

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

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

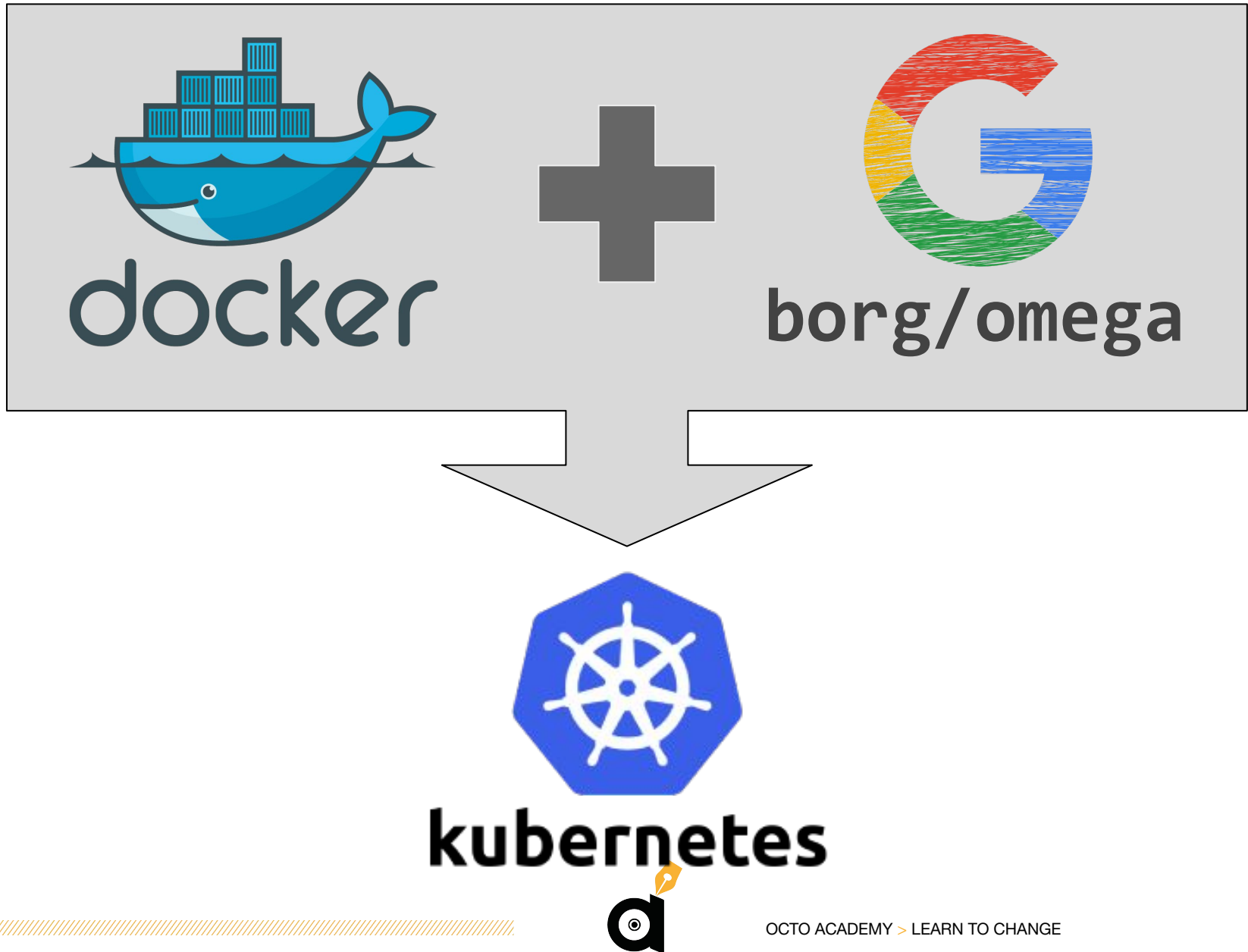
Le Continuous Delivery avec Kubernetes

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

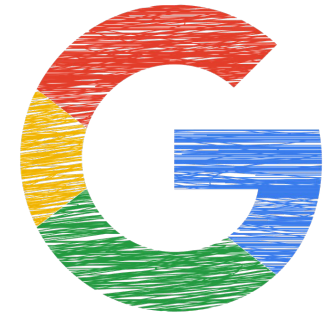
Eco-conception

Conclusion et Take Away

Genèse de Kubernetes : à la croisée des chemins



Genèse de Kubernetes : à la croisée des chemins



borg/omega

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 et annonce de la CNCF (Cloud Native Computing Foundation), organisation qui appartient à la Linux Foundation)

En 2018, Google cède le contrôle opérationnel à la CNCF

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 !

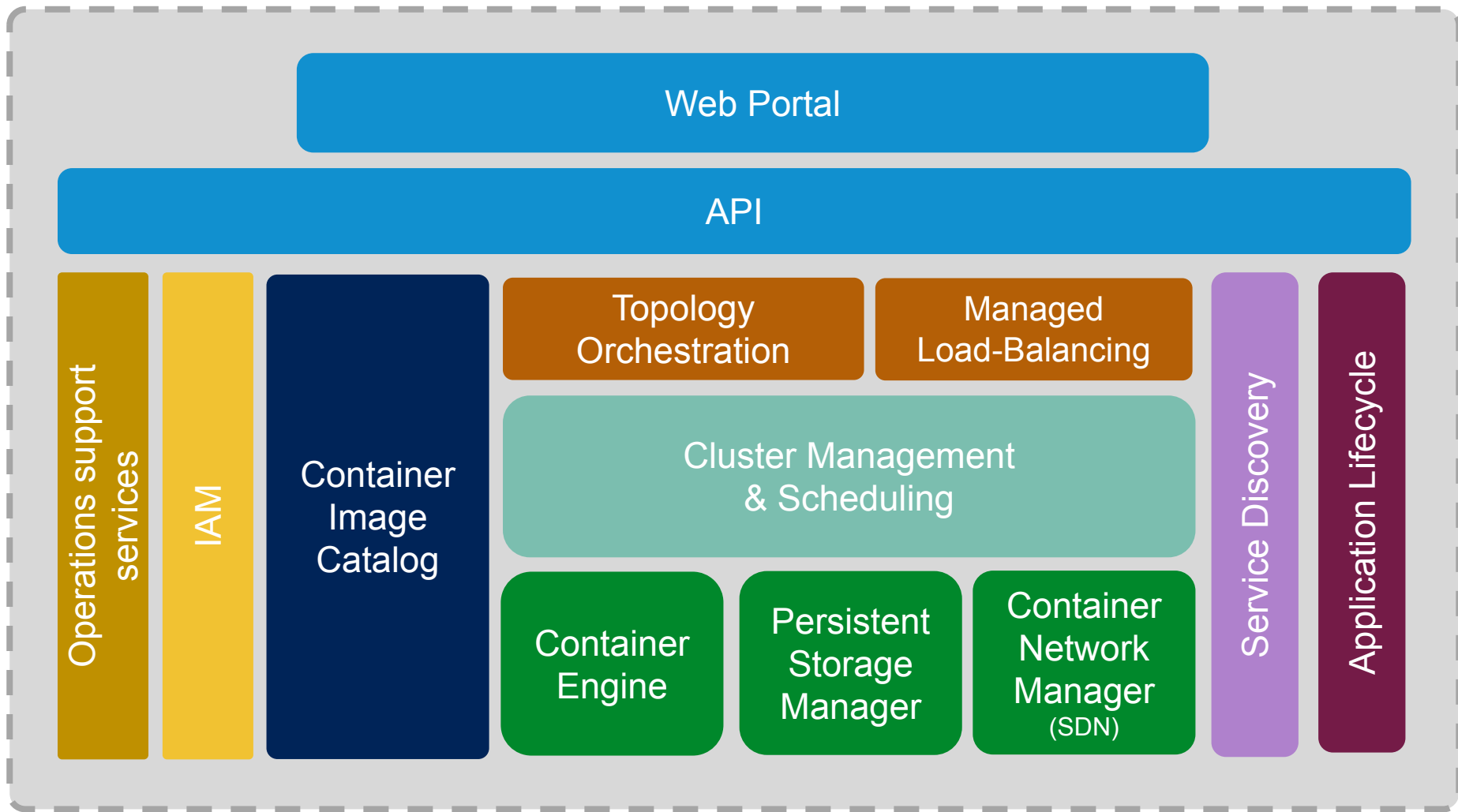


L'approche de K8s

- ▷ **Abstraction** des concepts pour élever 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)



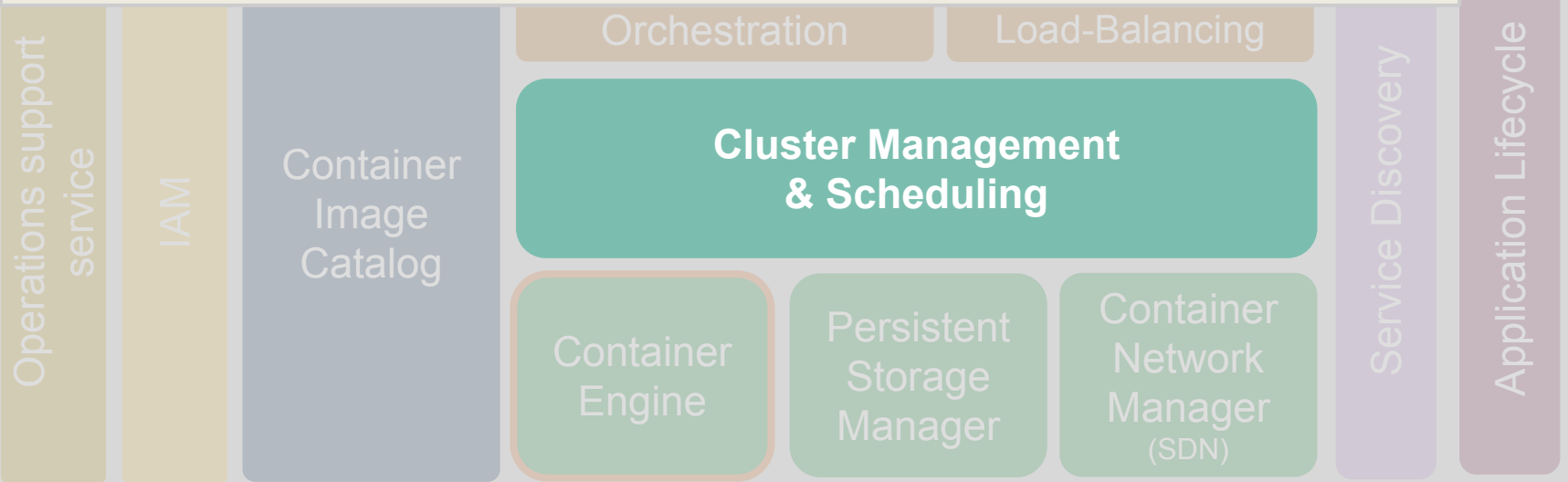
Cartographie fonctionnelle



Cluster Management

Rôle

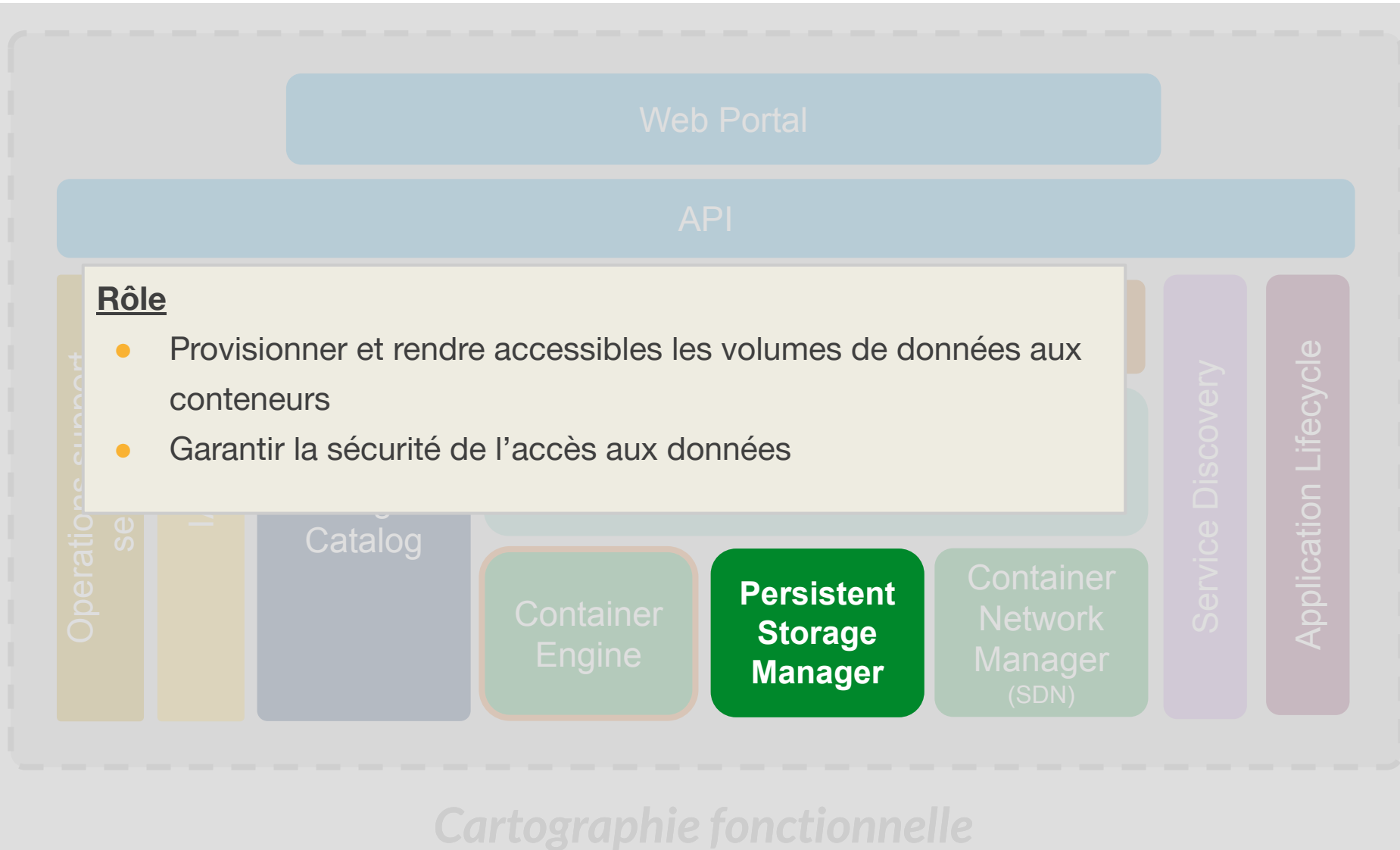
- Abstraire une flotte d'hôtes comme une ressource unique
- Déterminer et optimiser le placement des conteneurs sur les hôtes
- Surveiller et mettre à jour l'état du cluster
- Gérer les dysfonctionnements des nœuds du cluster



Cartographie fonctionnelle



Persistent Storage Manager



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

Catalog

Container
Engine

Persistent
Storage
Manager

**Container
Network
Manager
(SDN)**

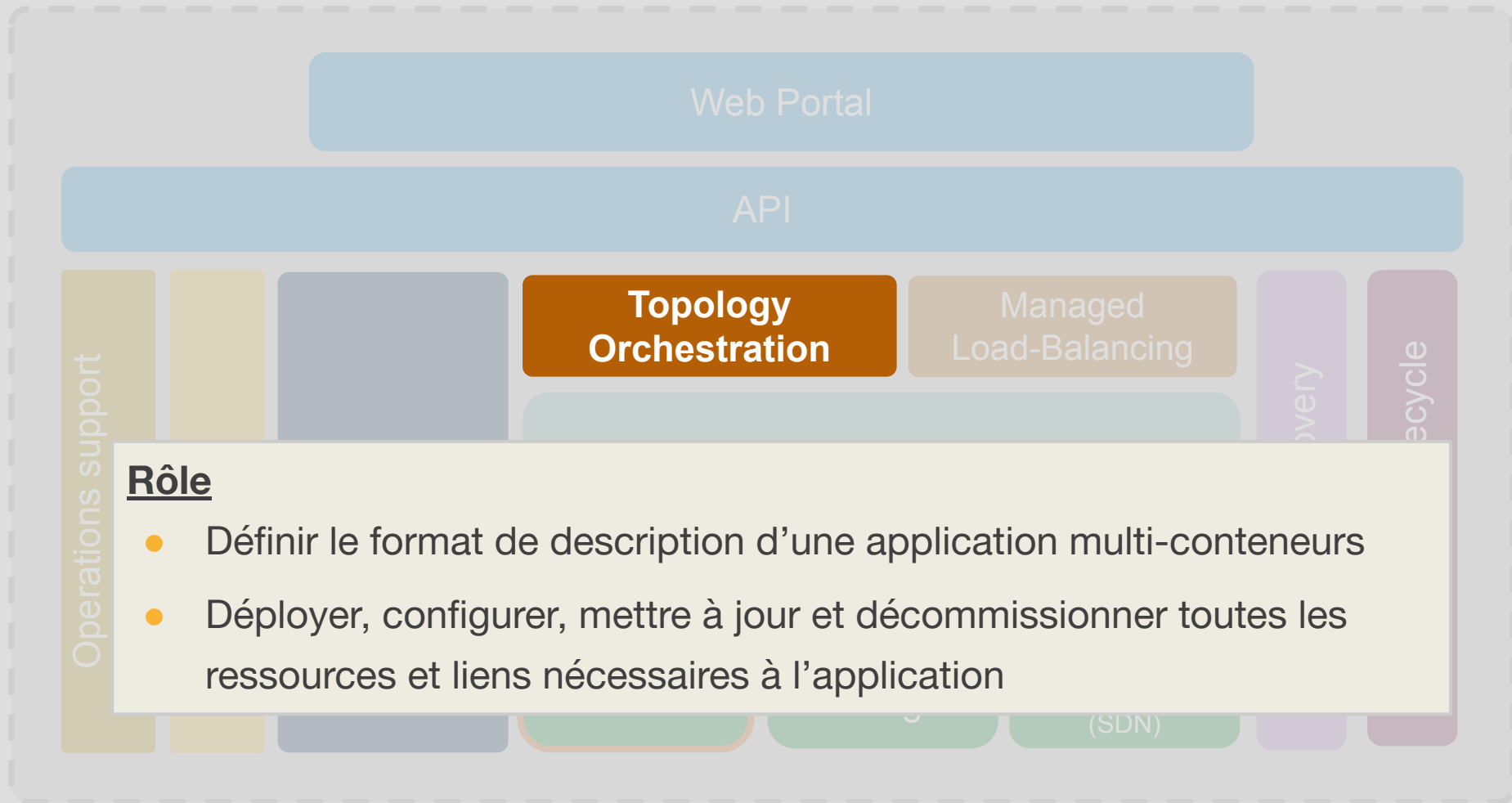
Service Discovery

Application Lifecycle

Cartographie fonctionnelle



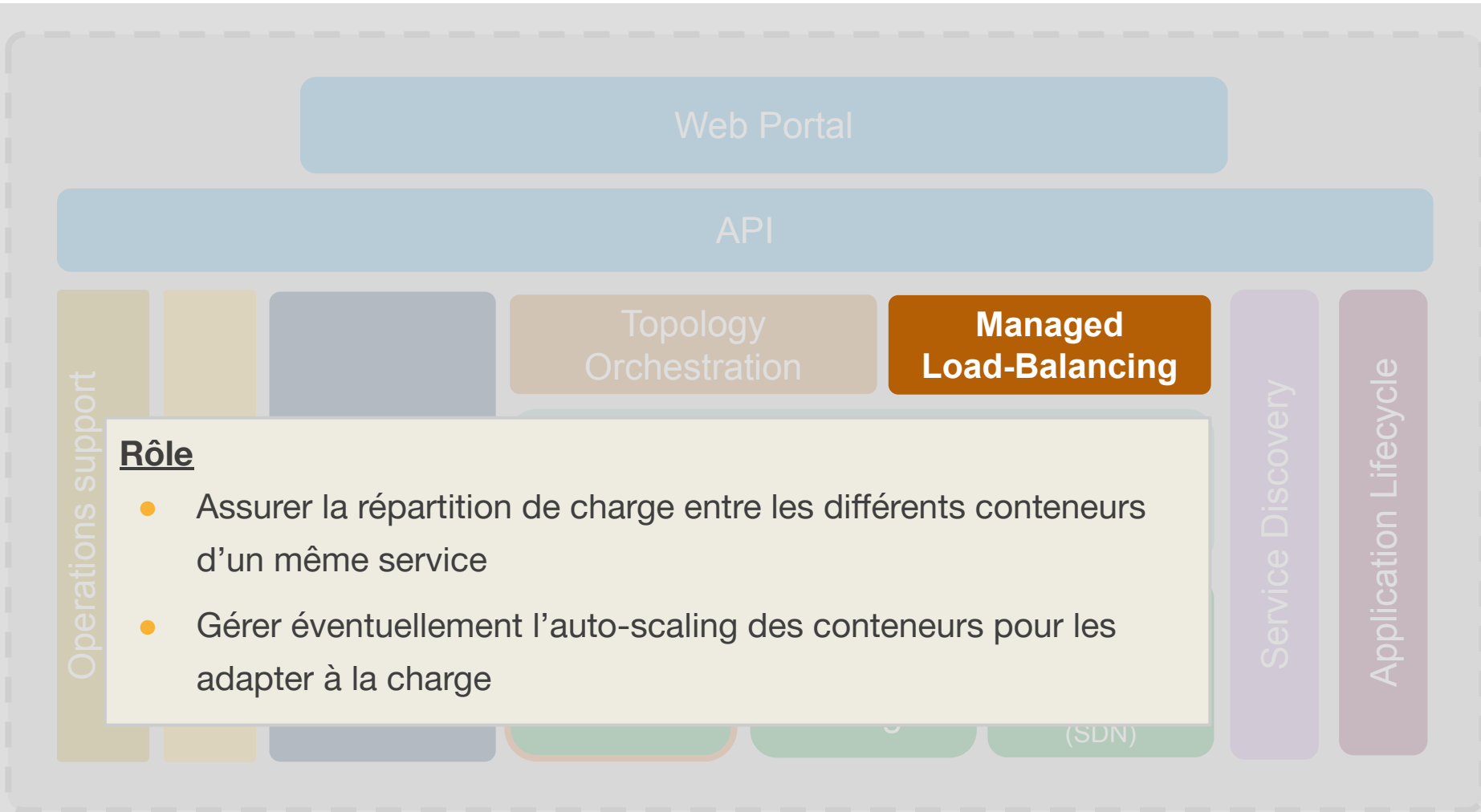
Topology orchestration



Cartographie fonctionnelle



Managed Load-Balancing



Cartographie fonctionnelle



Service Discovery

Web Portal

Rôle

- Maintenir les informations sur l'état du cluster de manière fiable
- Offrir des mécanisme d'élection, de détection d'indisponibilité et de publication/souscription aux composants de la plateforme

Operations
service

IAM

Image
Catalog

Container
Engine

Persistent
Storage
Manager

Container
Network
Manager
(SDN)

Service Discovery

Application Lifecycle

Cartographie fonctionnelle



Operations support services and IAM

Rôle

- Fournir les services nécessaires à l'exploitation de la plateforme (logging, monitoring, métrologie...)

Operations support
services

IAM

Container
Image

Orchestration

Load-Balancing

Cluster Management
& Scheduling

Discovery

Application Lifecycle

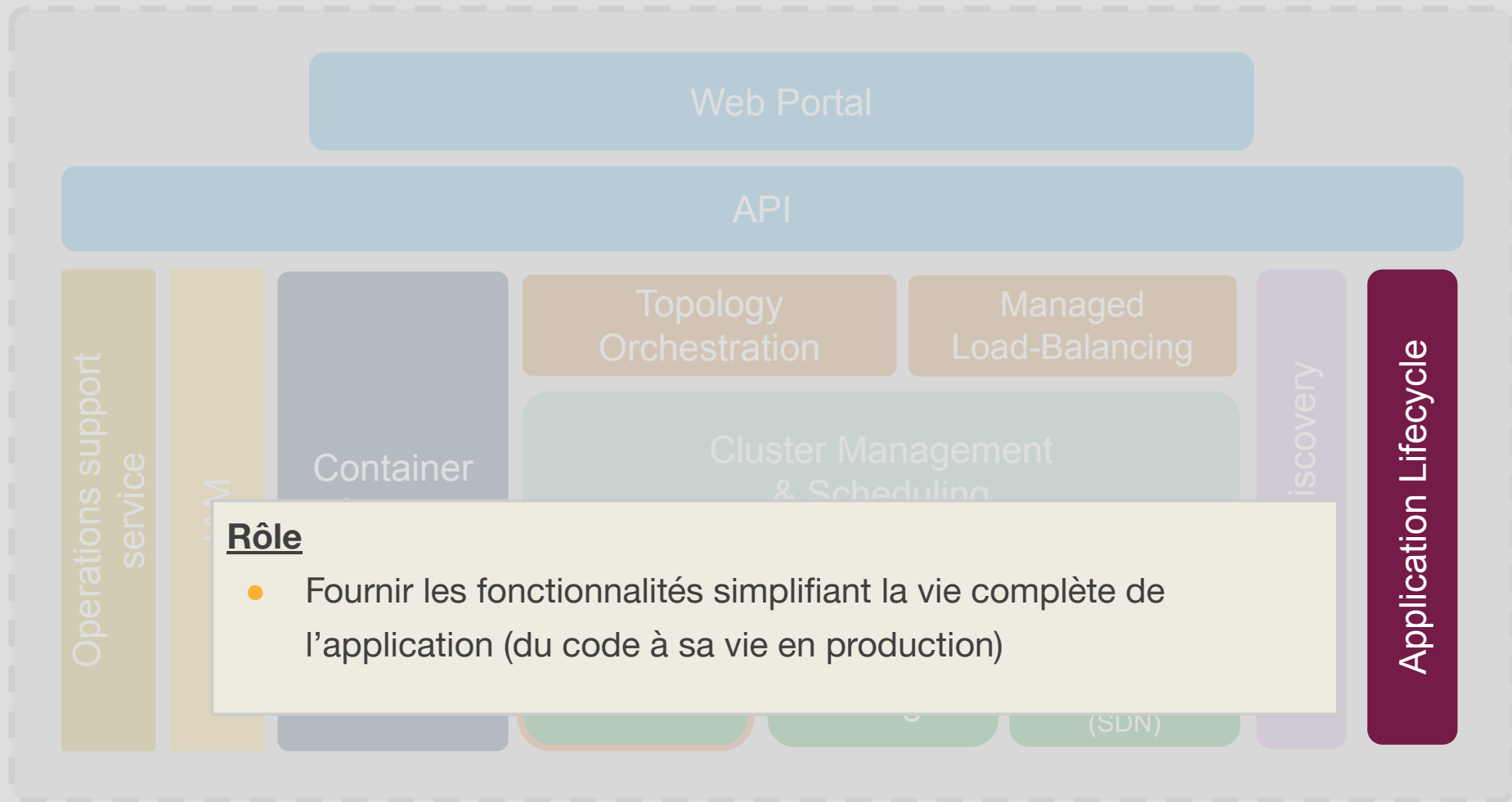
Rôle

- Fournir les fonctionnalités de gestion d'identités, de rôles et de contrôles d'accès notamment pour le support du *multi-tenant*

Cartographie fonctionnelle



Application Lifecycle



Cartographie fonctionnelle



Docker, en 2014/2015...

Non
implémenté

Implémentation
partielle

Implémentation
complète

Topology
Orchestration



Web Portal

API

Managed
Load-Balance



Swarm

Cluster Management
& Scheduling

Container
Image
Catalog



Registry

Container
Engine



Engine

Persistent
Storage
Manager

Container
Network
Manager
(SDN)

Service Disco

Application Lifecycle

Operations support
services

IAM



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



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*)



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 Desktop**)

- ▷ **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)
 - > Alibaba Cloud, Digital Ocean, Linode, IBM, OVH, Scaleway, etc
 - > 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/KaaS : un puzzle à reconstituer



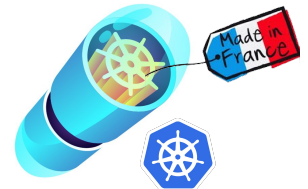
Azure Container Instances



Google Container Engine



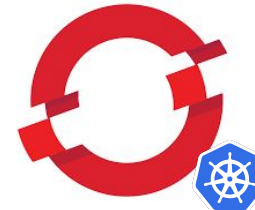
Cloud Run



Scaleway Kapsule



Docker Desktop
Mac et Windows



OPENSHIFT



RANCHER



IBM Cloud
Kubernetes Service



OVH Managed Kubernetes 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
 - ▷ Liveness et Readiness
 - ▷ Routes HTTP
 - ▷ Maîtrise des capacités
 - ▷ Monitoring applicatif
 - ▷ Log Management
- TP#4
TP#5
TP#6
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

Eco-conception

Conclusion et Take Away

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

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

Eco-conception

Conclusion et Take Away

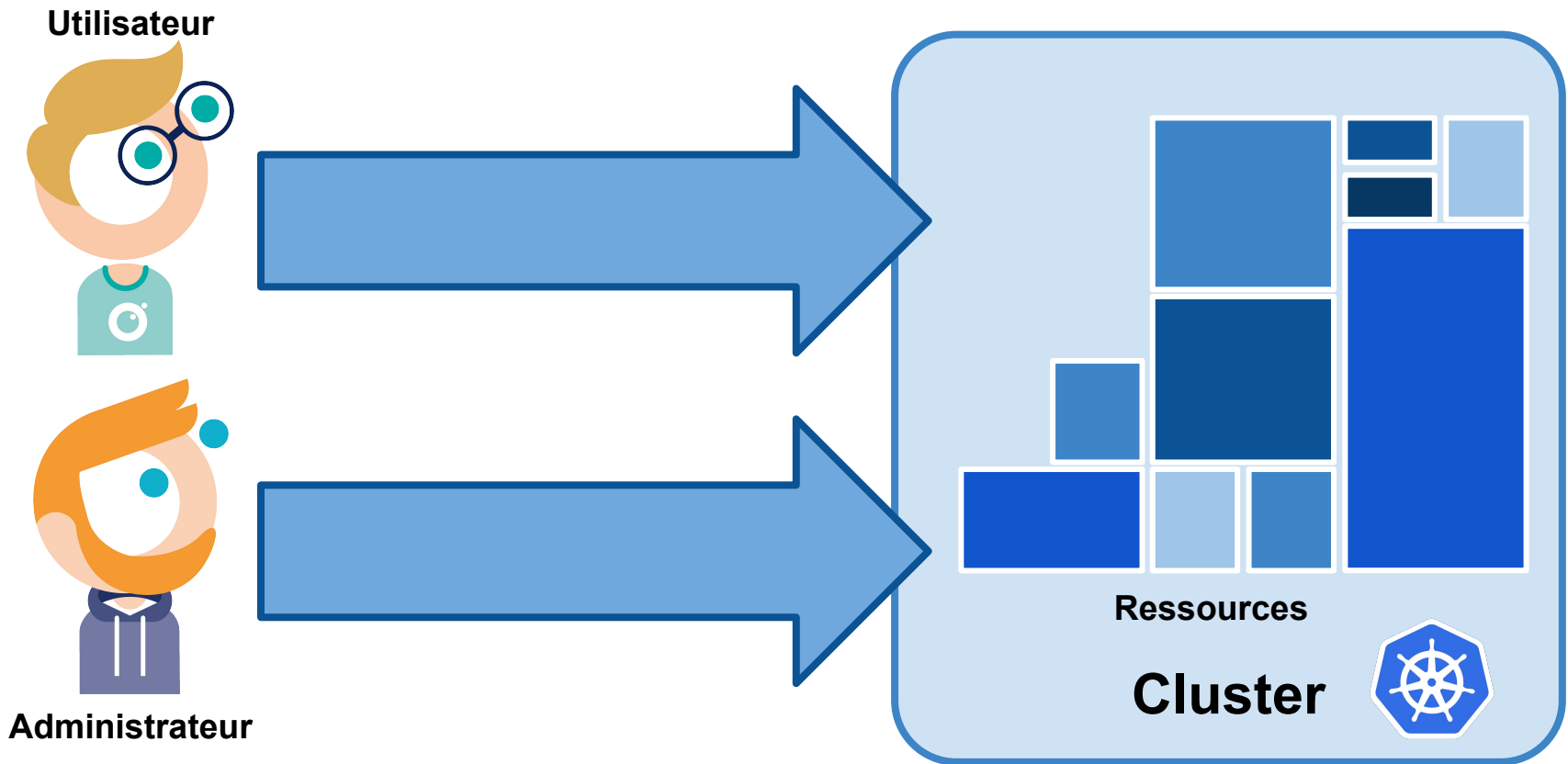
Comment interagir avec Kubernetes ?

- ▷ 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
 - > Extensible par plugins, dont un gestionnaire existe : **krew**
- ▷ Avec le **Dashboard**
 - > Visualisation (trop) simplifiée des ressources présentes dans le cluster
 - > Un “dashboard” en ligne de commande : **k9s**
 - > Alternatives intéressantes : Lens (k8slens.dev), *OpenLens*, *Weave Scope*

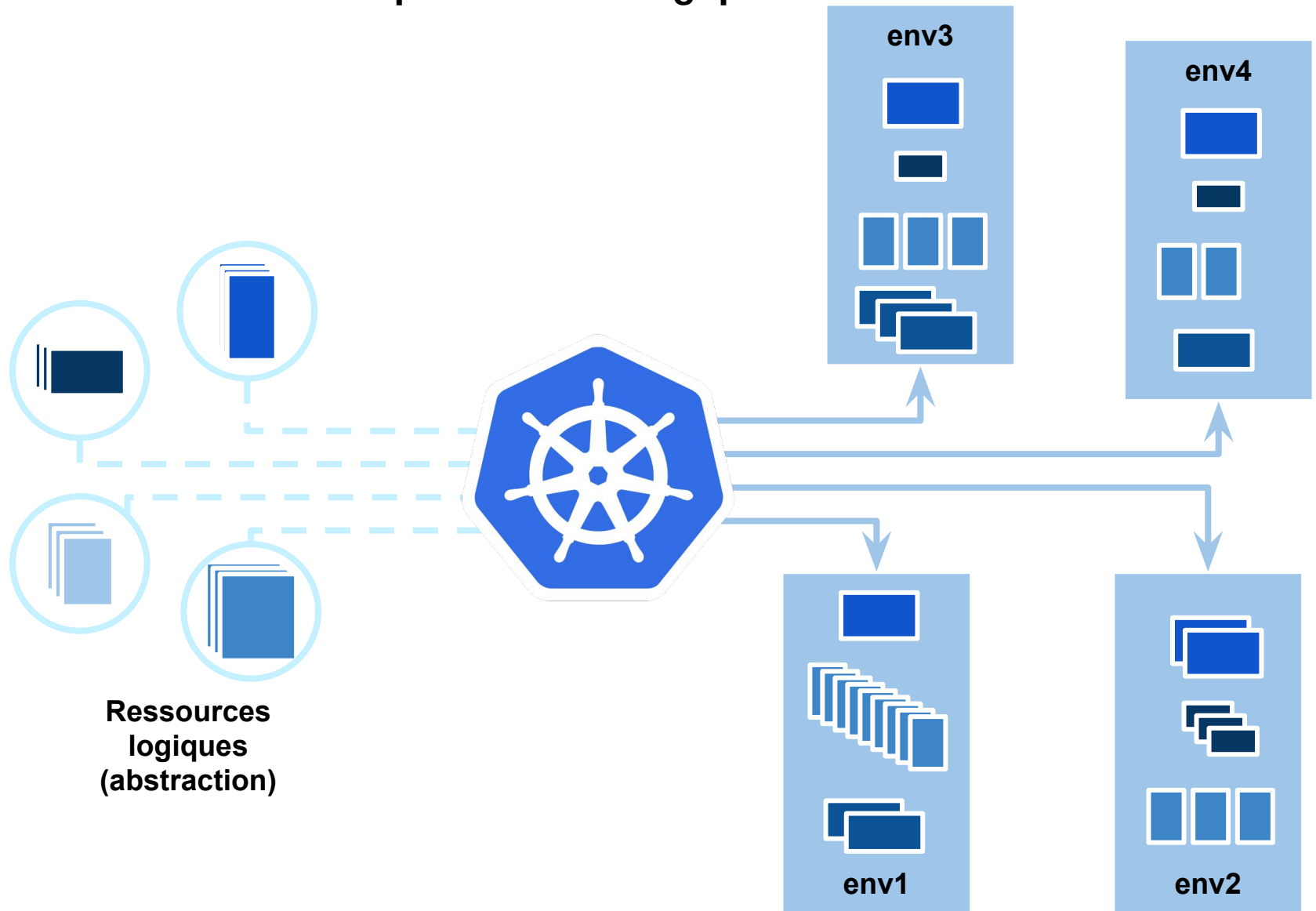


Utilisation de K8s : avant tout de la gestion de ressources

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



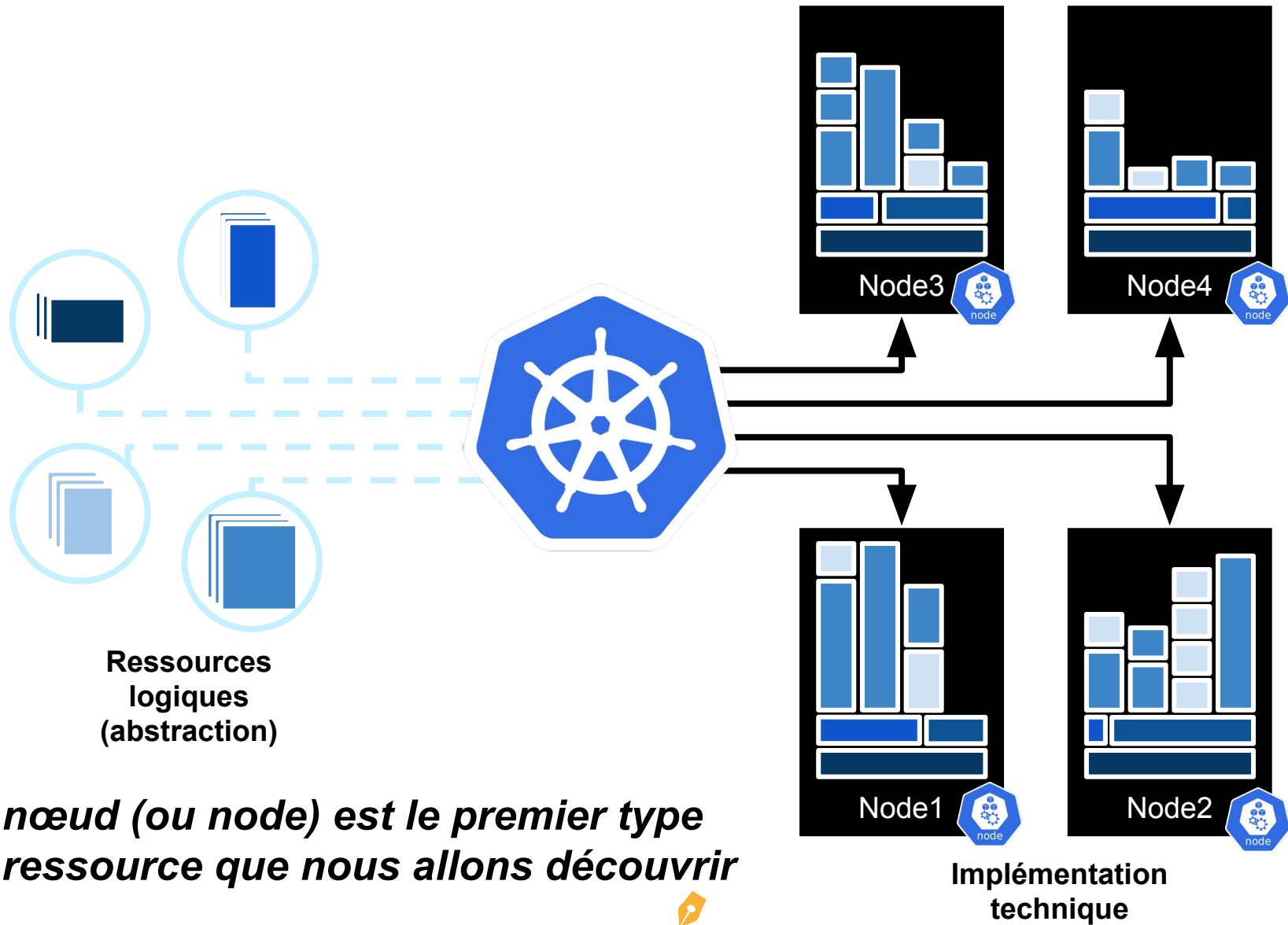
Utilisation de K8s : Implémentation logique



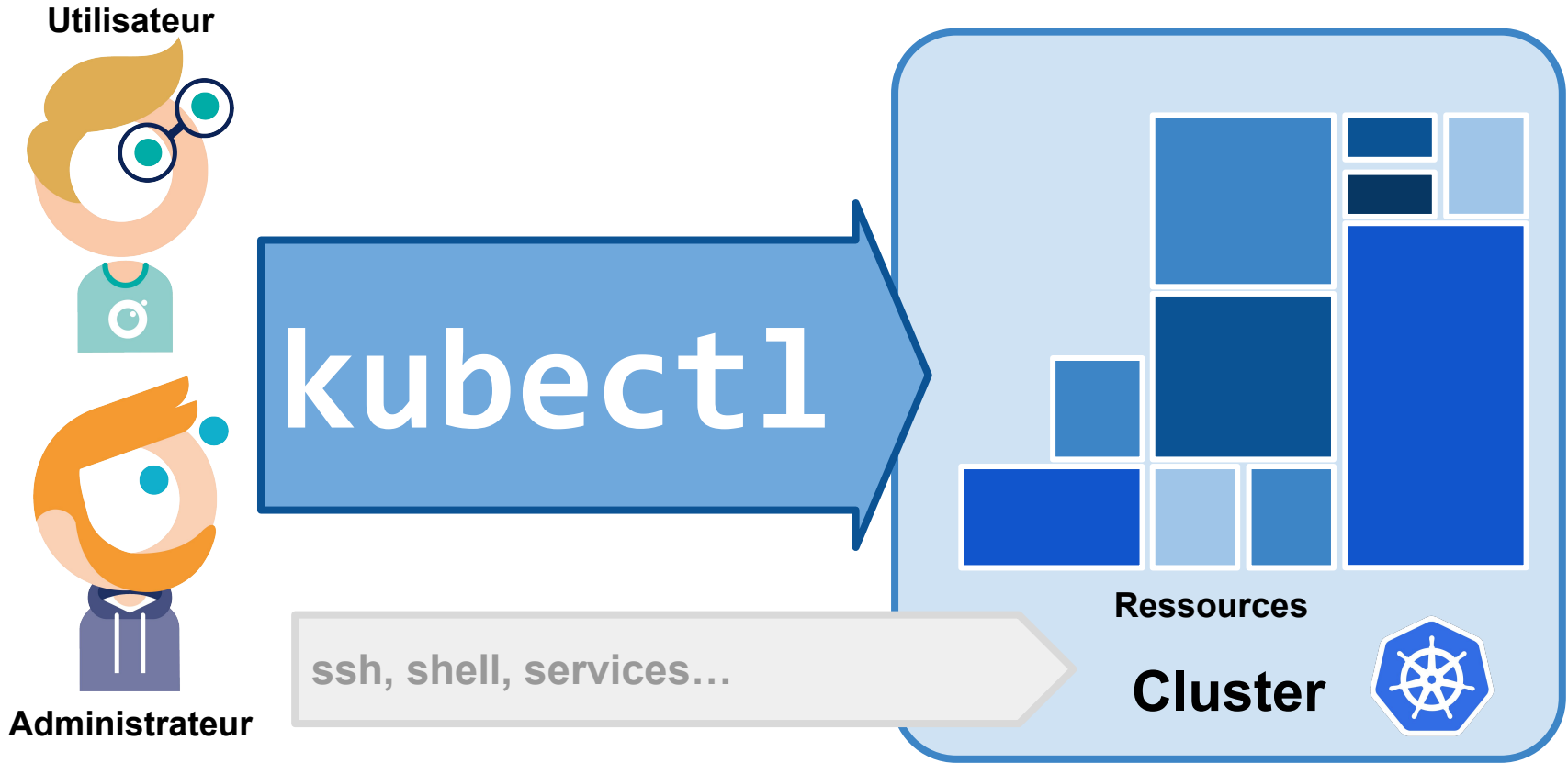
**topologies
logiques**



Utilisation de K8s : Implémentation technique



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 de version** 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 « automatique »

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

- ▶ Faire le ménage

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

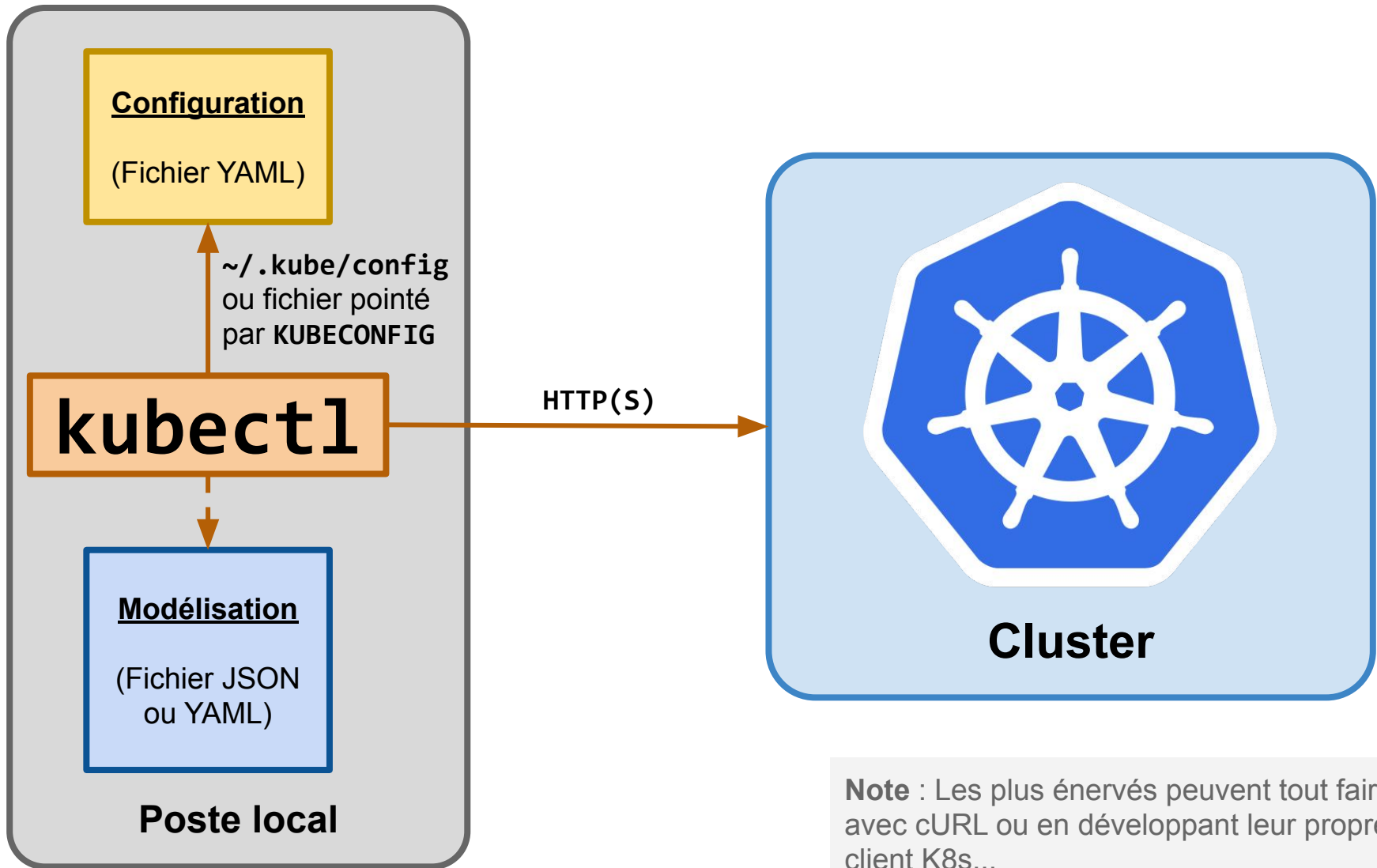
```
$ kubectl delete type6/NAME
```

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



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** (par défaut `.kube/config`) 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 ...`
 - > avec un plugin comme `kubectl ctx` (alias `kubectx`)

```
$ 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

apiVersion: v1

kind: Config

clusters:

- name: minikube

cluster:

certificate-authority: [snip]/ca.crt

server: https://192.168.42.209:8443

- name: cluster-dev

cluster:

certificate-authority-data: [snip]

server: https://35.187.116.50

users:

- name: minikube

user:

client-certificate: [snip]/client.crt

client-key: [snip]/client.key

- name: cluster-dev-spc

user:

token: [snip]

contexts:

- name: minikube

context:

cluster: minikube

user: minikube

- name: cluster-dev

context:

cluster: cluster-dev

user: cluster-dev-spc

current-context: minikube

contexte cluster-dev

contexte minikube

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

Eco-conception

Conclusion et Take Away

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

```
$ kubectl api-resources
```

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 a detailed description of that resource
(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....



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

```
$ kubectl api-resources
```

NAME	SHORTNAMES	APIGROUP	NAMESPACED	KIND
configmaps	cm		true	ConfigMap
namespaces	ns		false	Namespace
nodes	no		false	Node
persistentvolumes	pv		false	PersistentVolume
Pods	po		true	Pod
...				

alias -> kubectl get po

Indique si la ressource doit
être placée dans un
namespace



Documentation des ressources

```
$ kubectl explain pod
```

```
KIND:      Pod
```

```
VERSION:   v1
```

DESCRIPTION:

Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.

FIELDS:

`apiVersion` <string>

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info:
<https://git.k8s.io/community/contributors/devel/api-conventions.md#resources>

`kind` <string>

Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info:
<https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds>

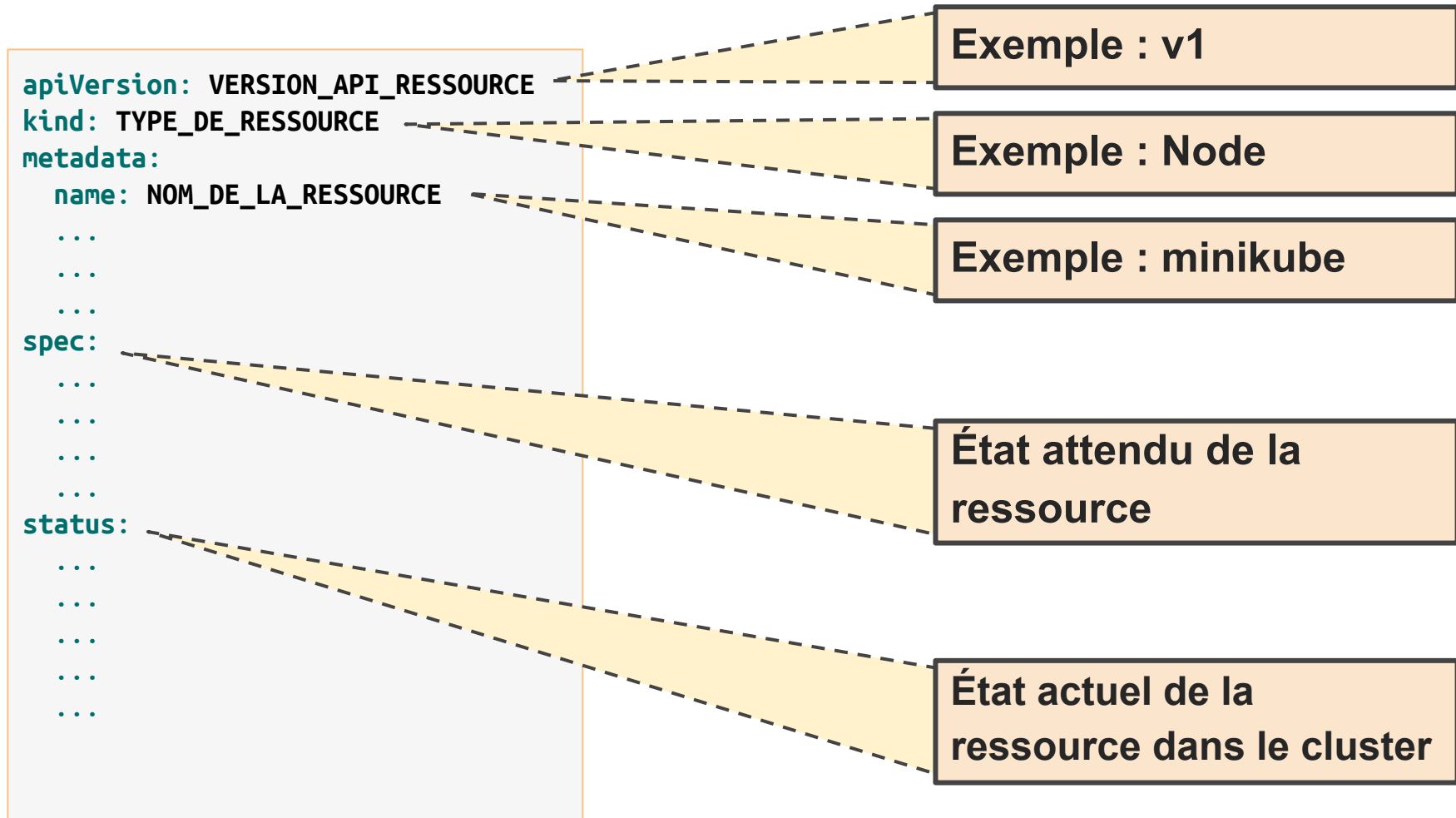
`metadata` <Object>

Standard object's metadata. More info:



Quelques généralités concernant les ressources K8s

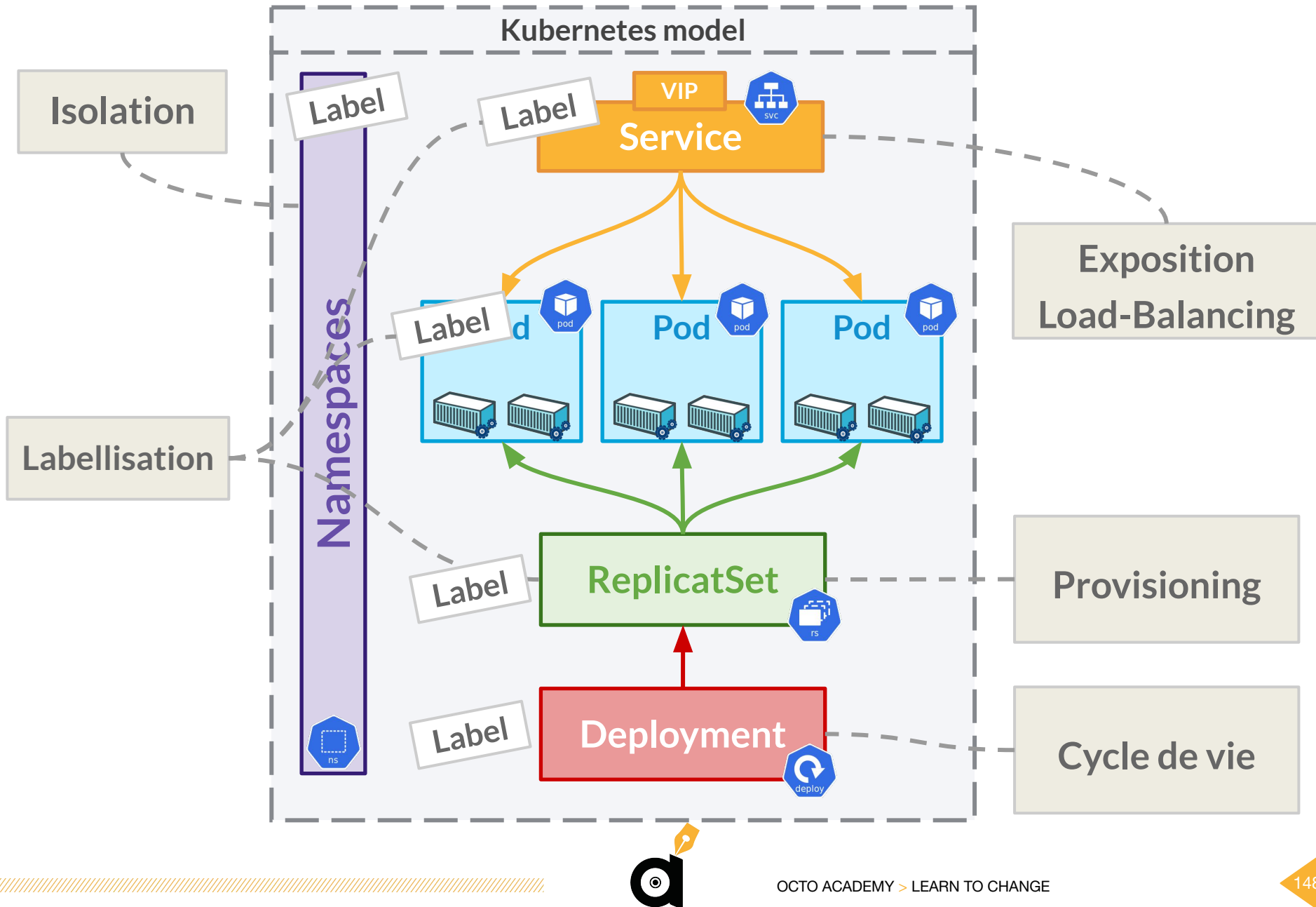
Dans Kubernetes, (presque) **toutes les ressources** sont structurées de façon identique



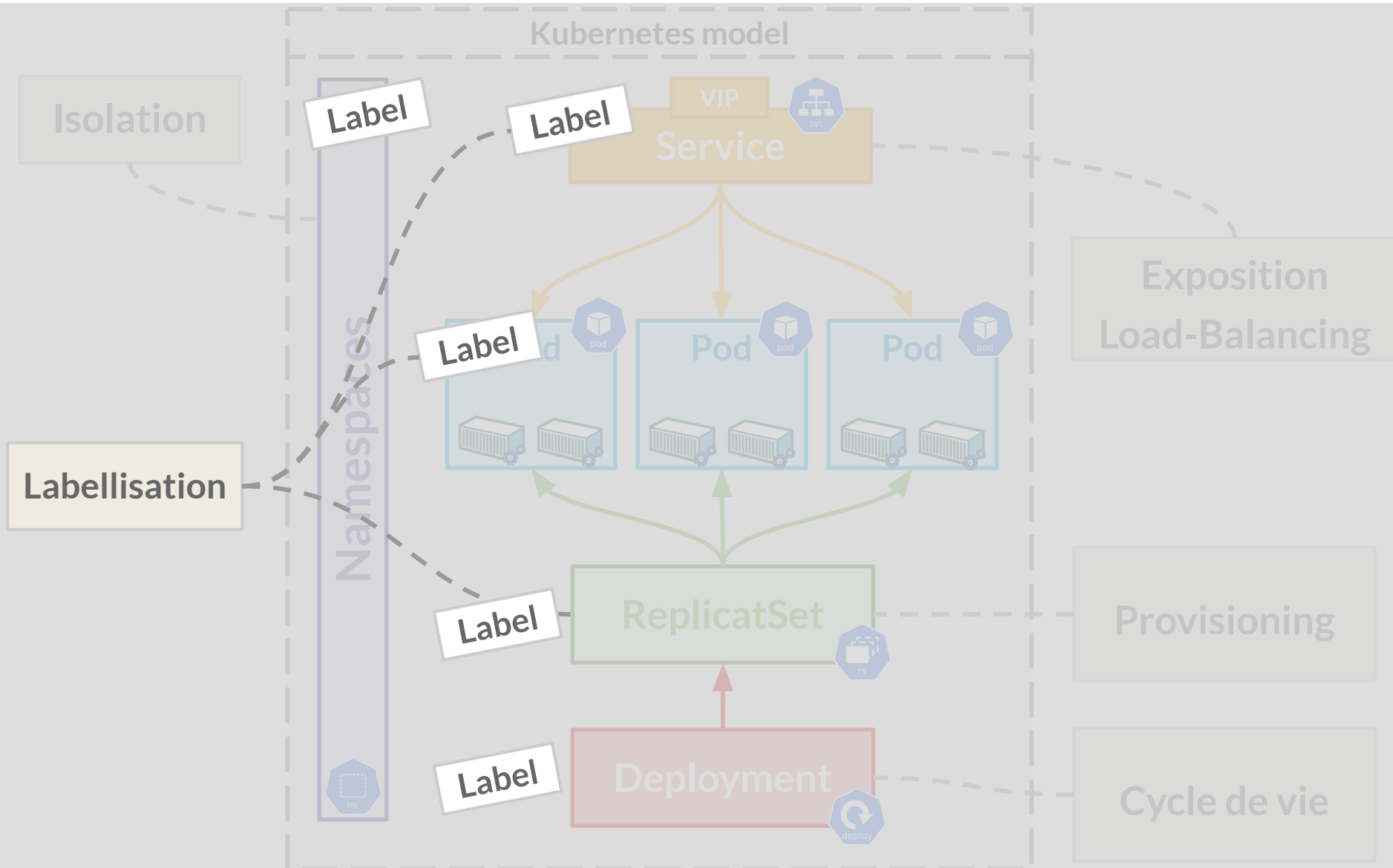
```
$ kubectl explain NOM_DE_LA_RESSOURCE --recursive
```

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 labellisées 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/node1 region=eu-west-1 zone=eu-west-1a
node "node1" labeled
```

```
$ kubectl get no -l region=eu-west-2
No resources found.
```

```
$ kubectl get no -l region=eu-west-1
NAME      STATUS    ROLES    AGE      VERSION
node1     Ready     <none>    44m      v1.8.0
```

```
$ kubectl get no -l region=eu-west-1 \
  -o=custom-columns=NAME:.metadata.name,ZONE:.metadata.labels.zone
NAME      ZONE
node1     eu-west-1a
```



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>
```

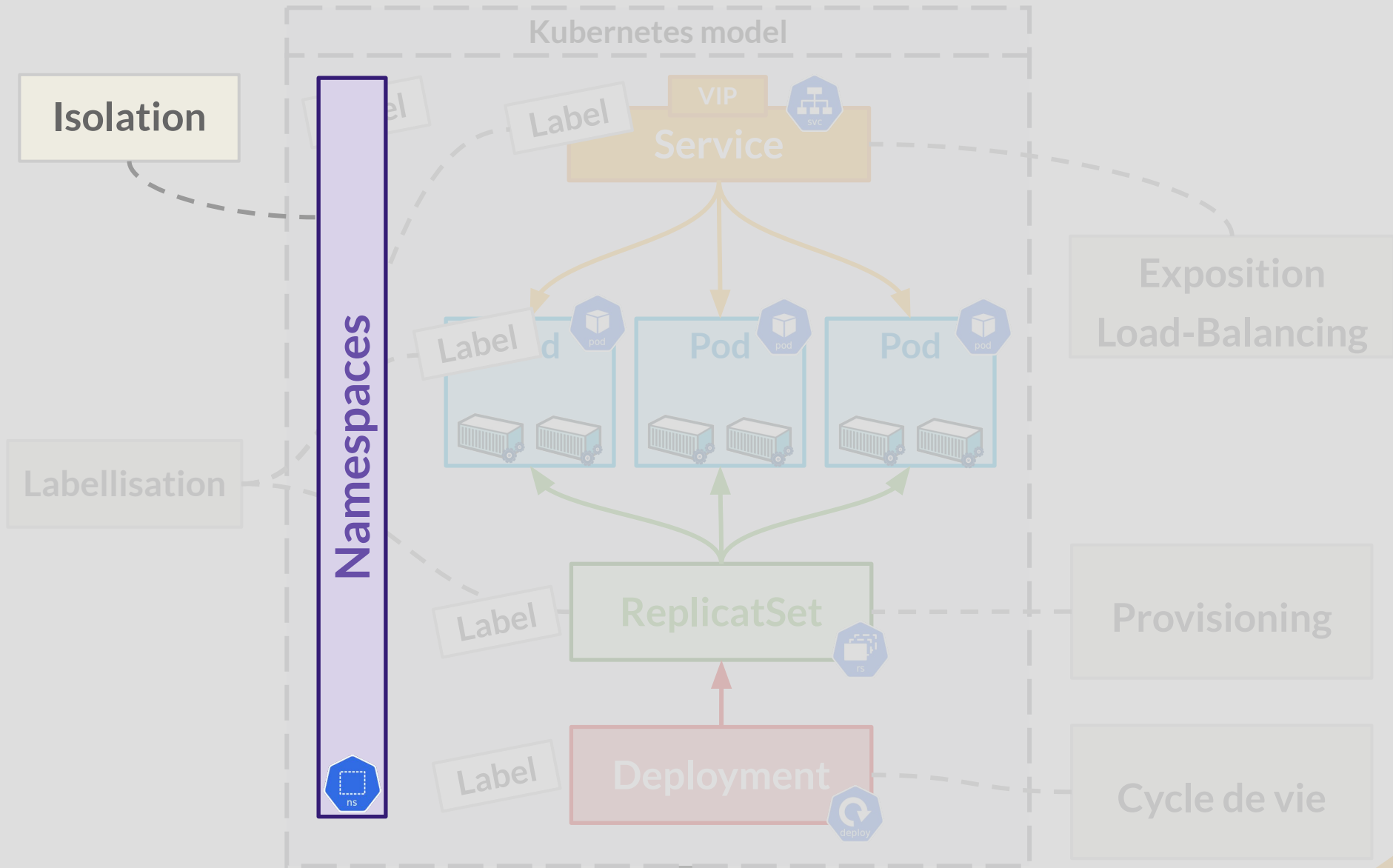


Labels vs. Annotations

- ▷ Le concept d'**annotations** est également présent 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...*





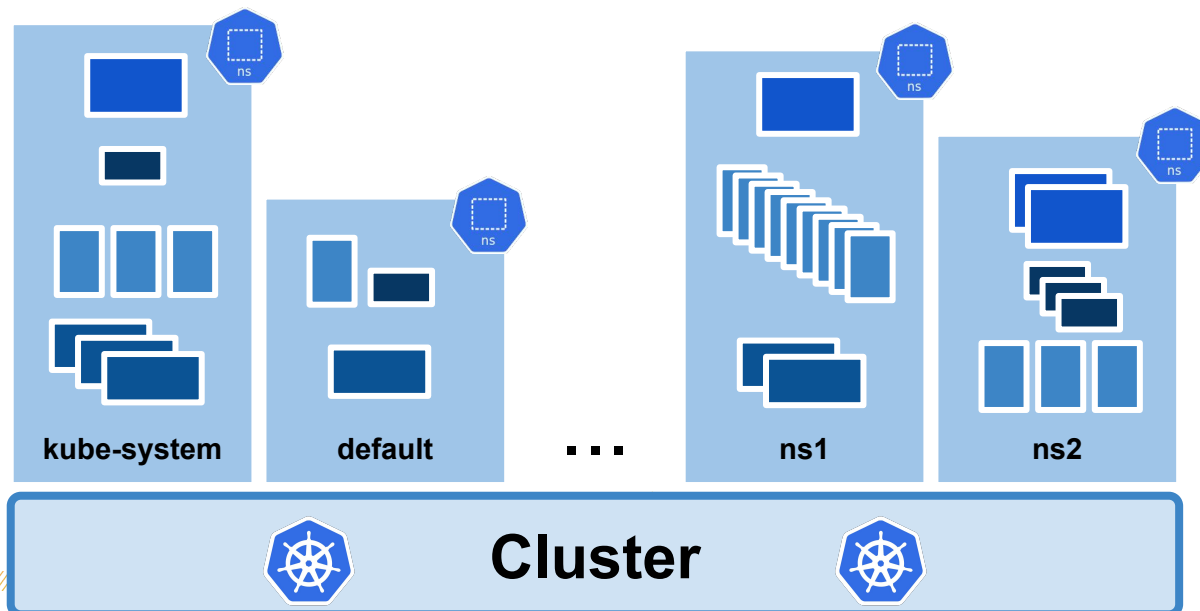
Les Namespaces (ns)

- ▷ *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**

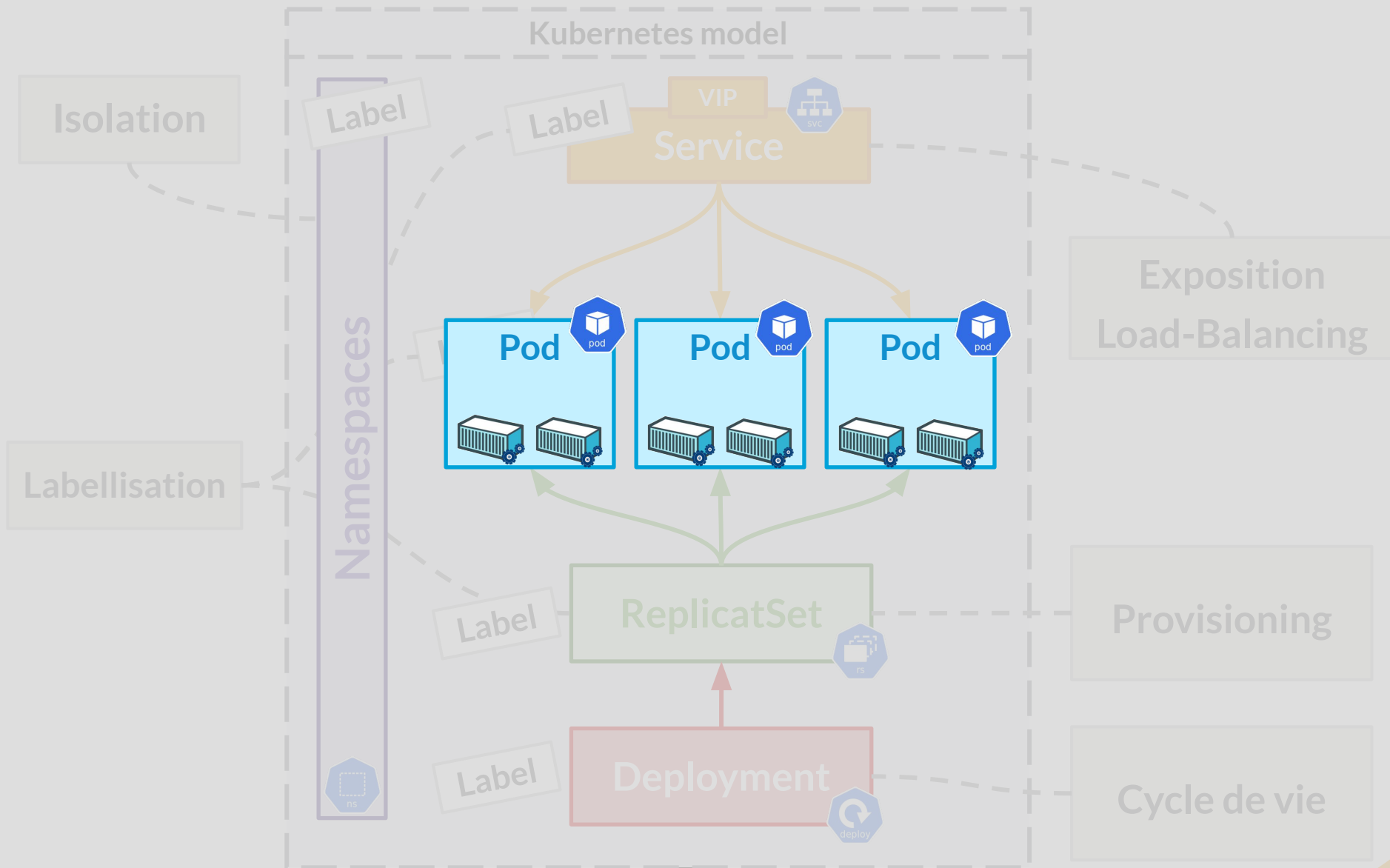


Les Namespaces (ns)

- 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)





- *Définition* : un ensemble de conteneurs ayant **un lien logique et une colocalisation**
- 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 principes du twelve-factor app**



Exemple de déclaration de Pod

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

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

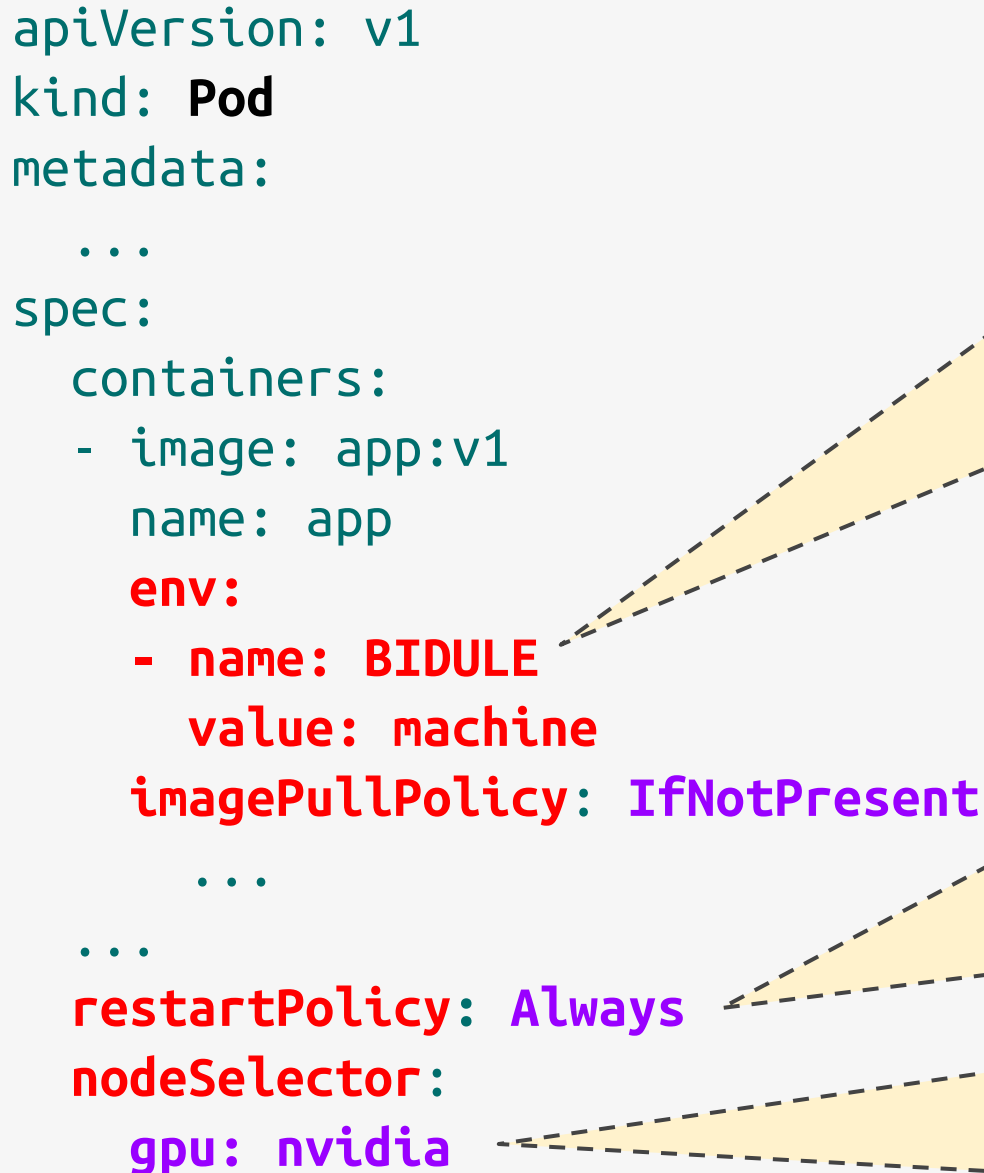
```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - image: node/app1:v1.0
    name: app1
  - image: memcached
    name: memcached
```

Liste des
conteneurs
du Pod



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:
    gpu: nvidia
```

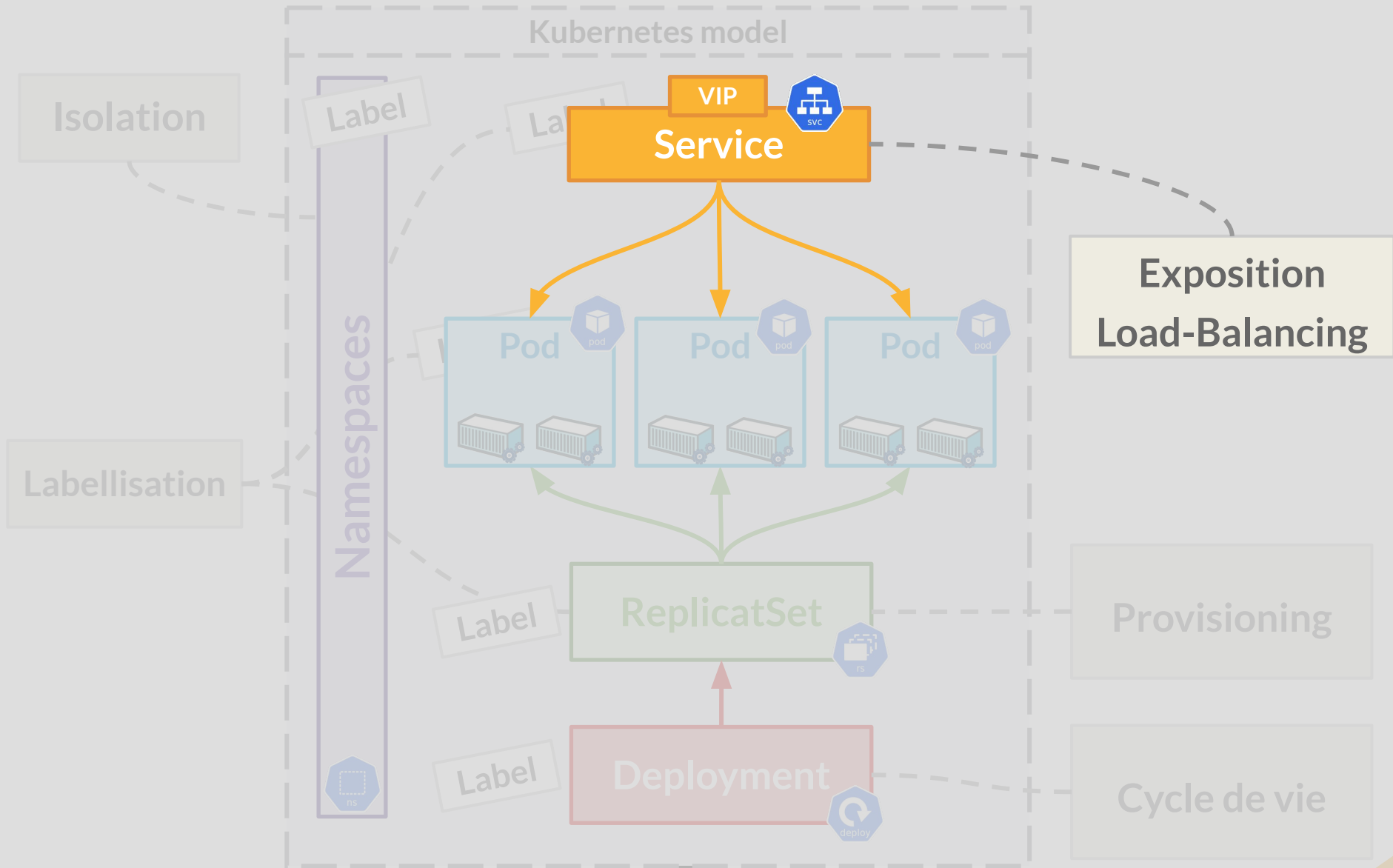


Injecter des variables d'environnement dans mon conteneur

- Always
- IfNotPresent
- Never

- Always
- OnFailure
- Never

Règle de placement sur les nœuds



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

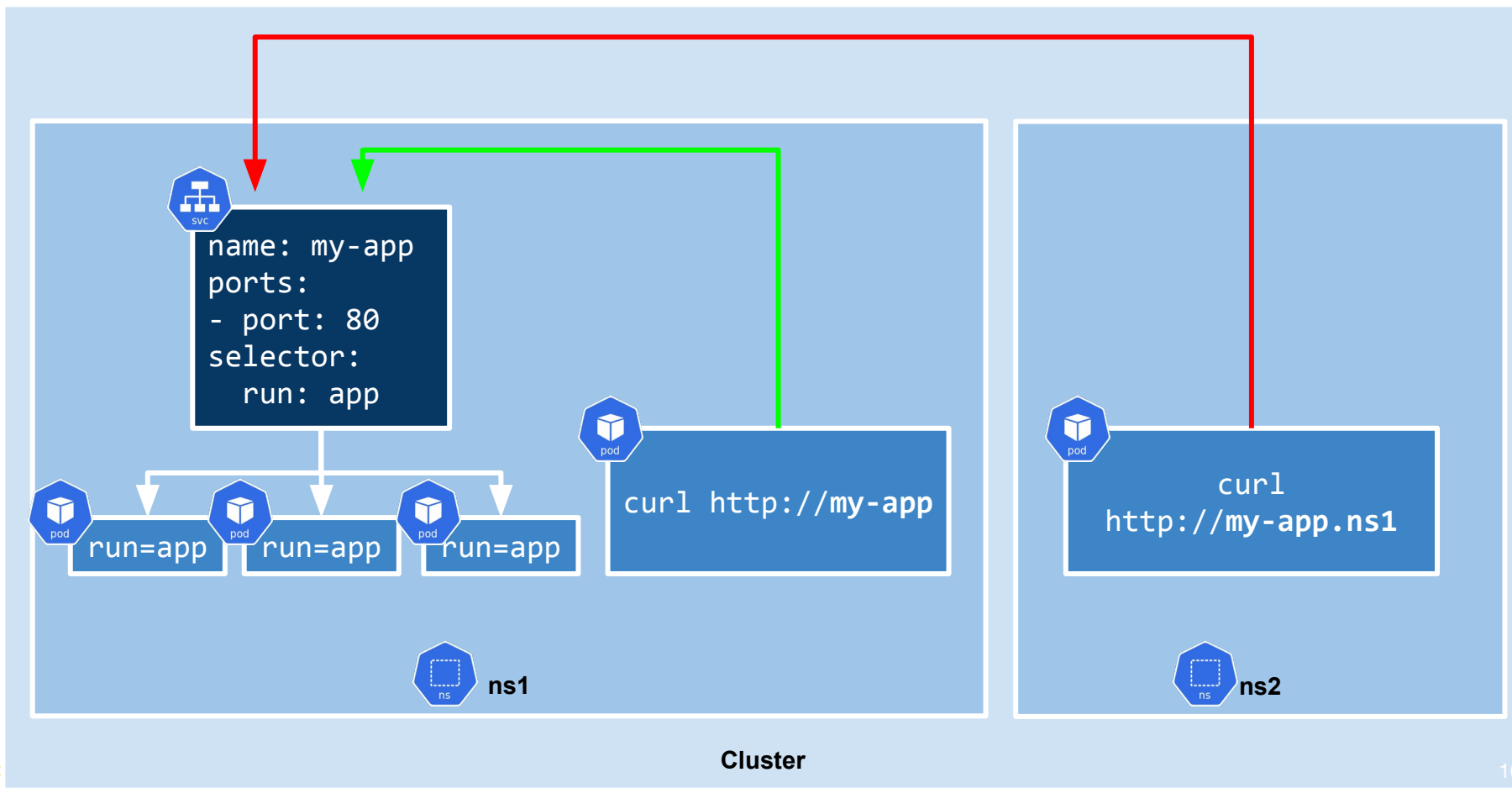
- Normalement, un pod ne parle **jamais à un autre pod directement**, il passe par un service qui l'« **expose** »
- Les autres namespaces peuvent résoudre les services des autres ns avec `${svc}.${ns}`

service

pod

namespace

cluster



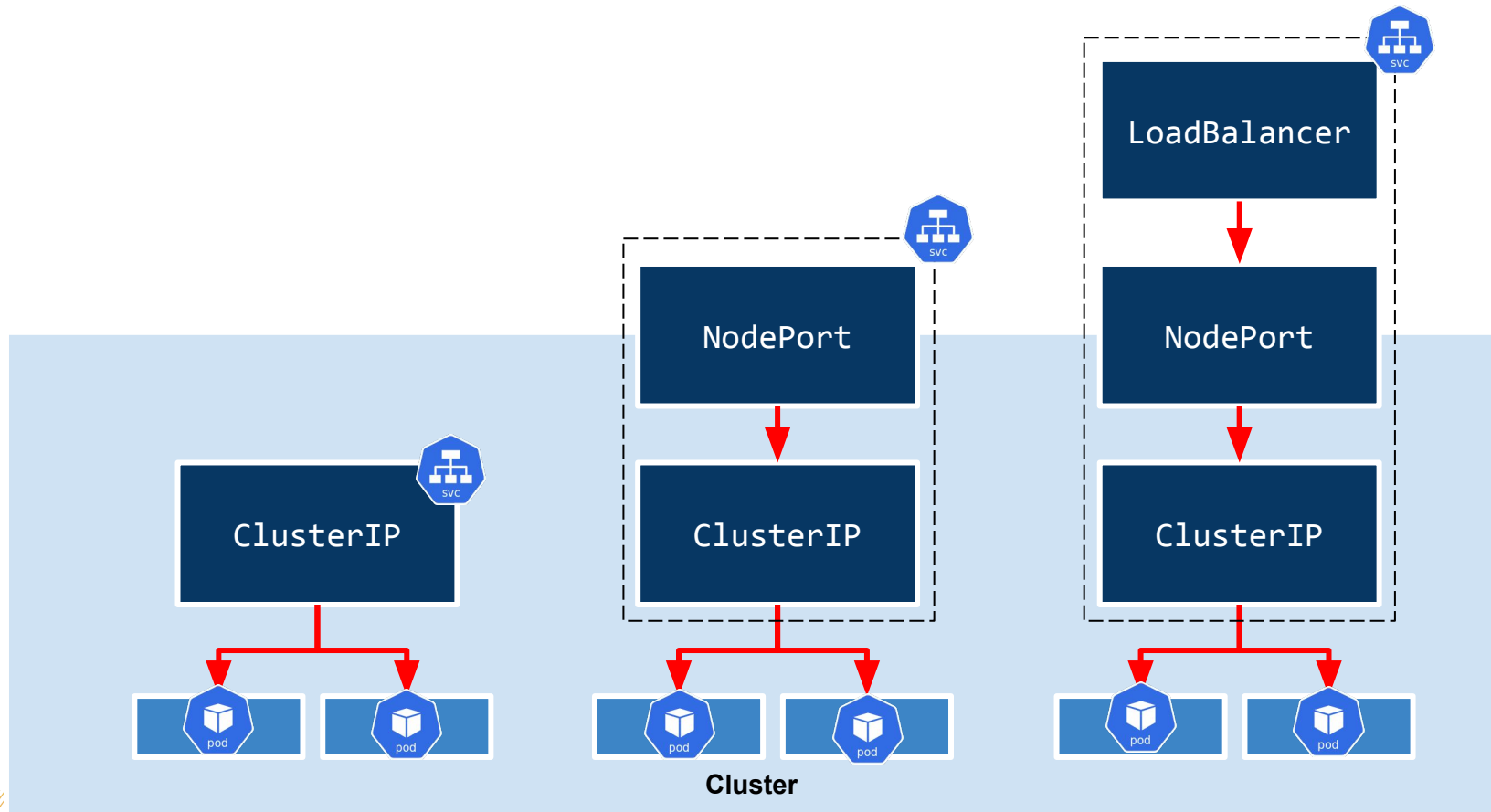
Différents types de services (1/2)

- **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*

service

pod

cluster



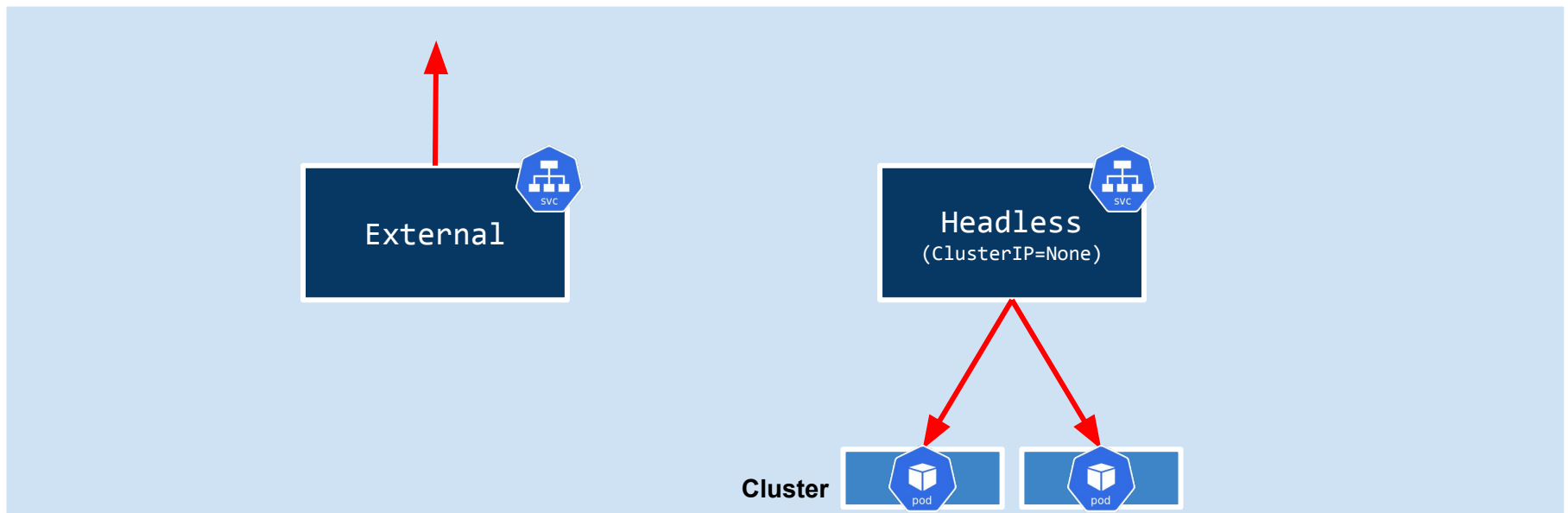
Différents types de services (2/2)

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

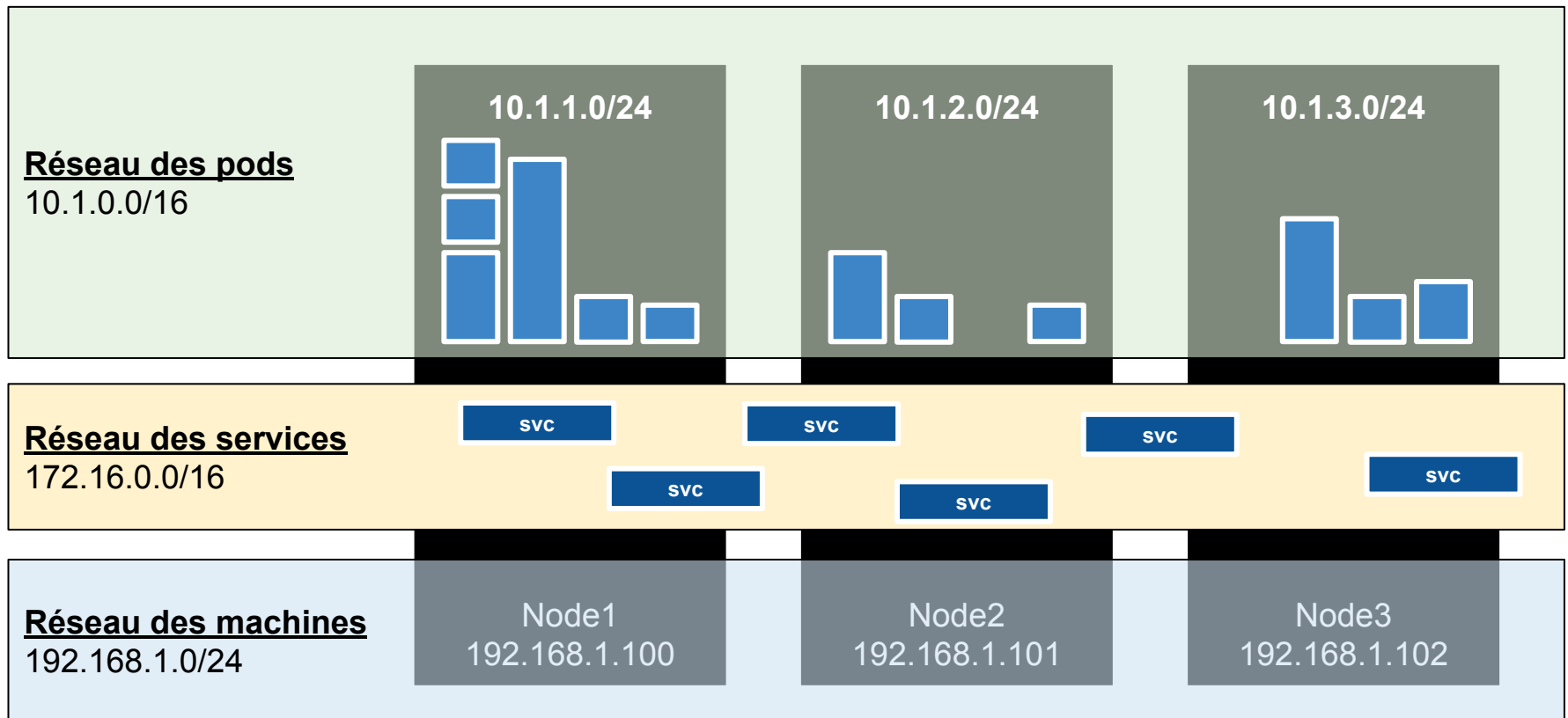
service

pod

cluster



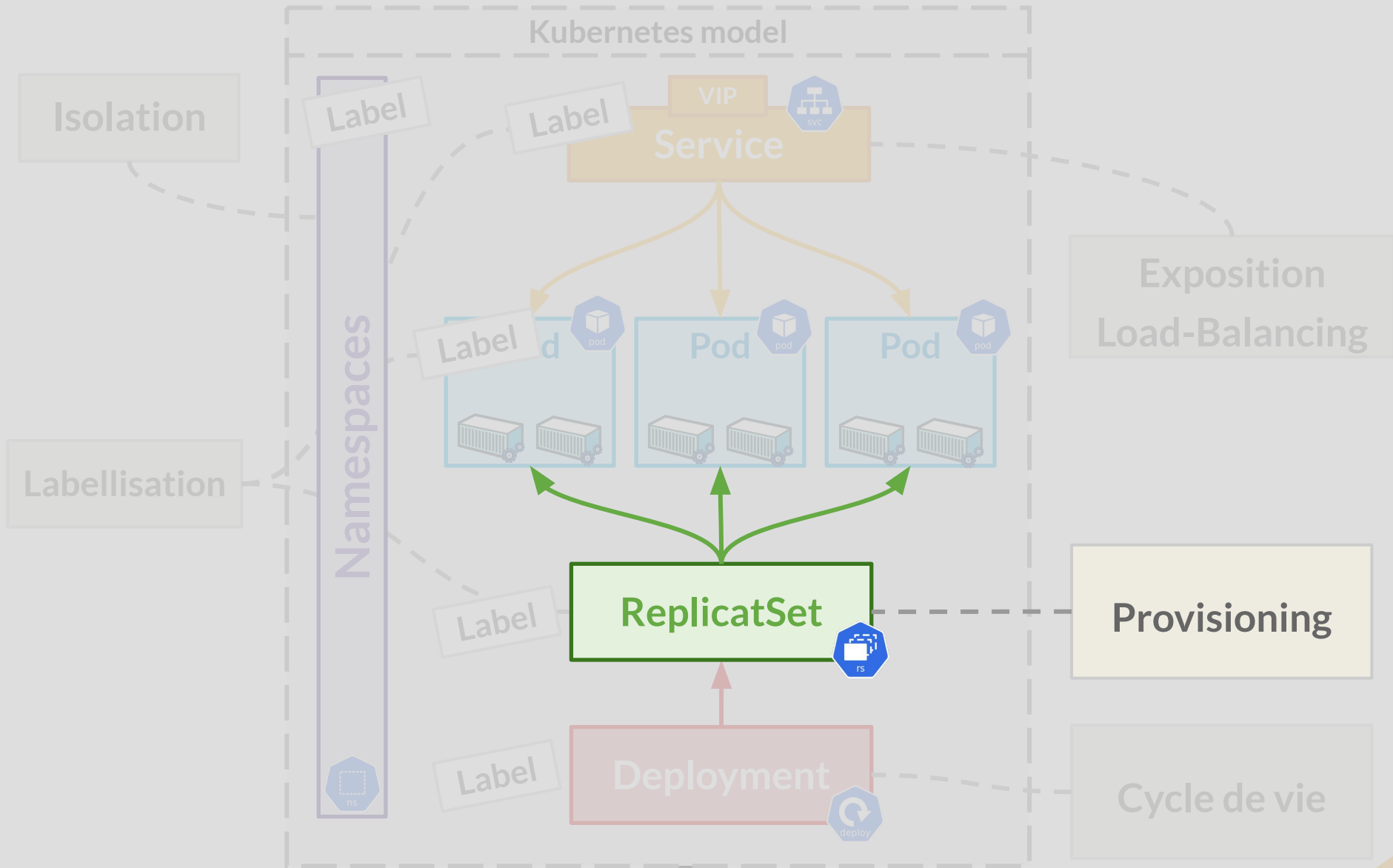
Le modèle réseau de K8s : 3 réseaux cohabitent



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-crétant 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

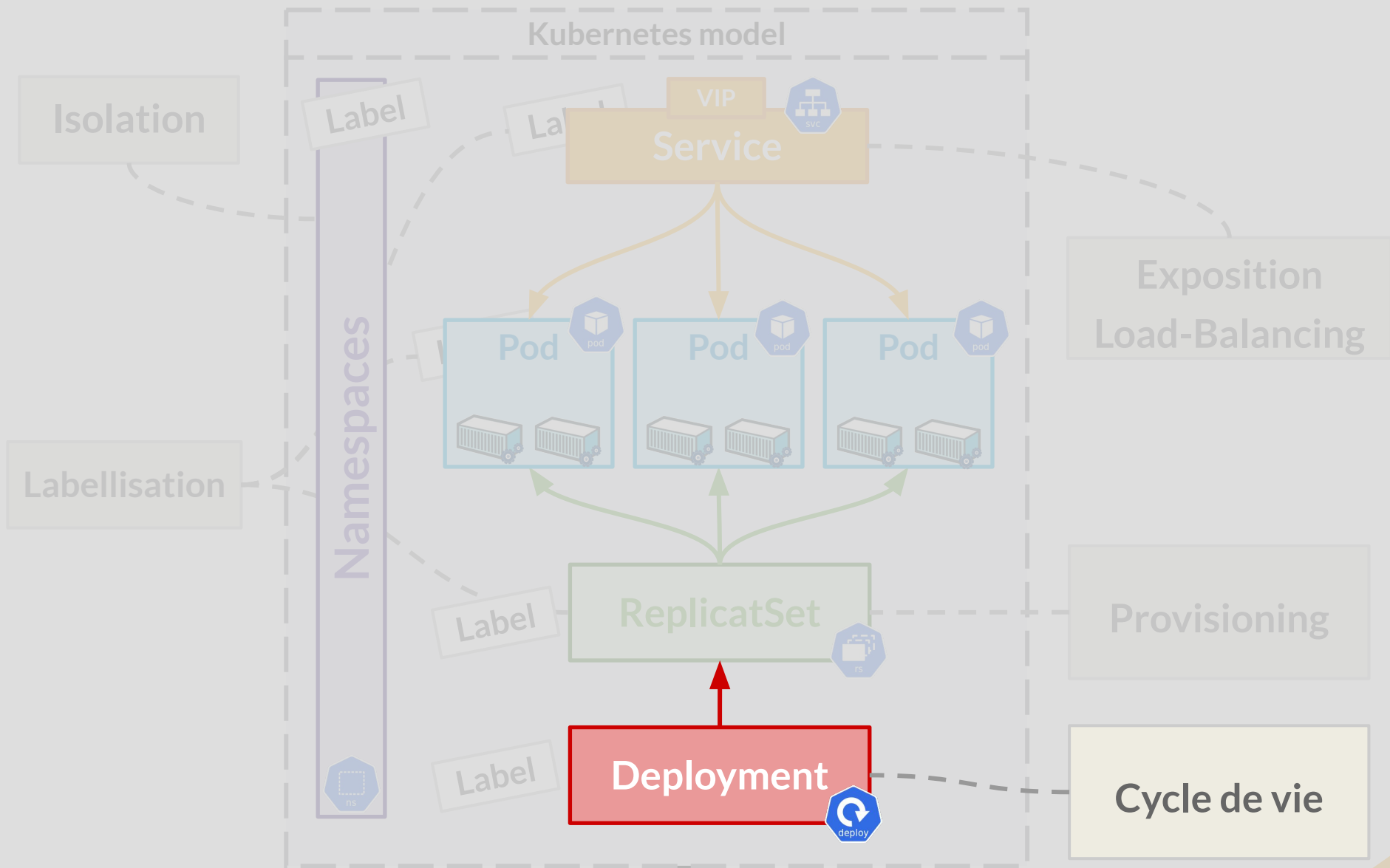
```
apiVersion: v1
kind: ReplicaSet
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
        - image: nginx:1.10
          name: nginx
          ports:
            - containerPort: 80
              protocol: TCP
```

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

**Nombre cible
et identification
des pods**



Les Deployments (deploy)



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

Légende

svc

po

ns

rs

deploy

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



Mécanisme du rolling-update : étape 2

Légende

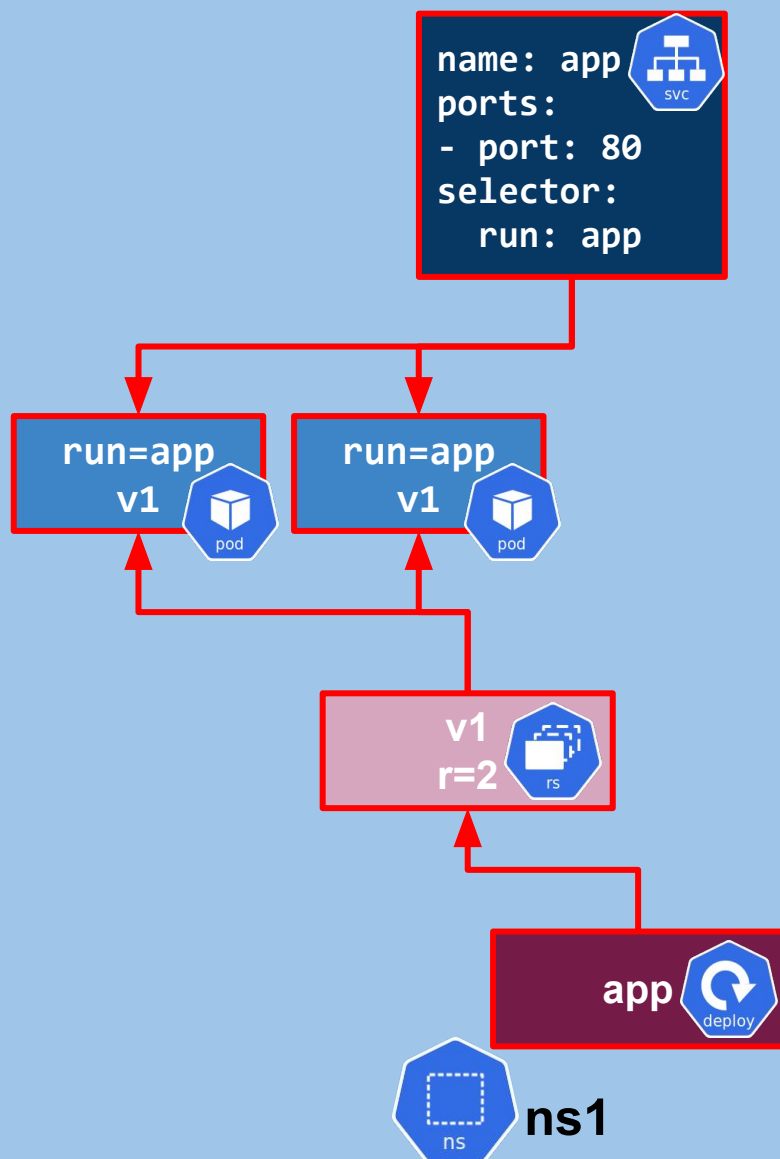
SVC

po

ns

rs

deploy



Mécanisme du rolling-update : étape 3

Légende

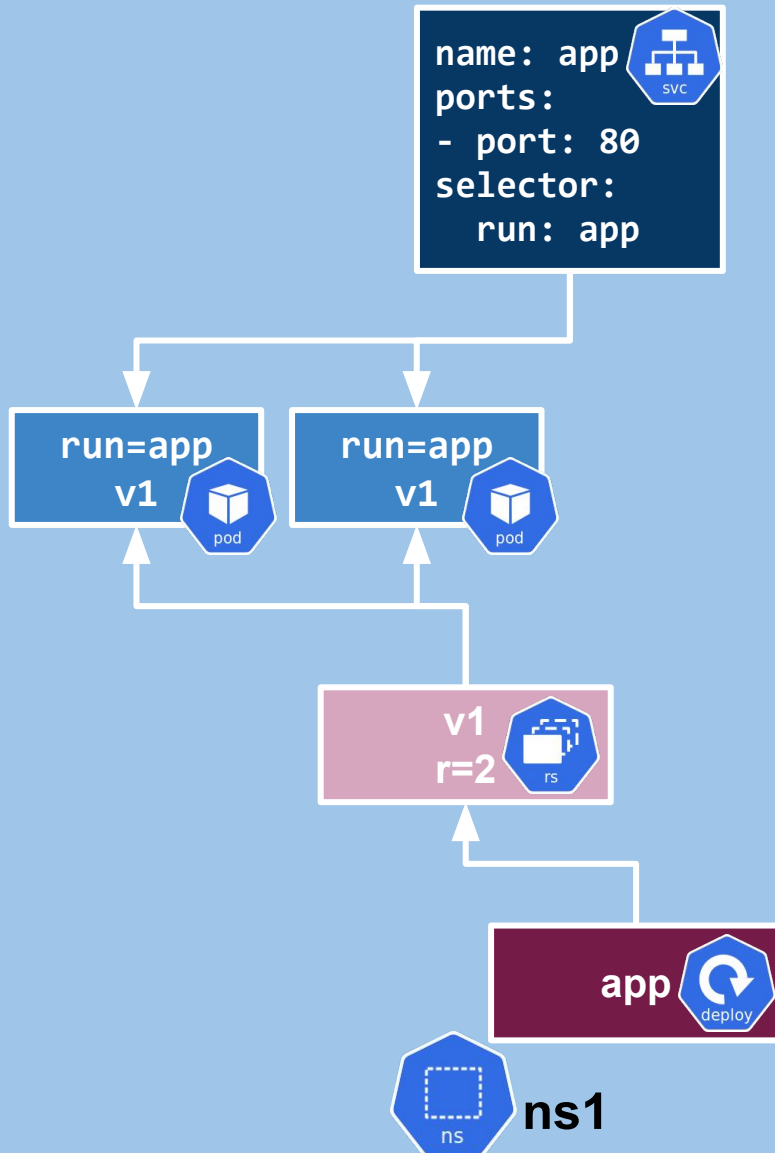
SVC

po

ns

rs

deploy



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



Mécanisme du rolling-update : étape 4

Légende

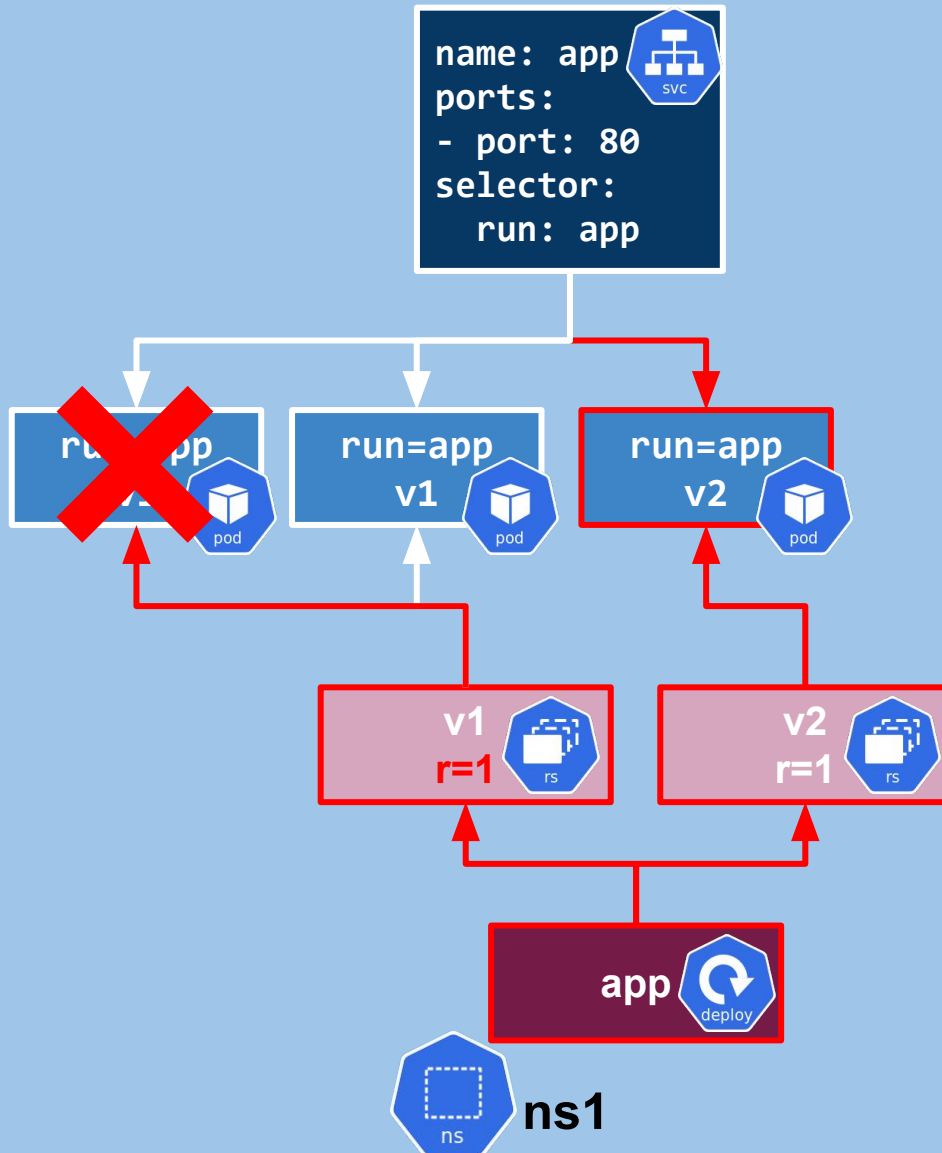
svc

po

ns

rs

deploy



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

Légende

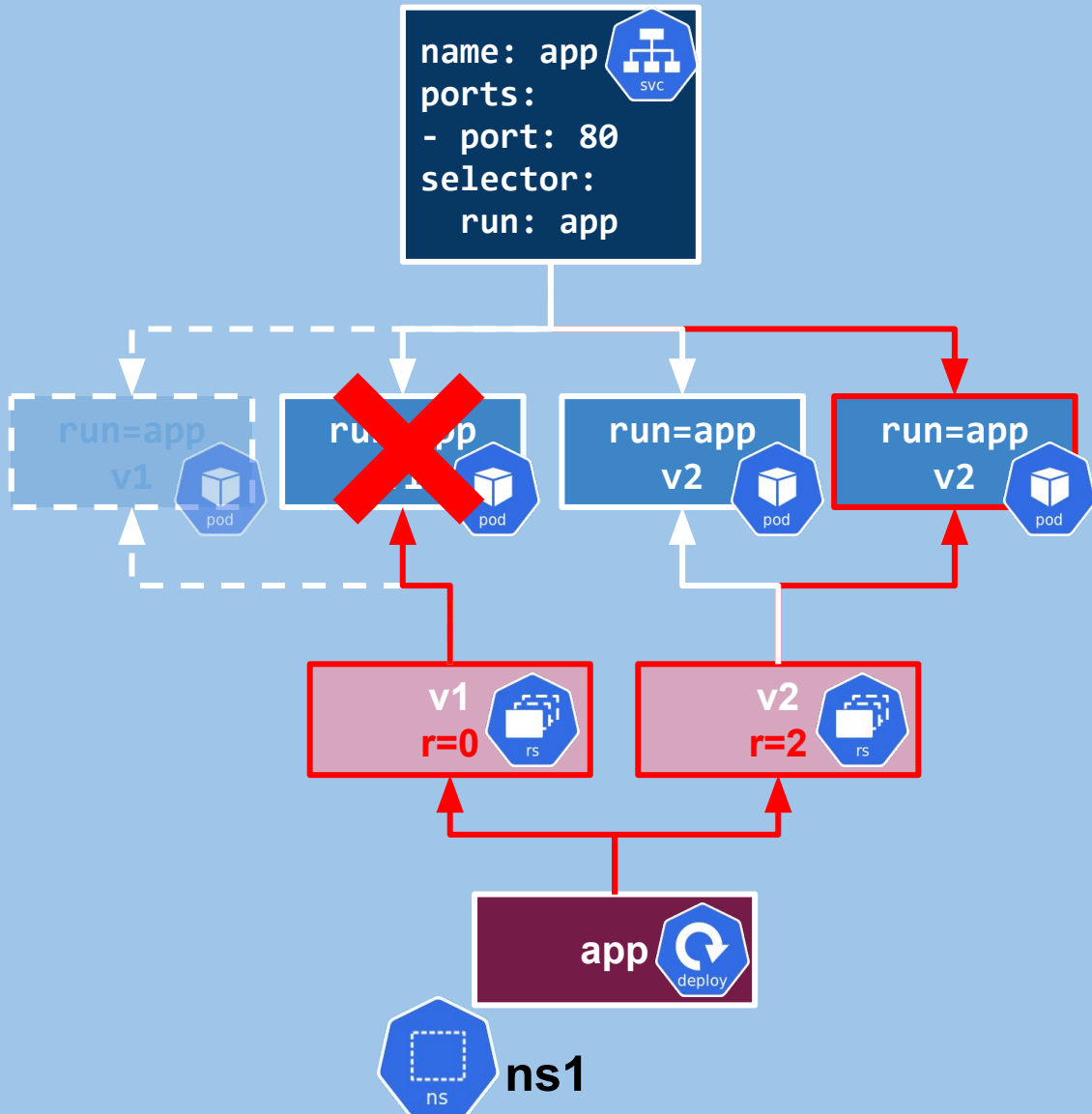
SVC

po

ns

rs

deploy



Mécanisme du rolling-update : étape 6

Légende

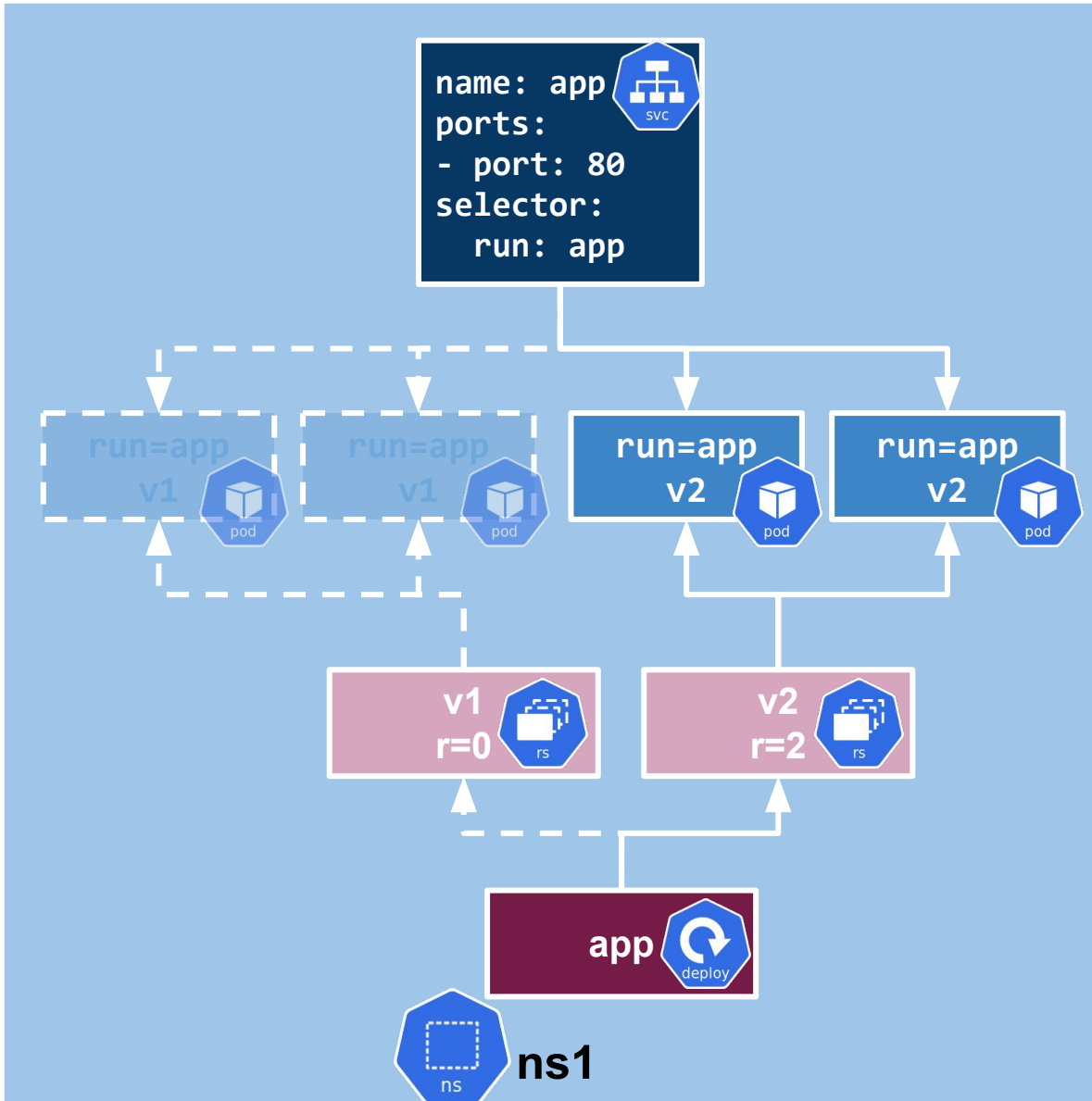
SVC

po

ns

rs

deploy



Mécanisme du rolling-update : étape 7

Légende

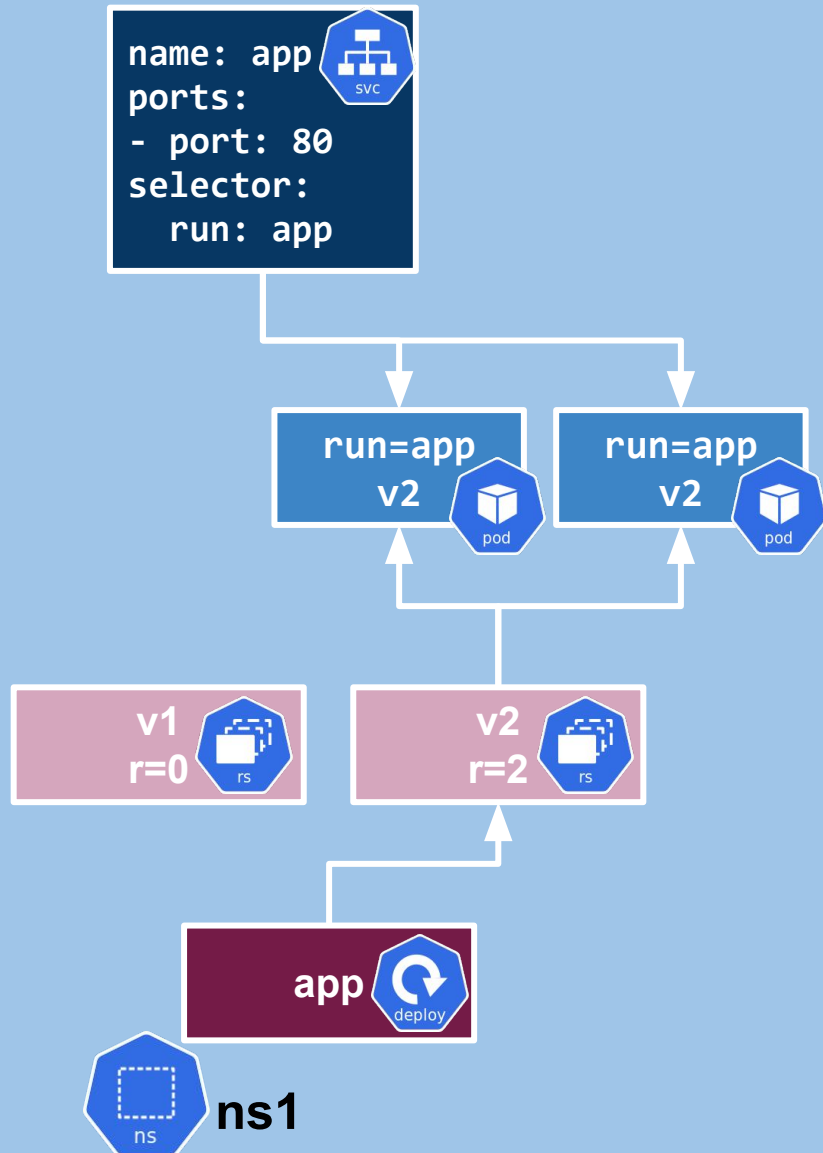
SVC

po

ns

rs

deploy



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

Eco-conception

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 données sensibles sont déclarées dans des **Secrets** afin de les mettre à disposition des applications, mais ils ne remplacent pas un coffre fort à secret
- ▷ Les **ConfigMap** sont gérées à l'identique, mais ont pour vocation à présenter des données de configuration non sensibles



Déclaration et utilisation d'un secret dans une variable d'environnement

Déclaration d'un secret

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
data:
  password: UEBzc3cwcmQ=
  username: Ym9i
type: Opaque
```

Secrets encodés
en base64

Accès au secret depuis le Pod

```
$ echo $SECRET_PASSWORD
P@ssw0rd
$ echo $SECRET_USERNAME
bob
```

Utilisation d'un secret dans un Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: app:v3.4
      env:
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: password
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: username
```


Déclaration et utilisation d'un secret dans un fichier

Déclaration d'un secret

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
data:
  password: UEBzc3cwcmQ=
  username: Ym9i
type: Opaque
```

Secrets encodés
en base64

Accès au secret depuis le Pod

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

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
```



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: mycontainer
      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 ConfigMap dans un fichier

Déclaration d'un ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: file-config
data:
  log.cfg: |
    [logs]
    level=INFO
```

Utilisation d'un cm dans un Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mycontainer
      image: app:v3.2
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
      volumes:
        - name: config-volume
          configMap:
            name: file-config
```

Accès au cm depuis le Pod

```
$ cat /etc/config/log.cfg
[logs]
level=INFO
```



TP #4