dive someproj:latest
```

- ▶ Pour le télécharger : <https://github.com/wagoodman/dive>



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ **Utilisation de Docker**
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

L'essentiel des commandes

Créer un conteneur et lancer une commande

docker container run

Démarrer un conteneur

docker container start

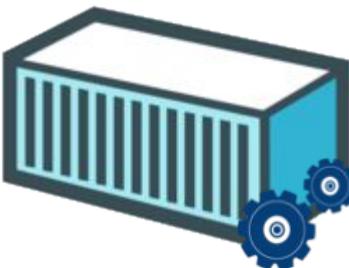
Se connecter à la console du conteneur

docker container attach

Supprimer un conteneur

docker container rm

Redémarrer le conteneur



docker container restart

Lister les conteneurs

docker container ls

Accéder aux logs produits

docker container logs

docker container stop

Stopper un conteneur

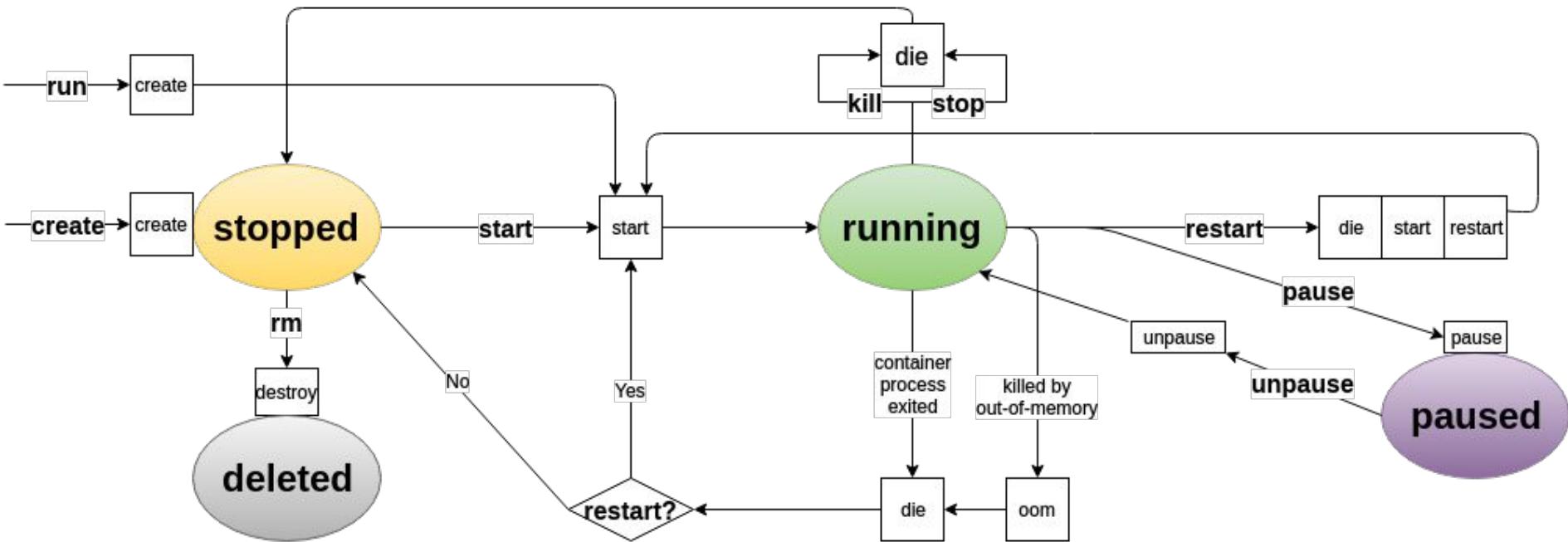


Le conteneur Docker

- ▶ Le conteneur est **la brique de base de Docker**
- ▶ Il est toujours **instancié à partir d'une image**
- ▶ Un conteneur **ne vit que pour les processus** qu'il contient
- ▶ **Si ces processus s'arrêtent**
 - Le conteneur est stoppé
 - Le contenu modifié subsiste tant que le conteneur n'est pas détruit
- ▶ **Une couche en écriture est créée à l'instanciation** du conteneur et supprimée à sa destruction



Cycle de vie d'un conteneur



Commande Docker : `docker container run`

Lancer un conteneur à partir d'une image

Exemple de lancement d'un conteneur ubuntu

```
$ docker container run -i -t ubuntu /bin/bash
```

Anatomie de la ligne de commande

- ◆ `docker container run` : lancer un conteneur
- ◆ `-i` : laisse l'entrée standard ouverte
- ◆ `-t` : assigne un terminal (tty) sur le conteneur
permet d'avoir un shell interactif
- ◆ `ubuntu` : utiliser l'image “ubuntu:latest”
- ◆ `/bin/bash` : lancer le shell bash à l'intérieur du conteneur



docker logs et la gestion des logs

- ▷ **Une application conteneurisée doit envoyer ses logs sur les sorties standard / erreur**
Plus de logging direct dans un fichier ou vers une solution de gestion de logs externe !
- ▷ **Récupération automatique des sorties par Docker** et sauvegarde dans un fichier local par défaut

Exemple de sortie de *docker logs* pour un conteneur wordpress

```
$ docker container logs wordpress
```

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.4. Set the 'ServerName' directive globally to suppress this message
PHP Warning: file_put_contents(/var/www/html/wp-content/themes//lib/css/theme.css) [
```



Docker et la limitation des ressources

Un système natif de limitation des ressources

- ▶ **Configurable au lancement du conteneur** via la commande `docker run`
- ▶ **Support des limites des ressources** CPU, Mémoire et I/O Disques
- ▶ **Basé sur la technologie cgroups** (*control groups*) du noyau Linux



Docker et la limitation des ressources

Des ressources finement configurables

► CPU

- quelle « part » (share) de CPU attribuer au conteneur
- assigner un conteneur à un ou des CPUs en particulier

► Mémoire

- selon le type de mémoire (applicative, noyau, swap, partagée)
- en mode limitation *hard* ou *soft* (limitation stricte ou en cas de contention)



Docker et la limitation des ressources

► I/O disques

- par poids relatif ou en absolue
- selon le type d'accès (lectures, écriture)
- pour le débit ou pour les entrées/sorties par seconde (IOPS)
- pour un disque donné ou pour l'ensemble des accès

Exemple de lancement du conteneur toto avec des limites CPU, mémoire et disques

```
$ docker container run --cpuset-cpus=0 \
    --memory=1g \
    --device-write-iops=/dev/sda2:5000\
    --device-read-bps=/dev/sda1:10mb \
    toto
```



Les commandes avancées

Créer un conteneur

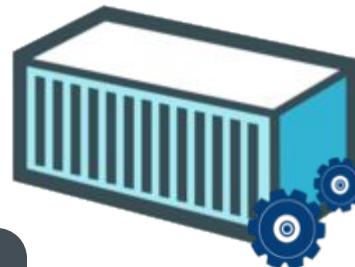
docker container *create*

Lister les processus

docker container *top*

Exécuter une commande dans le conteneur

docker container *exec*



docker container *diff*

docker container *kill*

Tue un conteneur

Affiche les différences dans le système de fichiers du conteneur

docker container *inspect*

Affiche des informations sur un conteneur

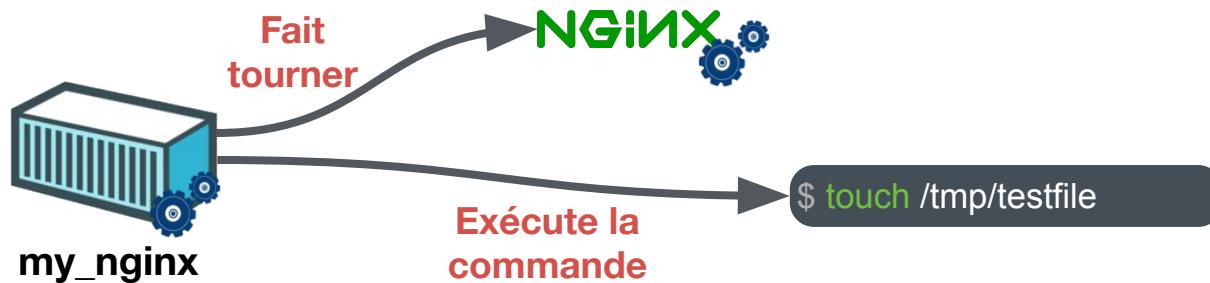


Commande Docker : *docker container exec*

Exécute une commande à l'intérieur
d'un conteneur en cours d'exécution

Exemple d'exécution de la commande “`touch /tmp/testfile`”

```
$ docker container exec my_nginx touch /tmp/testfile
```



- ▷ Permet de **faciliter le debugging** dans un conteneur
- ▷ À condition d'avoir au moins un shell dans le conteneur (bash, sh, ash)

Commande Docker : ***docker container diff***

Affiche les différences dans le système de fichiers du conteneur par rapport à l'image d'origine

```
$ docker container diff my_container
```

Exemple de sortie de la commande diff

```
C /tmp/FichierModifié  
D /tmp/FichierSupprimé  
A /tmp/FichierCréé
```

- ▷ **A** = Ajouté (*Added*) **D** = Supprimé (*Deleted*) **C** = Modifié (*Changed*)
- ▷ Très utile pour identifier rapidement les processus qui ne respectent pas les principes des conteneurs
(écriture de données dans le conteneur, utilisation d'un fichier de logs...)



Commande Docker : *docker container inspect*

Affiche des informations de bas niveau sur un conteneur

Exemple de commande inspect pour obtenir le fichier de logs du conteneur

```
$ docker container inspect --format='{{.LogPath}}' my_container
```

- ▷ Sortie de l'intégralité des informations **au format JSON**
- ▷ Possibilité de sélectionner les informations (--format)
- ▷ Quelques informations utiles à récupérer
 - Code de sortie du conteneur: '{{.State.ExitCode}}'
 - Adresse IP de l'instance: '{{.NetworkSettings.IPAddress}}'
 - Ports exposés: '{{.Config.ExposedPorts}}'
 - Variables d'environnement du conteneur: '{{.Config.Env}}'



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ **Les volumes**
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Gestion des données avec Docker

- ▶ Les conteneurs sont **légers et éphémères** et ne sont pas faits pour enregistrer des données de matières persistantes
- ▶ Les données doivent être stockées en dehors de l'image du conteneur dans des **volumes de données dédiés** à cet usage
- ▶ **2 techniques historiques pour accéder à des volumes** de données
 - > l'accès au système de fichiers de l'hôte,
 - > le volume lié au conteneur

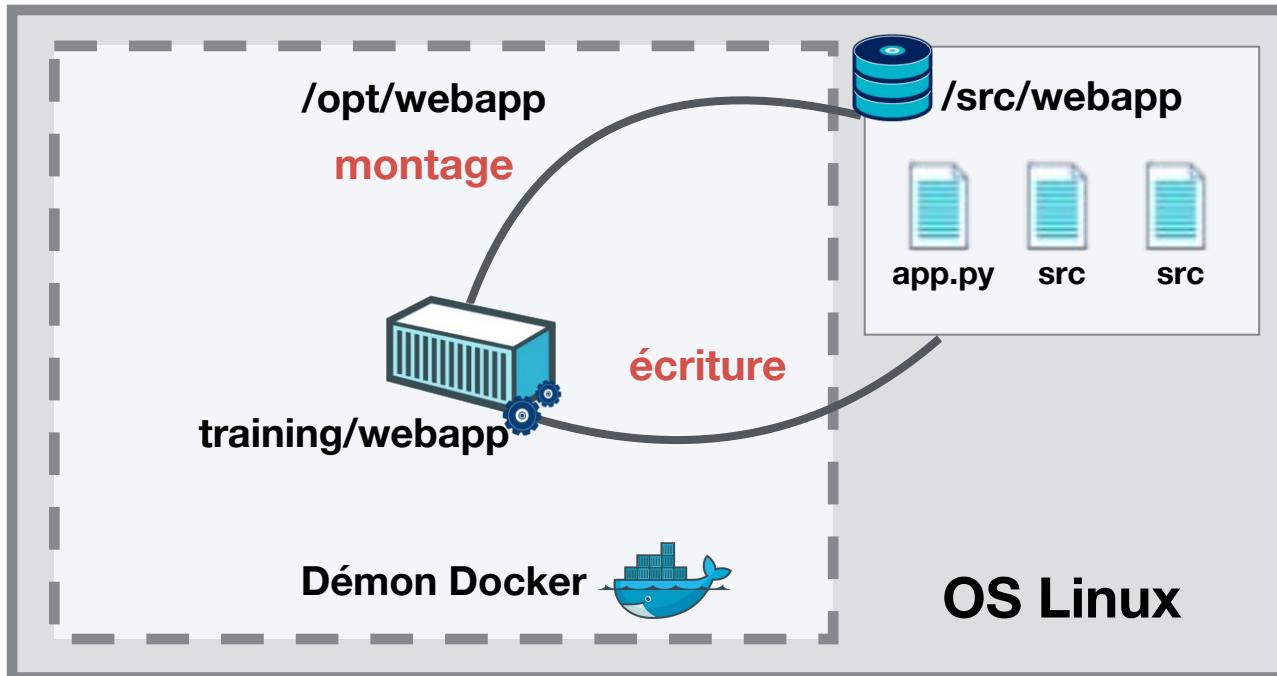


L'accès au système de fichiers de l'hôte

- Un répertoire ou un fichier de l'hôte est rendu accessible dans le conteneur

Exemple de création d'un conteneur avec un volume associé monté sur /opt/webapp

```
$ sudo docker container run -v /src/webapp:/opt/webapp training/webapp python app.py
```



Les volumes Docker

Objectifs :

- ▶ **Gérer des applications stateful** : pour les bases de données et les applications à architecture traditionnelle
- ▶ **Conserver l'indépendance avec les hôtes** : les données ne doivent pas être liées à un hôte et doivent suivre le déplacement des conteneurs qui y sont associés
- ▶ **Pouvoir conserver des données indépendamment de l'application** : pour gérer le cycle de vie de la donnée
- ▶ **Gérer les niveaux de services pour le stockage de la donnée** : SSD, IOPS garanties...
- ▶ **Faciliter les tâches opérationnelles** : snapshot, sauvegarde, restauration, copie...



L'essentiel des commandes

Créer un volume de données

`docker volume create`

Supprimer un volume

`docker volume rm`



Obtenir les informations
sur un volume

`docker volume inspect`

`docker volume ls`

Lister les volumes existants



Une gestion de volumes simplifiée

Exemples sur 2 cas d'utilisation:

- ▶ Création d'un volume de données pour MySQL

```
$ docker volume create --name mysql_data  
$ docker container run -d -v mysql_data:/var/lib/mysql mysql
```

- ▶ Sauvegarde puis restauration des données de MySQL

```
$ docker volume create --name mysql_backup  
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql  
mysql tar cjvf /backups/backup.tar.bz2 /var/lib/mysql
```

```
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql  
bash -c 'cd /var/lib/mysql && tar xvjf /backups/backup.tar.bz2'
```



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ **Création d'images et registres**
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Le Dockerfile

- Fichier contenant des suites d'instructions Docker et de commandes à exécuter pour construire un conteneur
- Permet par exemple d'installer tous les paquets requis par une application (Apache, Java, ...)

Exemple de fichiers *Dockerfile*

```
FROM ubuntu
MAINTAINER Coco LASTICOT
# Update the repository and install nginx
RUN apt-get update && apt-get install -y nginx
# Copy a configuration file from the current directory
ADD nginx.conf /etc/nginx/
EXPOSE 80
CMD ["nginx"]
```



Création des images

Sauvegarder les changements dans un conteneur dans une image

docker container commit



docker image build

Construire une image à l'aide d'un Dockerfile

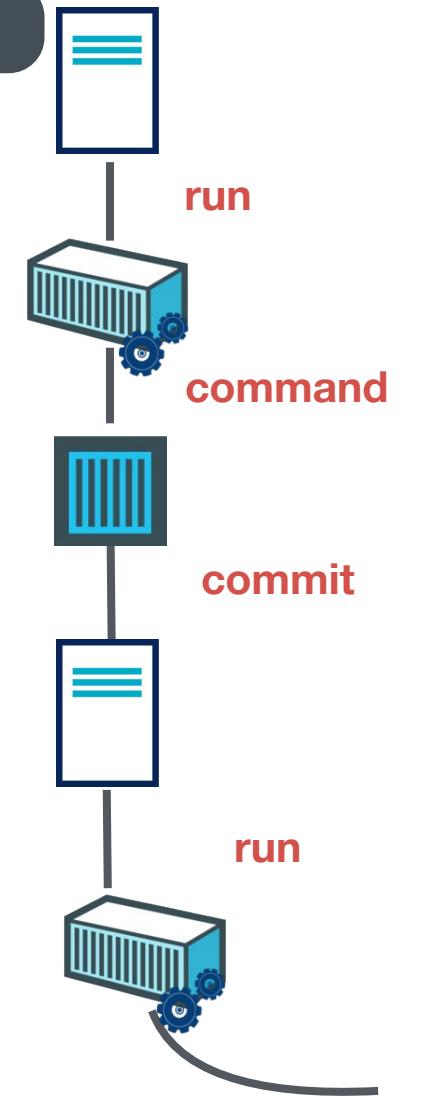


Le workflow d'un Dockerfile

```
$ docker image build -t username/myimage .
```

- ▶ Docker lance l'exécution d'un conteneur à partir d'une image
- ▶ Une instruction est exécutée et change le contenu en termes de fichiers
- ▶ Docker lance l'exécution d'un docker commit pour sauvegarder les changements de la nouvelle couche dans une image
- ▶ Docker lance un nouveau conteneur à partir de cette nouvelle image
- ▶ La prochaine instruction est exécutée... et ainsi de suite
- ▶ L'image finale est taggée “username/myimage”

•••



Dockerfile : instruction *RUN*

Exécute une commande puis créer une nouvelle image à partir des changements

```
RUN apt-get -y install apt-utils
```



Faire une étape de purge n'a pas de sens, il vaut mieux tout inliner dans une même commande :

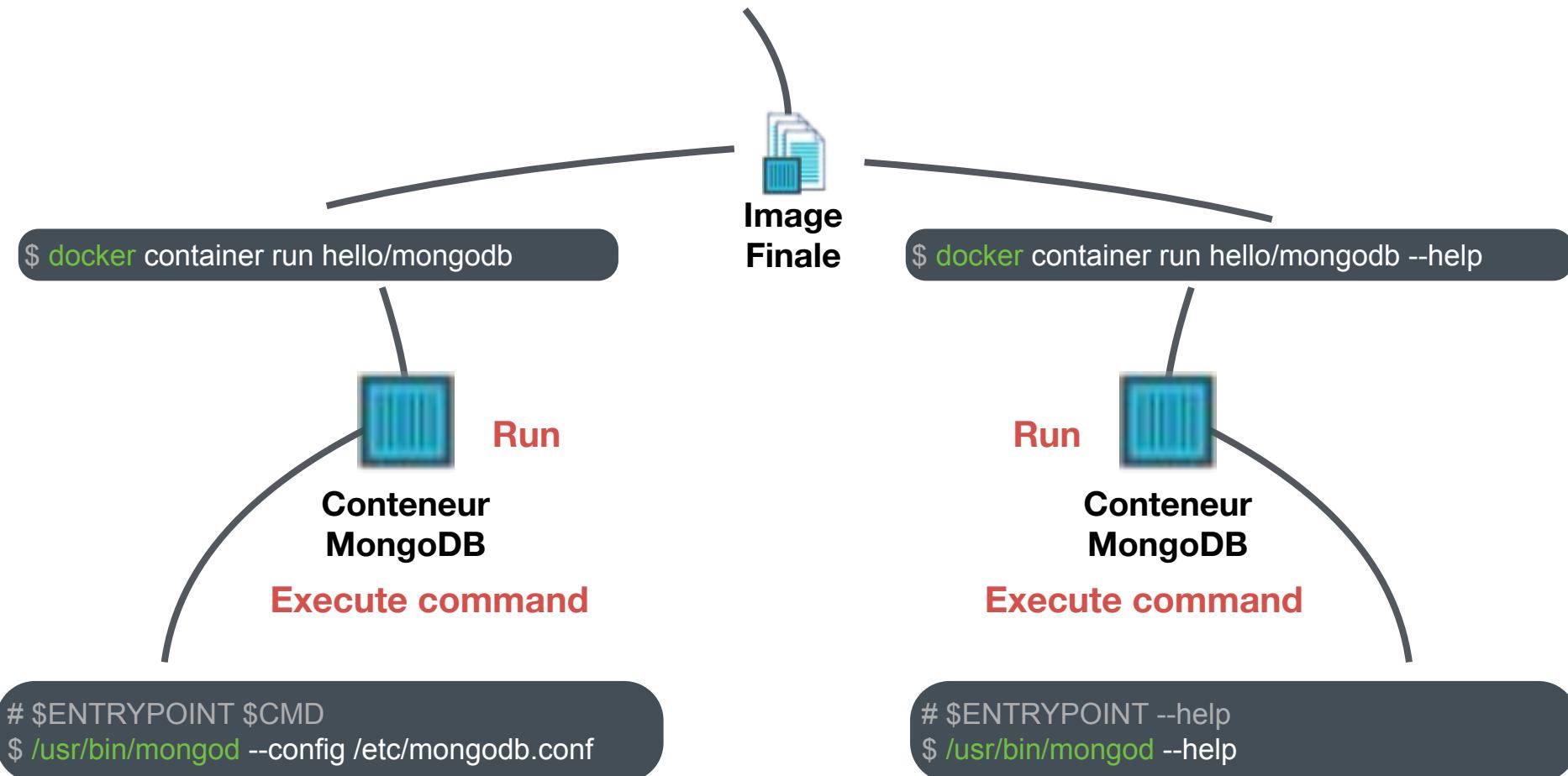
```
RUN apt-get -y update && apt-get install -y vim \  
&& rm -rf /var/lib/apt/lists/*
```



Dockerfile : instructions *ENTRYPOINT* et *CMD* ensemble

ENTRYPOINT ["/usr/bin/mongod"] Exécute un container comme un exécutable, potentiellement avec des arguments

CMD ["--config", "/etc/mongodb.conf"]



Dockerfile : instruction **WORKDIR**

Place le répertoire courant de travail dans le Dockerfile

Effectue un **syscall** pour changer de dossier

```
WORKDIR /etc/scalaris/
```

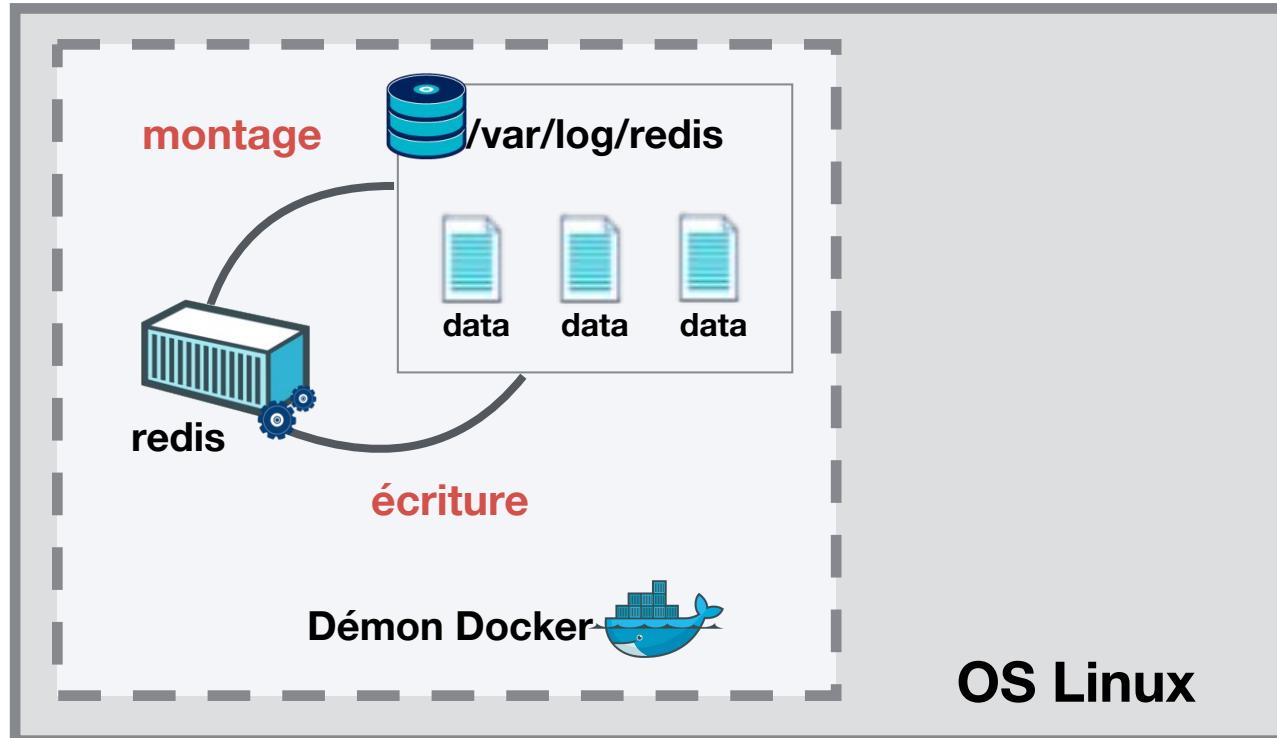
Toutes les commandes postérieur à un **WORKDIR** auront lieu dans le répertoire pointé



Dockerfile : instruction **VOLUME**

Déclare un volume secondaire au sein du conteneur pour sauvegarder des données (même effet que **docker run -v <path>**)

```
VOLUME [ "/var/log/redis" ]
```



Dockerfile : instruction **ENV**

Place une variable d'environnement pour toutes les commandes **RUN** qui suivent et pour la commande qui sera lancée par le conteneur

```
ENV JAVA_HOME /usr/share/java
```



Dockerfile : instruction **USER**

Spécifie le **user** à utiliser pour démarrer un conteneur d'application par CMD ou ENTRYPOINT

défaut = **root**

```
USER mongo # Not Kubernetes friendly
```

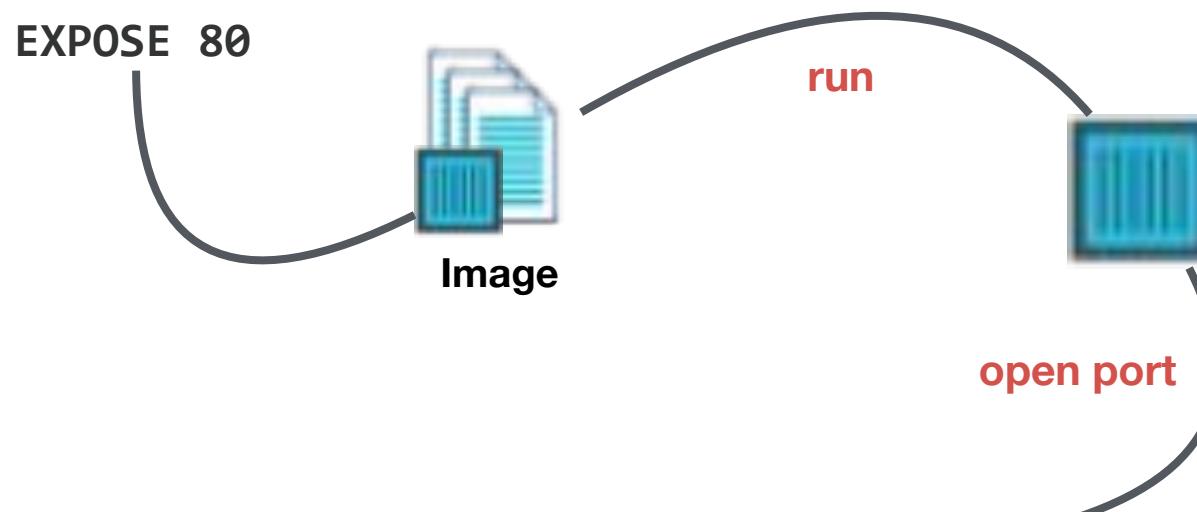
ou

```
USER 1000 # More Kubernetes friendly
```



Dockerfile : instruction EXPOSE

Déclare l'exposition d'un port du conteneur sur un port au hasard de l'hôte



```
$ iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j DNAT --to 192.168.1.2:8080 ACCEPT
```



Dockerfile : instruction COPY

Copie un fichier ou un dossier de l'Hôte dans le conteneur au chemin spécifié

Dockerfile : instruction ADD

Ajoute un fichier ou un dossier dans le conteneur au chemin spécifié.
Décomprime également les archives



Dockerfile : Un mot sur les images de base

- ▶ Toutes les distributions historiques fournissent des images de base : Debian, CentOS, Ubuntu...
- ▶ Ces images peuvent être considérées comme trop riches et complexes dans le cadre de la conteneurisation
- ▶ Des images encore plus minimalistes sont apparues. Exemple :



Dockerfile : Multistage build

- ▶ Depuis la version **17.05** de Docker Engine
- ▶ Objectif : faire des images finales les plus **légères** possibles et avec le **moins de failles** possible
- ▶ Principe : utiliser plusieurs images pour la construction d'une image définitive
 - Des images **intermédiaires** embarquent les **outils de build** (compilation, packaging, test, minification...)
 - L'image **finale** est allégée car ne contient que le **strict nécessaire à l'exécution** de l'application



Dockerfile : Exemple de Dockerfile pour un multistage build

Image :
 > 700Mo

```
FROM golang:1.11 as builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
```

Image :
 < 40Mo

```
FROM alpine:3.8
RUN apk --no-cache add ca-certificates
WORKDIR /
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
USER 1000
CMD [ "./app" ]
```





Utiliser le registre Docker



Le registre Docker

- **Un référentiel centralisé** qui stocke et rend accessible toutes les images avec leur différentes couches
- **Accessible via une API REST** formalisée et connue de tous les clients Docker
- **Permet le partage d'images** avec communauté ou au sein d'une entreprise
- Plusieurs implémentations existantes
 - > **Docker Hub** : registre public et gratuit
 - > **Docker Store** : registre d'images payantes
 - > **Docker Trusted Registry** : version entreprise on-premise
 - > **Docker Registry** : implémentation open-source
 - > **Docker Registry chez les Cloud providers** : Instanciation à la demande de Registries privées ou publiques (ex: AWS ECR Repository, GCP, Azure)
 - > **Nexus, Artifactory** peuvent également offrir le service de registre Docker
 - > **Gitlab** (une registry Docker pour chaque répo de code)
 - > **Portus, Harbour**



L'essentiel des commandes

Inscrit ou log dans
un registre

docker login



docker image push

Envoie une image
sur un registre

docker logout

logout d'un registre

docker search

Recherche une
image dans le
Docker Hub

docker image pull

Récupère une
image d'un
registre



Le registre officiel Docker - Docker Hub

- ▶ **Service de registre de Docker Inc. en mode SaaS**
- ▶ **Ouvert aux entreprises et aux utilisateurs**
(plan gratuit pour un unique dépôt privé (image en plusieurs versions) et pour un nombre illimité de dépôts publics)
- ▶ **Utilisé pour la distribution des images officielles**
(ubuntu, nodejs, ruby, gitlab...)
- ▶ Les fonctionnalités clés
 - **Automated builds** : construction automatique des images en cas de mise à jour d'un dépôt de source
 - **Webhooks** : lancement automatique d'action en cas de changements sur une images
 - **Dépôts privés** : dépôts non accessibles sans autorisation



Savoir rechercher une image - CLI

```
$ docker search java
```

Exemple de sortie de docker search

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
node	Node.js is a JavaScript-based platform for...	1465	[OK]	
java	Java is a concurrent, class-based, and obj...	555	[OK]	
develar/java		24		[OK]
anapsix/alpine-java	Oracle Java8 with GLIBC 2.21 over AlpineLinux	17		[OK]
isuper/java-oracle	This repository contains all java releases...	16		[OK]
netflixoss/java	Java Base for NetflixOSS container images	8		[OK]
nimmis/java-centos	This is docker images of CentOS 7 with dif...	7		[OK]
maxexcello/java	Docker framework container with the Oracle...	7		[OK]
nimmis/java	This is docker images of Ubuntu 14.04 LTS ...	6		[OK]
1science/java	Java Docker images based on Alpine Linux	5		[OK]
andreluiznsilva/java	Docker images for java applications	5		[OK]
lwieske/java-8	Oracle Java 8 Container	5		[OK]
aerath/java	Ubuntu with latest oracle java jdk, git, a...	1		[OK]
dwolla/java	Dwolla's custom Java image	1		[OK]

- ▶ **Stars** : nombre de votes par la communauté
- ▶ **Official** : dépôt dont la qualité a été vérifiée par Docker Inc et qui est mis à jour régulièrement
- ▶ **Automated** : image construite par Docker Hub à partir d'un processus automatique et vérifiable



Savoir rechercher une image - Interface web

Explore Help

Q java Sign up Log In

Repositories (7)

		Official		
	java official		555 STARS	1.9 M PULLS
	node official		1.5 K STARS	5.5 M PULLS
	tomcat official		399 STARS	2.6 M PULLS
	jetty official		58 STARS	365.5 K PULLS



Registre et workflow de développement

- ▶ **Le registre est le pivot central de Docker de toute utilisation industrialisée de Docker**
- ▶ Devient un référentiel dev/ops des images à déployer
- ▶ Nécessite d'être intégré dans les workflows de développement

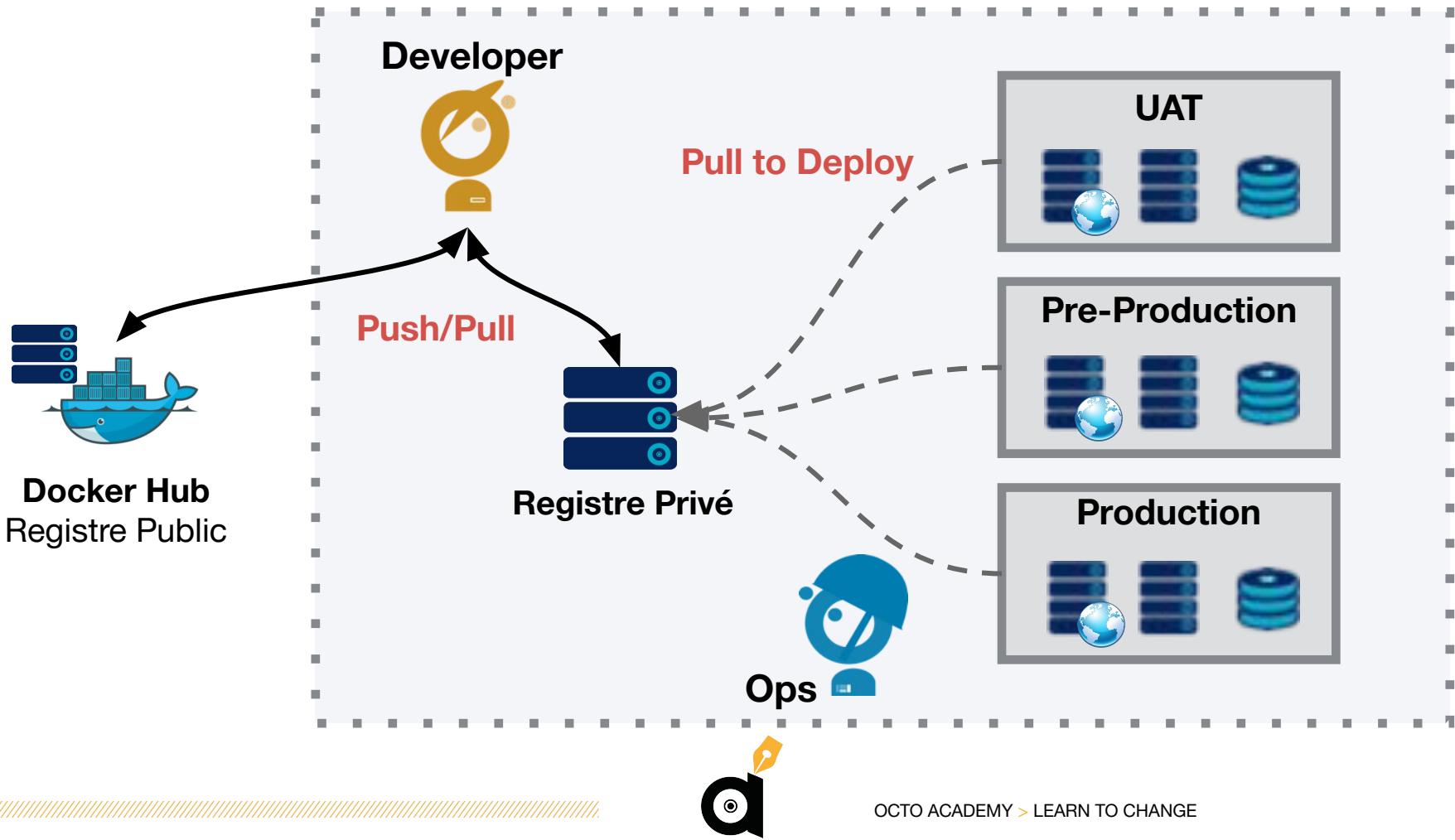
Exemple de *workflow* de développement simplifié

- Chargement d'une image de base (Pull)
- Développement en local
- Packaging de l'application dans une image (Build)
- Upload de l'image sur le Registre (Push)
- Téléchargement de l'image pour le déploiement sur les serveurs (Pull)



Registre et workflow de développement

- Collaboration Dev/Ops dans le cadre d'un workflow de développement avec Docker



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ **Docker Compose**

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

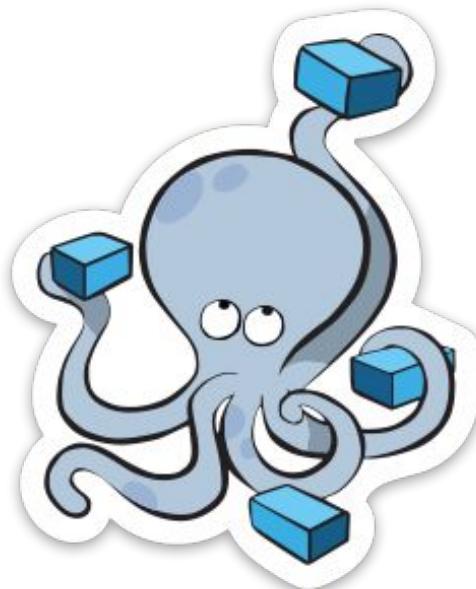
- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

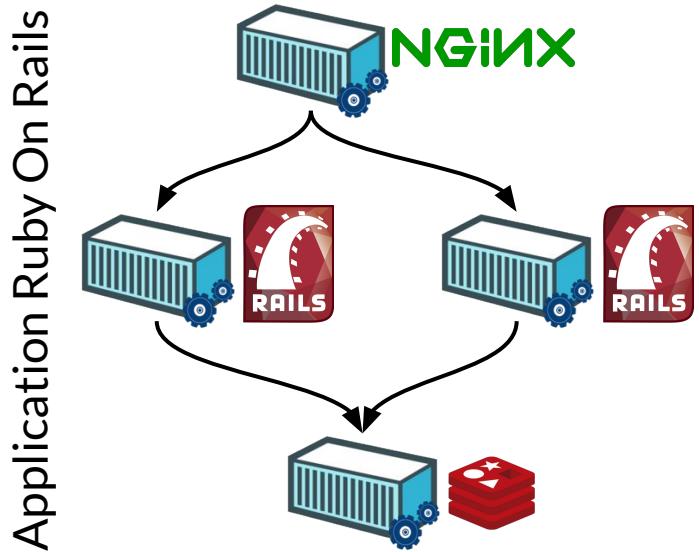
Conclusion et Take Away

Le déploiement de topologies avec Docker Compose



Pourquoi Docker Compose ?

- ▷ Une application moderne est généralement composée de **multiples conteneurs** avec de **nombreux liens** entre eux, qui utilisent potentiellement des **volumes**...
- ▷ **De nombreuses étapes à scripter** pour déployer une telle application nativement ... pour chaque environnement !



Exemple pour monter un environnement complet

```
$ cd ruby_on_rails_app
$ docker build --tag ruby_on_rails_app .
$ docker network create --driver overlay mynet
$ docker volume create --driver=volplugin --name=myvolume1 --opt size=40G
$ docker volume create --driver=volplugin --name=myvolume2 --opt size=40G
$ docker container run --name my_redis --network mynet --detach redis
$ docker container run --name ruby_on_rails_app_1 --network mynet --detach
--volume-driver=volplugin --volume myvolume1:/var/lib/data ruby_on_rails_app
$ docker container run --name ruby_on_rails_app_2 --network mynet --detach
--volume-driver=volplugin --volume myvolume2:/var/lib/data ruby_on_rails_app
...
...
```

La complexité du lancement d'un environnement complet peut vite devenir imposante et propice à des erreurs...

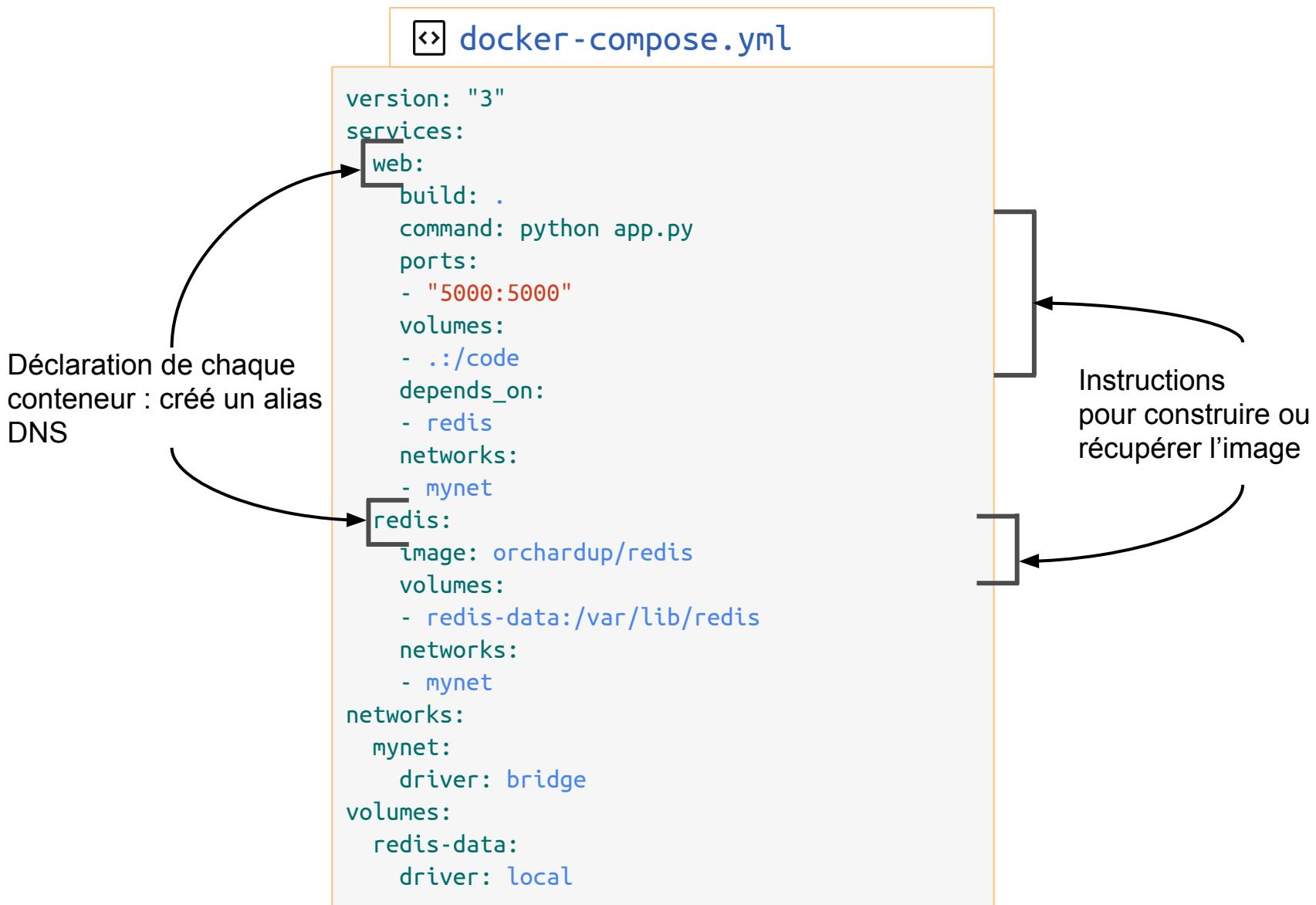


Qu'est ce que Docker Compose ?

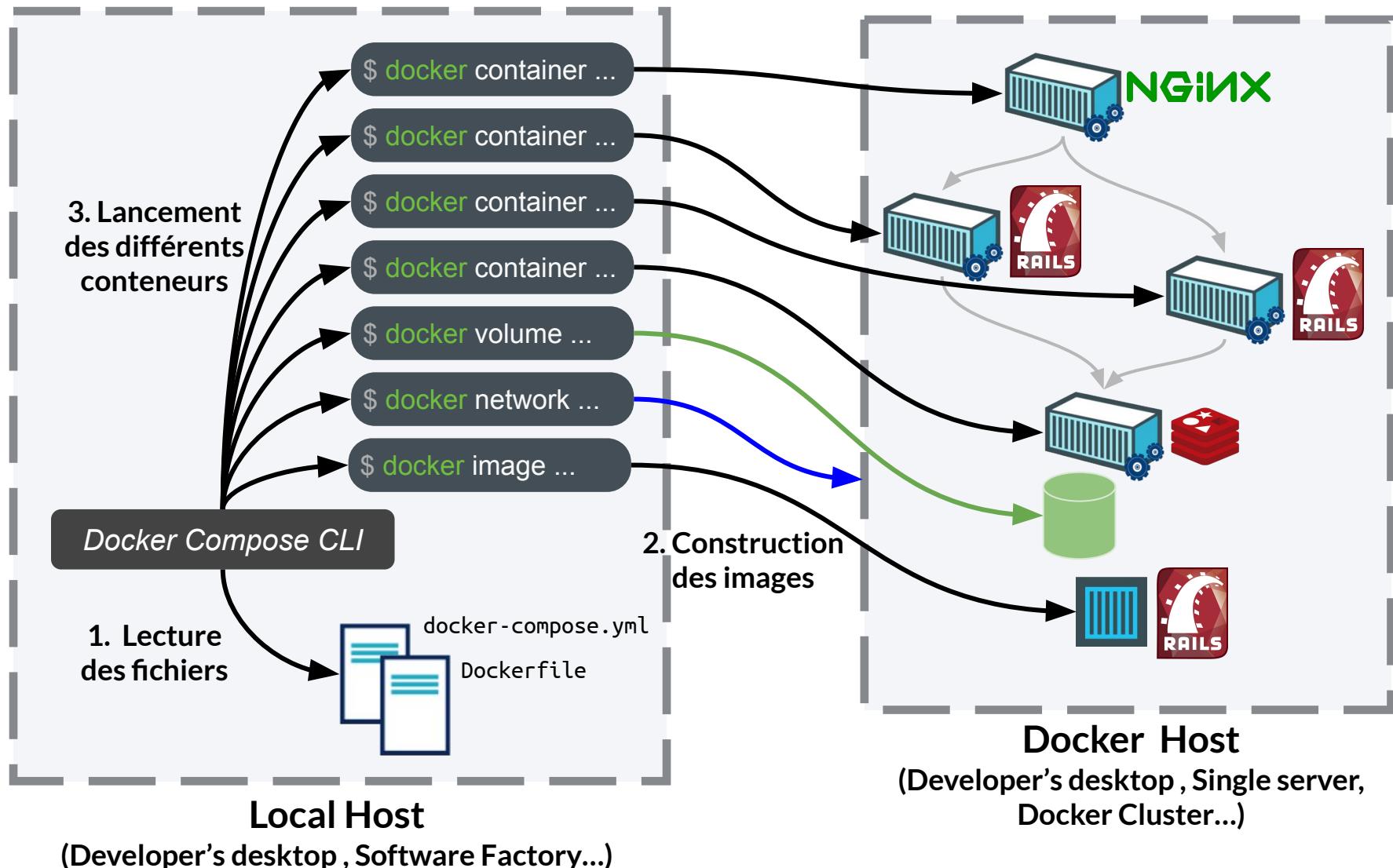
- ▷ Un outil Python en ligne de commande qui apporte
 - > **un format de description** d'une application multi-conteneurs
 - > **le déploiement d'applications multi-conteneurs** en local ou sur un cluster
- ▷ Initialement développé en dehors de Docker sous le nom de **Fig**, acquis par Docker Inc en 2014
- ▷ Permet notamment
 - > le lancement de **multiples versions d'un environnement**
 - > **le scaling (manuel)** des conteneurs stateless
- ▷ Actuellement en version 3



Le format de description



Fonctionnement de Docker Compose



L'essentiel des commandes

Créer un environnement de développement
avec un **docker-compose.yml**

Accéder aux logs des
conteneurs

docker-compose up [-d]

docker-compose logs

Stoppe/Démarre les services

docker-compose stop|start

Supprimer les
conteneurs stoppés

docker-compose rm

docker-compose ps

Pousse les
images
buildées
localement

Lister les conteneurs
lancés

docker-compose up --scale

docker-compose build

Redémarrer
les services

Ajouter des instances d'un conteneur

Reconstruit les
conteneurs



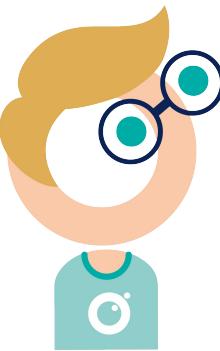
Les limitations de Docker Compose

- ▶ Le **scaling des conteneurs** ne peut être fait qu'*a posteriori*
- ▶ Une logique d'orchestration de déploiement simpliste et gérée en dehors de la plateforme de gestion des conteneurs : c'est le poste client qui orchestre !
- ▶ Non utilisable (directement) dans un contexte Kubernetes en dehors de Docker EE 2.0

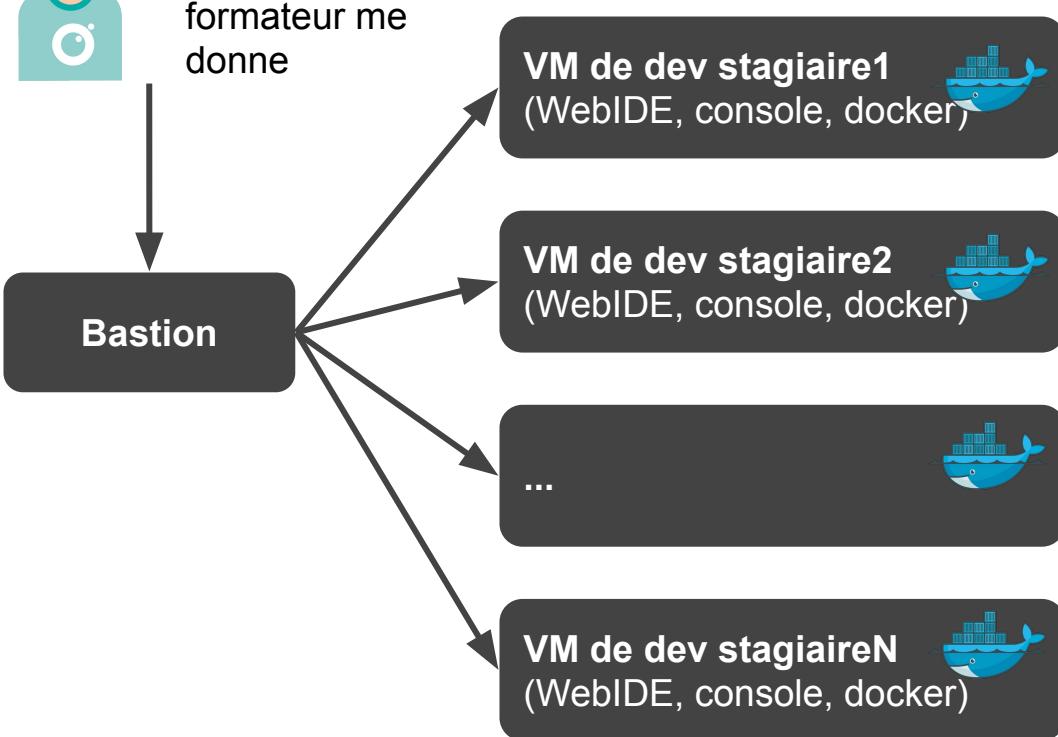
 On vous conseille de n'utiliser **Docker Compose** que sur votre poste de dev en local



Un mot sur les environnements



J'accède à mon environnement avec l'@ IP que le formateur me donne



TP#2

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ **Introduction & historique de K8s**
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Genèse de Kubernetes : à la croisée des chemins



Genèse de Kubernetes : à la croisée des chemins



Projets internes de conteneurisation chez Google (non opensource)

Utilisés comme plateformes pour tous les produits Google

Plusieurs années de REX chez Google

- => Énormes plateformes
- => Modélisation puissante

Omega exploite le principe d'un référentiel de configuration commun distribué basé sur Paxos



Genèse de Kubernetes : à la croisée des chemins



D'anciens développeurs de Borg écrivent K8s en Go

Directement pensé pour utiliser Docker (engine)

Directement dans l'optique d'en faire un projet OpenSource

Version 1.0 en Juin 2015 (donné à la CNCF, organisation qui appartient à la Linux Foundation)

Vient de κυβερνήτης : grec pour « timonier » ou « pilote »



L'objectif de Kubernetes

- ▷ Définir et déployer des **applications multi-conteneurs**
- ▷ Répartir les conteneurs sur **une flotte d'hôtes (nœuds)**
- ▷ **Optimiser et adapter le placement** des conteneurs
- ▷ **Surveiller la santé** des conteneurs
- ▷ Définir et appliquer des contraintes de **niveaux de services**
- ▷ Gérer **la disponibilité et la scalabilité** des conteneurs
- ▷ Gérer **le provisionnement et l'accès au stockage**
- ▷ **Isoler les conteneurs**
 - Limitation de ressources
 - Sécurité (vision multi-tenant)

Tout ça de manière dynamique et pour des milliers de conteneurs !

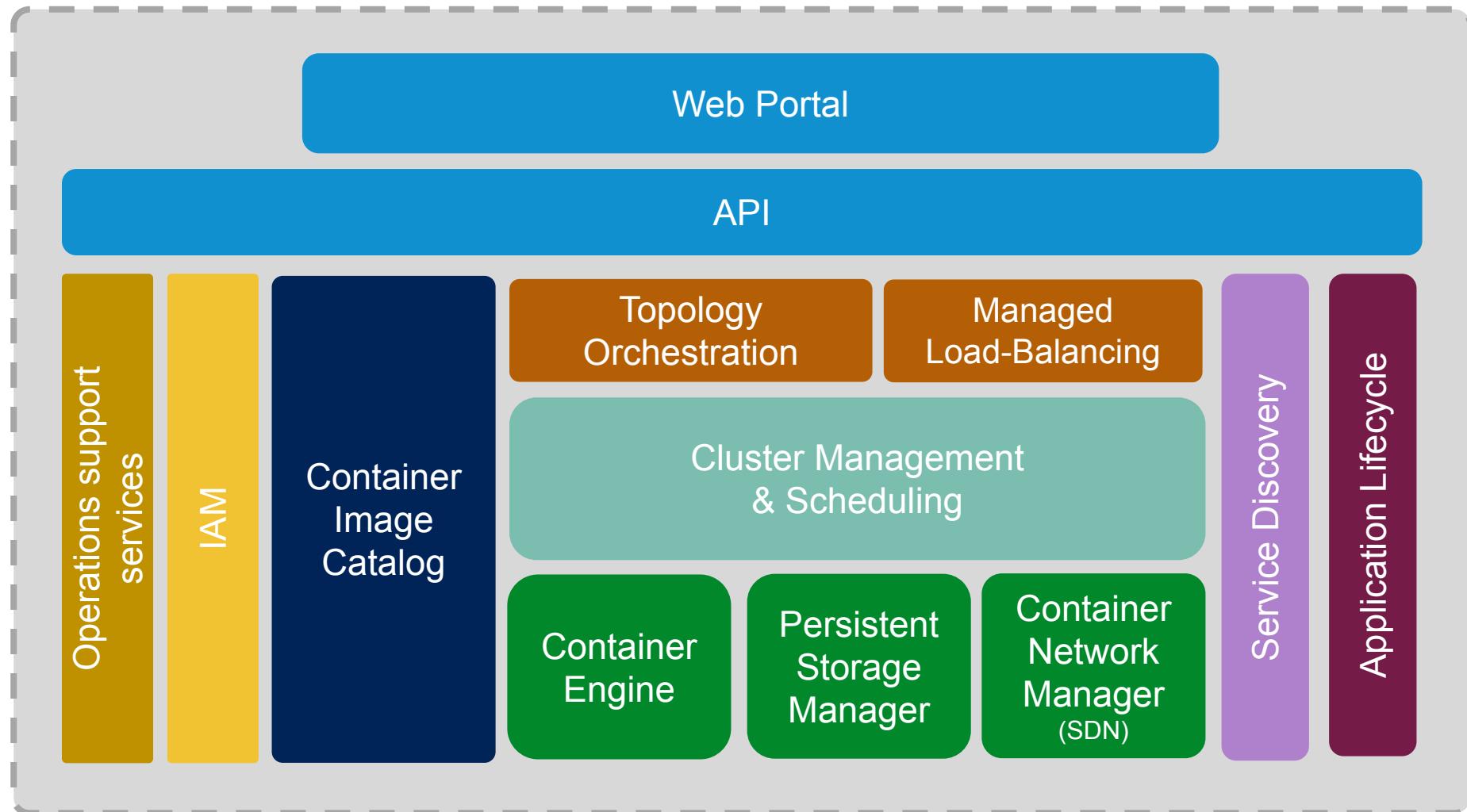


L'approche de K8s

- ▶ **Abstraction** des concepts pour éléver le niveau de modélisation
 - > *Apparition de différents types de ressources de haut niveau*
 - > *On ne manipule que très rarement la notion de conteneur directement*
- ▶ Approche **déclarative** plutôt que procédurale
 - > *On décrit ce que l'on souhaite, pas comment l'obtenir*
 - > *Notion de Desired State Configuration*



Les briques d'un orchestrateur de conteneurs (CaaS)



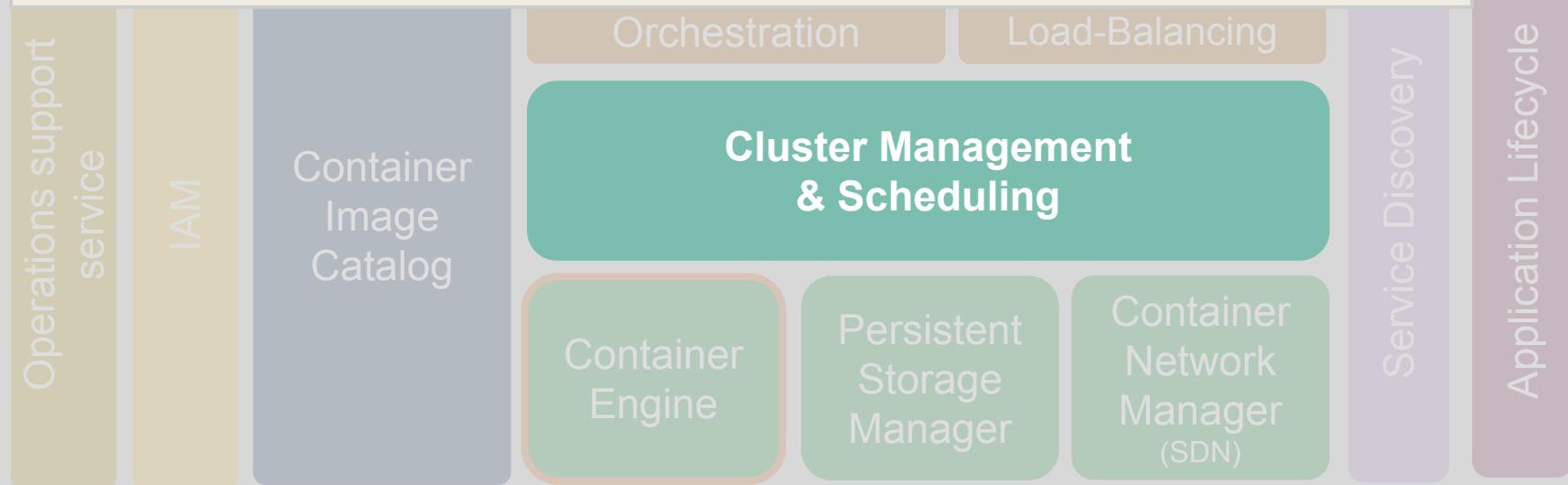
Cartographie fonctionnelle



Cluster Management

Rôle

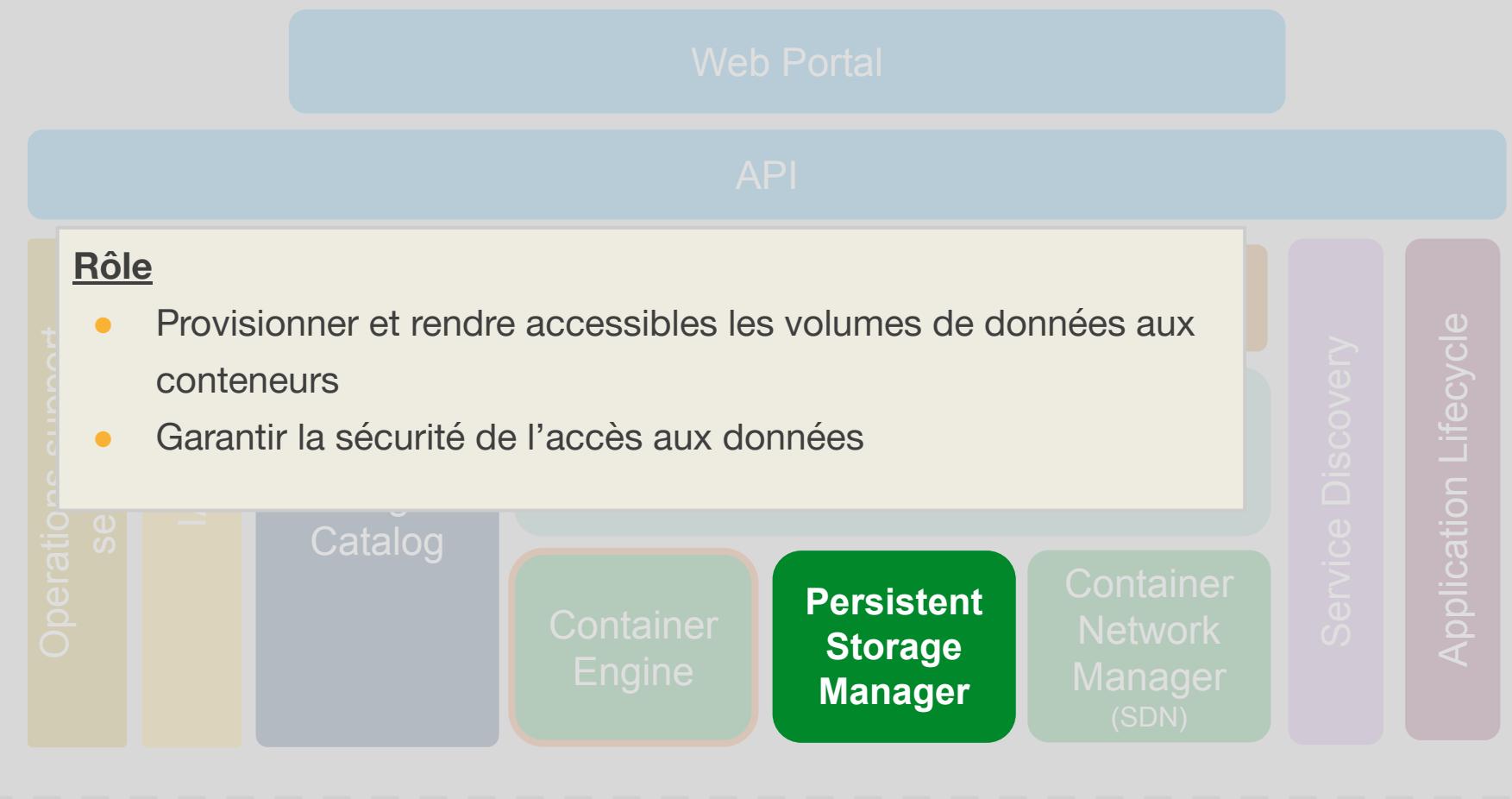
- Abstraire une flotte d'hôtes comme une ressource unique
- Déterminer et optimiser le placement des conteneurs sur les hôtes
- Surveiller et mettre à jour l'état du cluster
- Gérer les dysfonctionnements des nœuds du cluster



Cartographie fonctionnelle



Persistent Storage Manager



Cartographie fonctionnelle



Container Network Manager

Web Portal

Rôle

- Mettre en communication les différents conteneurs par des réseaux
- Assurer le routage des flux
- Garantir l'isolation réseaux des conteneurs

Operations support

Service

Catalog

Container Engine

Persistent Storage Manager

Container Network Manager (SDN)

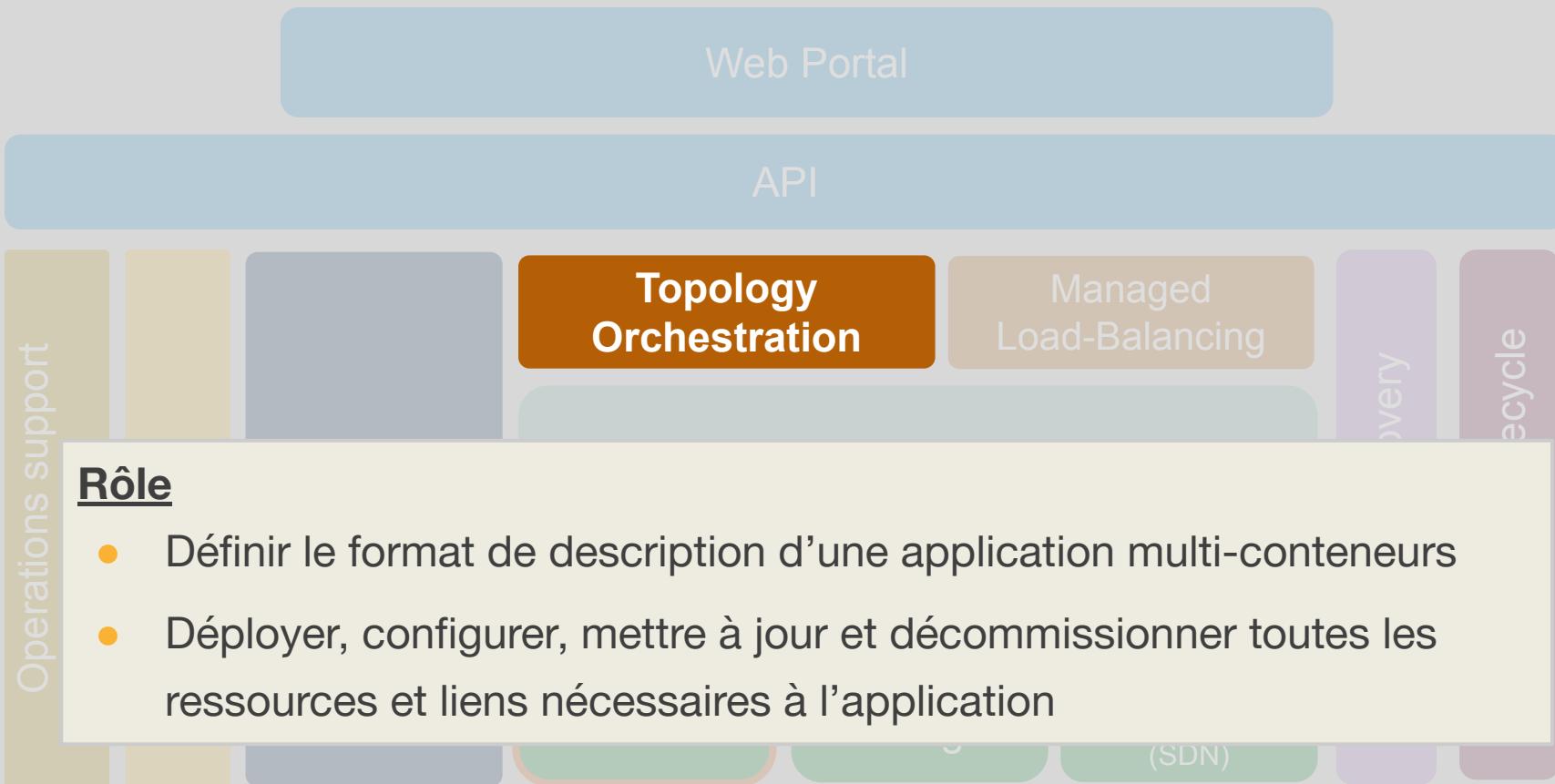
Service Discovery

Application Lifecycle

Cartographie fonctionnelle



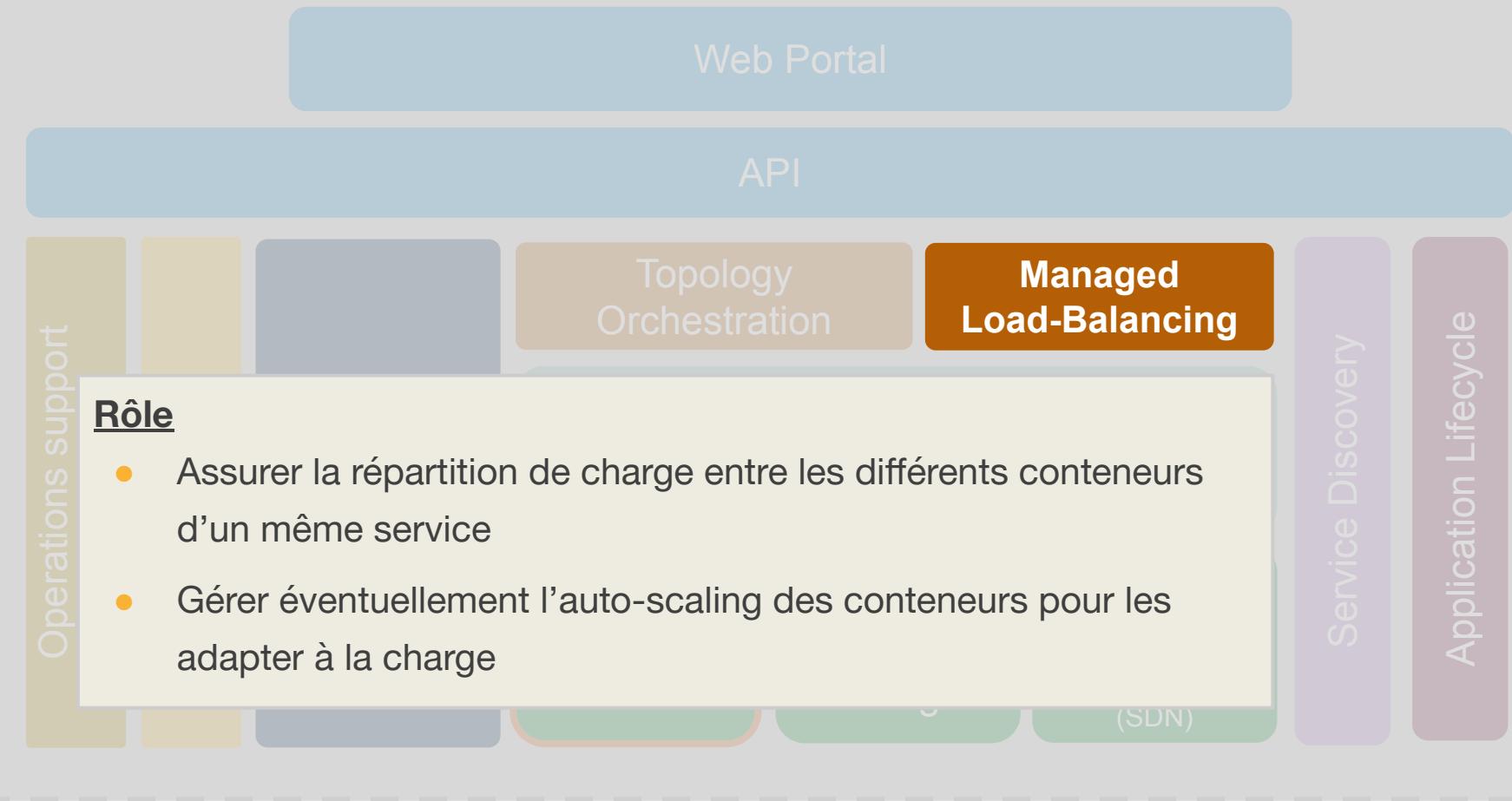
Topology orchestration



Cartographie fonctionnelle



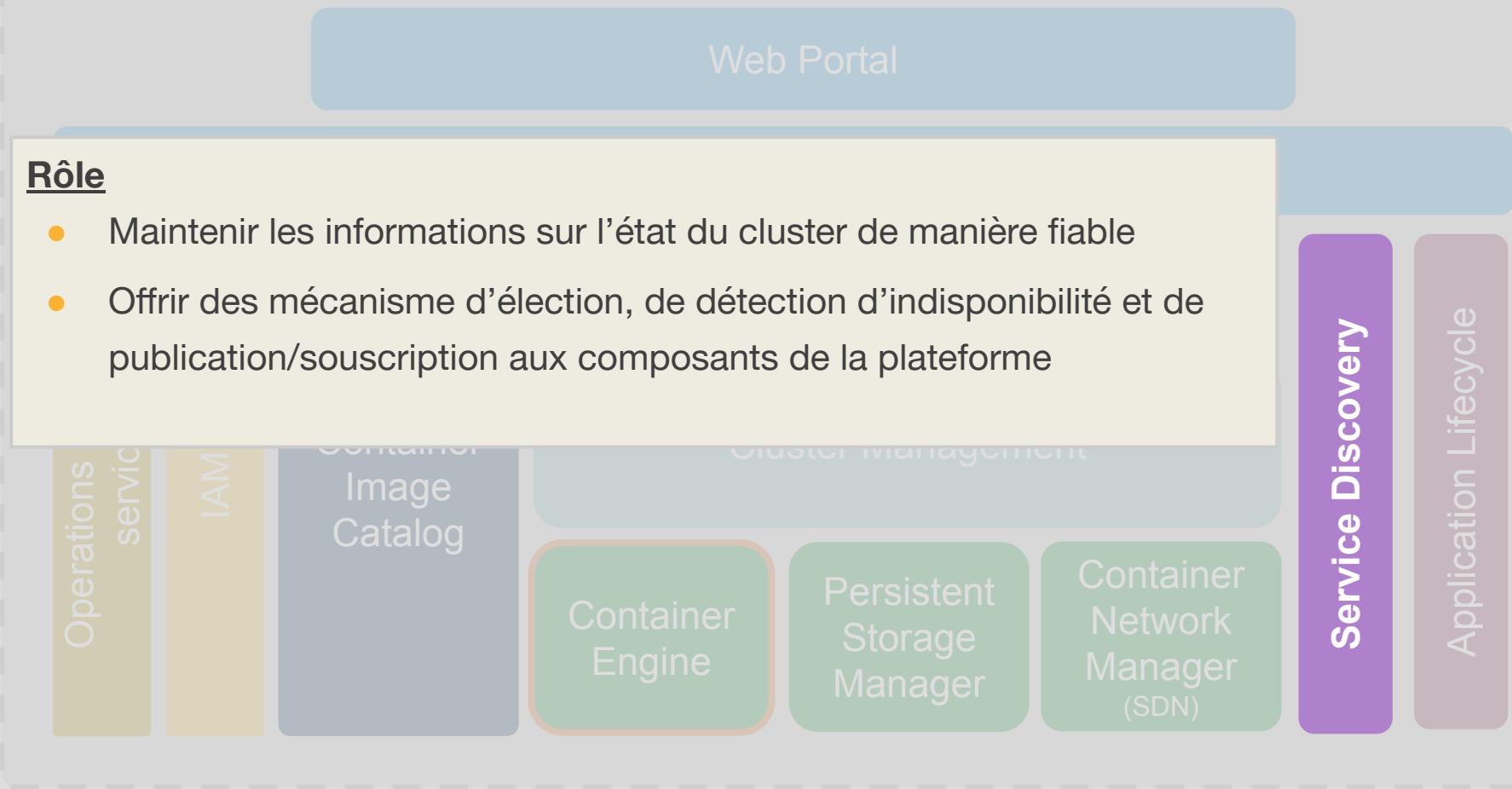
Managed Load-Balancing



Cartographie fonctionnelle



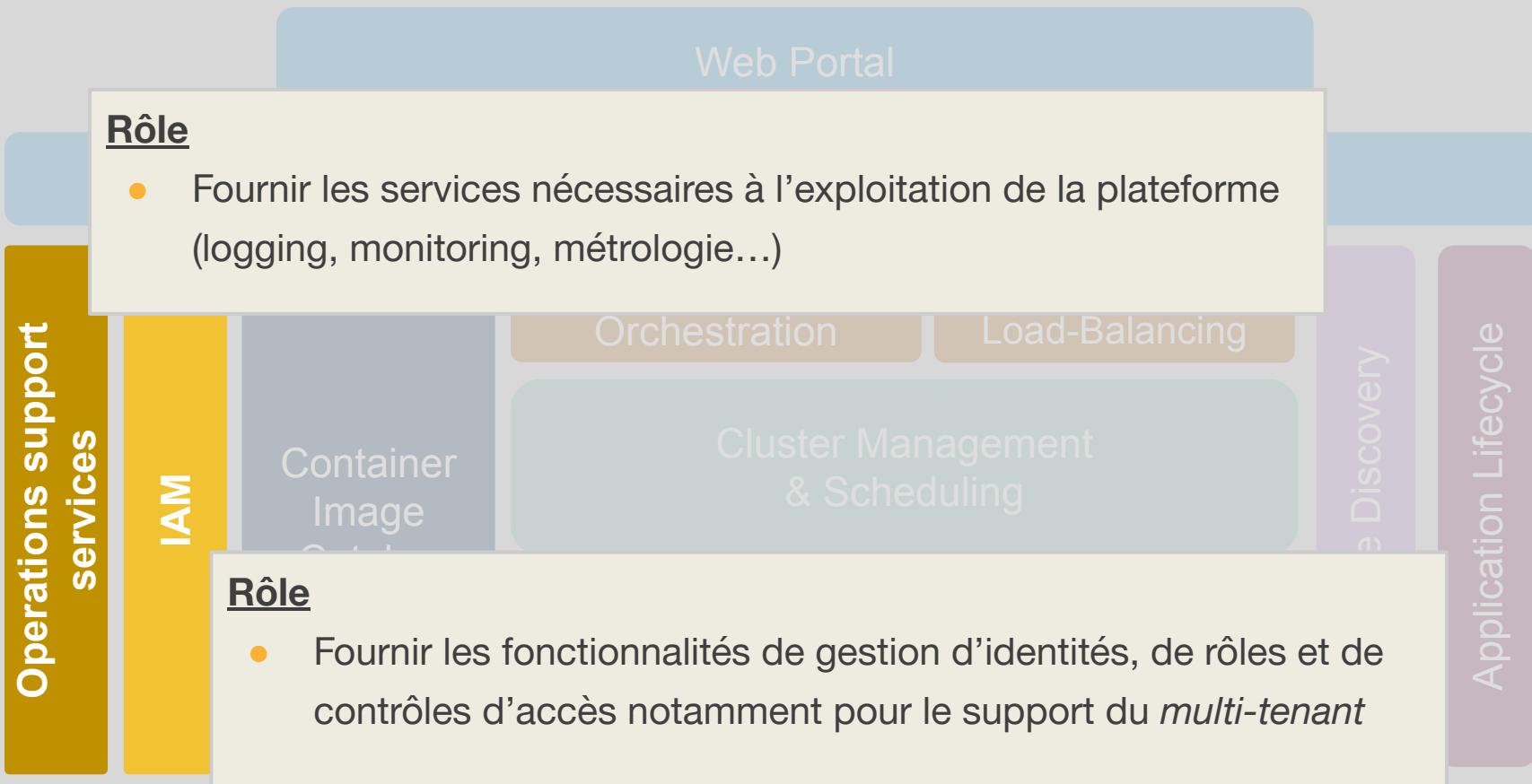
Distributed directory



Cartographie fonctionnelle



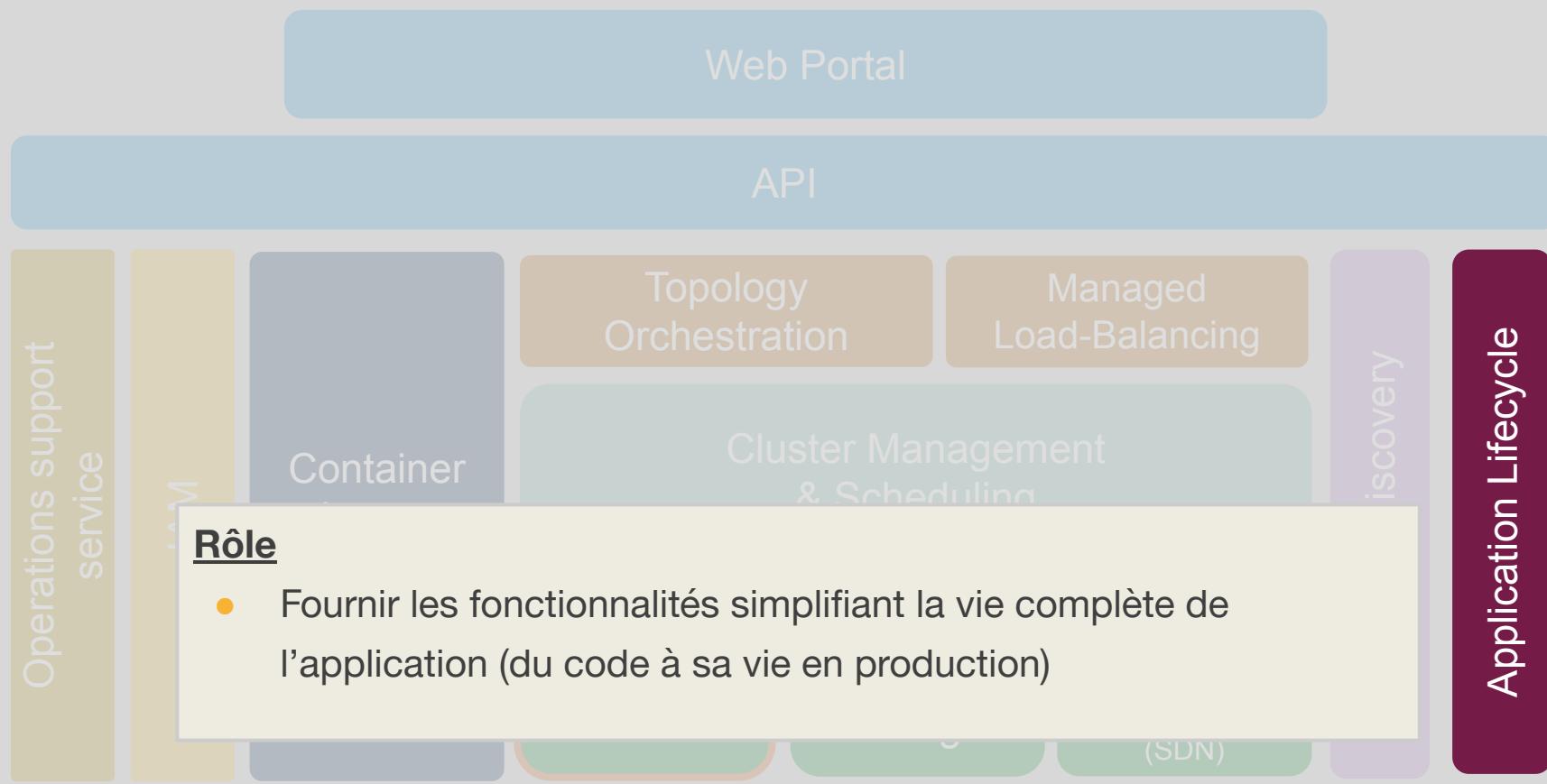
Operations support services and IAM



Cartographie fonctionnelle



Application Lifecycle



Cartographie fonctionnelle



Docker, en 2014/2015...

Non
implémenté

Implémentation
partielle

Implémentation
complète

Topology
Orchestration



Web Portal

API

Operations support
services

IAM

Container
Image
Catalog



Cluster Management
& Scheduling

Container
Engine



Persistent
Storage
Manager

Container
Network
Manager
(SDN)

Managed
Load-Balance



Service Disco

Application Lifecycle



Couverture fonctionnelle Kubernetes

Non implémenté

Implémentation partielle

Implémentation complète

Web Portal

API

Operations support services

IAM

Container Image Catalog

Topology Orchestration

Managed Load-Balancing

Cluster Management & Scheduling

Container Engine

Persistent Storage Manager

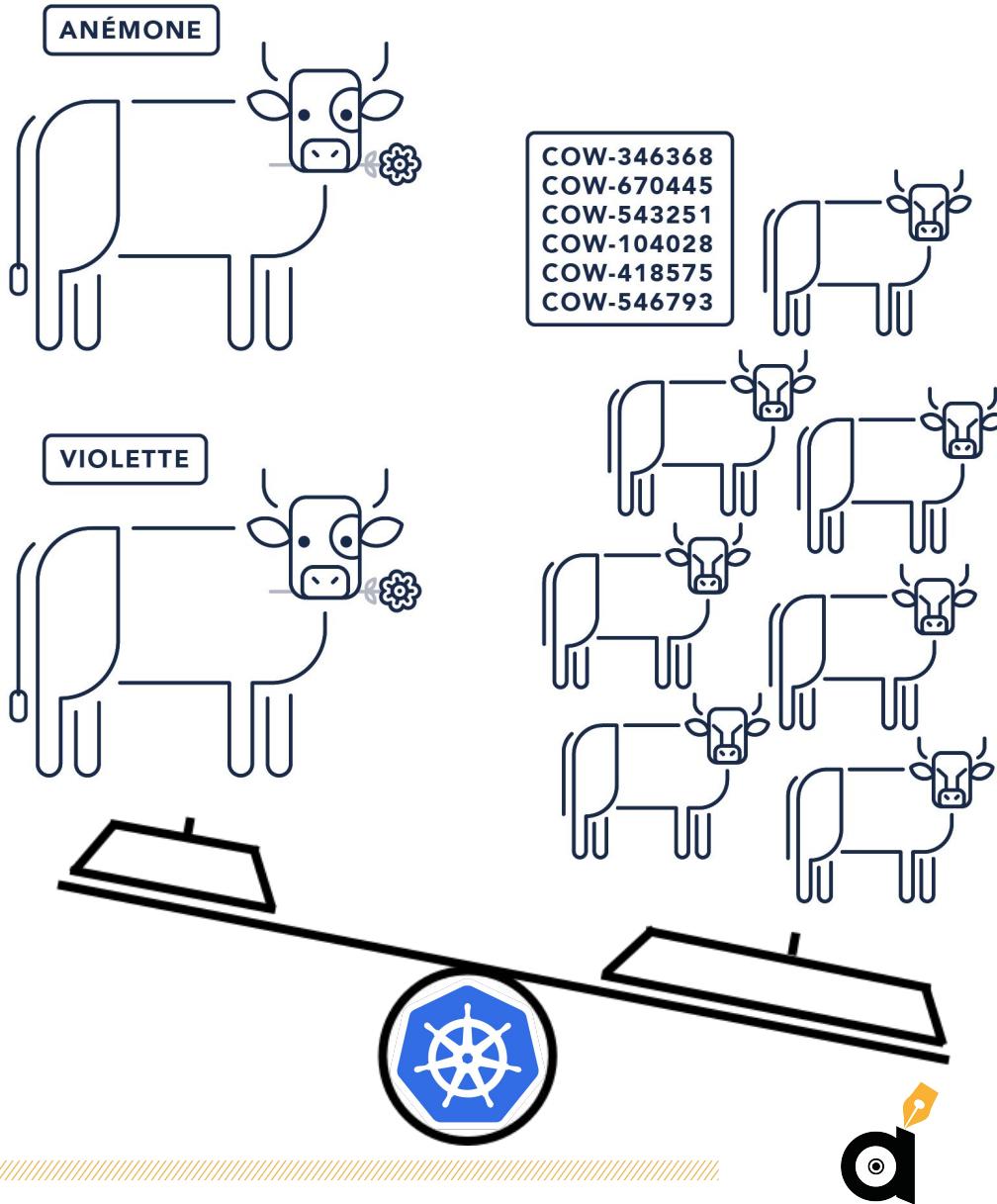
Container Network Manager (SDN)

Service Discovery

Application Lifecycle



Pets vs. Cattle



Comme Docker, K8s permet avant tout de déployer des composants de type **Cattle** :

- Scalables horizontalement
- Immuables
- Sans état
- Share nothing
- Création et destruction facile, rapide et à faible coût

Des architectures applicatives à revoir

- Des applications *Legacy* qui s'adaptent mal aux contraintes de la conteneurisation
- Des nouveaux patterns de conception à respecter pour tirer partie de la puissance de la conteneurisation
- Un ensemble de patterns déjà connus pour la conception d'applications Cloud (*Cloud Native Applications*)
- Un frein classique à l'adoption d'un PaaS ou d'un CaaS qui doit être anticipé dans la mise en œuvre



Les 12 facteurs

1. Avoir une **base de code** suivie avec un système de contrôle de version
2. Déclarer explicitement et isolez les **dépendances**
3. Stocker la **configuration** dans l'environnement
4. Traiter les **services externes** comme des ressources attachées
5. Séparer strictement les étapes de **Build, Release, Run**
6. Exécuter l'application comme un ou plusieurs **processus sans état**
7. Exporter les services via des **associations de ports**
8. **Concurrence** : Grossir à l'aide du modèle de processus
9. Créer des applications “**jetables**”
10. Garder le **développement, la validation et la production aussi proches que possible**
11. Traiter **les logs** comme des flux d'événements
12. Lancer les **processus d'administration** et de maintenance comme des one-off-processes

Un document de référence: [The Twelve Factor App](#)



Comment utiliser Kubernetes ?

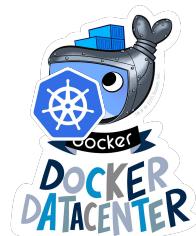
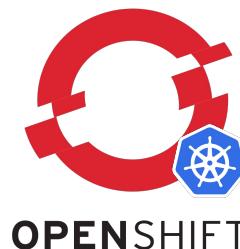
- ▶ **Chez soi**
 - Directement sur des serveurs physiques (bare metal)
 - Sur des VMs traditionnelles (VMWare)
 - Sur un cloud privé (OpenStack)
 - Sur son laptop (**minikube / Docker for Mac**)
- ▶ **Sur le cloud** (avec des capacités d'intégration avancées : LB, Volumes...)
 - **Amazon Web Services** (VM ou EKS)
 - **Google Cloud Platform** (VM ou GKE)
 - **Azure** (VM ou AKS)
 - Chez beaucoup d'autres fournisseurs de clusters Kubernetes managés

Des outils connexes au projet Kubernetes sont là pour aider les déploiements (kops, kubeadm...)



Le CaaS : un puzzle à reconstituer

Amazon EC2
Container Service



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ **Utilisation du client kubectl**

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

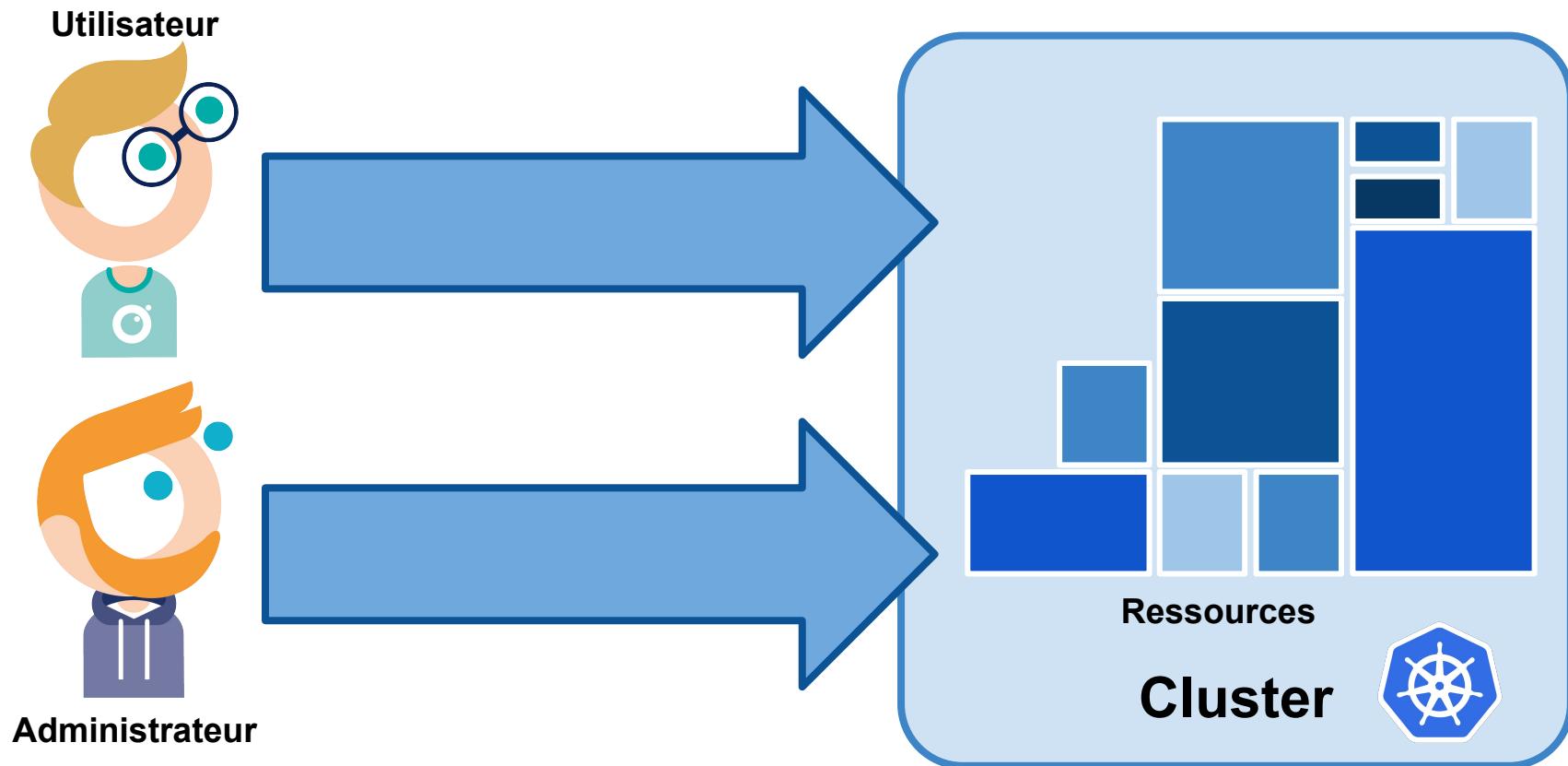
Comment interagir avec Kubernetes ?

- ▶ Avec l'**API**
 - Interaction programmatique pour manipuler les ressources
 - Approche déclarative de l'**État Attendu** (**Desired State**)
- ▶ Avec un **SDK** (Golang, Python...) qui s'appuie sur l'API
 - Terraform, Ansible
- ▶ Avec le **CLI**
 - Binaire **kubectl**
 - Utilise le SDK qui s'appuie sur l'API
- ▶ Avec le(s) **Dashboard(s)**
 - Visualisation simplifiée des ressources présentes dans le cluster
 - Pas complet, les ressources récentes de Kubernetes n'apparaissent pas en visualisation

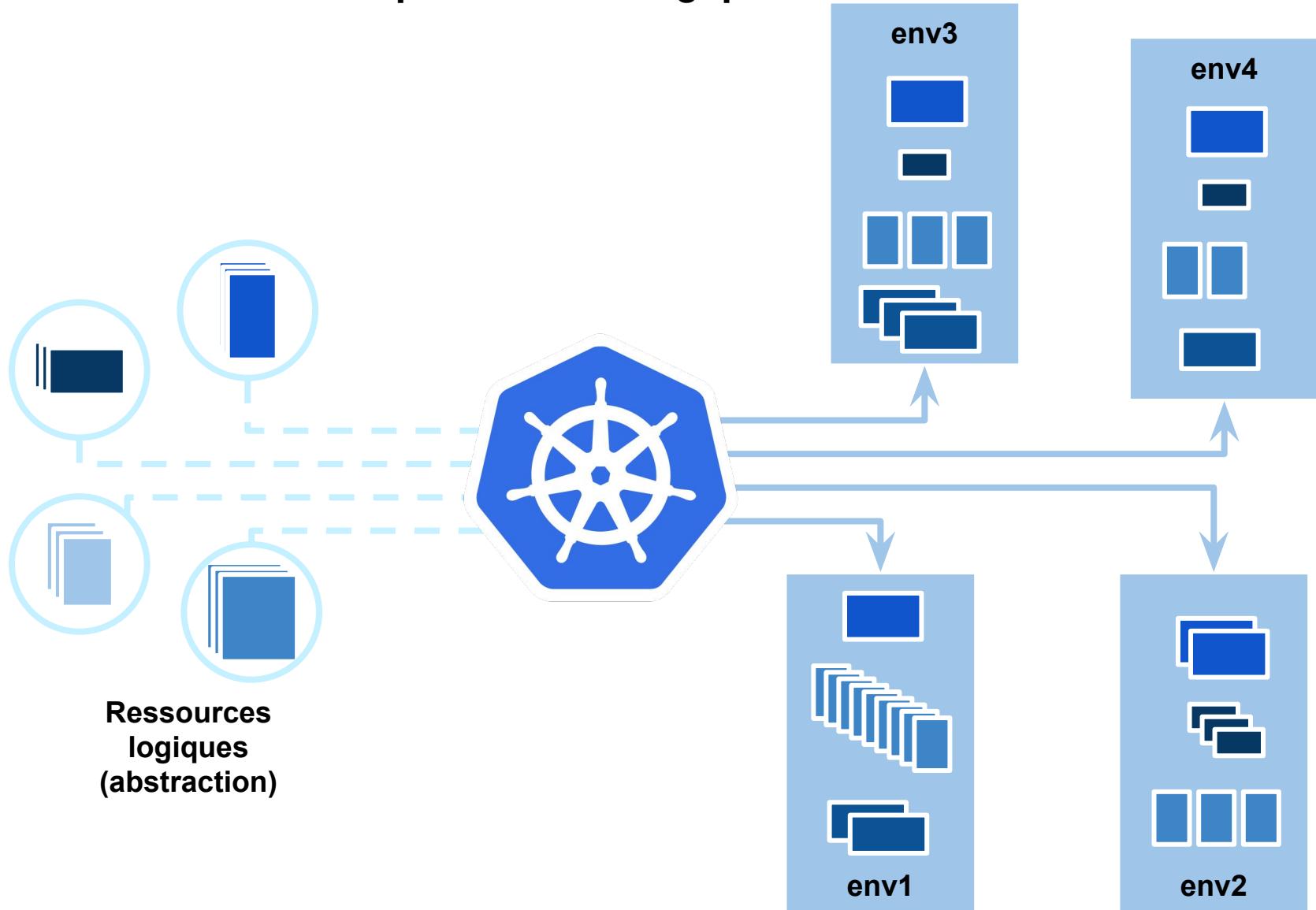


Utilisation de K8s : avant tout de la gestion de ressources

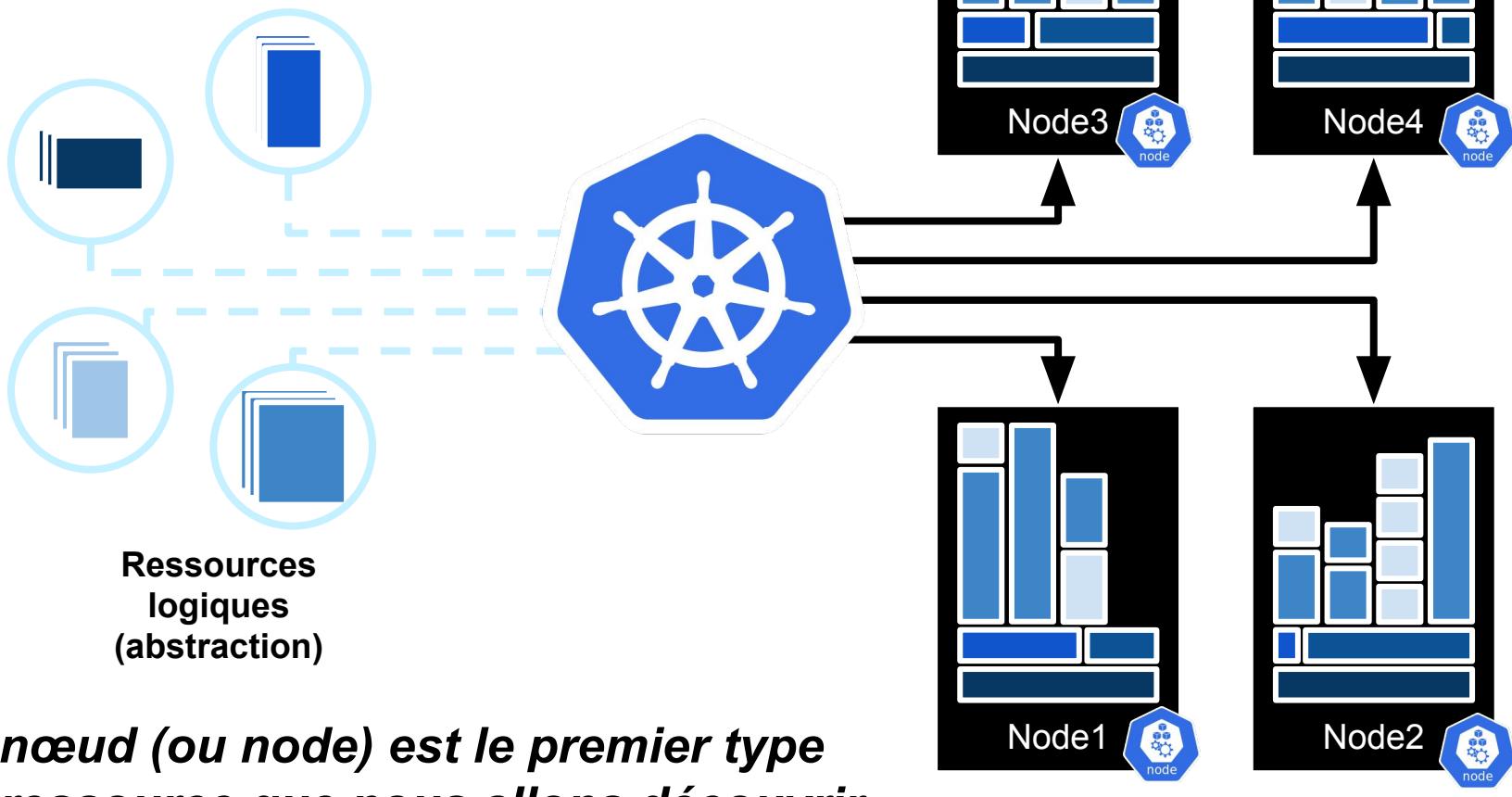
- Une **ressource** est un **concept logique** manipulé dans Kubernetes
- Il en existe beaucoup (20+)
- Certaines sont réservées aux administrateurs



Utilisation de K8s : Implémentation logique



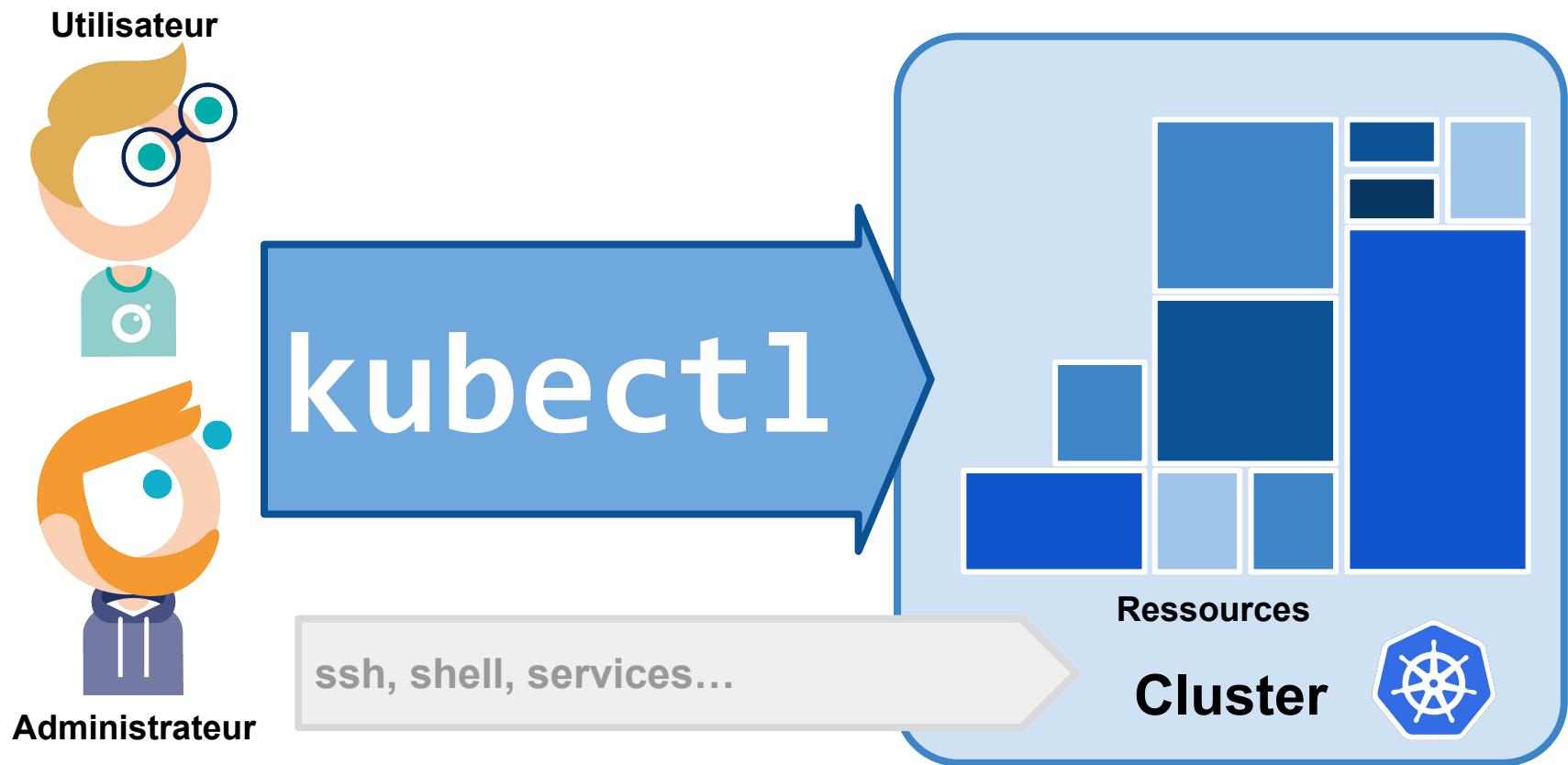
Utilisation de K8s : Implémentation technique



Le nœud (ou node) est le premier type de ressource que nous allons découvrir



kubectl : one CLI to rule them all



Un mot sur kubectl

- C'est un exécutable binaire écrit en **Go** qui existe pour la plupart des plateformes classiques (**Windows, Linux, MacOSX**)
- Pour le télécharger : <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- Le client kubectl suit la **même numérotation** que la partie serveur
- Pour avoir la dernière version stable :
<https://storage.googleapis.com/kubernetes-release/release/stable.txt>



1ère utilisation de kubectl : lister, afficher des ressources (get, describe)

- ▶ Lister toutes les ressources d'un type

```
$ kubectl get (type1|type2|type3|...)
```

- ▶ Lister une ou des ressource spécifique(s)

```
$ kubectl get type1/nom-ressource1 type2/nom-ressource2  
$ kubectl get type3 nom-ressource3
```

- ▶ Pour avoir plus de détails

```
$ kubectl get type1/nom-ressource -o wide  
$ kubectl get type1/nom-ressource -o (yaml|json)  
$ kubectl describe type8/nom-ressource
```



Utilisation de kubectl : les formats de sortie (option -o)

```
json  
yaml  
wide  
custom-columns=...  
custom-columns-file=...  
[go-]template=...  
[go-]template-file=...  
jsonpath=...  
jsonpath-file=...
```

```
$ kubectl get nodes/minikube -o=go-template \  
--template="{{.metadata.name}} est sous {{.status.nodeInfo.operatingSystem}}"  
minikube est sous Linux  
  
$ kubectl get nodes \  
-o=custom-columns=NAME:.metadata.name,CPU:.status.allocatable.cpu  
NAME      CPU  
minikube   2
```



Utilisation de kubectl : création / destruction de ressources

- En mode « automagique »

```
$ kubectl run [plein d'options]
```

- Faire le ménage

```
$ kubectl delete (type1|type2|type3|...) NAME
```

```
$ kubectl delete type6/NAME
```

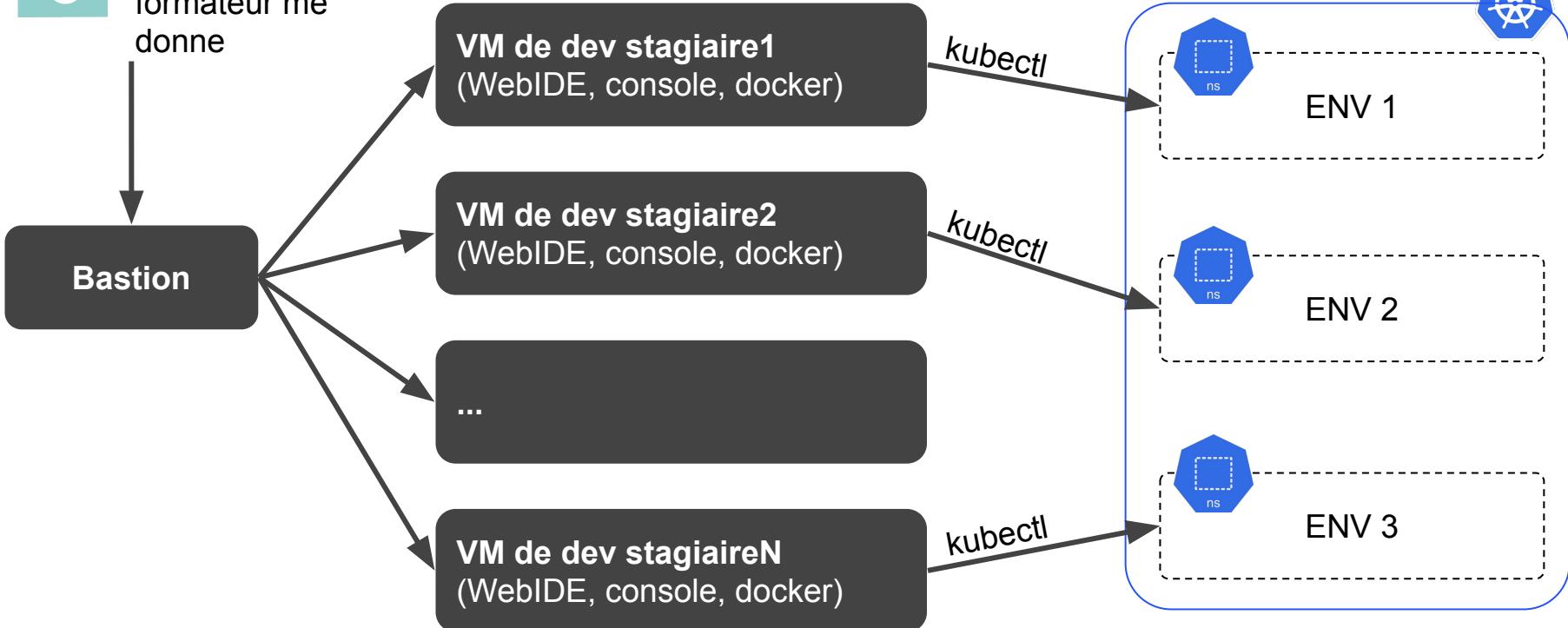
```
$ kubectl delete type8 --all
```



Un mot sur les environnements

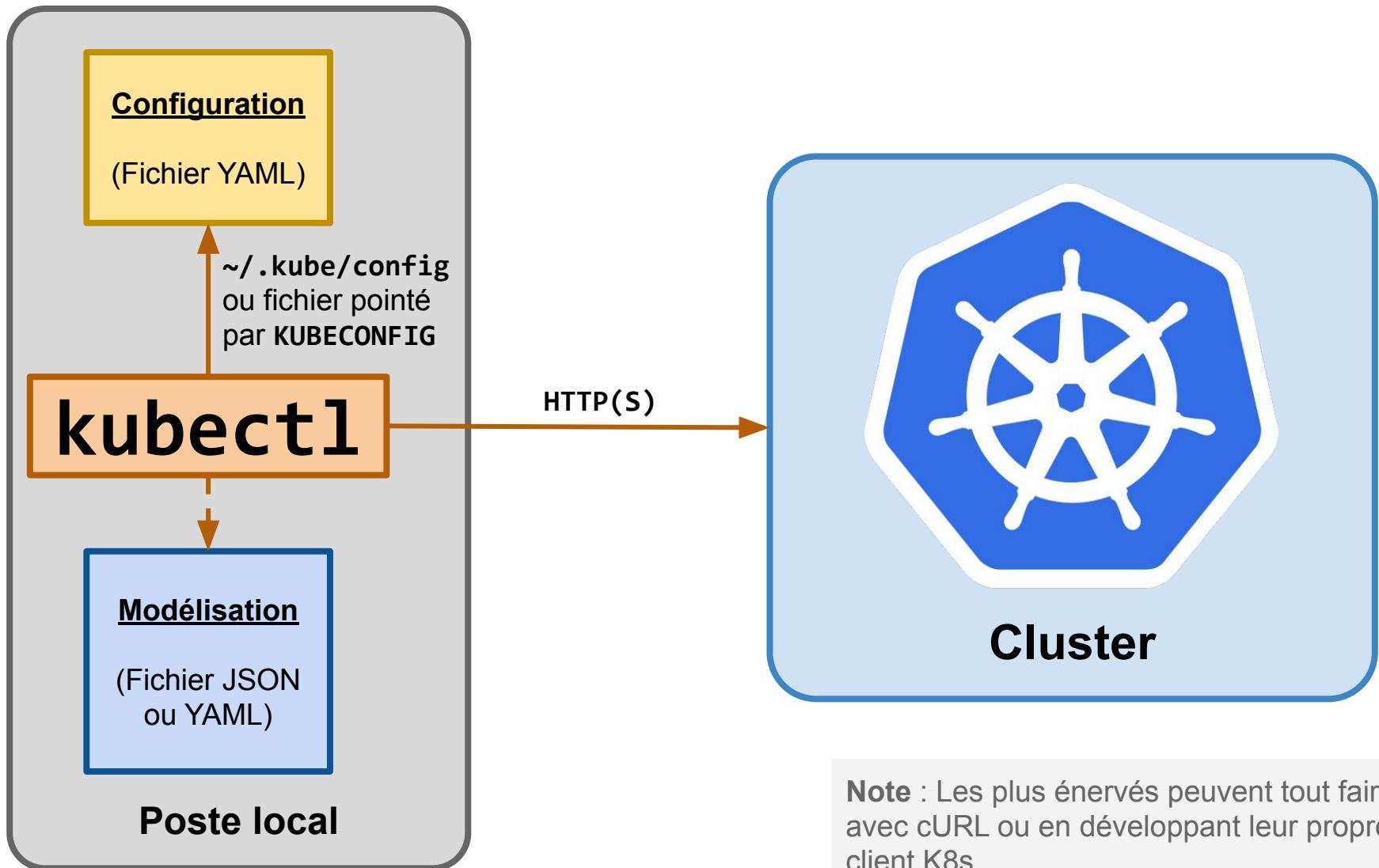


J'accède à mon environnement avec l'@ IP que le formateur me donne



TP#3

kubectl : le cli n°1 de K8s



Note : Les plus énervés peuvent tout faire avec cURL ou en développant leur propre client K8s...



Utilisation de kubectl : les contextes de configuration

- Un seul fichier de configuration **kubeconfig** permet de décrire plusieurs **contextes** d'exécution
 - Sur quel **cluster** se connecter
 - En tant que quel **utilisateur** et avec quelle **méthode d'authentification**
 - Login / mot de passe
 - Token
 - Certificat client + sa clé privée
- On peut choisir le **contexte** (**--context**)
- On peut surcharger le **cluster** (**--cluster**)
- On peut surcharger l'**utilisateur** (**--user**)
- Il y a un contexte **courant**, actif par défaut (**current-context**, **use-context**)



Utilisation de kubectl : les contextes de configuration

- La manipulation des contextes (création, suppression, activation...) peut se faire
 - À la main, avec un bon éditeur de texte
 - Avec les commandes `kubectl config ...`

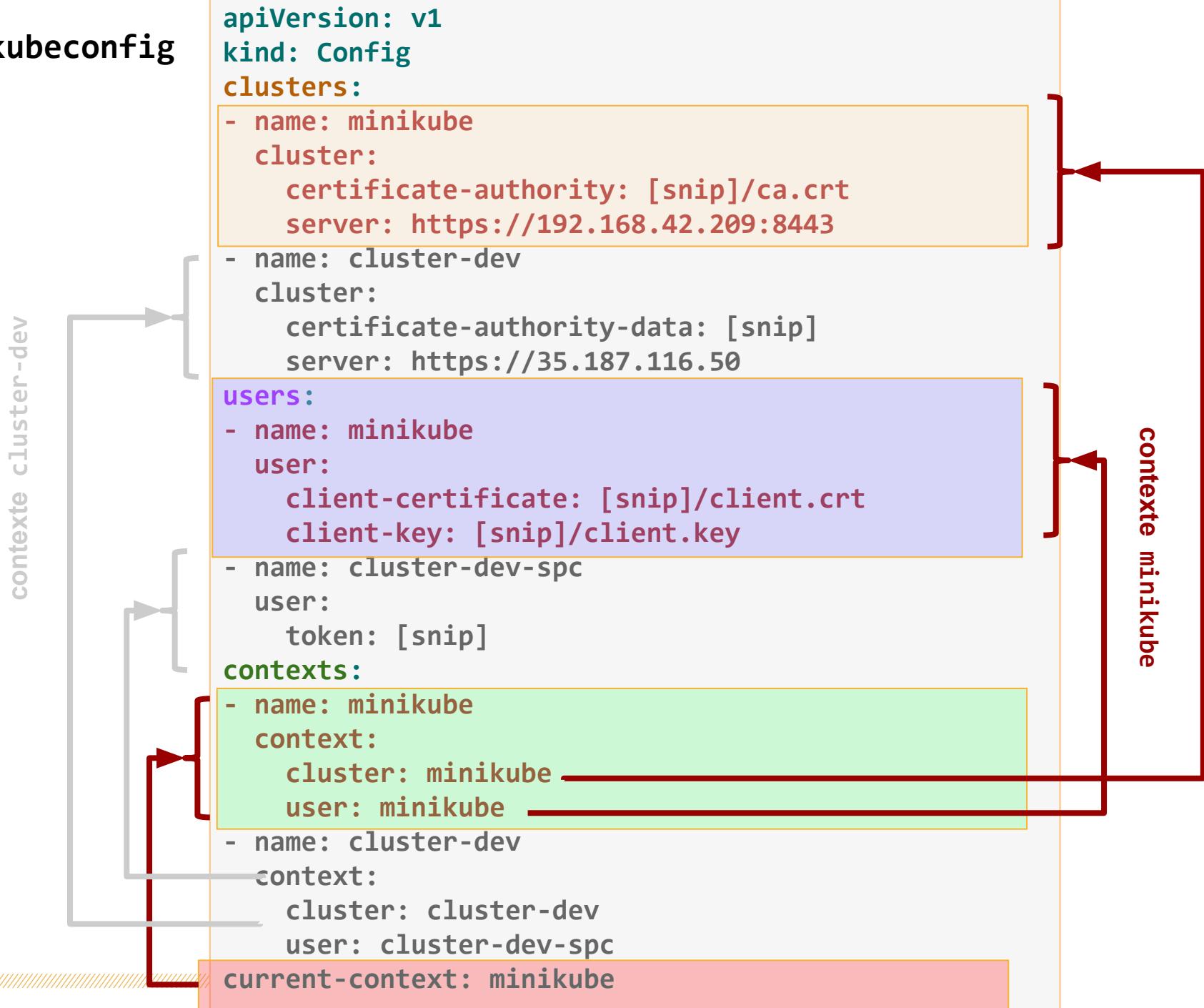
```
$ kubectl config current-context  
minikube
```

```
$ kubectl config get-contexts  
NAME  
cluster-dev  
minikube
```

```
$ kubectl config use-context cluster-dev  
Switched to context "cluster-dev".
```



kubeconfig



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Les types de ressources dans K8s (spoiler, il y en a beaucoup)

```
$ kubectl get
```

You must specify the type of resource to get. Valid resource types include:

* all

* certificatesigningrequests (aka 'csr')

* clusterrolebindings

* clusterroles

* clusters (valid only for federation apiservers)

* componentstatuses (aka 'cs')

* configmaps (aka 'cm')

* controllerrevisions

* cronjobs

* customresourcedefinition (aka 'crd')

* daemonsets (aka 'ds')

* deployments (aka 'deploy')

* endpoints (aka 'ep')

* events (aka 'ev')

* horizontalpodautoscalers (aka 'hpa')

* ingresses (aka 'ing')

* jobs

* limitranges (aka 'limits')

* namespaces (aka 'ns')

* networkpolicies (aka 'netpol')

* nodes (aka 'no')

* persistentvolumeclaims (aka 'pvc')

* persistentvolumes (aka 'pv')

* poddisruptionbudgets (aka 'pdb')

* podpreset

* pods (aka 'po')

* podsecuritypolicies (aka 'psp')

* podtemplates

* replicaset (aka 'rs')

* replicationcontrollers (aka 'rc')

* resourcequotas (aka 'quota')

* rolebindings

* roles

* secrets

* serviceaccounts (aka 'sa')

* services (aka 'svc')

* statefulsets

* storageclasses

error: Required resource not specified

Use "kubectl explain <resource>" for help and examples (e.g. kubectl explain pods).

See 'kubectl get -h' for help and examples....

* all

vrai nom

alias (pour les paresseux)

* namespaces (aka 'ns')

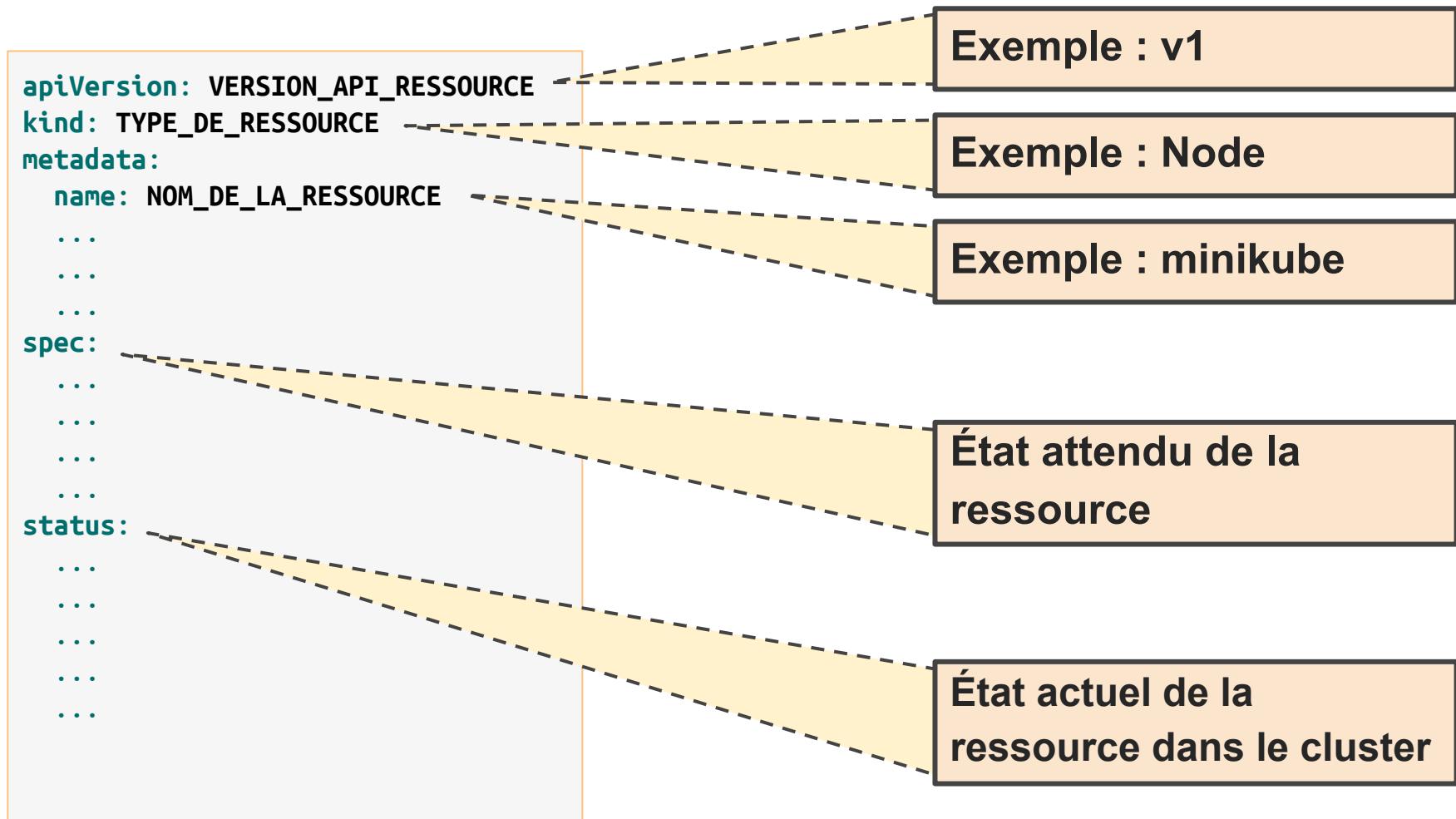
* nodes (aka 'no')

Use "**kubectl explain <resource>**" for a detailed description of that resource (e.g. **kubectl explain pods**).

See '**kubectl get -h**' for help and examples....

Quelques généralités concernant les ressources K8s

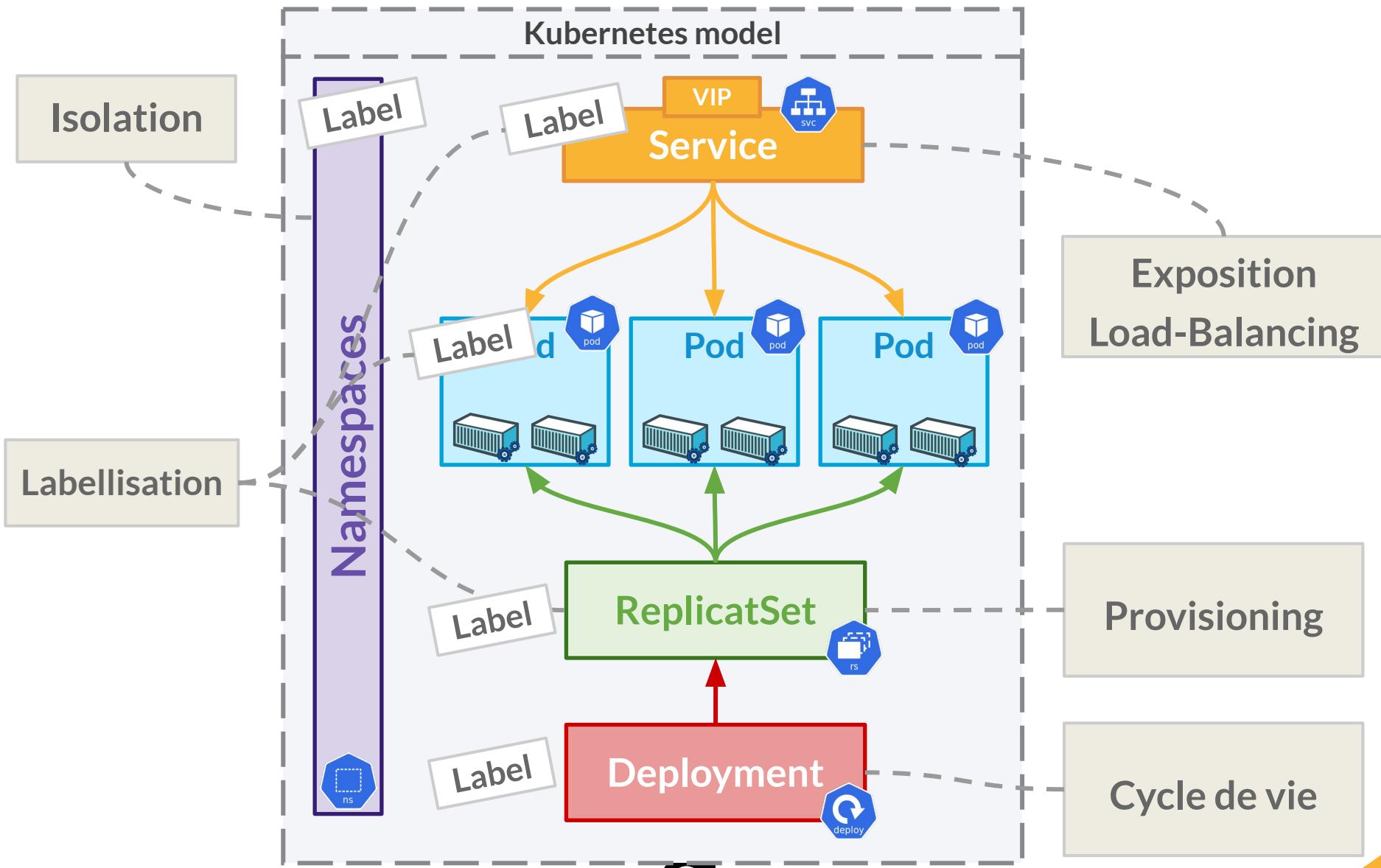
Dans Kubernetes, (presque) toutes les ressources sont structurées de façon identique



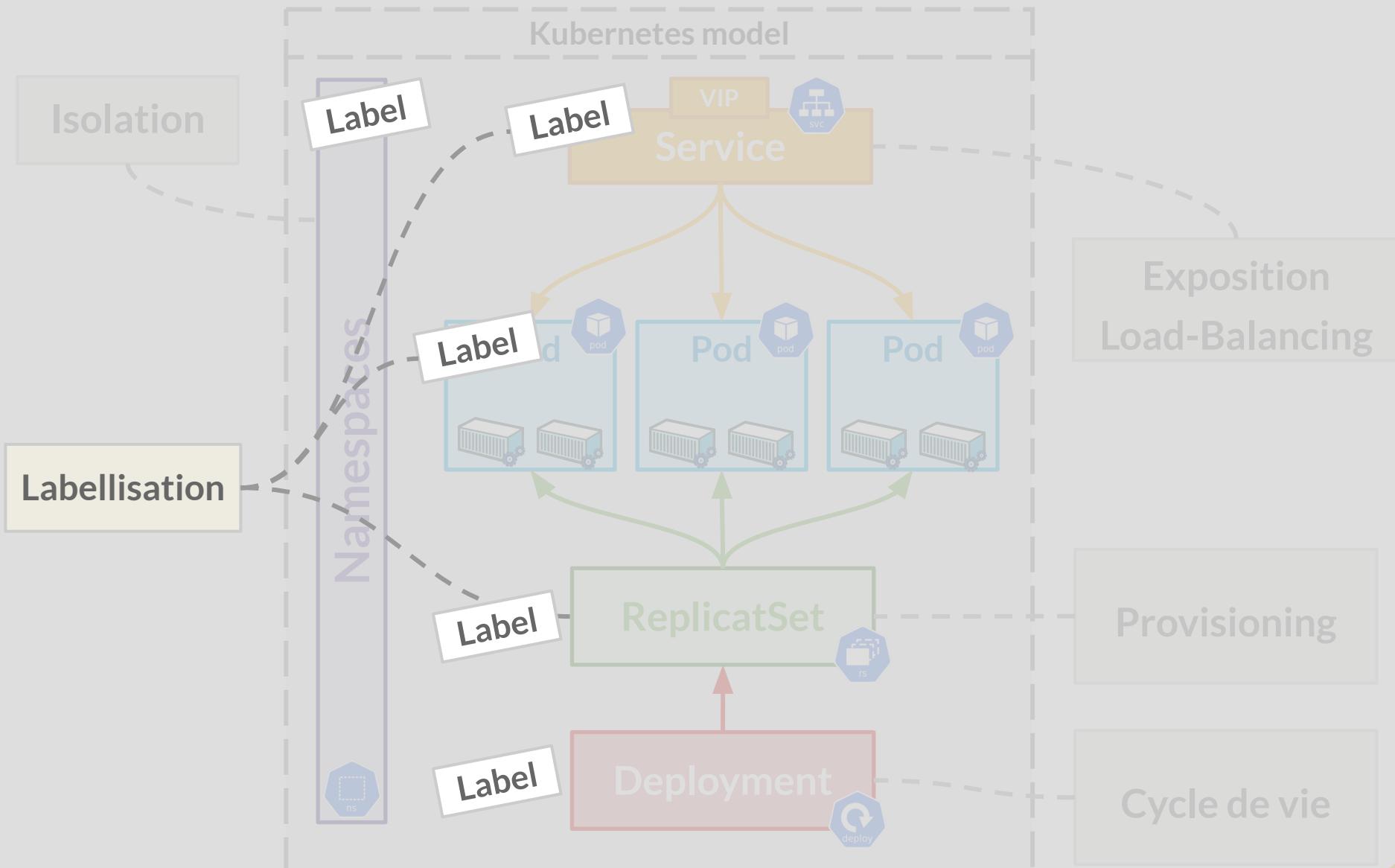
```
$ kubectl explain NOM_DE_LA_RESSOURCE --recursive
```

permet d'obtenir l'ensemble des champs possibles pour la ressource donnée

Les principales ressources K8s



Les Labels



Les Labels et les selectors

- Dans Kubernetes, toutes les ressources créées sont labellisés et labellisables
- Permet d'**associer** de manière souple et libre les **ressources Kubernetes** à des concepts
 - Localisation (dc1, dc2, eu-west...)
 - Environnements logiques (dev, qualif, prod...)
 - Produits / projets (app1, app2...)
 - Caractéristiques techniques (hdd, ssd...)
 - Architecture (front, back...)
 - Organisations (team1, team2...)
- Les labels sont **requêtables** au travers d'une syntaxe, appelé un **selector**
- Les labels et les selectors sont énormément **utilisés** et **nécessaires** au fonctionnement de nombreuses fonctions (exemple : le load-balancing)



Exemples de sélecteurs

- Opérateur de type égalité

```
'disklabel!=ssd'
```

- Opérateur ensembliste

```
'region in (usa, europe)'
```

- Présence / absence d'un label (*Attention au ! et au SHELL, protection par quote*)

```
'my_label'  
'!is_production_ready'
```

- multi critères, séparés par une virgule (&& => tous doivent matcher)

```
'is_backend, dc in (dc1,dc2),disk_type(ssd)'
```



Les Labels et les selectors

- Il est possible de poser un label à la **création** des ressources, mais aussi de modifier les labels en **cours de vie**
- L'option **-l** de `kubectl get` permet d'appliquer un sélecteur sur les ressources pour les filtrer

```
$ kubectl label no/minikube truc=machin trac=plop
node "minikube" labeled

$ kubectl get no -l truc=bidule
No resources found.

$ kubectl get no -l truc=machin
NAME      STATUS    ROLES      AGE      VERSION
minikube  Ready     <none>    44m     v1.8.0

$ kubectl get no -l truc=machin \
-o=custom-columns=NAME:.metadata.name,TRAC:.metadata.labels.trac
NAME      TRAC
minikube  plop
```



Les Labels et les selectors

- L'écrasement d'un label doit être confirmé

```
$ kubectl label no/minikube truc=bidule
error: 'truc' already has a value (machin), and --overwrite is false

$ kubectl label no/minikube --overwrite truc=bidule
node "minikube" labeled
```

- La suppression d'un label se fait en ajoutant un - à la fin du nom du label

```
$ kubectl label no/minikube truc-
node "minikube" labeled

$kubectl get no/minikube -o template \
--template='{{ .metadata.labels.truc }}'
<no value>
```



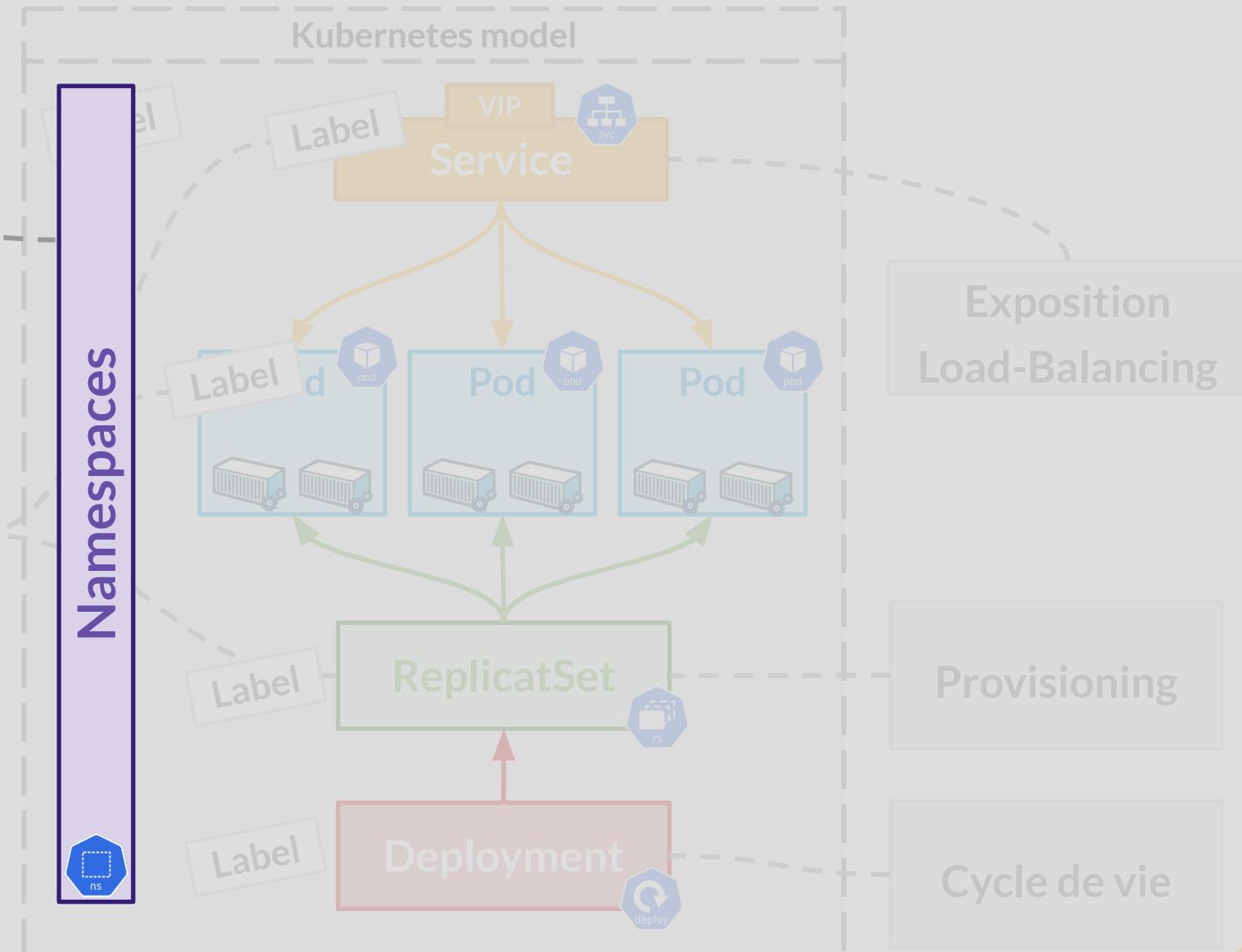
Labels vs. Annotations

- Le concept d'**annotations** est également présents sur tous les objets
- **Comme les labels**, elles permettent d'ajouter des informations descriptives aux ressources
- En général, on les utilise pour
 - Activer des **fonctions expérimentales**
 - Préciser des **comportements spécifiques** du cluster
 - Tracer l'**historique** de certains changements sur les objets
- À la différence des labels, elles ne peuvent pas être utilisées pour filtrer les objets

*Nous aurons l'occasion d'en reparler car le rôle des labels est **majeur** dans K8s...*



Les Namespaces (ns)



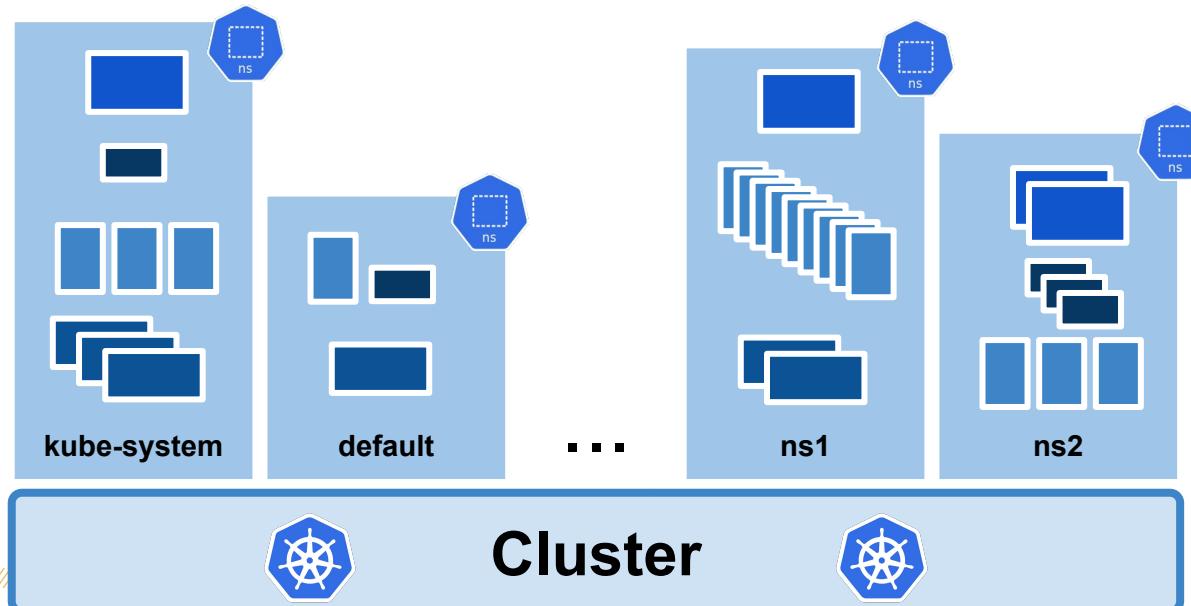
Les Namespaces (ns)

- ▷ **Définition** : l'unité de regroupement des ressources pour représenter des équipes / des projets, des environnements...
- ▷ C'est sur les *namespaces* que se positionnent les **limitations de ressources et quotas**
- ▷ Des objets de même nom dans deux *namespaces* différents ne sont **pas en conflit** et ne se voient pas directement
- ▷ **Les namespaces ne peuvent pas s'imbriquer**
- ▷ Les **contextes** kubeconfig permettent de fixer le namespace à utiliser **par défaut**
- ▷ Des **Règles** peuvent s'appliquer par namespace pour en **restreindre les accès**

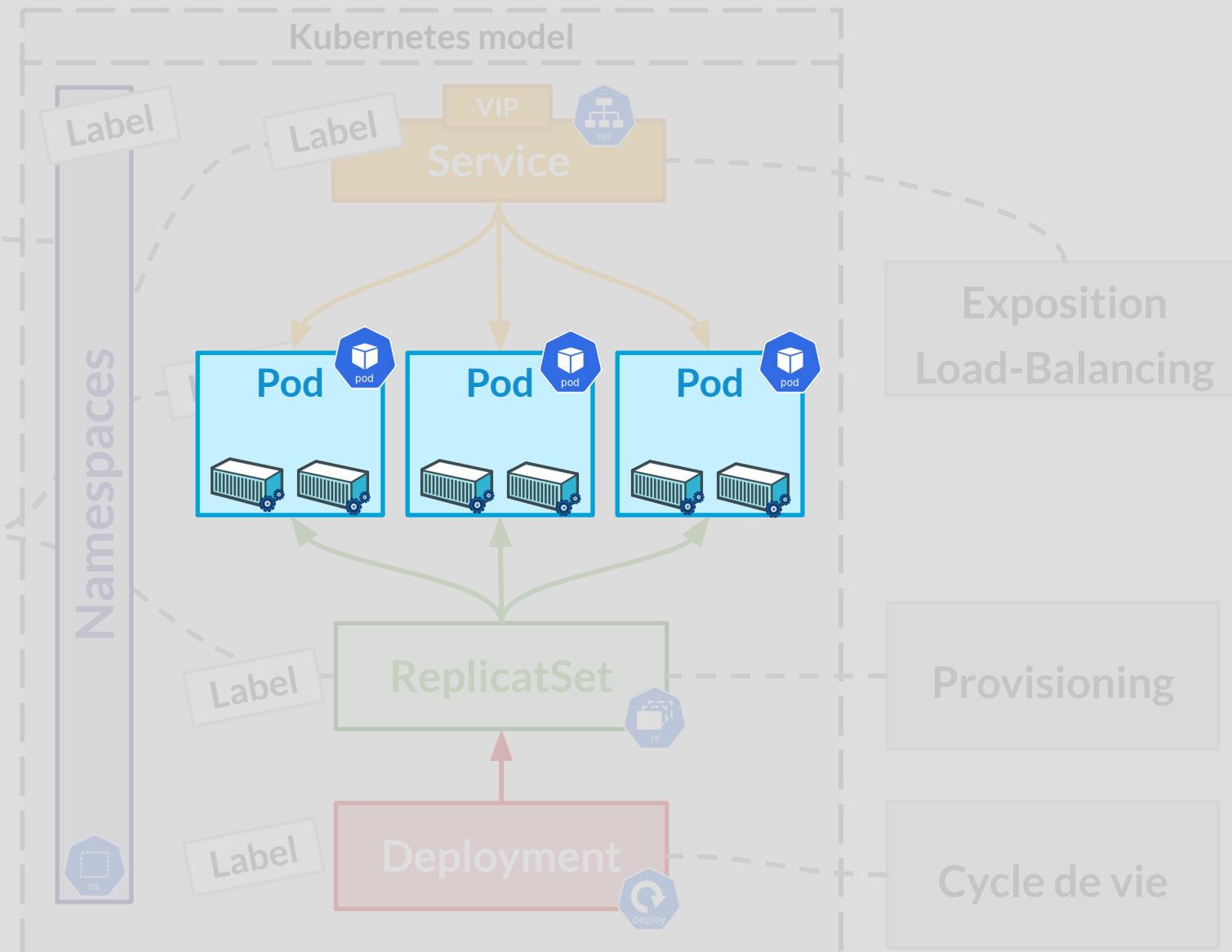


Les Namespaces (ns)

- ▷ Même si ce n'est pas explicitement décrit, (presque) **toutes les ressources** sont dans **un (et un seul) namespace**
 - > Les **nodes** sont une des **exceptions**
 - > Supprimer un **ns** supprime les **ressources** qu'il contient
 - > Une ressource **ne peut pas être déplacée** d'un ns à un autre
- ▷ Dans un cluster Kubernetes, il existe généralement au moins trois namespaces
 - > **kube-public**
 - > **kube-system**
 - > **default**



Les Pods (po)



Les Pods (po)



- *Définition* : un ensemble de conteneurs ayant **un lien logique et une colocationalisation**
- **Une abstraction supplémentaire** au-dessus des conteneurs
- **Objet éphémère**, se construit et se détruit à un coût négligeable
- **Les conteneurs d'un même Pod partagent des composants** comme le réseau (ex: même adresse IP, même boucle locale)
- La plupart du temps en pratique: **1 pod = 1 conteneur**
- En pratique K8s ajoute un conteneur **technique** dans chaque pod. Ce conteneur **technique** porte l'IP. Cette complexité est masquée à l'utilisateur.



Le **pod** est un objet très **technique** qui n'est que très rarement directement manipulé. D'autres concepts de plus haut niveau sont là pour le faire à notre place...



Le **pod** servira des applications très souvent **stateless**, et **sans notion de dépendances** (vis à vis de l'OS sous jacent). Ce que l'on retrouve dans les **2eme et 6eme twelve-factor app**



Exemple de déclaration de Pod

Exemple de pod co-localisant une application et un cache mémoire

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - image: node/app1:v1.0
      name: app1
    - image: memcached
      name: memcached
```

Liste des
conteneurs
du Pod

C'est un exemple
uniquement. Faire ça
ressemble à une
mauvaise idée...



Autres propriétés des Pods

```
apiVersion: v1
kind: Pod
metadata:
  ...
spec:
  containers:
    - image: app:v1
      name: app
      env:
        - name: BIDULE
          value: machine
      imagePullPolicy: IfNotPresent
      ...
    ...
  restartPolicy: Always
  nodeSelector:
    disk: ssd
```

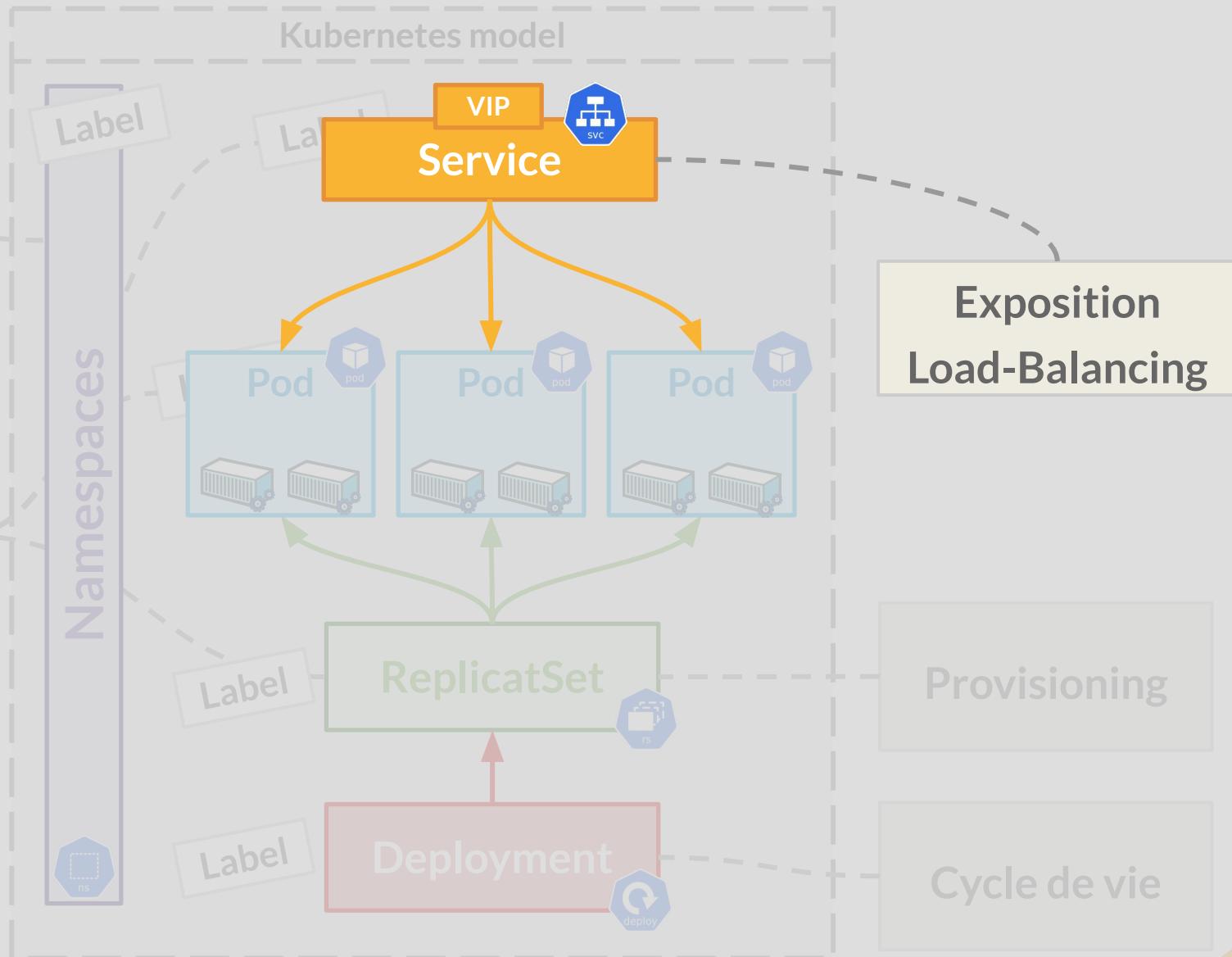
Injecter des variables d'environnement dans mon conteneur

- Always
- IfNotPresent
- Never

- Always
- OnFailure
- Never

Règle de placement sur les nœuds

Les Services (svc)



Les Services (svc)

- ▷ Définition : une interface **nommée** permettant d'accéder à un groupe de pods
- ▷ Le service sert à
 - > **Nommer un groupe de conteneurs**
 - > **Agir en tant que Load-Balancer** devant des Pods
- ▷ Il est **accessible depuis tous les pods du même namespace par son nom DNS court (nginx-svc)**

Exemple d'un service nginx

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
  selector:
    run: nginx
```

Port exposé
par le service

Choix des pods
du service



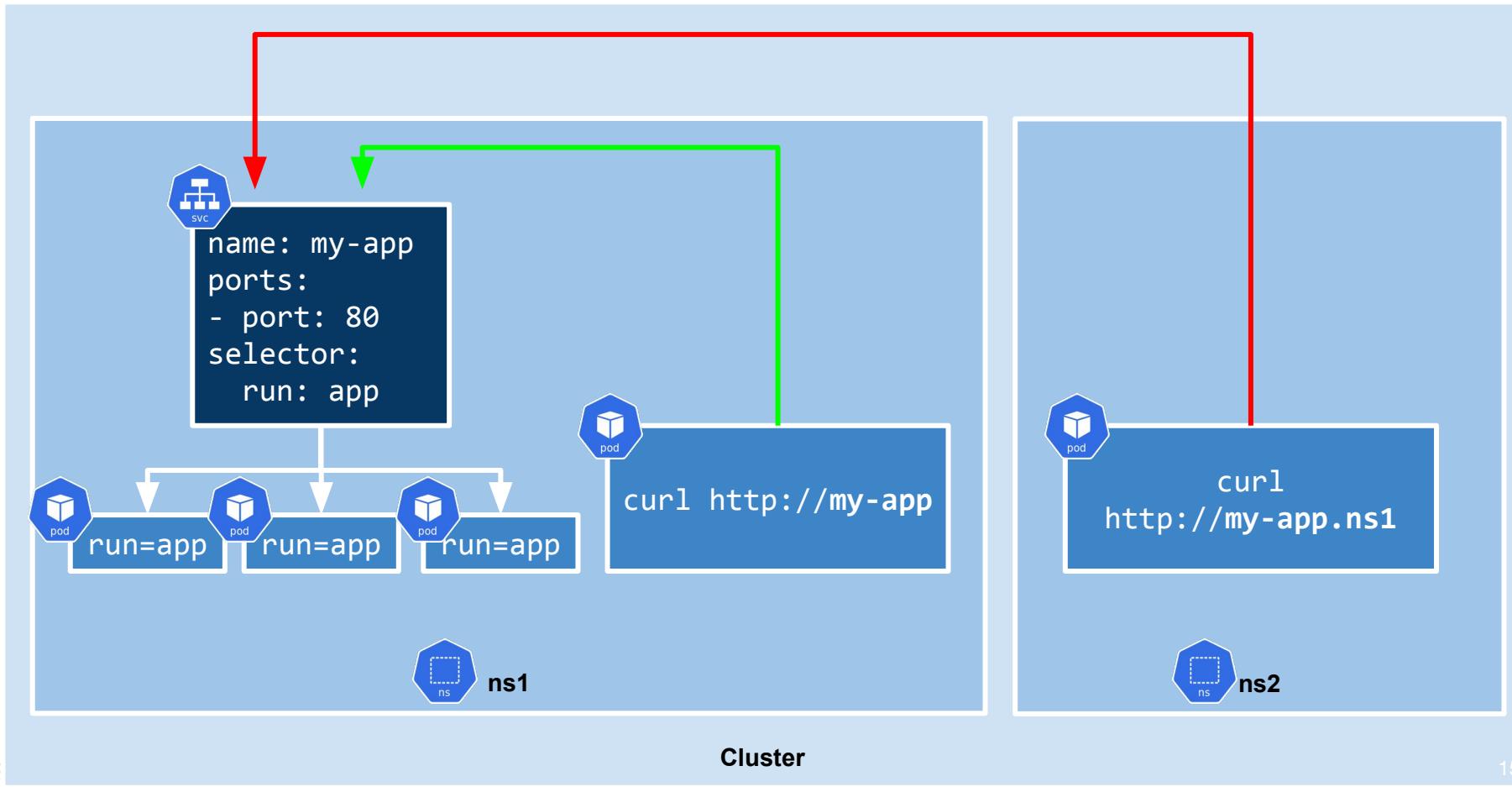
Il s'agit du 7eme des twelve-factor app :
Associations de ports



Les Services et le nommage

- Normalement, un pod ne parle **jamais à un autre pod directement**, il passe par un service qui l'« **expose** »
- Les autres namespaces peuvent résoudre les services des autres ns avec `${svc}.${ns}`

service
pod
namespace
cluster



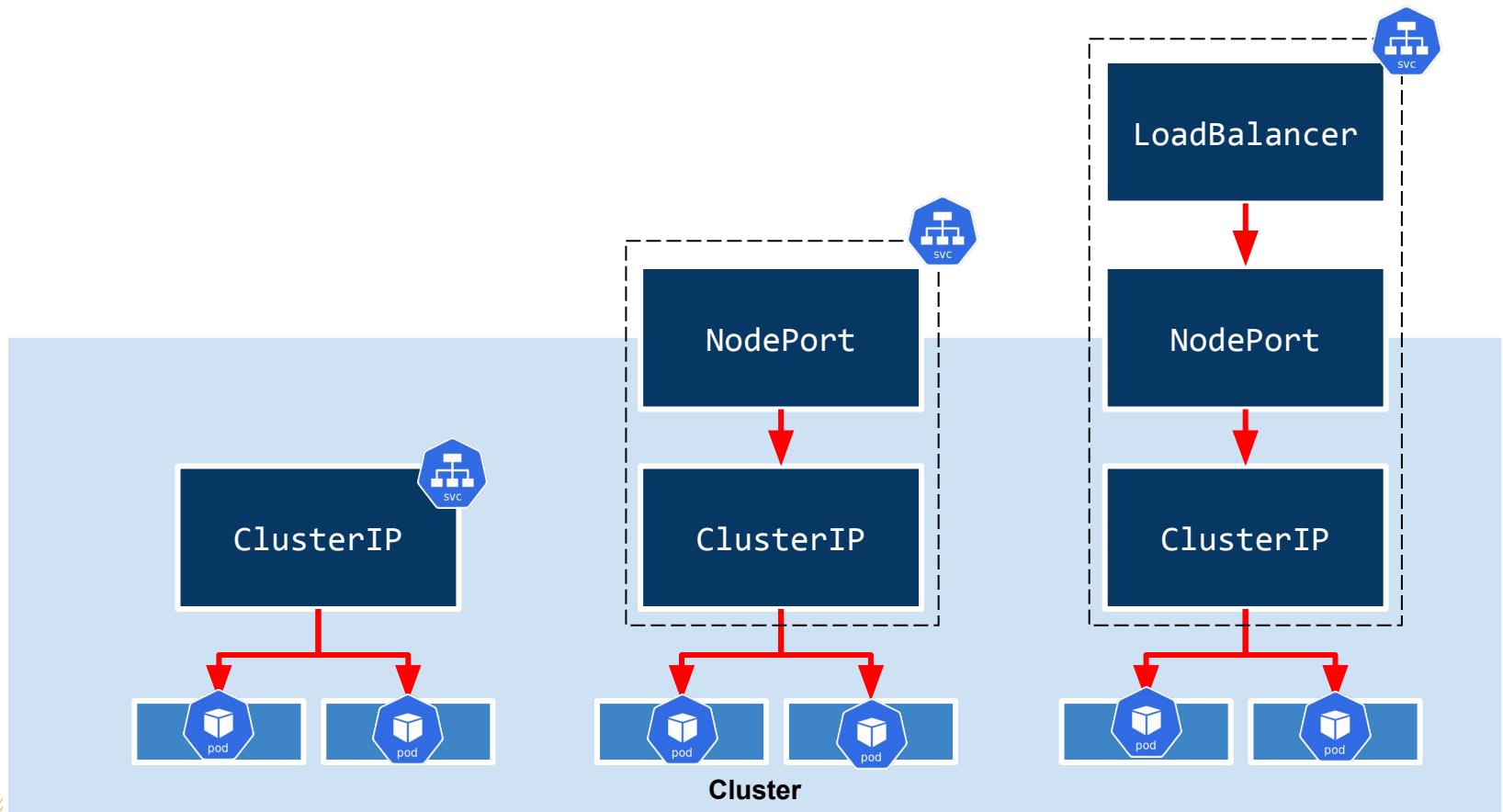
Différents types de services (1/2)

service

pod

cluster

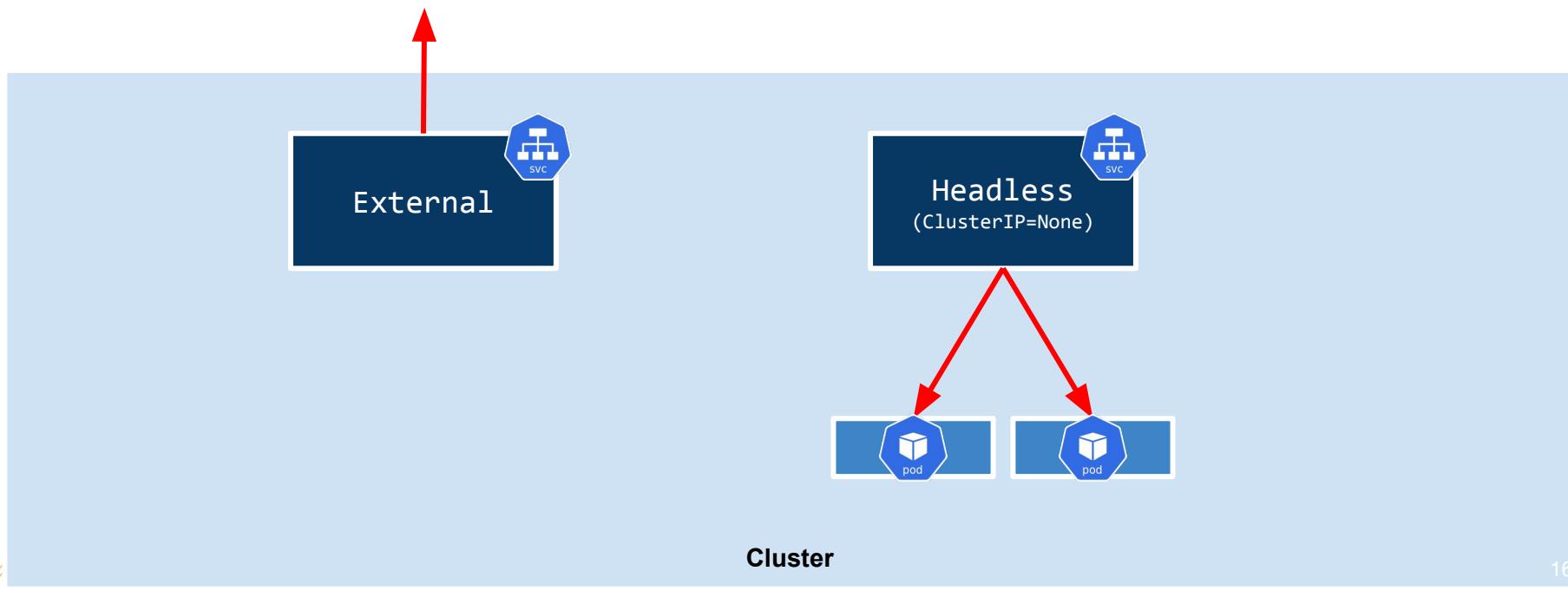
- **ClusterIP (défaut)** : allocation d'une adresse Interne au Cluster, uniquement accessible par d'autres pods
- **NodePort** : allocation d'un port spécifique (par défaut 30000-32767) sur tous les Nodes => permet l'accès par des composants externes au cluster
- **LoadBalancer** => Crée un Load-balancer externe, via le *cloud provider*



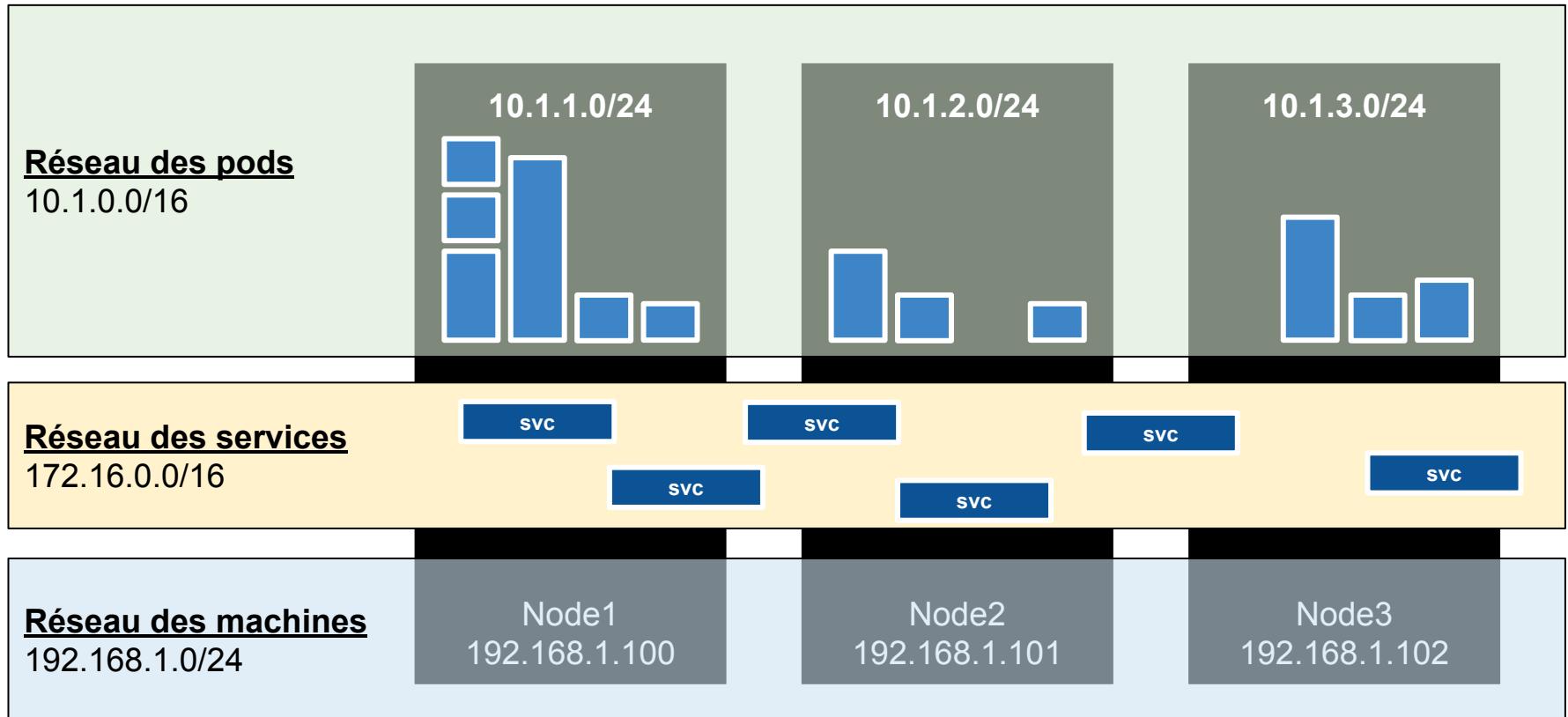
Différents types de services (2/2)

- **External** : pointeur DNS (statique, manuel) vers un service externe
- **ClusterIP (Headless)** : pas d'allocation d'une adresse Interne, simple liste ou *round-robin* DNS vers les pods

service
pod
cluster



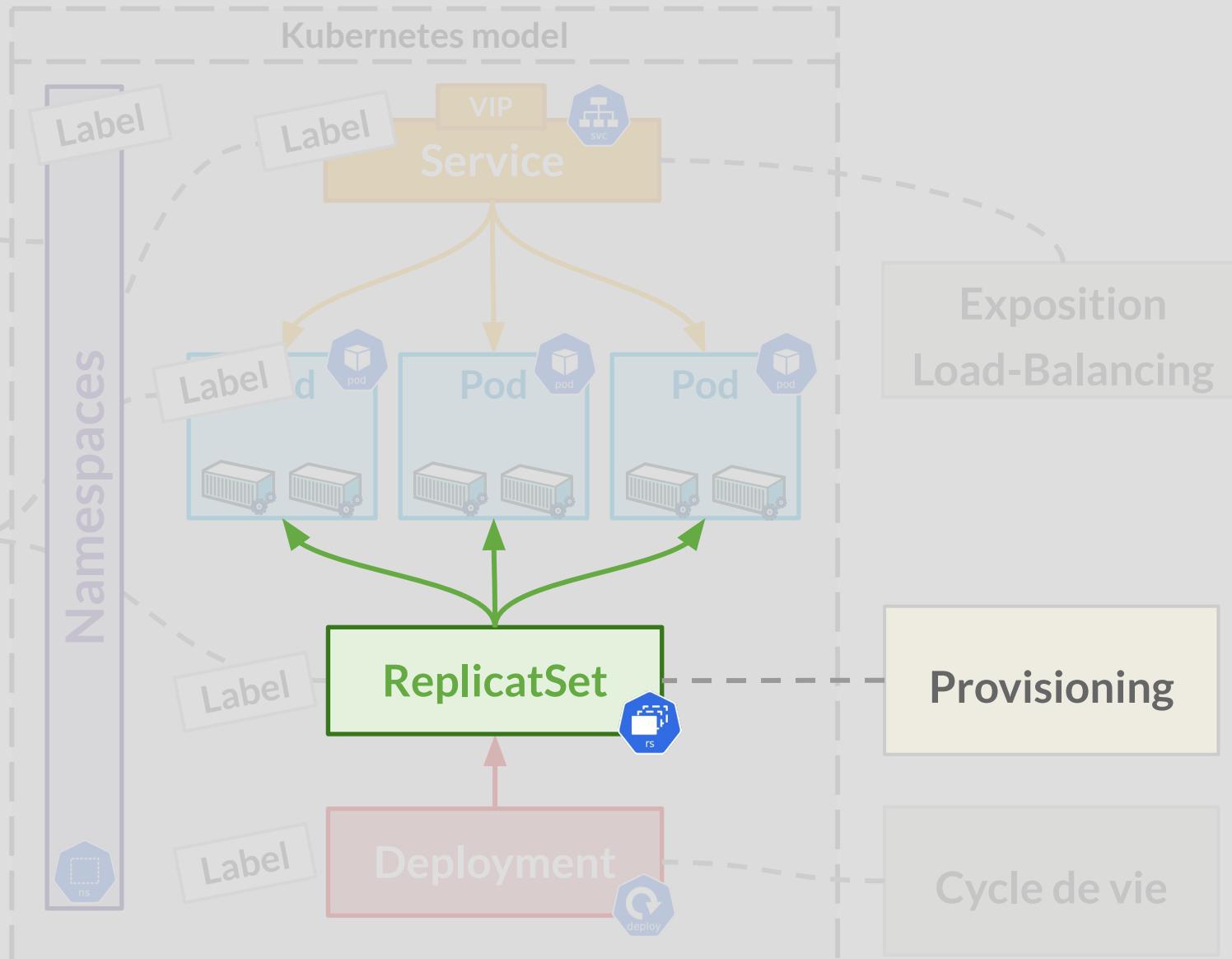
Le modèle réseau de K8s : 3 réseaux cohabitent



Le réseau des machines est le seul à être obligatoirement accessible de l'extérieur d'un cluster. Les réseaux des pods et des services ne le sont généralement pas.



Les ReplicaSets (rs)



Les ReplicaSets (rs)

- Ils garantissent la (re)création des pods en encapsulant la définition d'un ou de plusieurs pod(s)
 - Notion de **template** de pods à instancier
- Ils s'assurent du respect du « **taux de réPLICATION** » **attendu** en re-créant ou supprimant des pods au besoin

```
$ kubectl scale rs/rs1 --replicas=5  
replicaset "rs1" scaled
```



La gestion de **création** ou **recréation** d'un certain nombre de **réPLICAS** suit les principes édictés dans le **8eme et 6eme twelve-factor app: Concurrence**.



Exemple de ReplicaSet

Spécification des pods à créer
(voir exemple pods pour format)

```
apiVersion: v1
kind: ReplicaSet
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
        - image: nginx:1-10
          name: nginx
          ports:
            - containerPort: 80
              protocol: TCP
```

Nombre cible et identification des pods



Les Deployments (deploy)



Kubernetes model

Isolation

Exposition
Load-Balancing

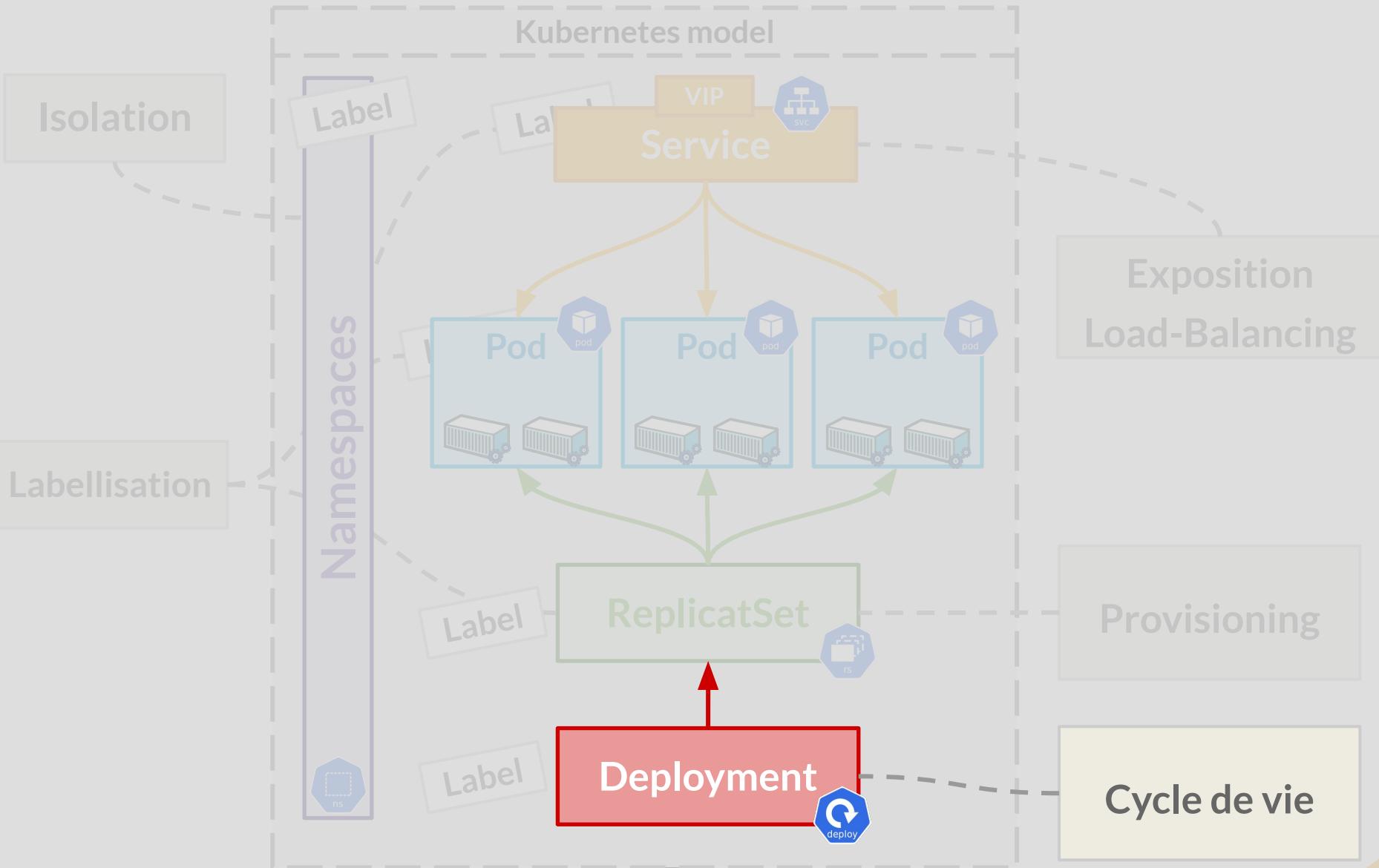
Labellisation

Provisioning

Namespaces

Deployment

Cycle de vie



Les Deployments (deploy)

La manipulation directe des **ReplicaSets**, même si elle est possible, est déconseillée, au profit d'un objet qui l'encapsule : le **Deployment**

Le **Deployment** est la gestion du **cycle de vie** et du **versioning** d'un ReplicaSet

Les **Deployments** peuvent adopter plusieurs stratégies pour les montées de version :

- ▷ **Recreate**
- ▷ **Rolling updates**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: truc
spec:
  replicas: 2
  selector:
    matchLabels:
      run: truc
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        run: truc
    spec:
      containers:
        - image: my_image:v1
          name: truc
```



Mécanisme du rolling-update : étape 1

SVC

po

ns

rs

deploy

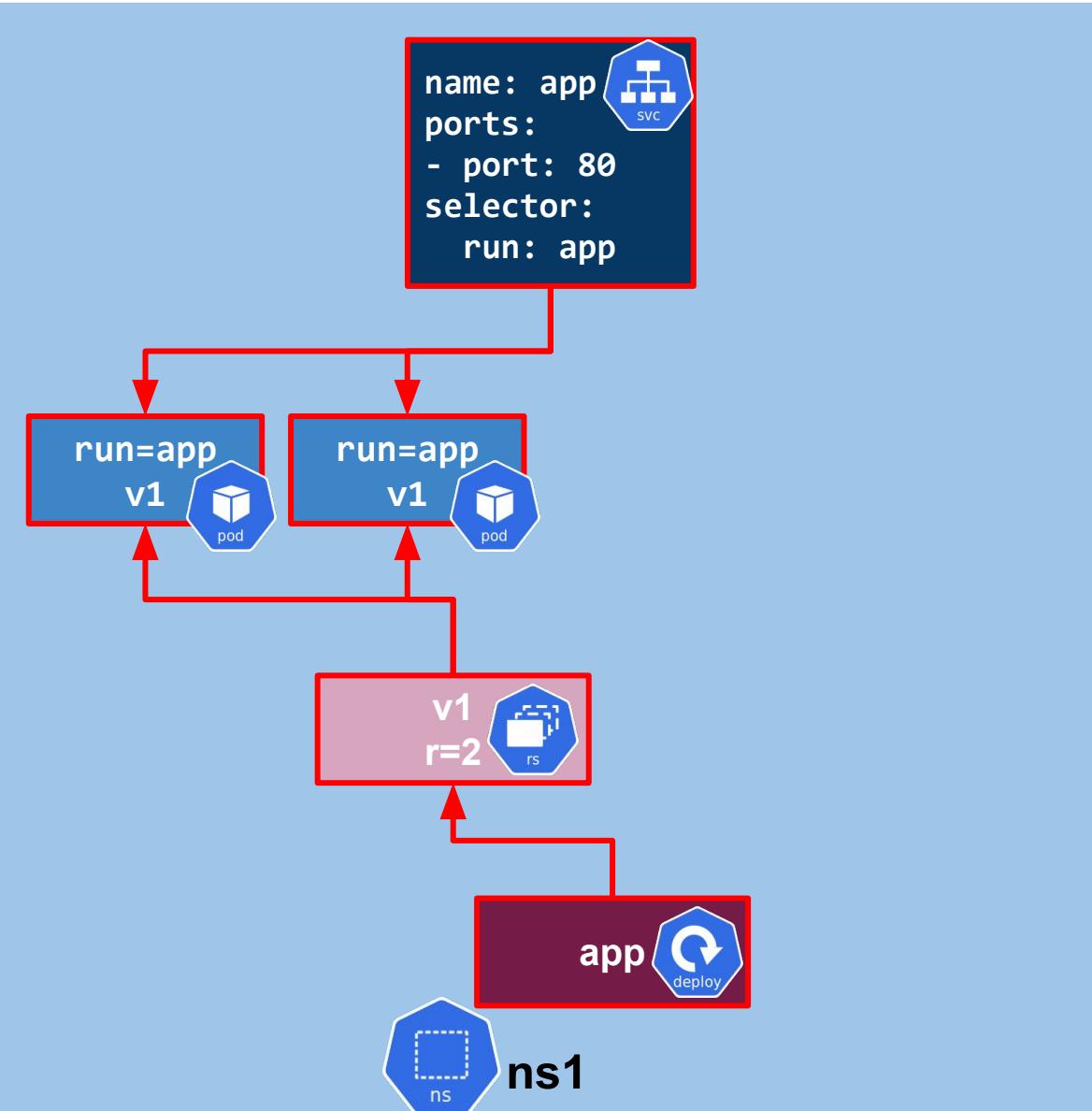


ns1

```
$ kubectl run  
--image=img:v1 \  
-r=2 \  
--expose \  
--port=80 \  
app
```



Mécanisme du rolling-update : étape 2



SVC

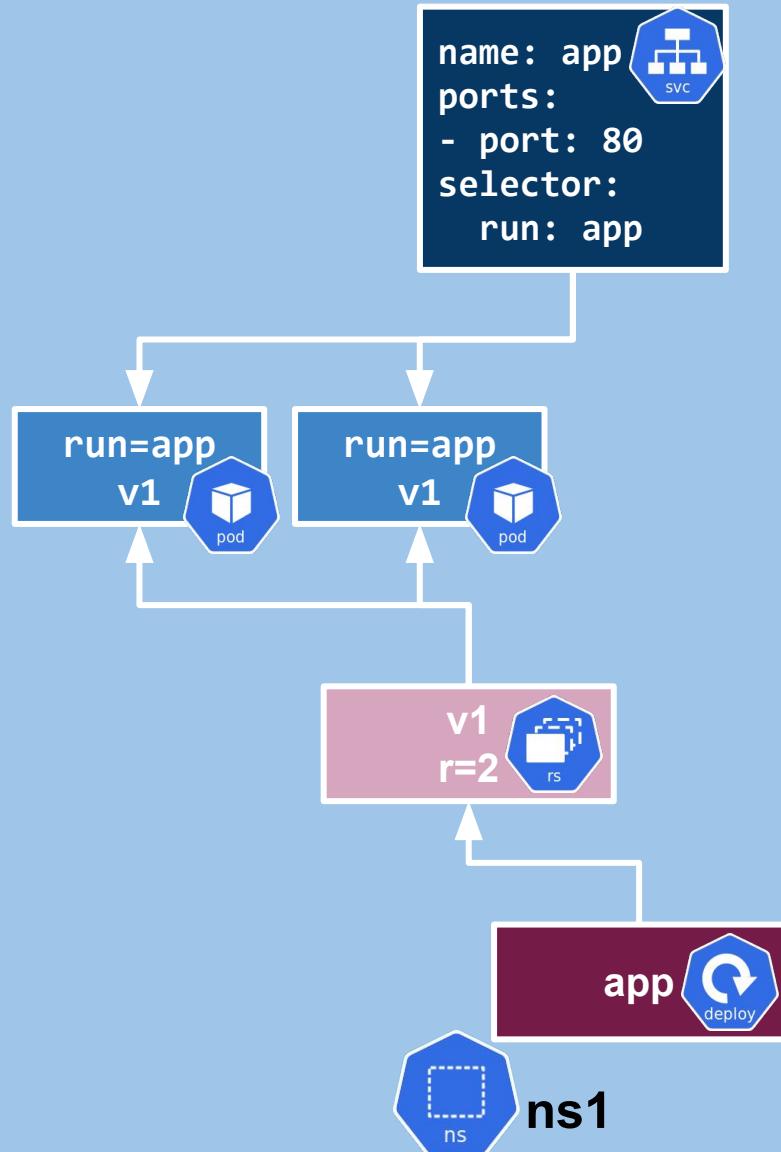
po

ns

rs

deploy

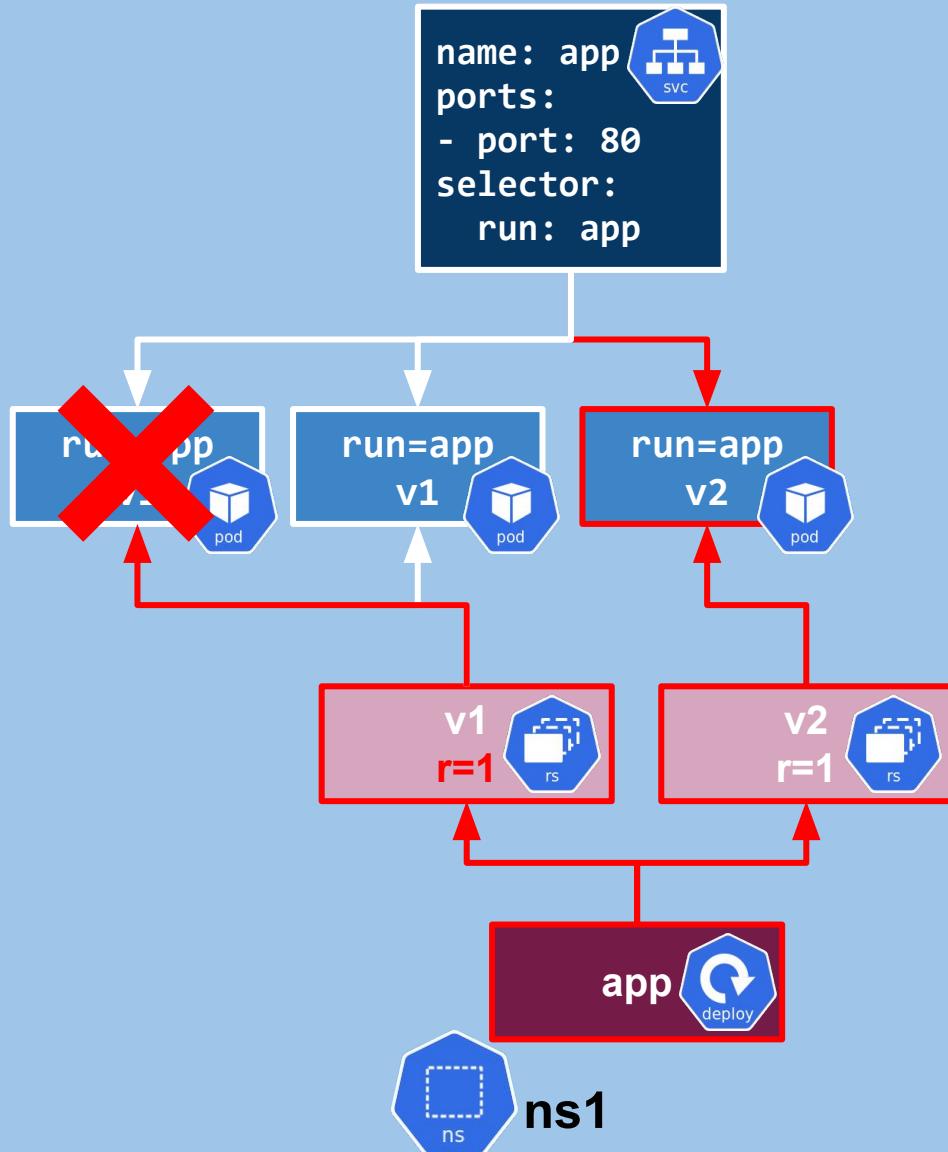
Mécanisme du rolling-update : étape 3



```
$ kubectl set image \
  deploy/app \
  app=img:v2
```

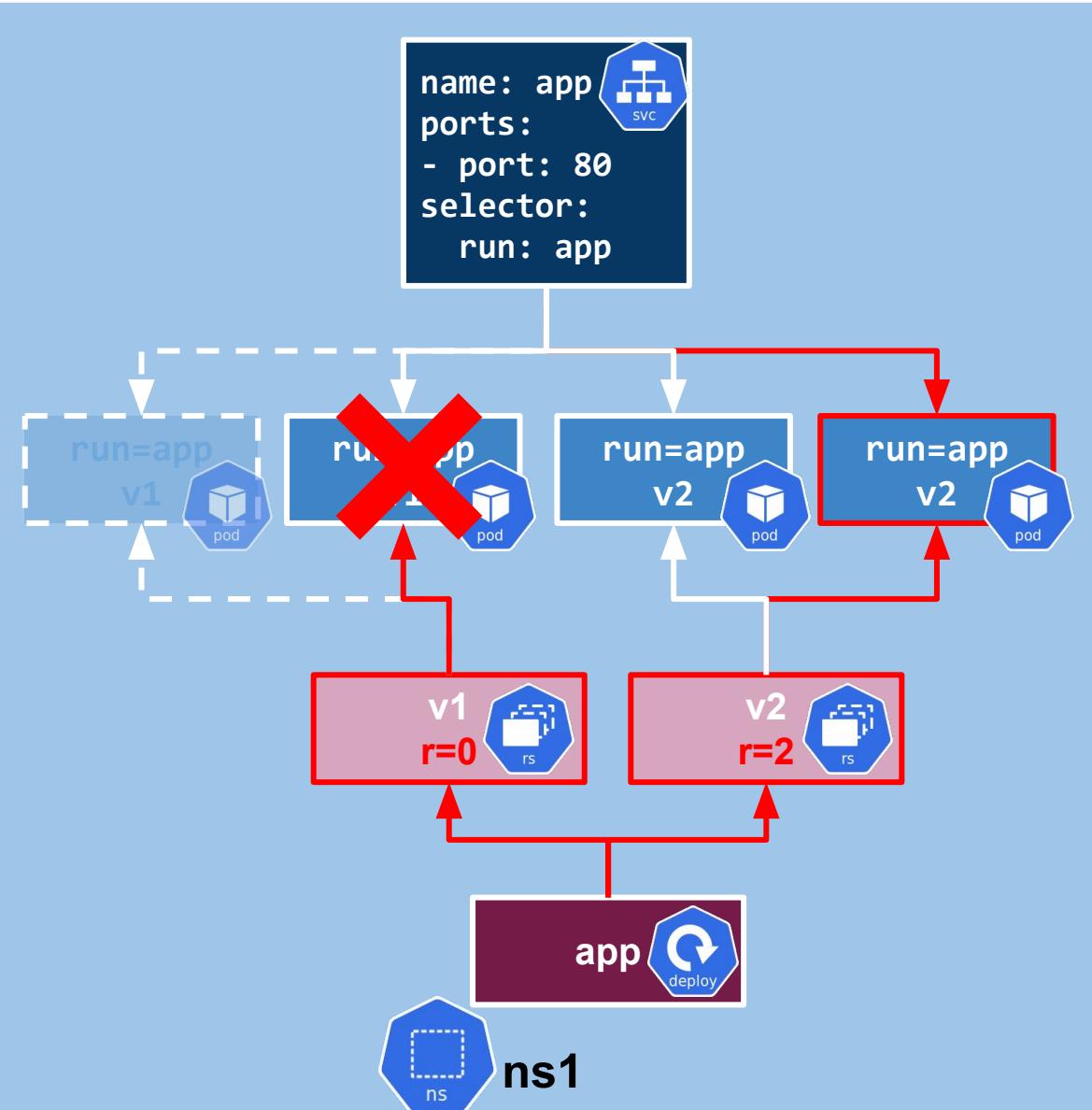


Mécanisme du rolling-update : étape 4



Note : le **svc** pointe sur des **pods** qui ont été provisionnés par **deux rs**. C'est là que la magie des labels / sélecteurs opère...

Mécanisme du rolling-update : étape 5



SVC

po

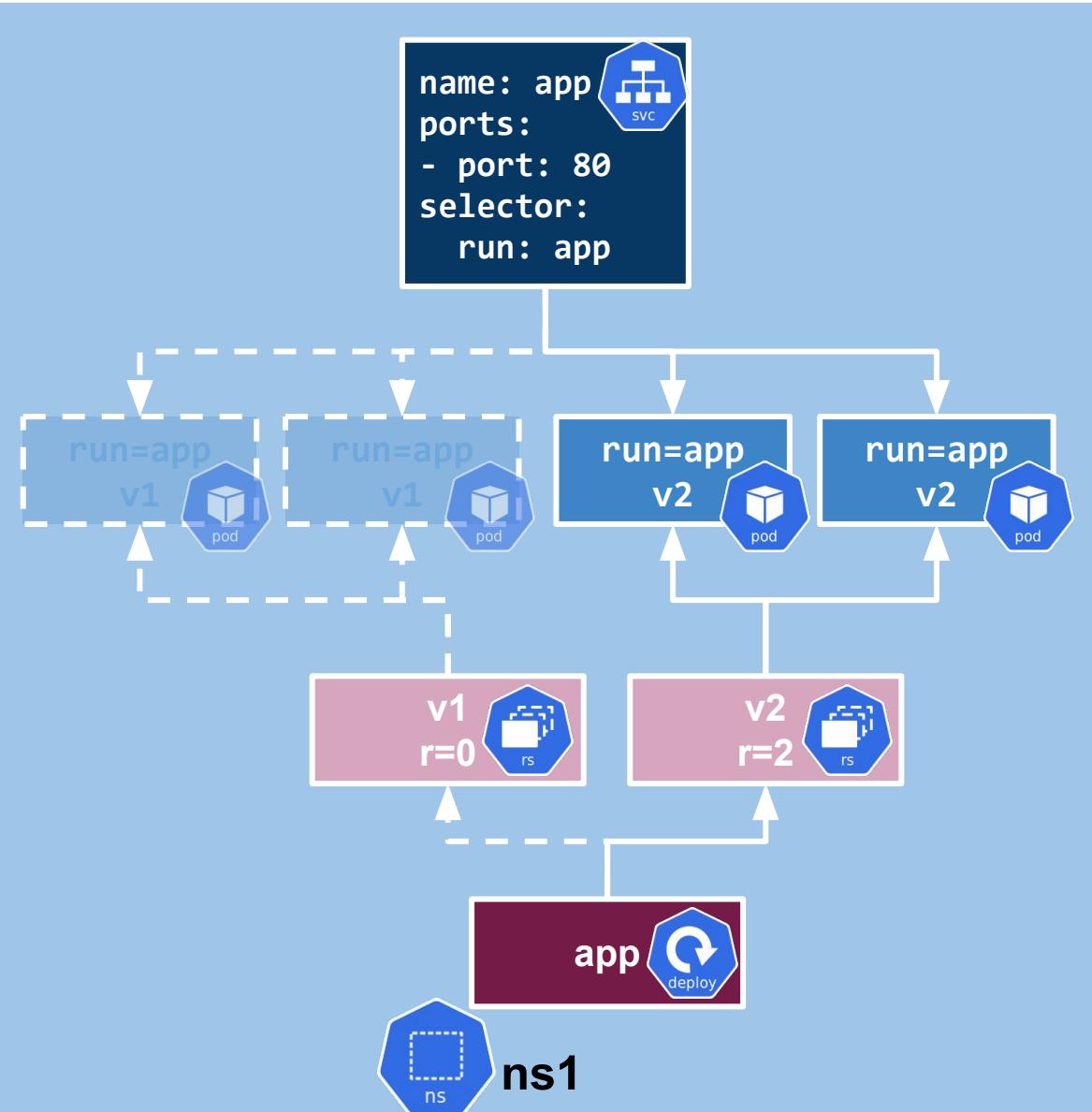
ns

rs

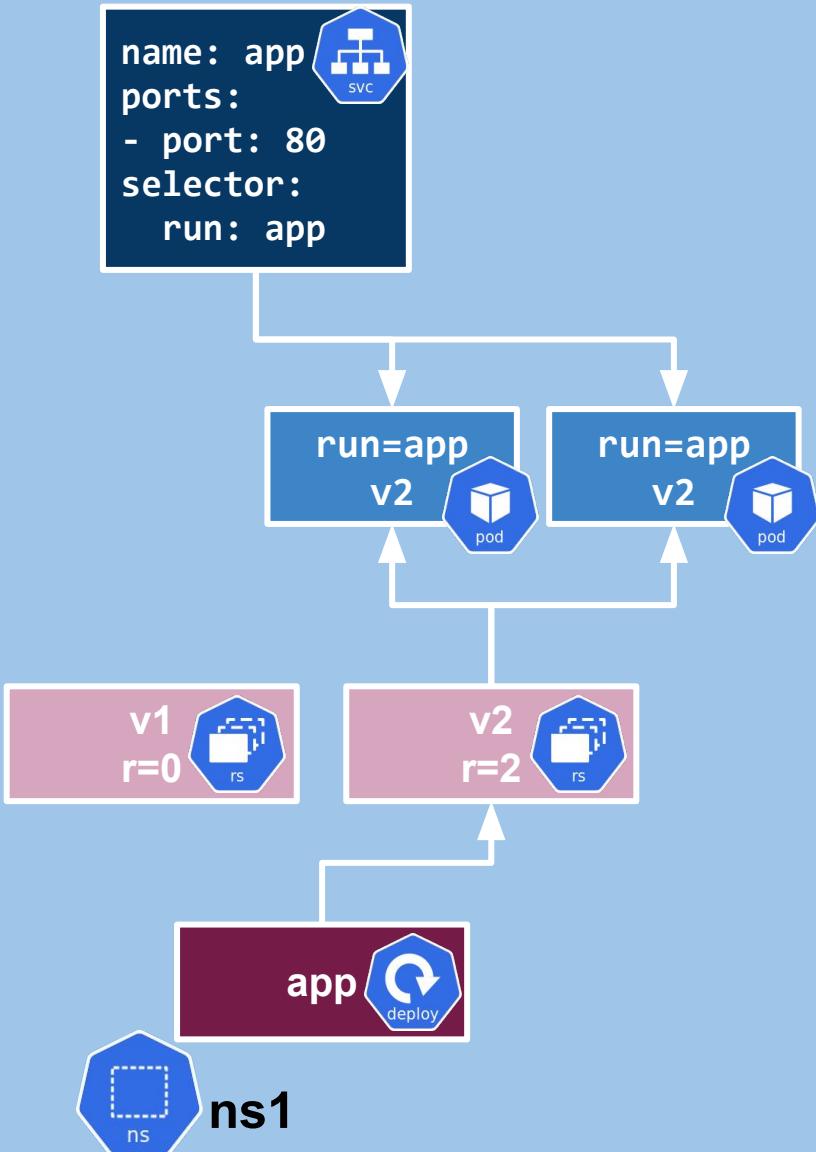
deploy



Mécanisme du rolling-update : étape 6



Mécanisme du rolling-update : étape 7



Note : le rs de la v1 reste présent, il permet d'effectuer rapidement un rollback.

Voir la propriété
revisionHistoryLimit

Les ReplicaSets et les Deployments

- Voir l'historique des versions d'un Deployment

```
$ kubectl rollout history deploy/app  
deployments "app"  
REVISION  CHANGE-CAUSE  
1          kubectl run app --image=nginx:1.12 --replicas=2 --expose=true --port=80 --record=true  
2          kubectl set image deploy/app app=nginx:1.13 --record=true
```

- Faire un rollback sur une version précédente

```
$ kubectl rollout undo deploy/app --to-revision=1
```

- Mettre en pause / reprendre un changement de version d'un **ReplicaSet**

```
$ kubectl rollout (pause|resume) deploy/app
```



Manipulation avancée des ressources

Pour l'instant, nous avons utilisé kubectl principalement sous sa forme qui masque la structure des ressources

- ▷ `kubectl run`
- ▷ `kubectl set`
- ▷ `kubectl scale`
- ▷ ...

Mais il est très souvent (tout le temps en fait) nécessaire d'être beaucoup **plus fin** dans la définition ou la **gestion des ressources**

Nous allons voir comment manipuler les ressources sous forme de **fichiers (JSON, YAML)**



Utilisation de kubectl : manipulation avancée des ressources

- ▷ Tricher pour avoir un squelette de fichier à adapter

```
$ kubectl run ... --dry-run -o yaml > ressource.yaml
```

- ▷ Création d'objet à partir d'un fichier (ou d'un répertoire)

```
$ kubectl create -f ressource.(yaml|json)
```

- ▷ En mode « idempotence »

```
$ kubectl apply -f ressource.(yaml|json)
```

- ▷ En mode interactif (lance vim) => jamais en prod !!

```
$ kubectl edit type/ma_ressource
```

- ▷ En mode différentiel

```
$ kubectl patch -f ressource.(yaml|json)
```

- ▷ Suppression d'un objet à partir d'un fichier (ou d'un répertoire)

```
$ kubectl delete -f ressource.(yaml|json)
```



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Secrets et les ConfigMaps (cm)



- **Rôle : Distribuer les données (éventuellement sensibles)** (configuration, clés ssh, certificats, login / mot de passe...)
 - Pour de la **configuration technique** (connexion à une registry, certificats TLS...)
 - Pour les mettre à disposition des **applications**
- Les secrets sont déclarés dans Kubernetes qui fait office de **coffre-fort à secrets**
- Les **ConfigMap** sont gérées à l'identique, mais ont pour vocation à présenter des données de configuration non sensibles



Secrets et les ConfigMaps (cm)

- ▷ Ils peuvent être **mis à disposition des Pods sous forme de**
 - > **Fichiers** dans des volumes spéciaux montés dans les pods
 - > **Variables d'environnement**
- ▷ **La mise à jour** d'un secret (ou d'un cm)
 - > sous forme de **variables d'environnement** nécessite le **redéploiement** des conteneurs qui l'utilisent
 - > sous forme de **fichiers** va se faire automatiquement, mais peut prendre plusieurs dizaines de secondes



Le **3eme** des **twelve-factor app : configuration** peut être appliqué avec les les **Secrets** et les **ConfigMaps**.



Déclaration et utilisation d'un secret dans un fichier

Déclaration d'un secret

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
data:
  password: YW16NGV2ZXI=
  username: YW16
type: Opaque
```

Secrets encodés
en base64

Utilisation d'un secret dans un Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: app:v3.4
      volumeMounts:
        - name: db-creds
          mountPath: "/etc/db-creds"
          readOnly: true
      volumes:
        - name: db-creds
          secret:
            secretName: db-credentials
```

Accès au secret depuis le Pod

```
$ cat /etc/db-creds/password
P@ssw0rd
```



Un ConfigMap dans une variable d'environnement

Déclaration d'un ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config
data:
  log_level: INFO
```

Utilisation d'un cm dans un Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: app:v3.2
      env:
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: env-config
              key: log_level
```

Accès au cm depuis le Pod

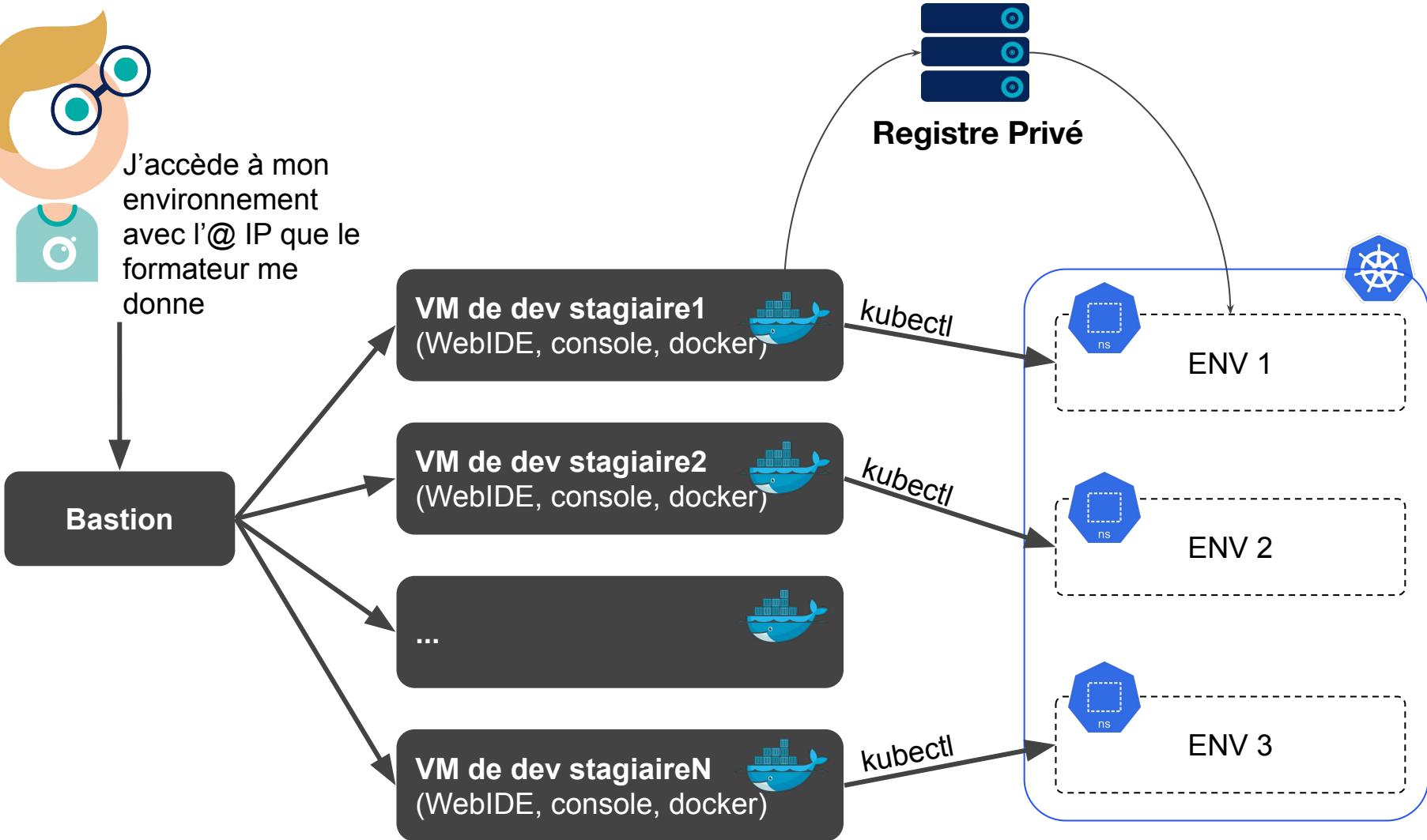
```
$ echo $LOG_LEVEL
INFO
```



Un mot sur les environnements



J'accède à mon environnement avec l'@ IP que le formateur me donne



TP#4

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ **Liveness et Readiness**
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Autres propriétés des Pods : les probes

```
apiVersion: v1
kind: Pod
metadata:
...
spec:
  containers:
    - image: app:v1
      name: app
      livenessProbe:
      readinessProbe:
        initialDelaySeconds: 8
        periodSeconds: 1
```

httpGet:
path: /healthz
port: 8080

tcpSocket:
port: 8080

exec:
command:
- cat
- /tmp/healthy



Autres propriétés des Pods : Container Lifecycle Hook

- ▶ 3 modes possibles
 - > **InitContainers** : Exécution de commandes dans une autre image avant l'initialisation du pod
 - > **PostStart** : Exécution d'une commande au démarrage du pod
 - > **PreStop** : Exécuter avant la terminaison du pod

```
apiVersion: v1
kind: Pod
metadata:
...
spec:
  containers:
  - name: nginx
    image: nginx
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo Start"]
      preStop:
        exec:
          command: ["/usr/sbin/nginx", "-s", "quit"]
```



Par nature, un pod a un cycle de vie très court. On le considère comme **jetable** et l'application conteneurisée doit pouvoir être **éteinte gracieusement**. C'est le **9eme des twelve-factor app.**



Autres propriétés des Pods : Container Lifecycle Hook

► Exemple d'un InitContainer

```
apiVersion: v1
kind: Pod
metadata:
  ...
spec:
  containers:
    - name: myapp-container
      image: busybox
      command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
    - name: init-myservice
      image: busybox
      command: ['sh', '-c', 'until nslookup myservice; do echo waiting for
myservice; sleep 2; done;']
```



Le 12^{eme} des twelve-factor app: Processus d'administration peut être appliqué avec les InitContainer. Par exemple pour lancer les migration de base de données.



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ **Routes HTTP** TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Les Ingresses (ing)



- ▷ Les Ingresses définissent des **routes** (ou **règles de routage**) HTTP(s) vers des services K8s
- ▷ Un peu comme les services, mais au niveau L7 - HTTP => met en place des règles dans un **Reverse Proxy** managé **dynamiquement** par K8s
- ▷ Une ingress possède plusieurs informations principales
 - Le **vhost** à router (www.example.com)
 - Le **chemin** (path) à router (/ , /myapp...)
 - Le **service** vers lequel router le trafic => le **backend** (myservice, port 8080)
 - Optionnellement, des informations pour le **TLS** (certificats + clé privée, via un secret)



Les Ingresses (ing), exemple (1/2)

```
---  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: grafana  
  namespace: monitoring  
spec:  
  rules:  
    - host: grafana.trololo.com  
      http:  
        paths:  
          - path: /  
            backend:  
              serviceName: grafana  
              servicePort: 3000
```

« Tout le trafic à destination de **grafana.trololo.com** est envoyé sur le port **3000** du service **grafana** »

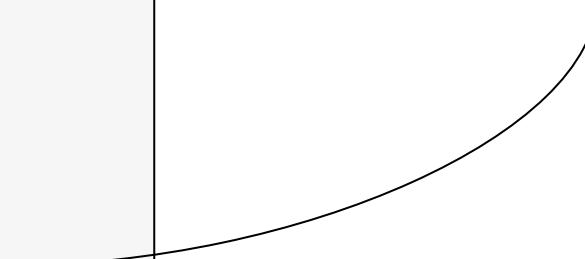


Les Ingresses (ing), exemple (2/2)

```
---  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: grafana  
  namespace: monitoring  
spec:  
  tls:  
    - secretName: ssl-ingress  
  rules:  
    - host: grafana.trololo.com  
      http:  
        paths:  
          - path: /  
            backend:  
              serviceName: grafana  
              servicePort: 3000
```

Création d'un **Secret** à partir du certificat

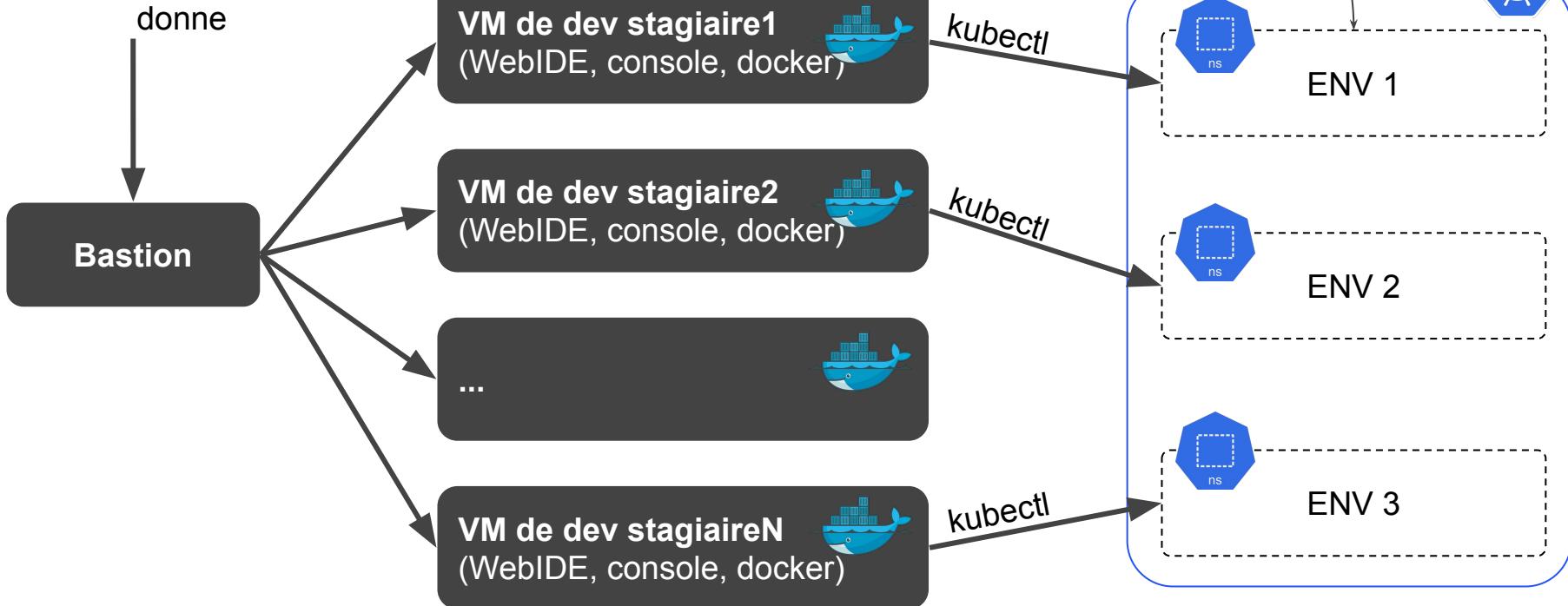
```
$ kubectl create secret tls ssl-ingress \  
--cert=path/to/cert/file \  
--key=path/to/key/file
```



Un mot sur les environnements



J'accède à mon environnement avec l'@ IP que le formateur me donne



TP#5

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ **Maîtrise des capacités**
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Maîtrise des capacités

La maîtrise des capacités repose sur :

- ▷ La mise à disposition de **capacités informatiques** du cluster : l'**offre**
 - > l'unité de *cpu*
 - > *memory*
 - > Les **PersistentVolumeClaims**
 - > Du stockage local pour les pods (expérimental)
- ▷ L'expression de **demandes** et de **contraintes** à l'allocation des conteneurs
 - > **requests** : le minimum dont un pod a besoin pour fonctionner
 - > **limits** : ce qu'il s'engage à ne pas dépasser (et que le cluster contrôle)
- ▷ Le positionnement de **quotas de ressources** et de **limites** sur des **namespaces**

Ces limitations au sein de vos namespaces sont **gérées par les administrateurs** de votre cluster.



Impacts des requests et limites sur la vie d'un pod

- ▶ les **requests** impactent avant tout le choix du **node** au moment du **placement** du pod
- ▶ Les **limits** impactent le **bridage** du **CPU** et le risque de se faire **tuer** en cas de dépassement de la mémoire

Le fait de positionner des **limits** de CPU est un prérequis à la mise en œuvre de l'autoscaling de pods au travers de HPA ou **HorizontalPodAutoscaler**

```
$ kubectl run plop --image=plop \
  --requests=cpu=50m,memory=256Mi \
  --limits=cpu=200m,memory=256Mi
[...]
$ kubectl autoscale deploy/plop \
  --min=2 \
  --max=5 \
  --cpu-percent=80
```



Mettre en place des quotas et des limites sur les namespaces

Plusieurs types de limitations sont possibles sur les namespaces

- ▶ **Les ResourceQuotas (quota)**
 - Quotas agrégés : somme des **requests** et des **limits** de tous les pods du namespace
 - Quotas sur les **nombres d'objets** instanciables dans le namespace
- ▶ **Les LimitRanges (limits)**
 - Bornes **min** et **max** (et valeurs par **défaut**) par **requests** et par **limits**



À partir du moment où des **ResourceQuotas** ont été positionnés par vos administrateurs, il devient impossible de créer des ressources sans **requests** et/ou **limits** sur les capacités concernées.



L'utilisation de **LimitRange** permet de mettre des valeurs par **défaut** et peut simplifier la vie des développeurs.



Suivre sa consommation dans un namespace

```
$ kubectl describe ns kubernetes-application
Name:           kubernetes-application
Labels:         app=python
Annotations:    <none>
Status:        Active

Resource Quotas
Name:          test-rq
Resource      Used   Hard
-----
configmaps    0       10
limits.cpu    0       20
limits.memory 0       20Gi
persistentvolumeclaims 0       4
pods          2       40
replicationcontrollers 0       20
requests.cpu   0       10
requests.memory 0       10Gi
secrets        2       10
services       0       10
services.loadbalancers 0       0
services.nodeports 0       2

No resource limits.
```



En fonction des droits que vous possédez, vous pourrez peut-être voir les détails des quotas qui s'appliquent en regardant les **ResourceQuota** et **LimitRanges** présents dans votre namespace



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ **Monitoring applicatif** TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Monitoring de K8s

- ▷ Le monitoring de K8s est primordial car
 - > La plateforme repose sur de **nombreux composants**
 - > On ne part pas en prod sans monitoring
 - > Il est nécessaire pour certaines **fonctionnalités** du cluster
 - Afficher des **graphes** dans le **dashboard**
 - Permettre l'**autoscaling**
- ▷ Le monitoring doit s'effectuer à **tous les niveaux**
 - > **Matériel**
 - > **Système**
 - > **Cluster** d'un point de vue **technique** (services K8s)
 - > **Composants logiques** du cluster (pods, applications...)

C'est cette partie qui nous intéresse !!

Beaucoup de solutions existent

- S'appuyer sur l'existant vs. nouvelle implémentation
- On-prem vs. SaaS

Nous proposons ici **quelques principes** et une **implémentation type**



Quelques principes sur le monitoring

- ▷ Vers la généralisation des URLs de métriques au format **OpenMetrics**

```
$ curl http://localhost:8080/metrics
```

```
# HELP rest_client_request_status_codes Number of http requests, partitioned by metadata
# TYPE rest_client_request_status_codes counter
rest_client_request_status_codes{code="200",host="10.240.0.10:6443",method="DELETE"} 3
rest_client_request_status_codes{code="200",host="10.240.0.10:6443",method="GET"} 2758
rest_client_request_status_codes{code="200",host="10.240.0.10:6443",method="PATCH"} 3
rest_client_request_status_codes{code="200",host="10.240.0.10:6443",method="PUT"} 1305
rest_client_request_status_codes{code="201",host="10.240.0.10:6443",method="POST"} 58
```

Works with **K8s**



Works with your **OS** [prometheus / node_exporter](#)

Works with **Etcd**

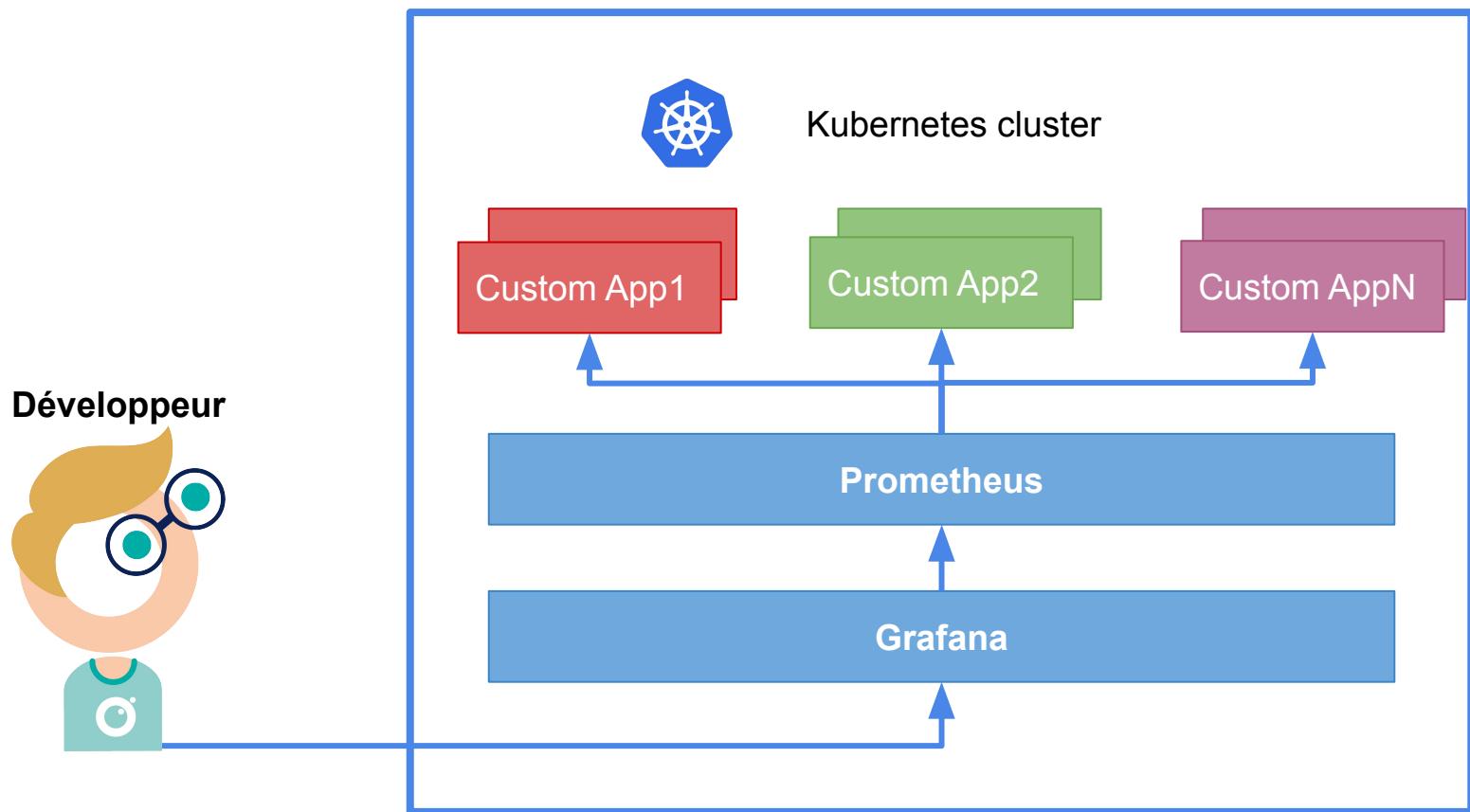


Works with **your app** [Acme](#)



Architecture générale de Monitoring (Suggestion)

- ▶ **Prometheus** est une **TSDB (Time Series DataBase)** qui a pour rôle de collecter (scrapper) les métriques exposée notamment par les applications (et bientôt la vôtre !)
- ▶ **Grafana** est une web UI qui permet de structurer l'affichage de vos données sous forme de dashboard



Prometheus : Un moteur de gestion orienté TSDB (Time Series DataBase)

- Collecte via les URLs de métriques
- Stockage
 - Moyen terme (6h - 15j)
 - Push possible sur d'autres backends
- Requêteage : PromQL
- Alerting
 - Sur base de requêtes PromQL
 - Plusieurs vecteurs de sortie
- Présentation
 - Web UI simple pour mettre au point ses requêtes
- Configuration
 - Statique (fichier de conf)
 - Dynamique (**K8s**, Consul...)

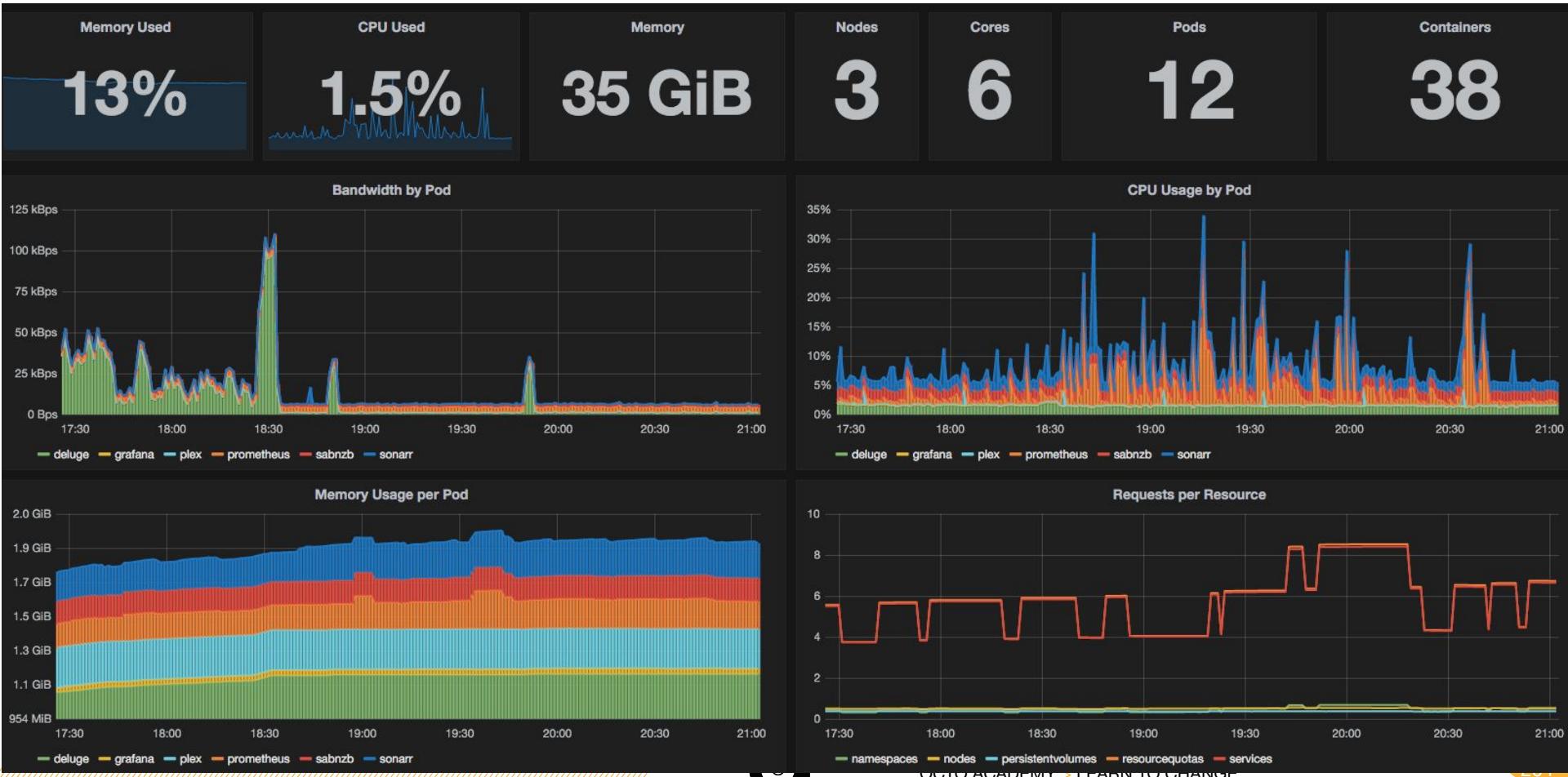


L'intégration Prometheus/K8s ajoute des labels spécifiques à K8s aux métriques (**ns**, **pod**...)



Grafana

- Le dashboard **multi-technologies** de référence
- De nombreux dashboards communautaires disponibles pour Docker, K8s...
- Configurable par **API**
- Peut aussi implémenter des **alertes**



Comment collecter ses métriques applicatives ?

Il existe **beaucoup de solutions** plus ou moins élégantes pour configurer Prometheus en vue de collecter vos métriques

- ▷ Configuration **manuelle** (par les admins)
- ▷ Configuration **automatique** (par les admins ou les dev)

Ces possibilités dépendent de comment Prometheus a été déployé avec votre cluster

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: kubernetes-application
spec:
  endpoints:
    - interval: 10s
      port: prometheus
  namespaceSelector:
    matchNames:
      - kubernetes-application
  selector:
    matchLabels:
      app: kubernetes-app
```

Quelle fréquence ?

Quel port ?

Quel namespace ?

Quels pods derrière quel service ?



TP#6

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Gestion des logs

- ▷ La gestion des logs concerne à nouveau **plusieurs couches** sur un cluster K8s
 - > Le système
 - > Les composants techniques du cluster
 - Services système (**Kubelet**)
 - pods techniques (statiques)
 - pods addons (via des **DaemonSets**, **Deploys**...)
 - > **Les pods applicatifs**
- ▷ C'est cette partie qui nous intéresse !!
- ▷ Ce que l'on attend de Kubernetes
 - > Simplifier la **collecte** et la **consolidation** des logs
 - > **Labelliser** / Décorer / Annoter les logs avec des informations propres au cluster (namespace du pod par exemple)
 - > Faire tout ça avec des conteneurs qui ont de **durée de vie** potentiellement **courte**
- ▷ **L'outillage** permettant de traiter vos logs n'arrive **pas par défaut** dans un cluster Kubernetes. Une **action d'installation** d'une stack spécifique **doit être effectuée par vos administrateurs**



Production des logs applicatifs

La convention de production des logs de Docker prévoit de maximiser la **portabilité des conteneurs** en les rendant **agnostiques** de la méthode de centralisation des logs

- ▶ Les logs sont produits uniquement sur **STDOUT** et **STDERR**
- ▶ **Pas de** production de **fichiers** dans /var/log, même si c'est un volume monté
- ▶ **Pas d'envoi** des logs vers un système tiers (syslog, E/S) spontanément **de la part de l'application**



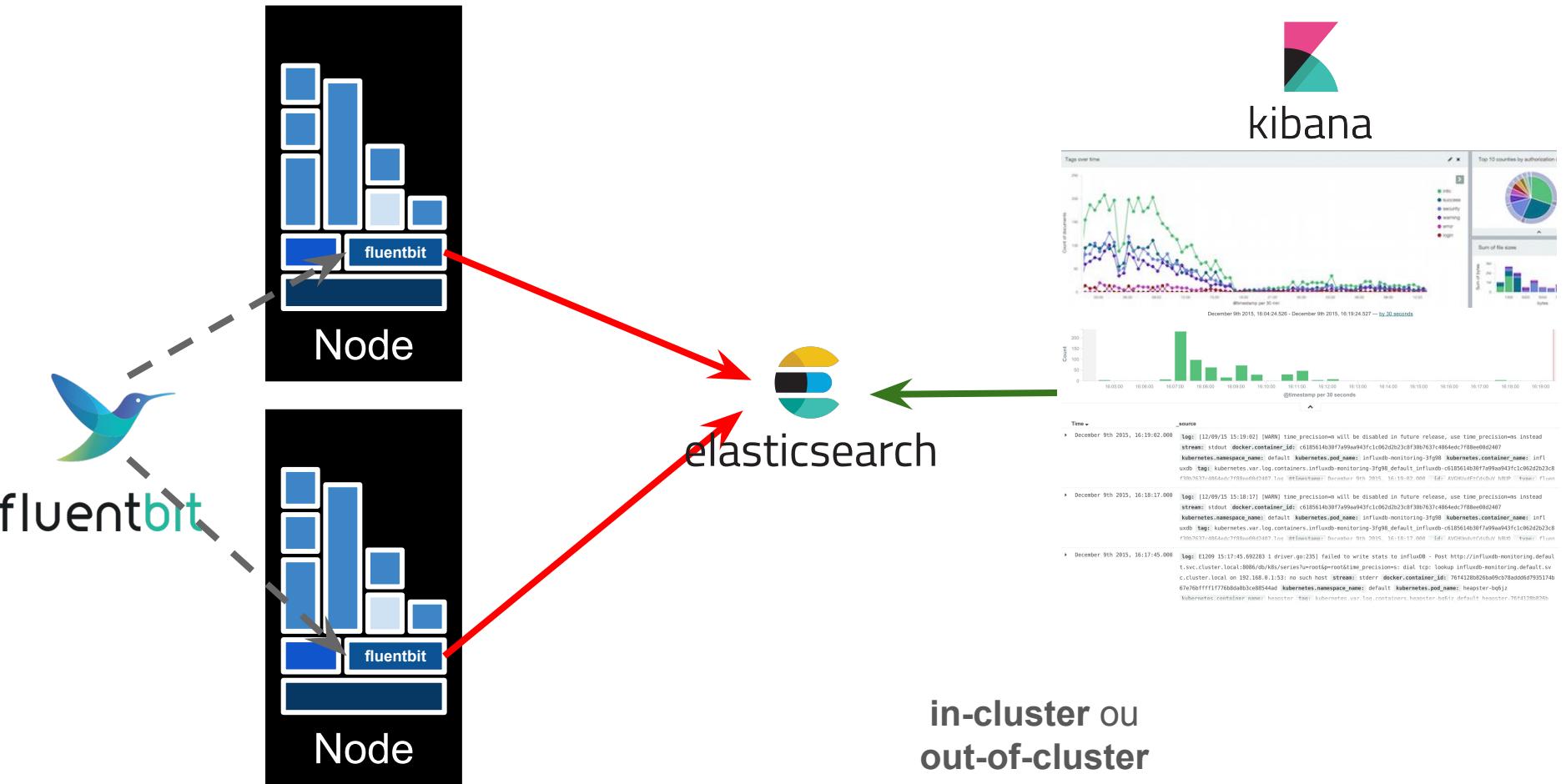
Il s'agit du **11eme des twelve-factor app** :
Traitez vos logs comme un flux d'événement



Le modèle EFK : ElasticSearch / Fluentbit / Kibana



- Une variante du très classique **ELK**
 - **Fluentbit** => un équivalent de Logstash
 - > autres outils utilisés : **FileBeat**, **Fluentd**



TP#7

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Les pods à état

- De la même manière que les conteneurs, les données d'un pod **sont perdues** si celui-ci **meurt ou redémarre**
- Sur K8s, les pods sont **volatiles** et peuvent redémarrer à tout moment (scaling, passage d'un nœud à un autre, erreur dans le pod)
- Il est possible de monter des **volumes** sur les pods
- Kubernetes est capable de gérer des montages de nombreuses sortes
 - > hostPath (répertoire monté sur le nœud)
 - > emptyDir (répertoire vide sur le nœud)
 - > nfs
 - > awsElasticBlockStore
 - > cephfs et ceph
 - > **secret (vu précédemment)**
 - > gitRepo (déprécié)
 - > azureFileVolume
 - > **configMap (vu précédemment)**
 - > ...



Exemple : Volume NFS et pod

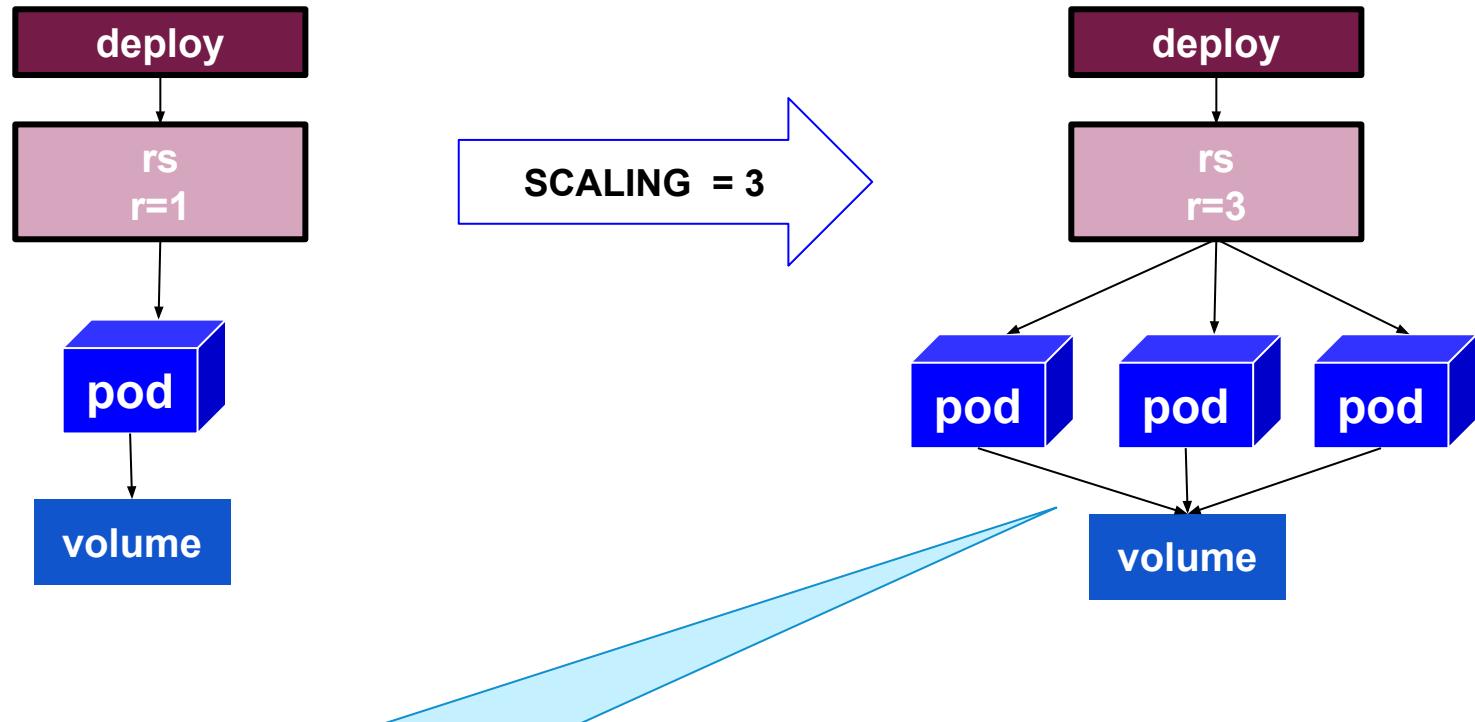
```
apiVersion: v1
kind: Pod
metadata:
  ...
spec:
  volumes:
    - name: postgres-data
      nfs:
        server: 10.244.2.5
        path: /data/postgres
  containers:
    - image: postgres:10
      name: postgres
      volumeMounts:
        - mountPath: /var/lib/postgres/data
          name: postgres-data
```

Ça marche, mais
niveau abstraction,
on a déjà vu mieux...



Les volumes

Attention toutefois au comportement en cas de scaling de votre **deployment / replicaset**



Pour des architectures **legacy** historiques, ça peut faire le boulot, mais attention au type de volume utilisé !!



Comment lever ces deux limitations ?

Le manque d'**abstraction** entre les pods et les volumes est géré par **deux concepts**

- ▶ Les **PersistentVolumes (pv)** qui seront gérés par les administrateurs du cluster
- ▶ Les **PersistentVolumeClaims (pvc)** qui seront gérés par les développeurs au sein de leur namespace

La gestion de volume ajoute également une problématique de **scaling**

- ▶ Certains applicatifs (MySQL, PostgreSQL) ne peuvent pas être simplement scalés
- ▶ Le concept de **StatefulSets (sts)** permet de répondre à cette problématique de façon élégante

Attention à toujours respecter les principes d'architecture consistant à **isoler**

- ▶ Les composants **stateless**
 - vos applicatifs (scalabilité horizontale)
- ▶ Des composants **stateful**
 - les solutions de persistance de tout poil

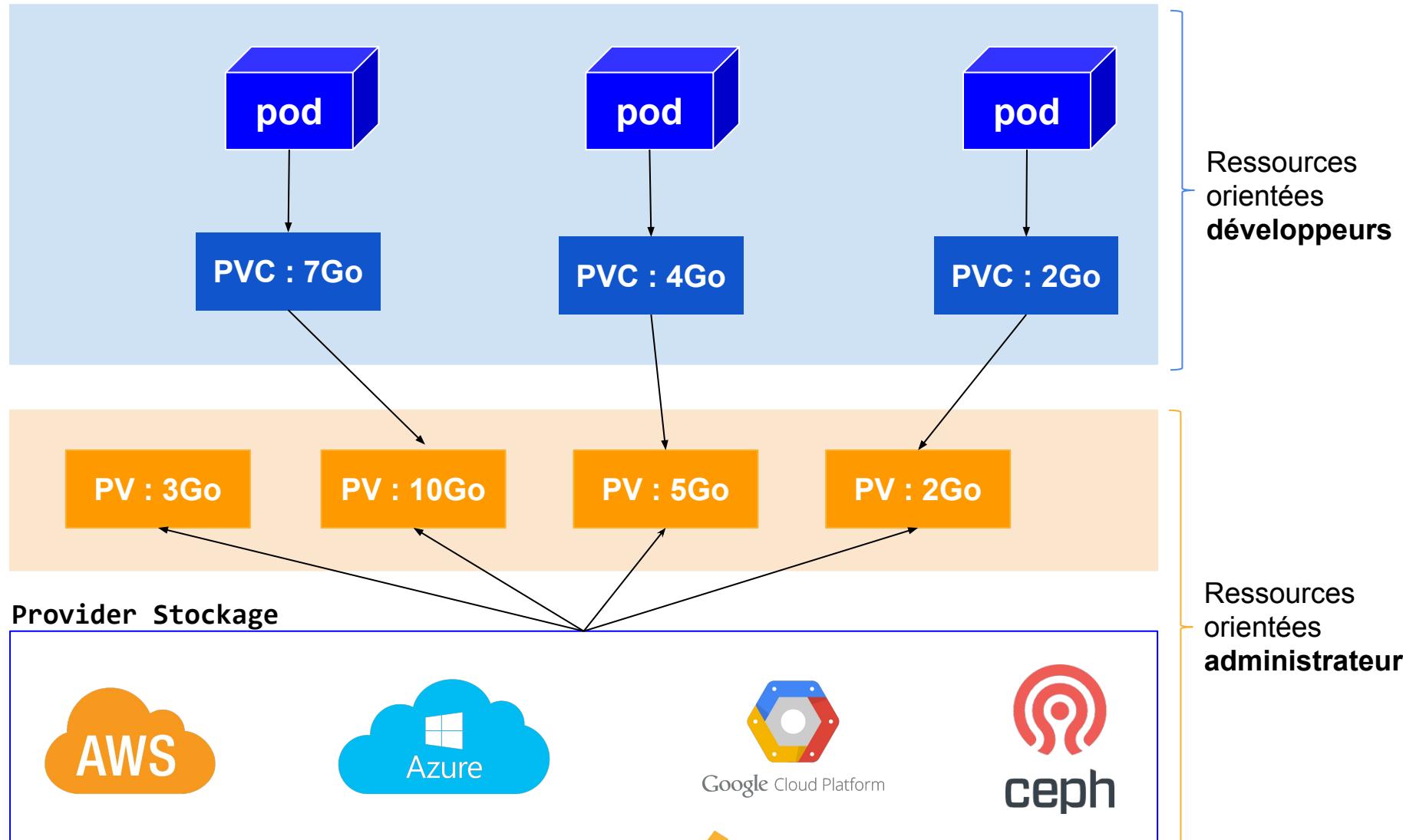


PersistentVolumeClaim et PersistentVolume

PersistentVolumeClaim



PersistentVolume



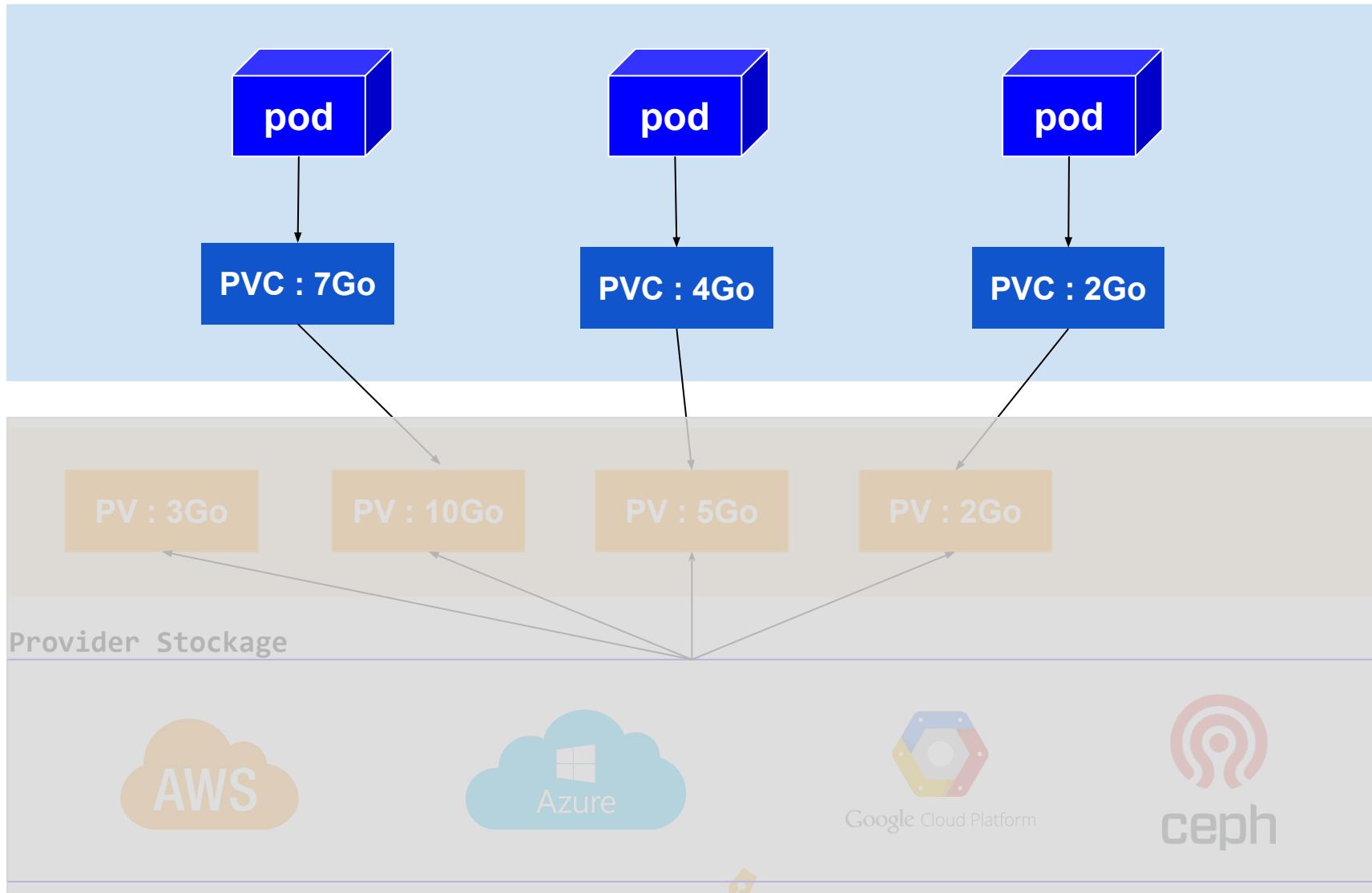
Les PersistentVolumeClaims



PersistentVolumeClaim



PersistentVolume



Les PersistentVolumeClaims (pvc)

- Le PersistentVolumeClaim est une **ressource faite pour les développeurs**
- Le PersistentVolumeClaim permet d'ajouter une **couche d'abstraction** sur le stockage : l'utilisateur monte le **PersistentVolumeClaim** sans se soucier du type de stockage sous-jacent
- Les informations nécessaires pour un émettre un **pvc**
 - La **taille** souhaité (en Go)
 - Le **mode d'accès** (K8s permet 3 modes d'accès, tous les providers de stockage ne permettent pas ces 3 modes)
 - **RWO** : Read Write Once
 - **ROX** : Read Only Many
 - **RWX** : Read Write Many
- Le **PersistentVolumeClaim** peut être configuré pour demander des **PersistentVolumes** spécifiques via des sélecteurs sur leurs labels



Montage des PersistentVolumeClaims dans les pods

Déclaration du pod

```
apiVersion: v1
kind: Pod
metadata:
  ...
spec:
  volumes:
    - name: postgres-data
      persistentVolumeClaim:
        claimName: postgres-claim
  containers:
    - image: postgres:10
      name: postgres
      volumeMounts:
        - mountPath: /var/lib/postgres
          name: postgres-data
```

Déclaration du PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```



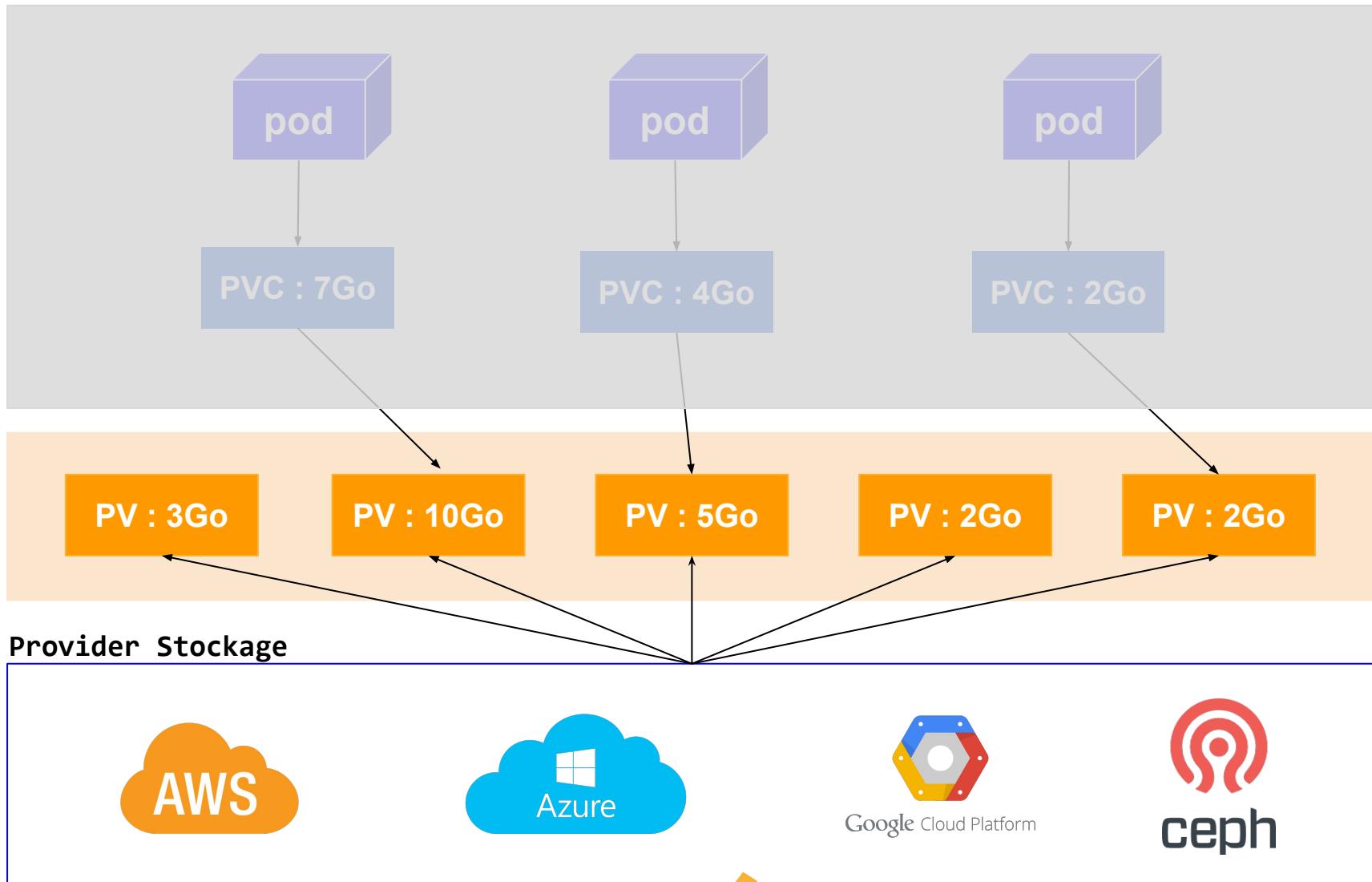
Les PersistentVolumes



PersistentVolumeClaim



PersistentVolume



Les PersistentVolumes

- ▷ Les **PersistentVolumes** sont des ressources **faites pour les administrateurs de votre cluster Kubernetes**
- ▷ Ils **n'appartiennent pas** à un namespace particulier
- ▷ Le **PersistentVolume** représente l'abstraction du stockage et masque les détails d'implémentation (NFS, Ceph, EBS, AzureFile...)
- ▷ Il est de la responsabilité de l'administrateur d'en gérer la **disponibilité**
- ▷ 2 méthodes d'approvisionnement : statique et dynamique



Les PersistentVolumes

- 4 phases dans le cycle de vie d'un **PersistentVolume** (**Available**, **Bound**, **Released**, **Failed**)
- Lors de la phase de **Released** Kubernetes applique la **Reclaim Policy** définie lors de la création du **PersistentVolume**, parmi
 - **Retain** : Quarantaine
 - **Recycle** (déprécié) : Le volume est nettoyé (rm -rf volume_path/*)
 - **Delete** : Le composant assurant le stockage est détruit (Volume EBS, CEPH ...)



Il faut donc retenir qu'en tant que développeur, à partir du moment où vous supprimerez votre ressource **PersistentVolumeClaim**, les données stockées seront potentiellement perdues définitivement selon la configuration des **PersistentVolume**.



PersistentVolume et binding

Déclaration du PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    server: 10.244.2.5
    path: /data/postgres
```

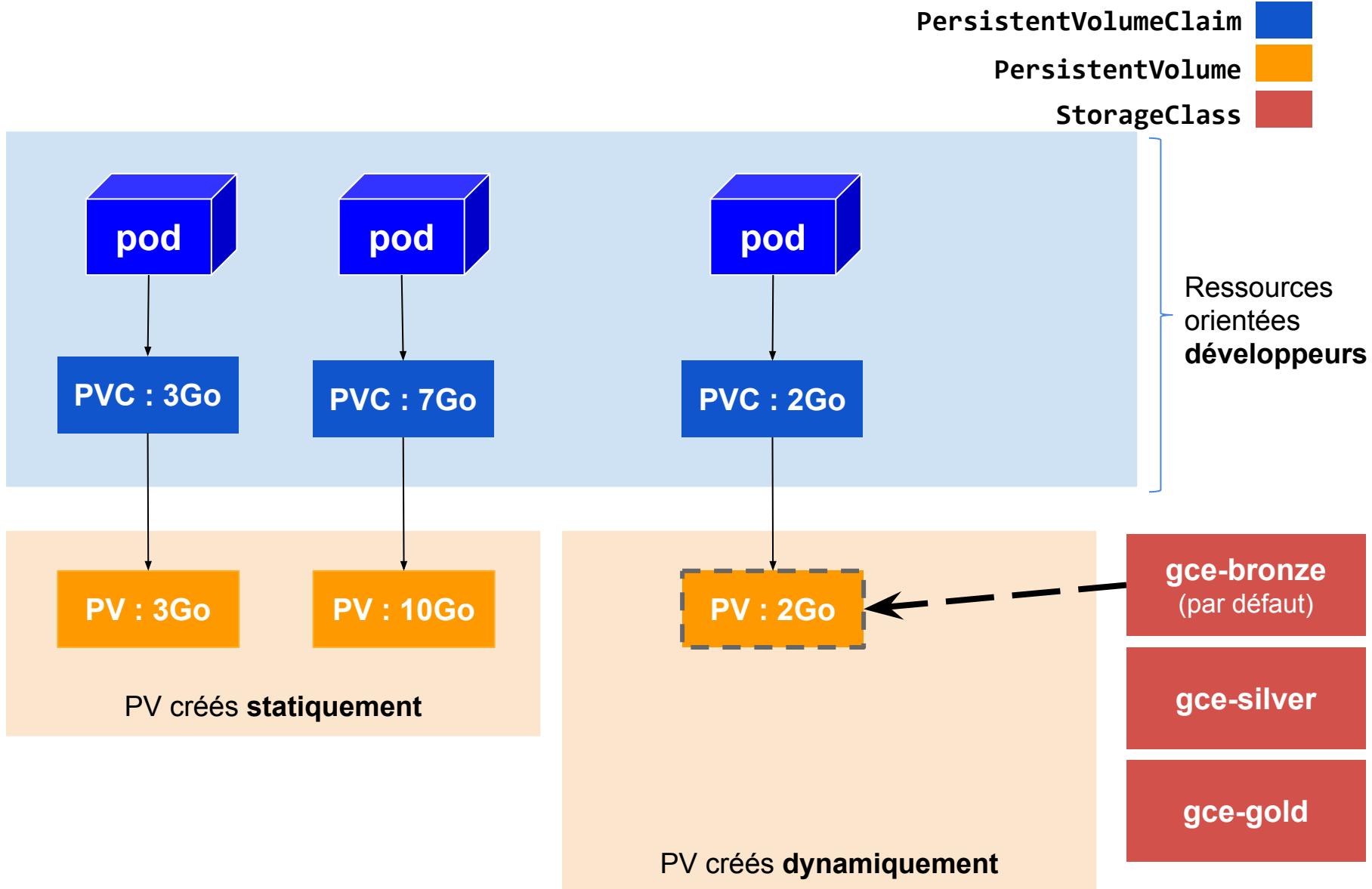
Déclaration du PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-claim
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

```
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM           STORAGECLASS   REASON   AGE
pv1       10Gi        RWO            Delete          Available   <none>         manual        5s
$ kubectl create -f pvc.yml
persistentvolumeclaim "postgres-claim" created
$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM           STORAGECLASS   REASON   AGE
pv1       10Gi        RWO            Delete          Bound      default/postgres-claim   manual        13s
```



PersistentVolumeClaims et approvisionnement



Utilisation de la StorageClass dans le PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-claim
spec:
  storageClassName: gce-gold
  accessModes:
  - ReadWriteOnce
  resources
  requests:
    storage: 10Gi
```

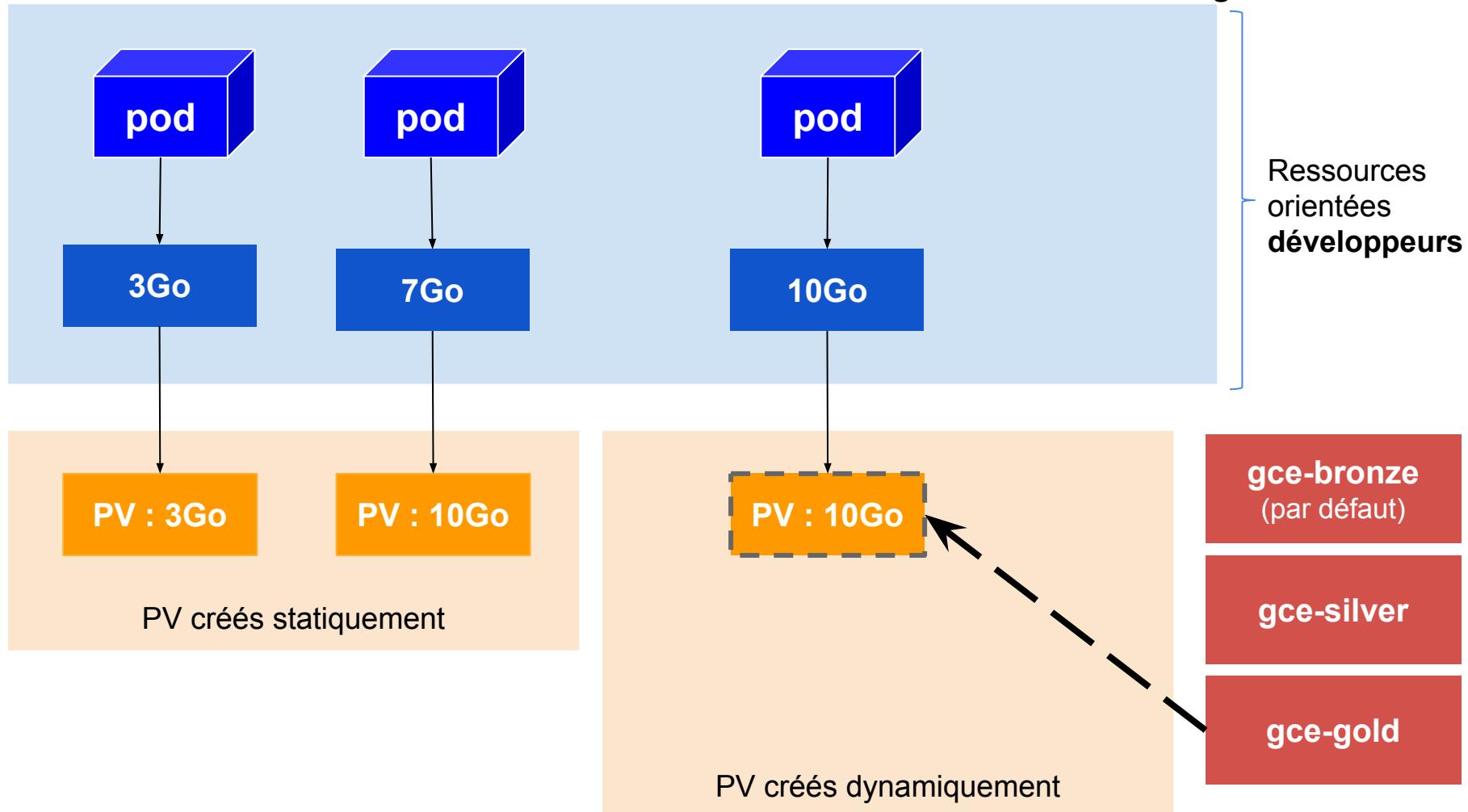
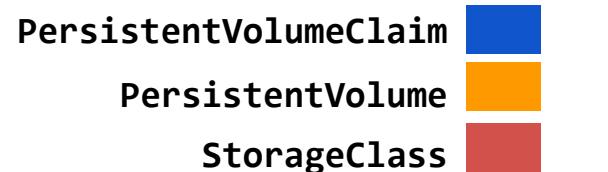
Spécification de la
StorageClass à
utiliser



- Ce modèle permet de ne pas se soucier du provisionnement en **PersistentVolume**
- Les ressources de stockage ne sont **ni infinies ni gratuites**
- Nécessité de mettre des **limitations** et des **quotas** en place



PersistentVolumeClaims en précisant la classe



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ **Les statefulsets**
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Le modèle des StatefulSet (sts)



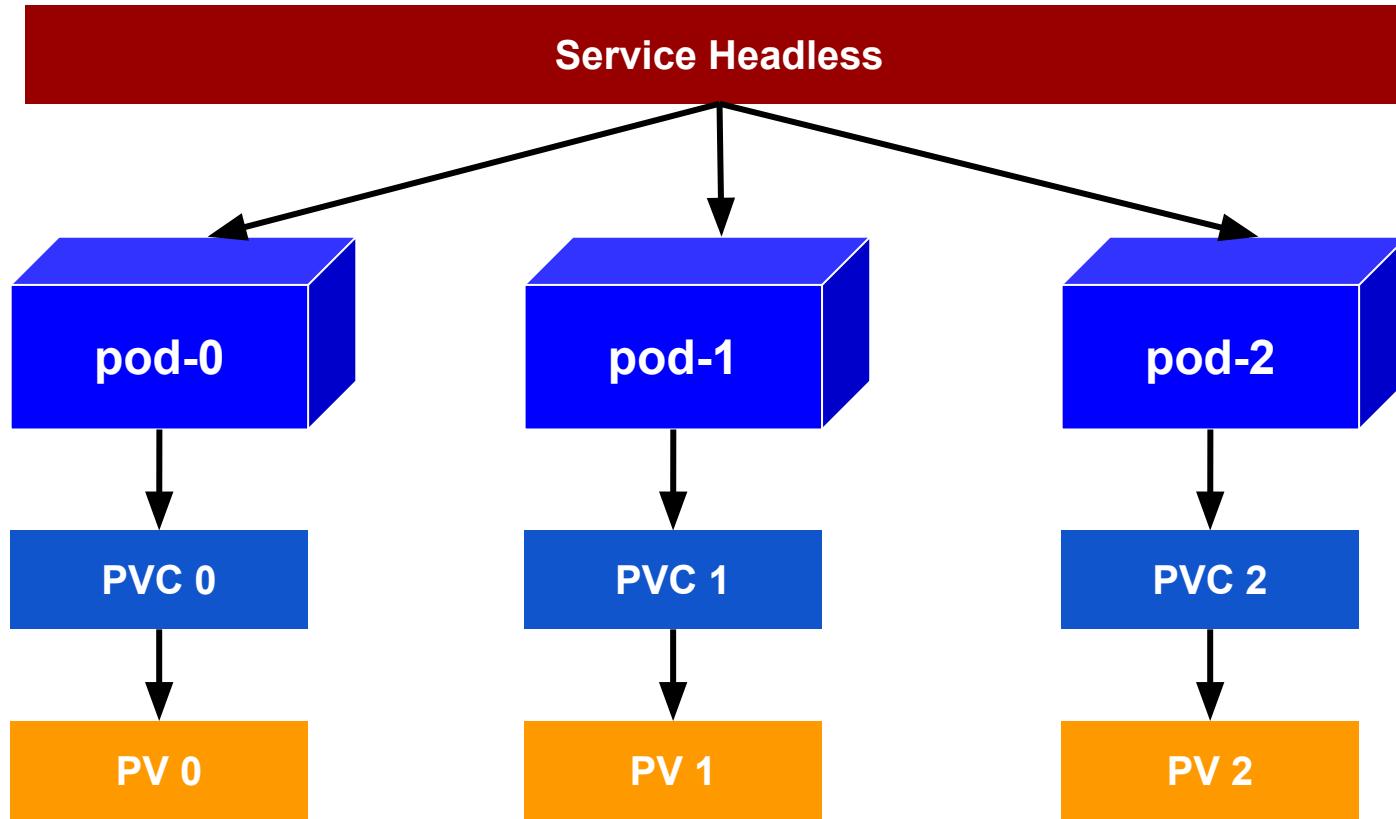
- ▷ Comme les **Deployments**, les **sts** gèrent, déplacent et font écheler les pods en se basant sur leurs spec
- ▷ À la différence des **deploy**, les pods générés par un **StatefulSet** ne sont pas **interchangeables**, ils ont une identité propre, stable, qui suit les pods même en cas de rescheduling
- ▷ Composant K8s (**kubectl get statefulsets**) ou (**kubectl get sts**)



Supprimer un **StatefulSet** ne supprime pas les **PersistentVolumes** associés



Architecture générée par un StatefulSet (replicas = 3)



- La nomenclature de nommage des pods est la suivante :
 - <**Nom du StatefulSet**>-{0..N}
 - Elle suit toujours cette logique, le nom des Pods est **prédictible**

En tant que dev, faut-il gérer ses StatefulSets à la main ?

- ▶ Réponse courte : « **NON** »

Commencer à **manipuler des volumes sans une stratégie et une architecture solides** relève généralement de la **mauvaise idée**.

Répondre à ce besoin ne s'improvise pas sur un coin de table.

ATTENTION !

Ne joue pas avec les volumes et les sts tout seul : tu risques de casser la prod.



Faut-il gérer ses StatefulSets à la main ?

- ▶ Réponse longue : « peut-être en **dev**, probablement **pas en prod** »

Heureusement, de nombreux éditeurs proposent de **masquer** (une partie de) **la complexité de gestion** des clusters de persistance :

- ▶ Déploiement
- ▶ Mise en haute-disponibilité
- ▶ Montée de version
- ▶ *Backup / monitoring*
- ▶ *Scale up, scale down*
- ▶ Plein d'autres trucs compliqués en fonction de la techno



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Ajouter de la logique à la gestion des StatefulSets

Pour ça, il est possible de définir de **nouveaux types de ressources** dans Kubernetes : les **CustomResourceDefinitions (crd)**

Exemple de **crd** :

- ▷ Clusters MySQL / PostgreSQL
- ▷ Clusters Elasticsearch
- ▷ Clusters etcd
- ▷ Cluster Redis
- ▷ *u-name-it*
- ▷ Plein d'autres trucs pour les ops



- L'installation de **crd** et des agents qui vont traiter ces nouvelles ressources (opérateurs) est de la **responsabilité des administrateurs**.
- Cela demande des **droits très avancés** sur le cluster.



Exemple de CRD



Nouvelles ressources non standard

```
$ kubectl get crd  
NAME                                         CREATED AT  
elasticsearchclusters.enterprises.upmc.com    2018-09-21T07:54:49Z  
redisfailovers.storage.spotahome.com           2018-09-25T08:10:31Z
```

Instanciation d'un objet non standard
(par un `kubectl apply`)

```
apiVersion: storage.spotahome.com/v1alpha2  
kind: RedisFailover  
metadata:  
  name: mon-cluster-redis-qui-dechire
```



TP#8

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

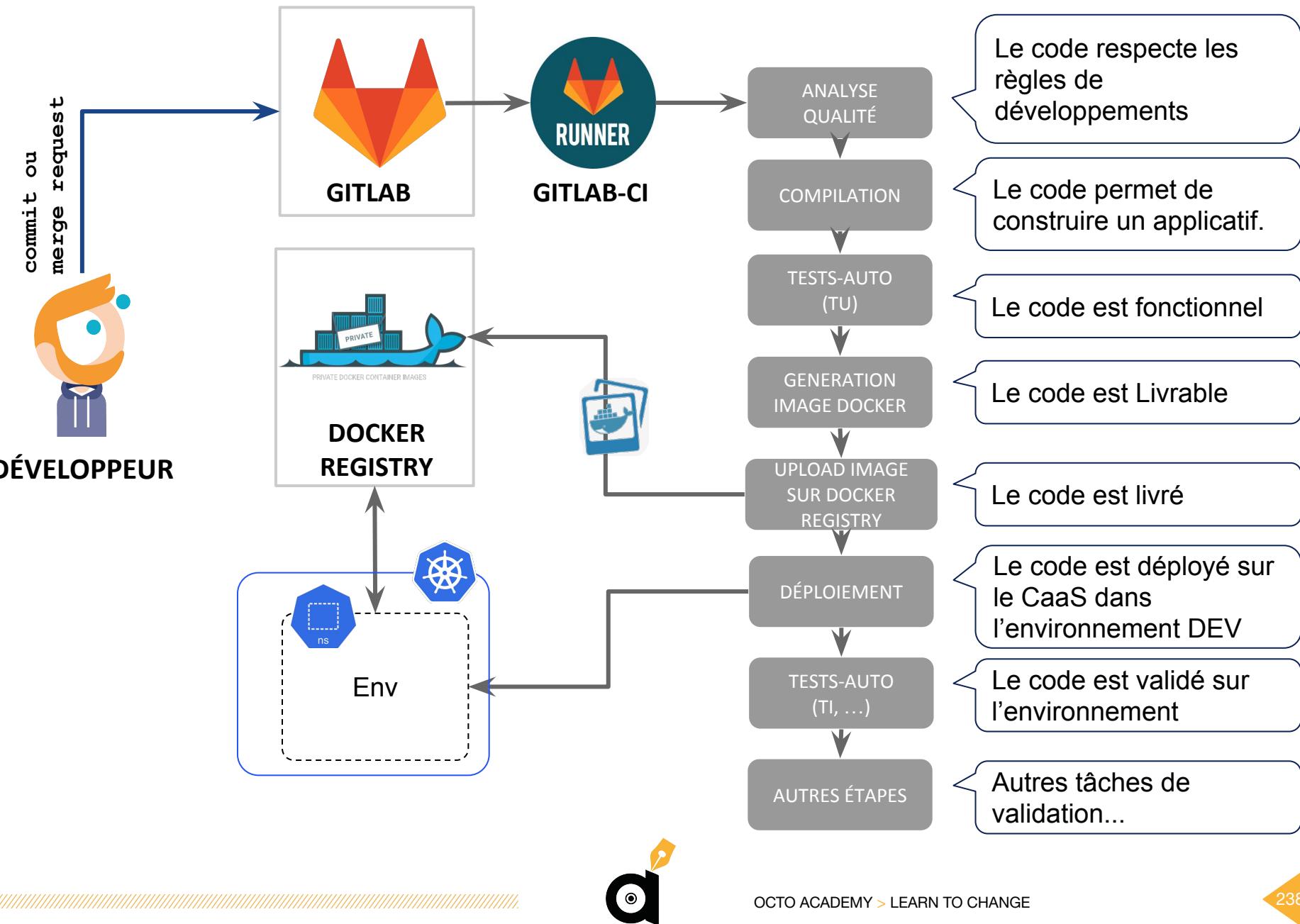
- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Exemple de workflow Kubernetes : Pipeline CI



Pipeline CI : plusieurs modèles possibles avec Kubernetes

- ▶ La gestion des branches de votre **code git** peut être intégrée dans votre pipeline
- ▶ Elle permet de tester votre **application** dans un **environnement similaire** à votre **production** avant de merger dans une branche à plus forte visibilité (dev, recette,..., production)

Nous vous proposons **deux modèles** (mais il en existe bien plus)



La possibilité de déployer une application en fonction d'une **branche du système de contrôle de version** est le **1er des twelve-factor app.**

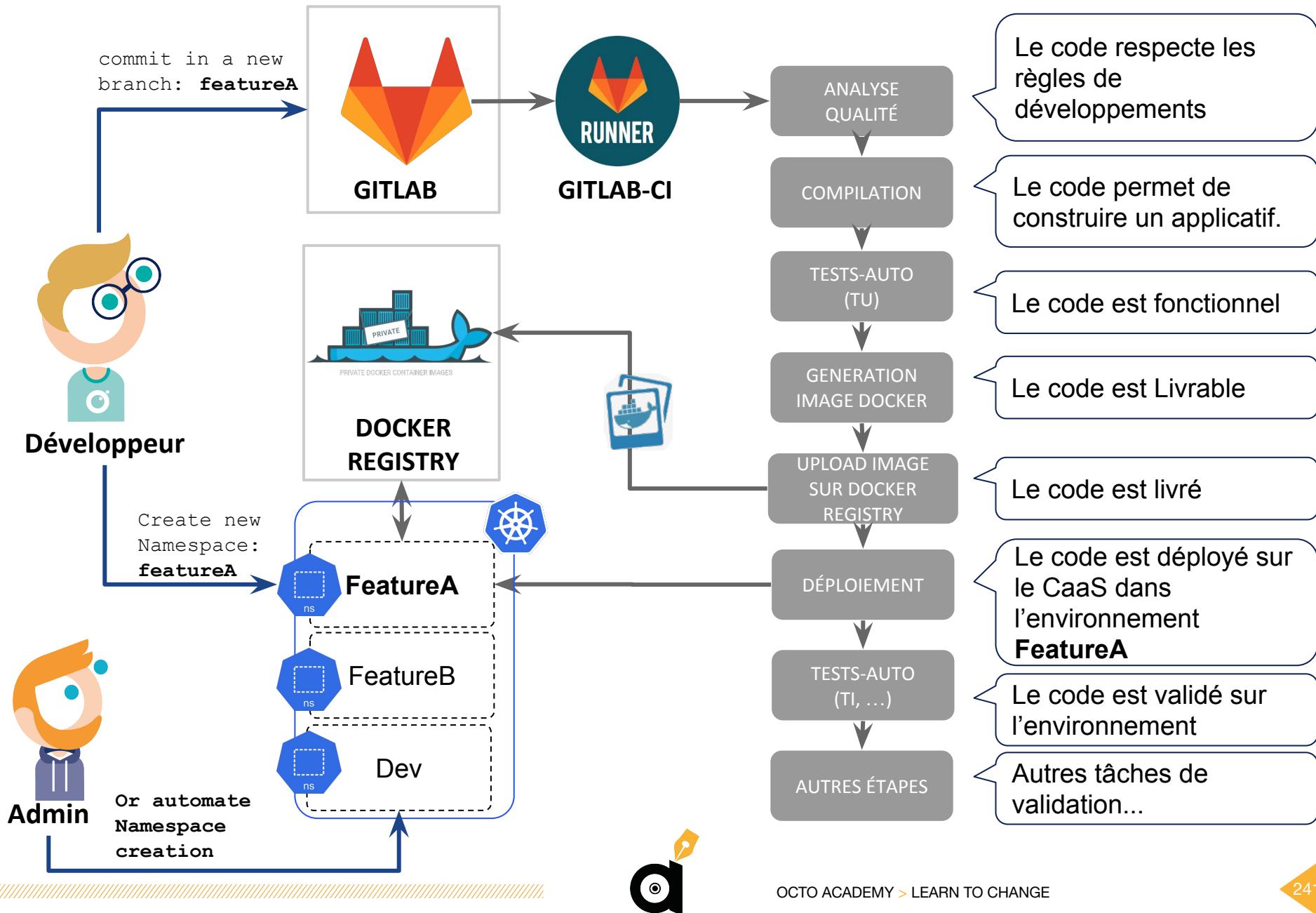


Pipeline CI : Modèle une branche = un namespace

- ▶ Les développeurs ont des droits étendus sur le **cluster** et peuvent créer leurs propres namespaces ...
- ▶ ... ou les **administrateurs** de votre cluster **automatisent la création dynamique d'un namespace** à chaque création de branche dans votre repo git



Pipeline CI : Une branche = un namespace

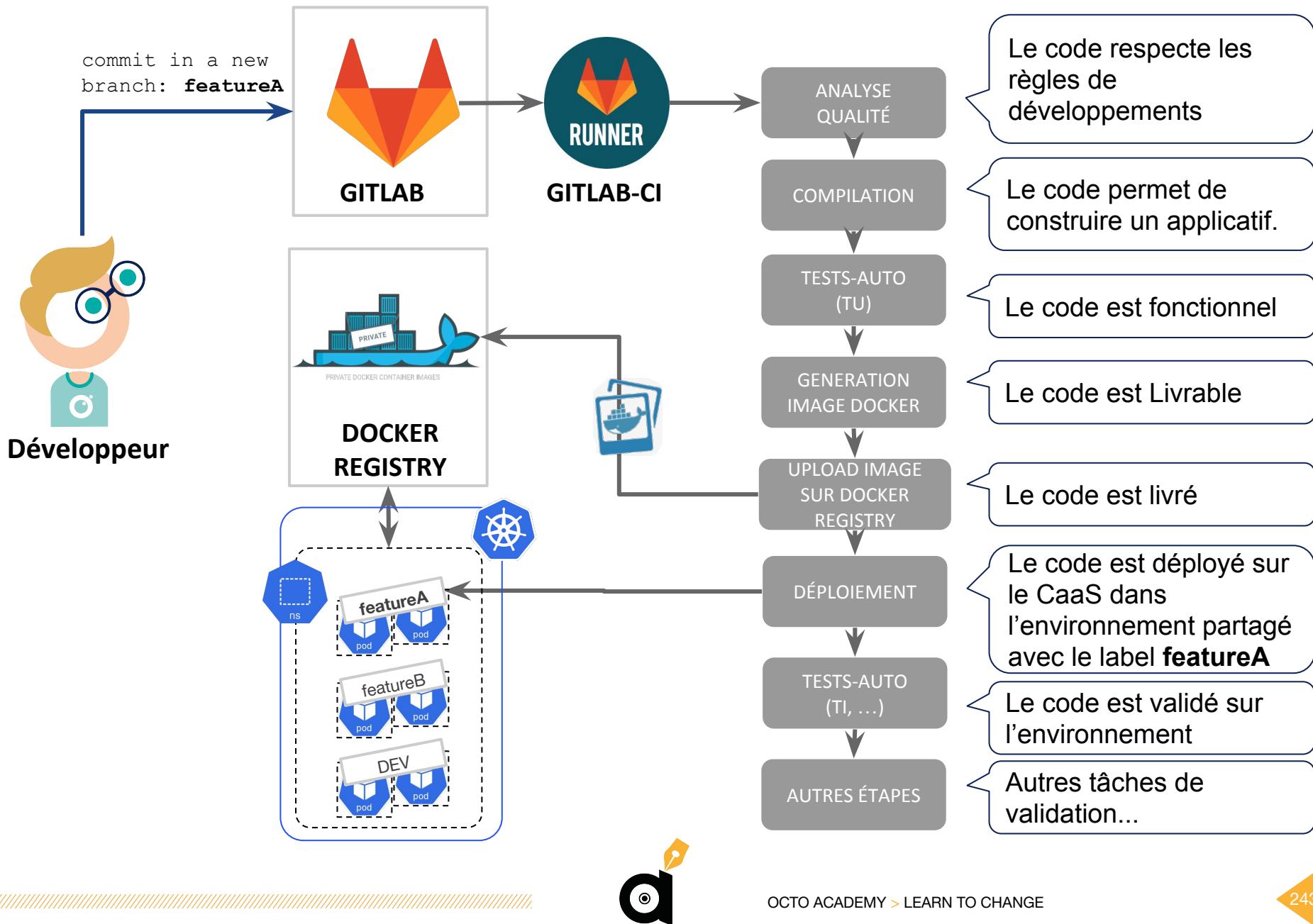


Pipeline CI : Modèle une branche = un label

- Une **labellisation** et un **nommage fin** de vos ressources en **adéquation** avec votre **branche** dans **un seul et même namespace**
- Vous retrouverez **plusieurs fois la même topologie**, avec une **séparation des branches** représentée par **plusieurs labels**



Pipeline CI : Une branche = Une gestion fine des labels



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Exemple de workflow Kubernetes : Pipeline CD

- Le **continuous deployment** est une **usine de déploiement** qui suite aux différents tests effectuées par la CI, aura pour rôle de livrer le code dans **differents environnements**
- Une **bonne pratique** que nous aimons appliquer est de **construire l'application une fois**, pour la **déployer n fois** dans les **differents environnements**.

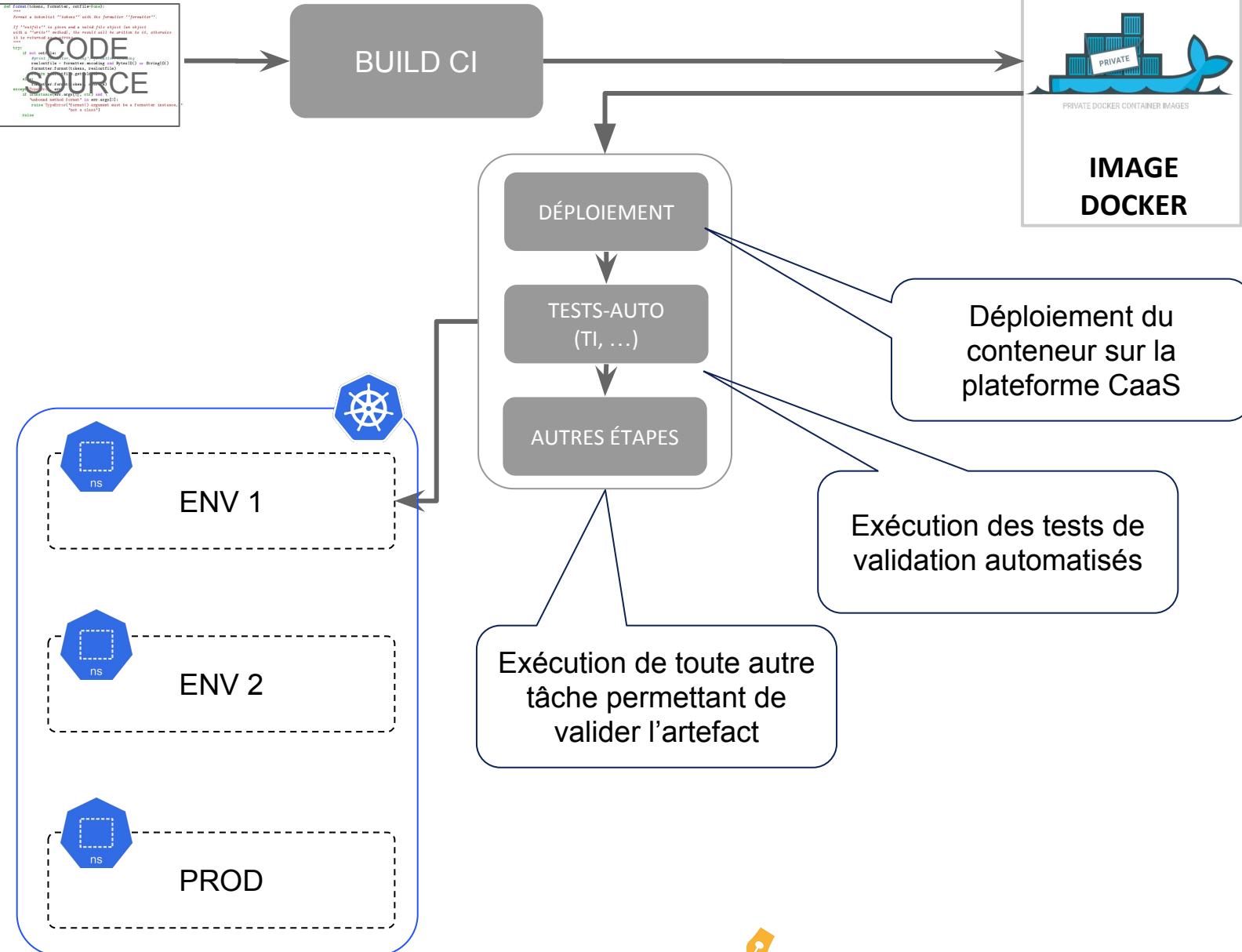


Deux principes des **twelve-factor app** s'appliquent avec le **déploiement continu**:

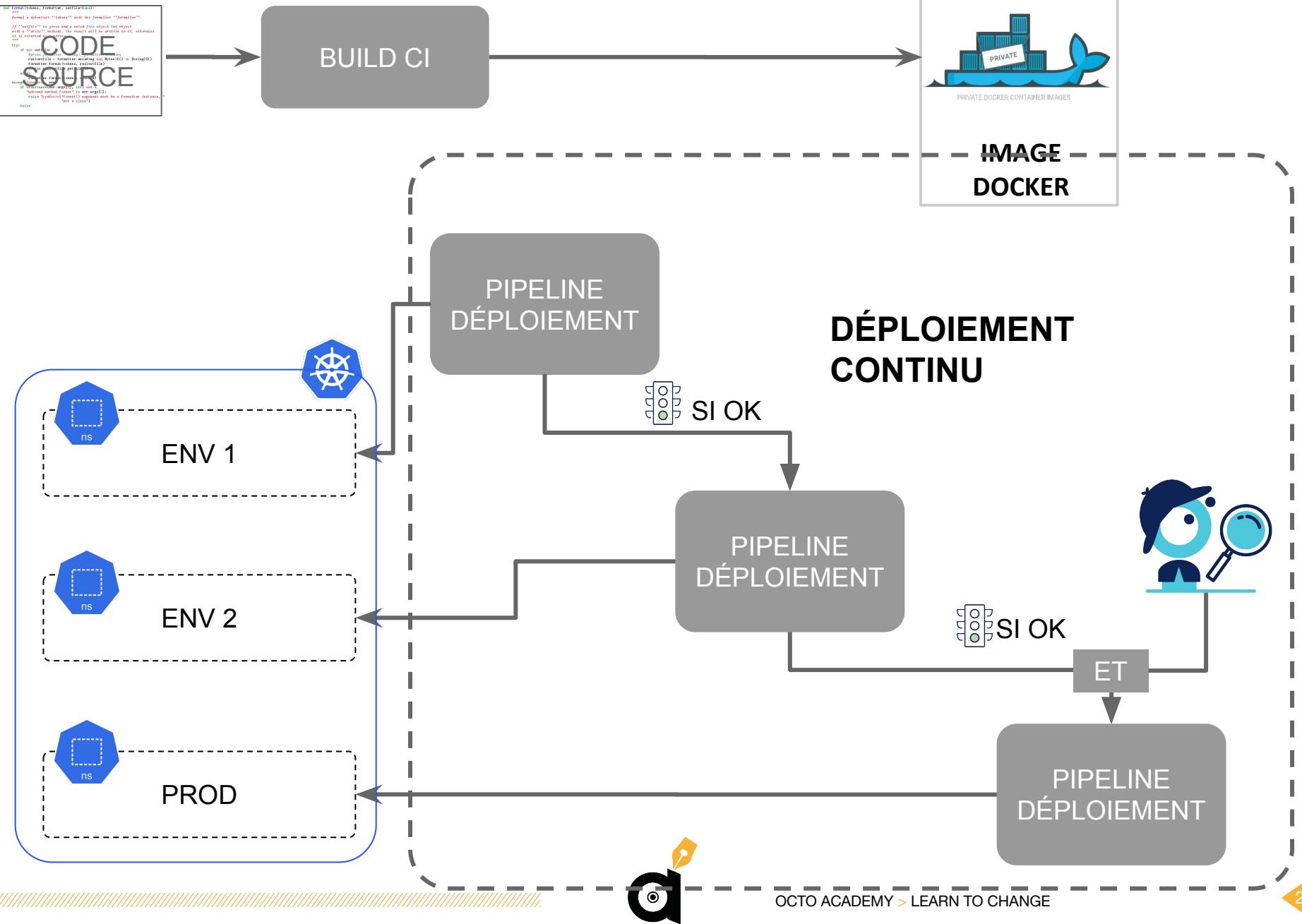
- > Le **5eme** : **Build, release, run** puisque l'on sépare les étapes d'assemblage
- > Le **10eme** : **Parité dev/prod** puisqu'on donne le moyen de livrer un package unique de la dev à la prod



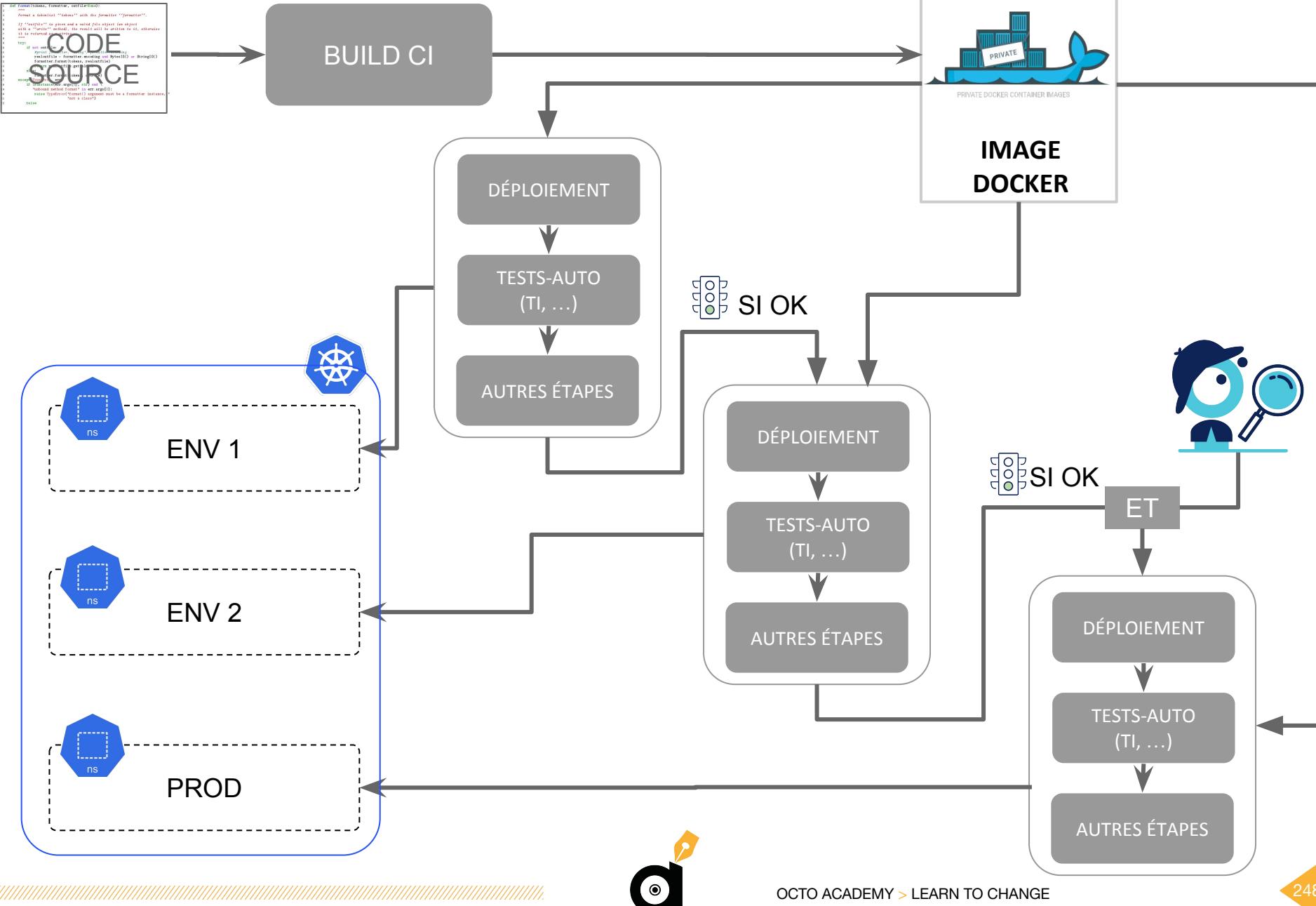
Exemple de workflow Kubernetes : Pipeline CD



Exemple de workflow Kubernetes : Pipeline CD



Exemple de workflow Kubernetes : Pipeline CD



CD : Comment déployer dans Kubernetes ?

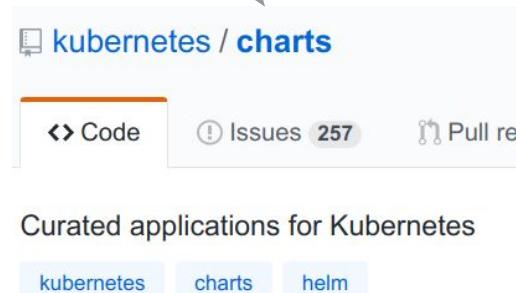
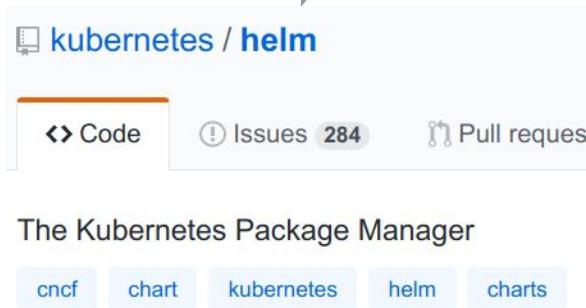
Aujourd’hui, il existe de nombreux moyens de déployer ses applications dans K8s. Toutes les solutions sont modulaires et componables pour être lancées dans un pipeline de CI/CD (Jenkins, Gitlab-CI...)

- ▷ Du simple shell qui lance des `kubectl apply`
- ▷ Ansible
- ▷ Terraform
- ▷ Helm
- ▷ Skaffold
- ▷ Gitkube
- ▷ Pulumi



Un mot sur Helm...

- ▶ Helm est un **outil** de déploiement, avec un **catalogue** d'architectures-types pour K8s



```
$ helm init  
[...]  
$ helm install -n mariadb --namespace myns stable/mariadb
```



Un mot sur Helm...

- ▷ Helm permet de déployer des **topologies applicative complètes** dans **Kubernetes**
 - deployments, services, configmaps...
- ▷ Ces topologies, appelées chartes, sont soit
 - Récupérées auprès de la communauté (**helm search <name>** && **helm install <name>**)
 - Crées à partir de la commande **helm init**
- ▷ Le déploiement d'une charte se fait sous forme de **releases**
 - elles sont déployées à partir d'un composant installé dans Kubernetes appelé **Tiller**
 - chaque release possède un numéro de version
 - chaque déploiement se fait par **topologie complète (sans notion d'idempotence)**
- ▷ Helm est un outil très intéressant dans le cas du **déploiement de composants complexes** comme les **statefulsets** (car c'est une mine d'exemples)...



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

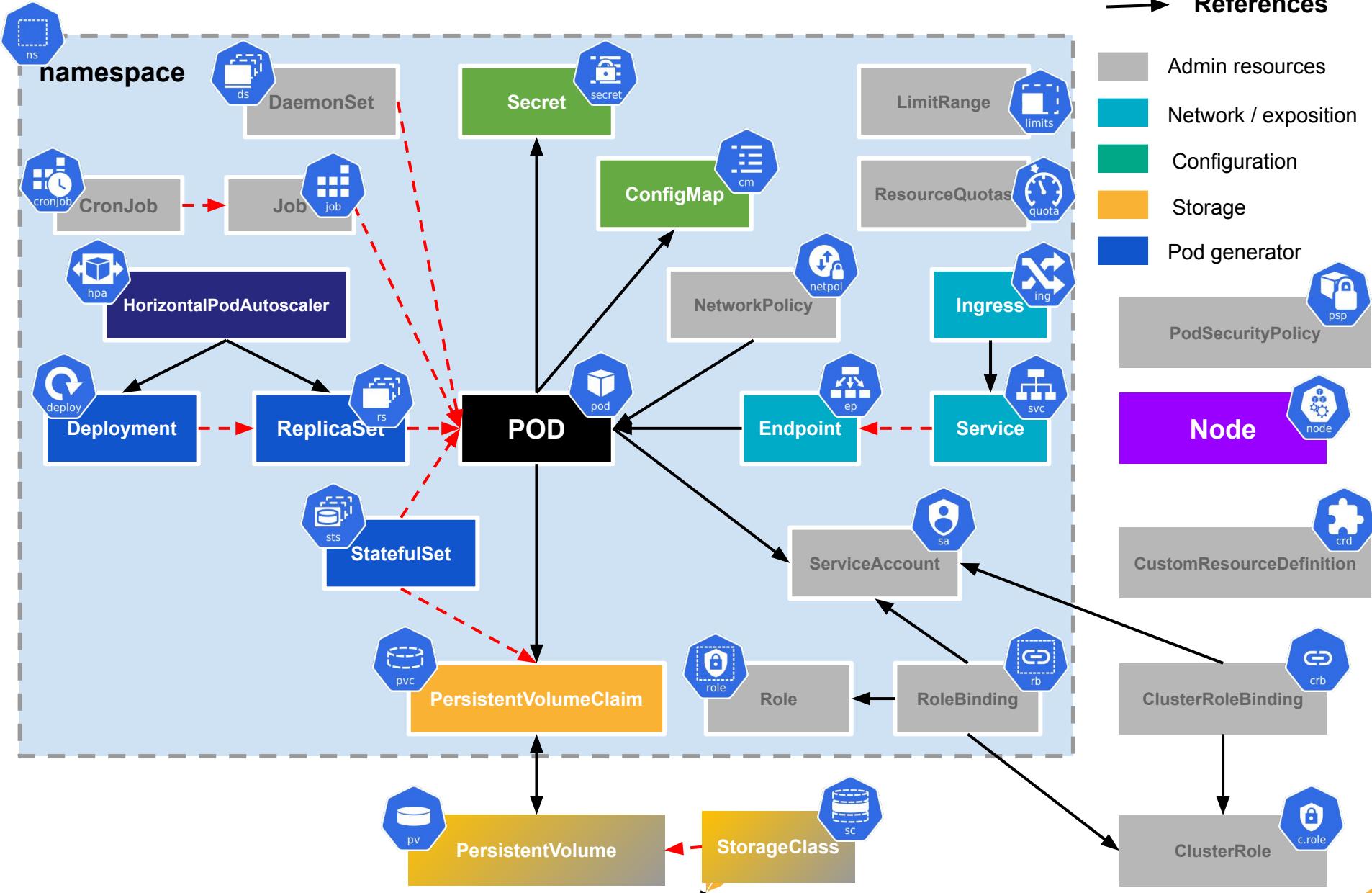
Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Conclusion et Take Away

Kubernetes Ressources Map

→ Creates
→ References



What's next

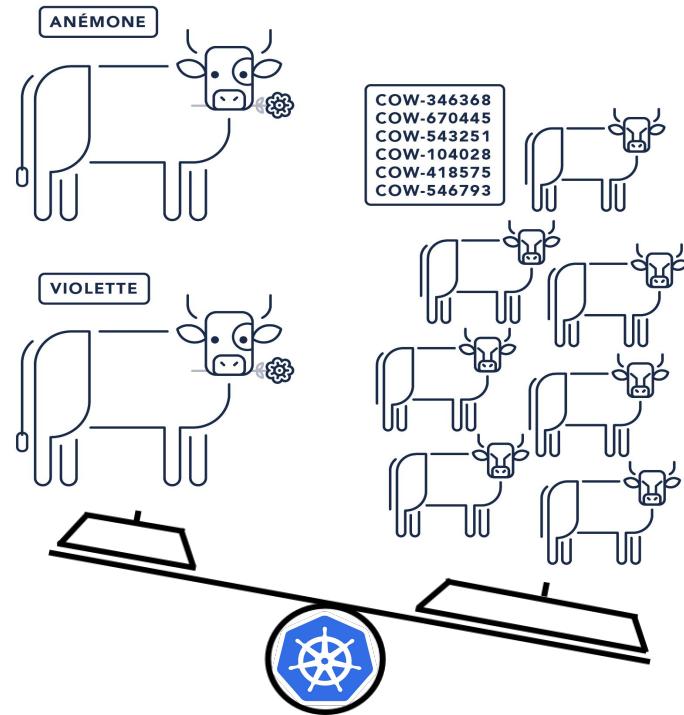
- ▷ La **1ère étape** avant de se lancer dans le déploiement de votre **application dans Kubernetes**, c'est réfléchir au **cycle de vie de bout en bout** de votre **application (CI/CD...)**
- ▷ Plus d'une **douzaine de ressources** restent à découvrir pour compléter votre **connaissance de Kubernetes**. Celles-ci sont généralement **réservées** aux **administrateurs**
- ▷ La communauté Kubernetes s'étend jour après jour et nous apporte son lot d'outils dans un contexte de développement, nous retiendrons notamment:
 - > **minikube** qui vous permet de déployer un kubernetes localement
 - > **skaffold** qui vous permet de faciliter le cycle de vie de votre application du build au déploiement
 - > **telepresence** qui vous permet de lancer votre code, encore en développement dans Kubernetes
 - > ...



Take Away : Les bonnes pratiques sur K8s

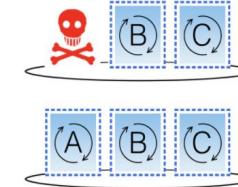
- Docker :
 - Ephémère par principe.
 - Minimiser le nombre de couches.
 - Utilisez le multi-stage build pour des images plus petites et sécurisées
 - Ne pas redémarrer l'applicatif en cas d'échec, planter proprement à la place.
 - Log vers stdout & stderr,
 - Préciser un utilisateur avec les droits minimum nécessaire.
- Déploiement K8s:
 - Utiliser des Labels.
 - Penser aux sondes readiness & liveness.
 - Ajouter des URL de métriques.
 - Toujours ajouter des limitations de ressources et quotas.
- Exposition :
 - Utiliser le type LoadBalancer uniquement quand nécessaire, ClusterIP ou NodePort peut être suffisant.
 - Ingress = votre reverse proxy managé.
 - Utiliser le service External pour mappez vos services externe à votre cluster.

Pour aller plus loin :
<https://blog.octo.com/the-twelve-factors-kubernetes/>

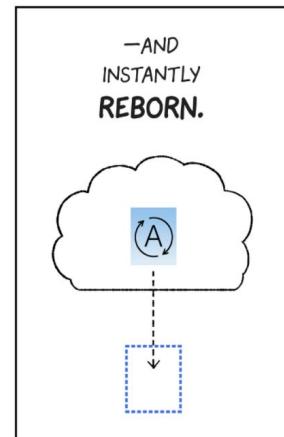


IF ANY VIOLATION OR
INCONSISTENCY IS
DETECTED...

BAM.



SOMETHING'S GONNA
BE TERMINATED—



Source image

