Kubernetes

LE TOUR DE TABLE

Pré-requis

Pour assister à cette formation, il est nécessaire de connaître et comprendre les notions de base associées aux conteneurs. Vous êtes capable de construire une image (par exemple avec un Dockerfile), lancer un conteneur, l'arrêter, inspecter ses logs.

- Un navigateur
- Un compte GCP (recommandé)

Agenda

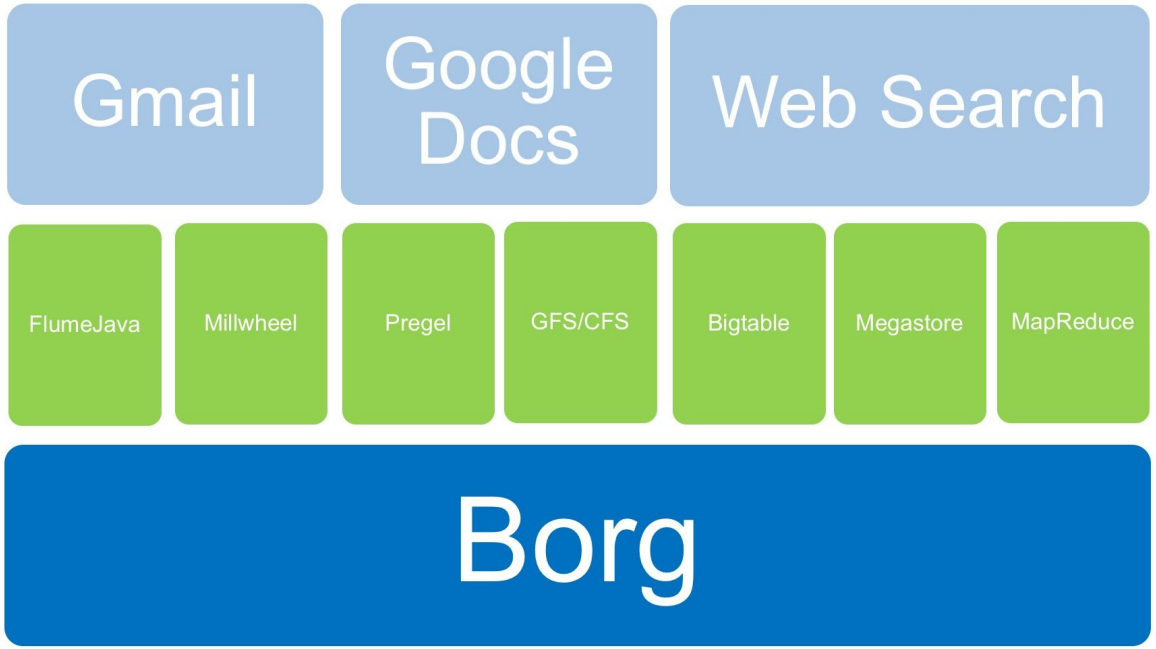
- Rappel sur Docker
- Kubernetes : les origines
- Premier aperçu
- Architecture interne
- Démarrage du cluster
- Getting started
- Mise à l'échelle et mise à jour

Test de positionnement

https://forms.gle/Q9h6iag4Q9snwJPr7

Kubernetes : les origines

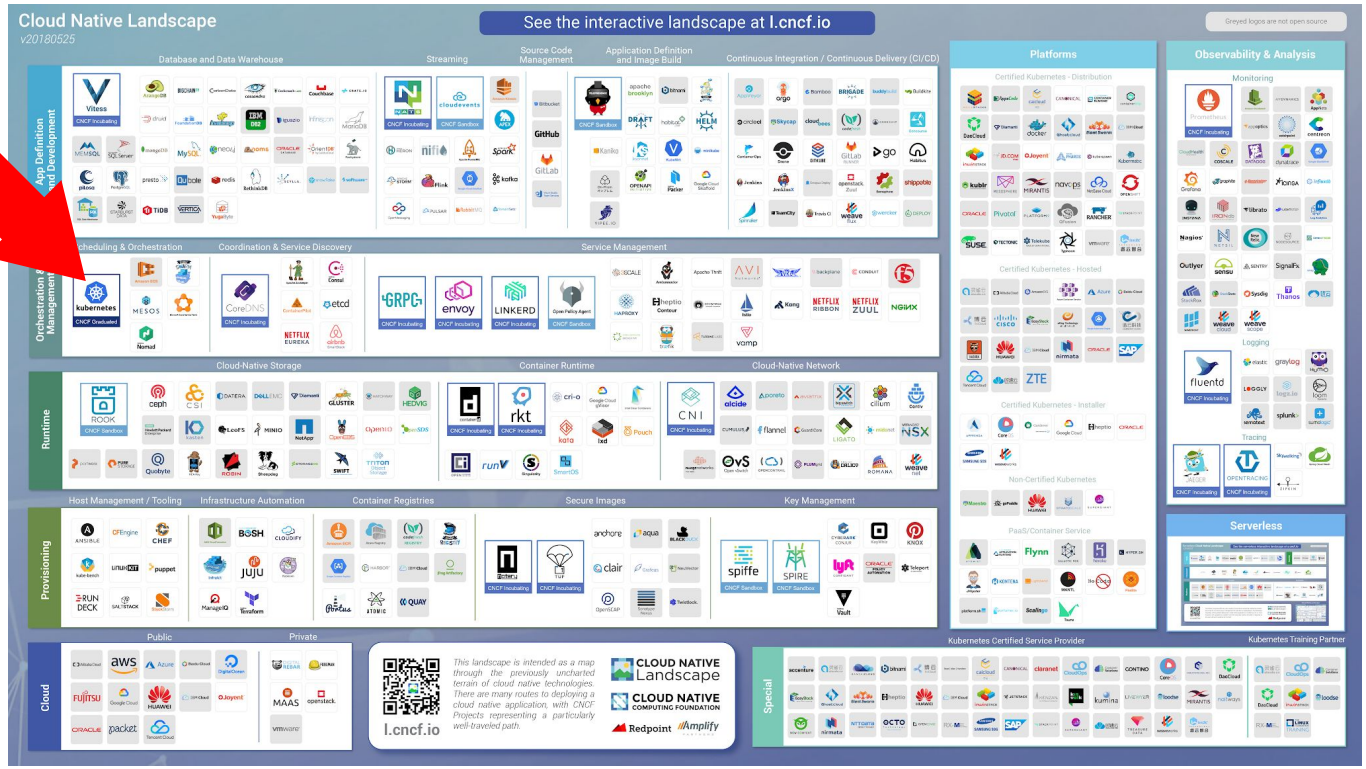
Google : 22 ans de containers



Cloud Native Computing Foundation

kubernetes

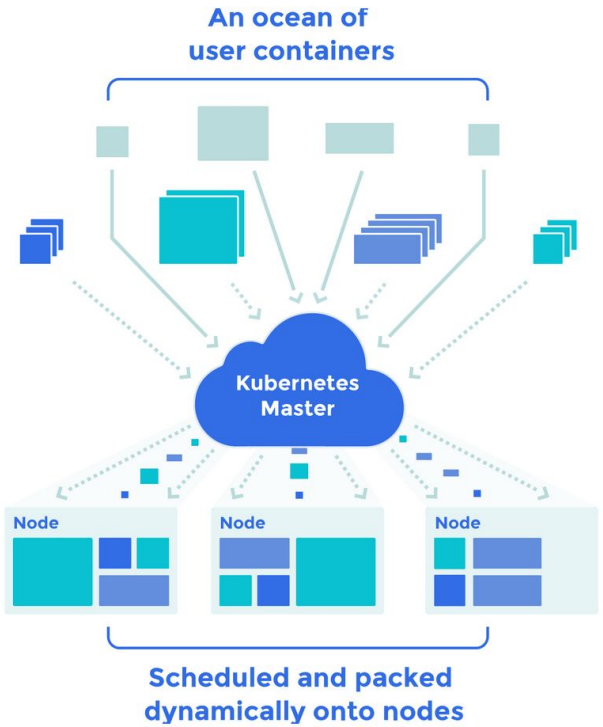
CNCF Graduated



landscape.cncf.io

[=] School

Kubernetes, pour quoi faire ?



- Lancer 5 containers basés sur l'image `redis:4.0`
- Mettre en place un load-balancer interne au cluster pour servir ces 5 containers
- Lancer 10 containers `webapp:1.0`
- Mettre en place un load-balancer public pour permettre d'accéder aux containers de l'extérieur du cluster
- Augmenter le nombre de containers `webapp` pendant les soldes 😉
- Continuer à servir les requêtes pendant la mise à jour vers `webapp:2.0`

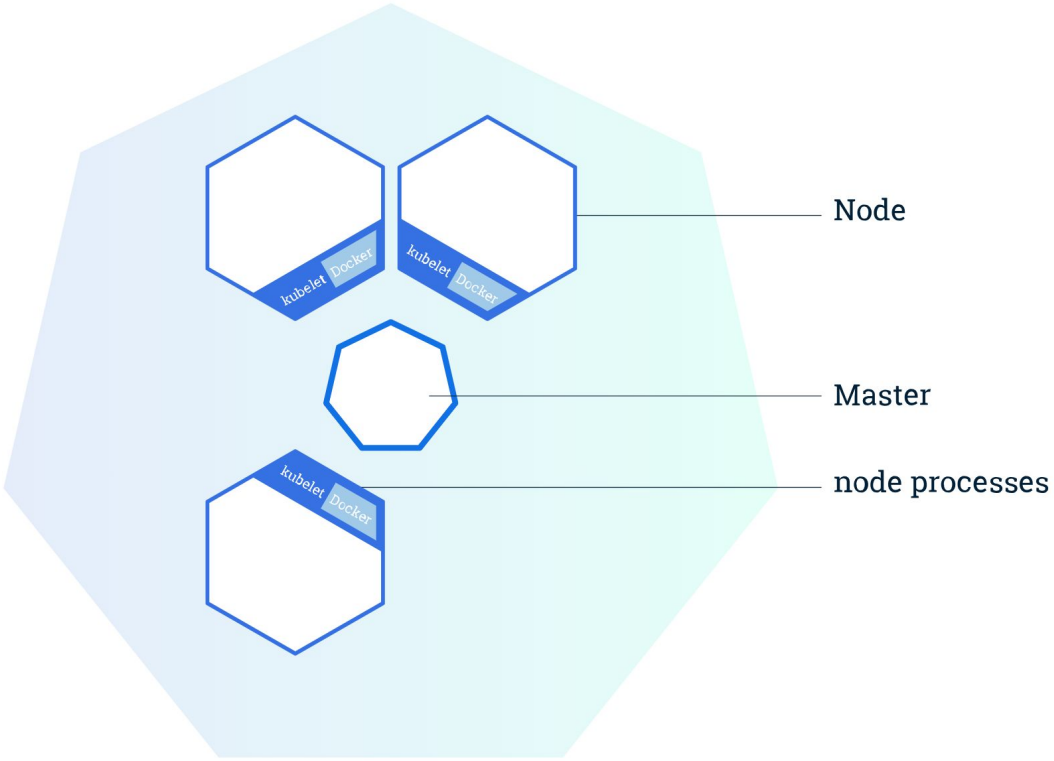
[=] School

Mais aussi

- Mise à l'échelle automatique
- Déploiement "Blue/green" et "Canary"
- Exécution de traitements unitaires ou répétés
- Prioriser les tâches en cas de manque de ressources sur le cluster
- Exécuter des services qui persistent des données sur disque
- Contrôler l'accès aux différentes ressources
- Automatiser les tâches complexes ("operators")

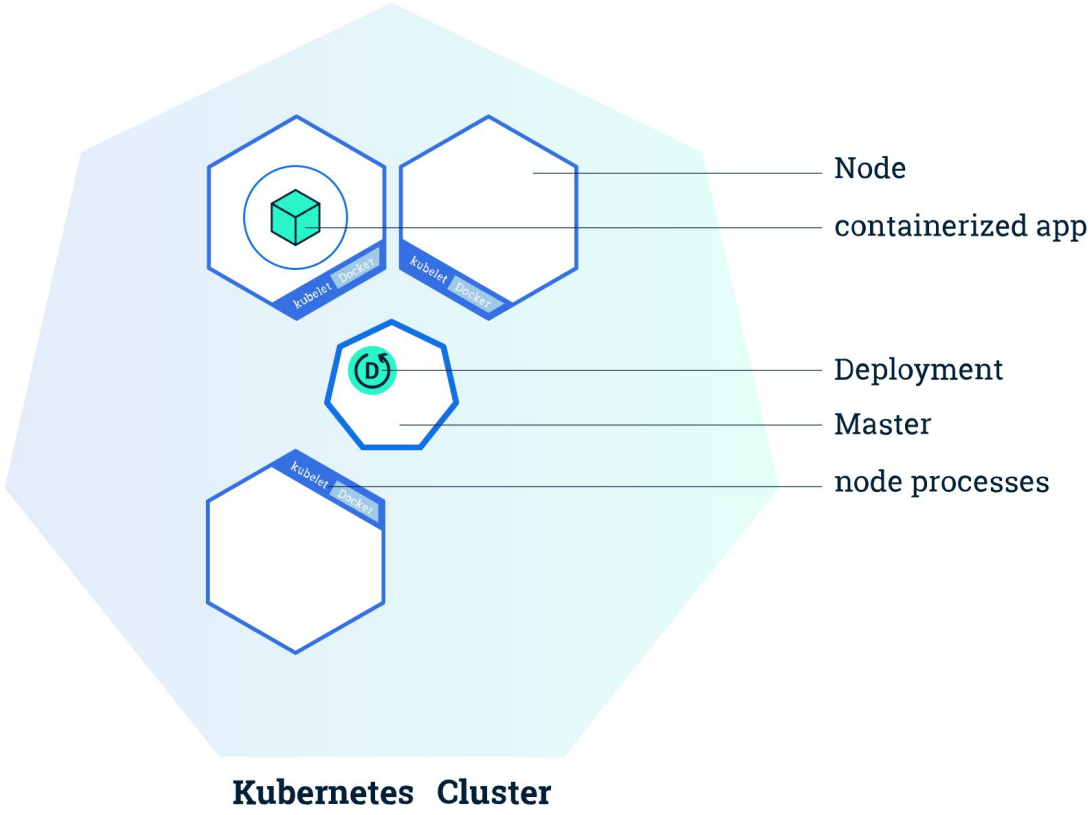
Premier aperçu

Cluster

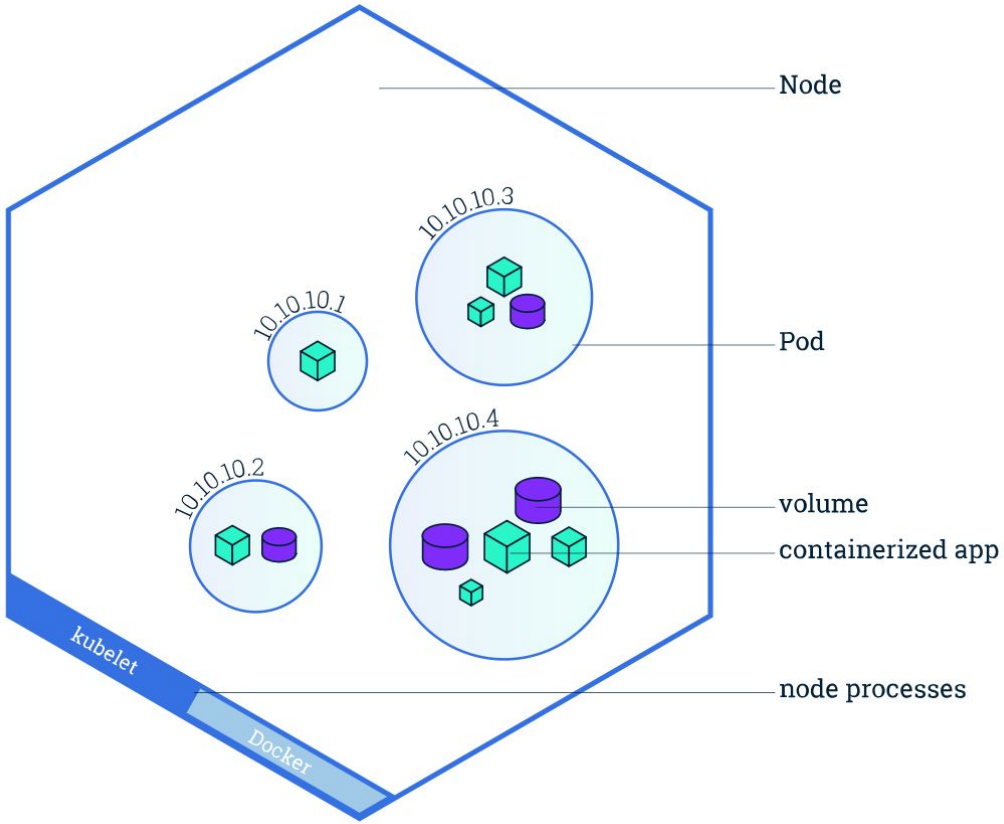


Kubernetes cluster

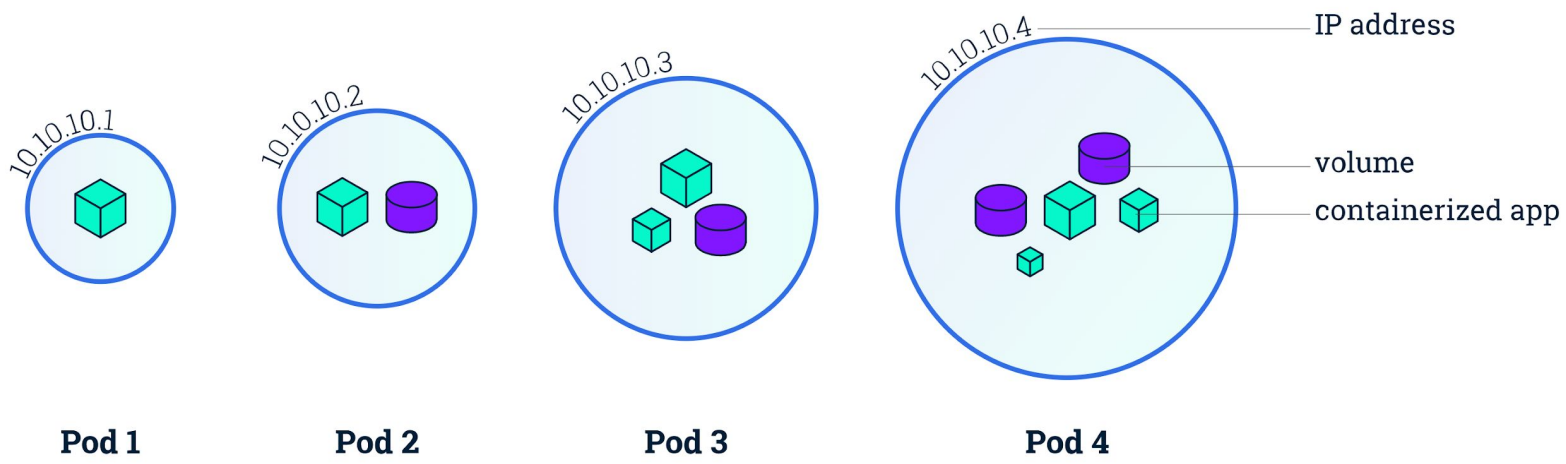
Exécuter des applications



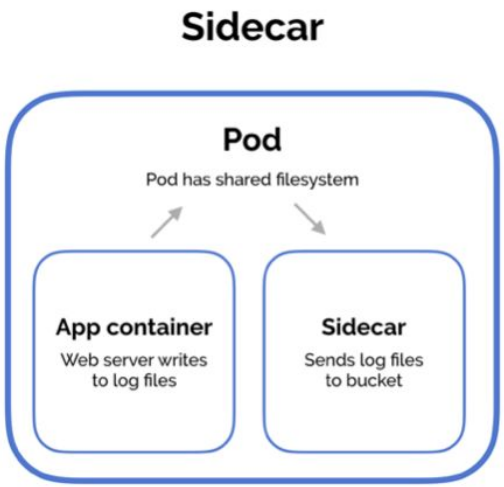
Node

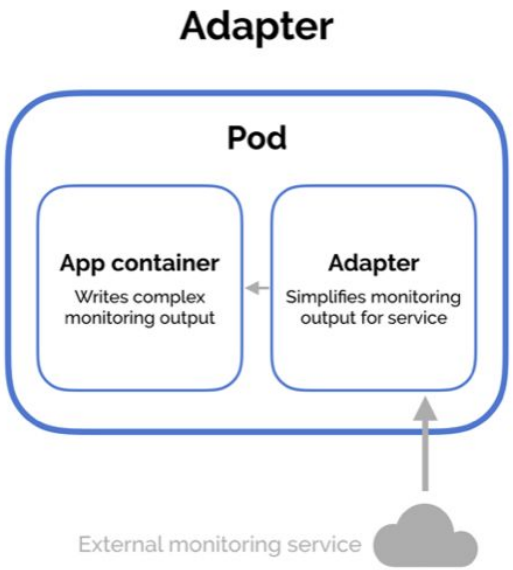


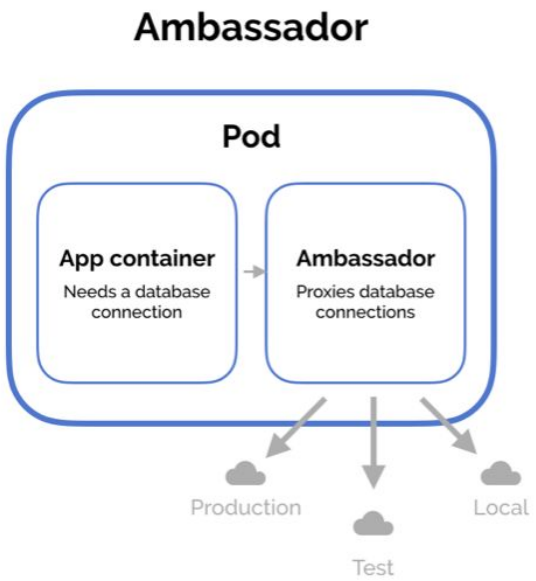
Pod


10.10.10.1
10.10.10.2
10.10.10.3
10.10.10.4
IP address
volume
containerized app
Pod 1
Pod 2
Pod 3
Pod 4

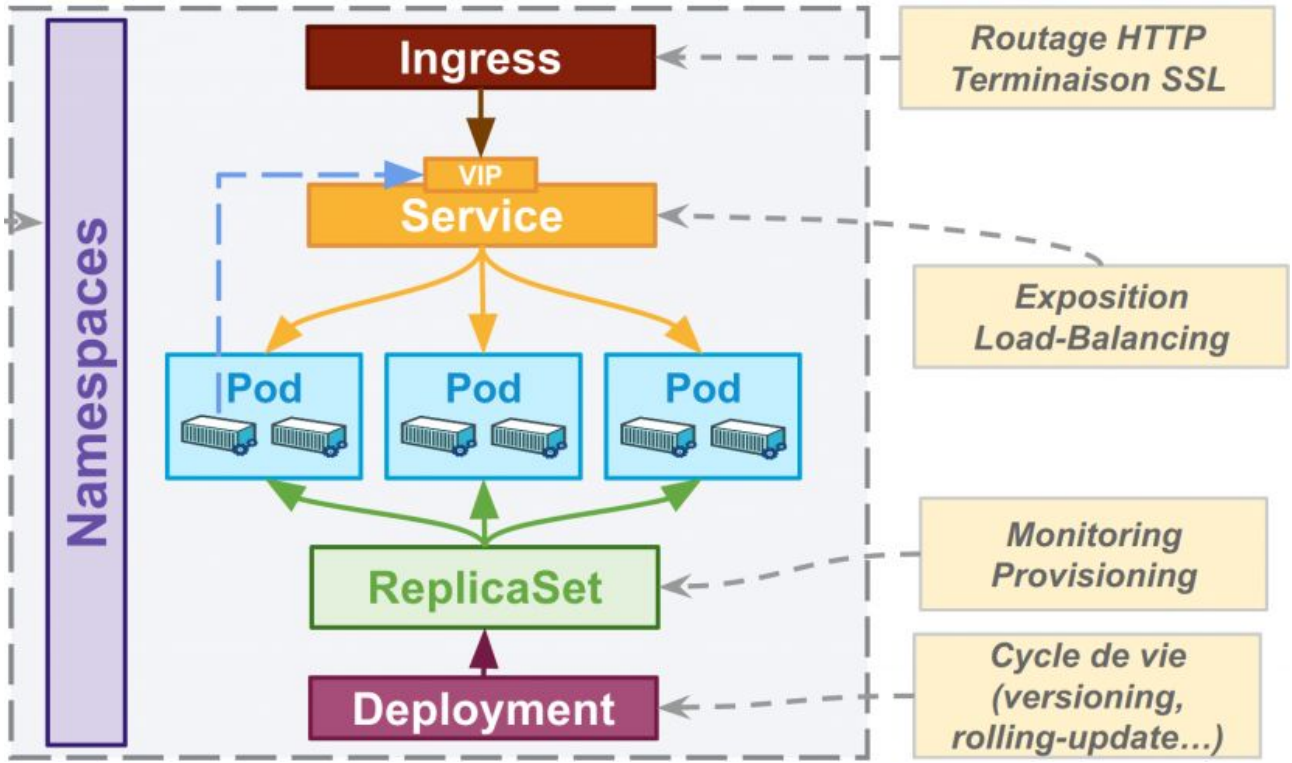
Patterns





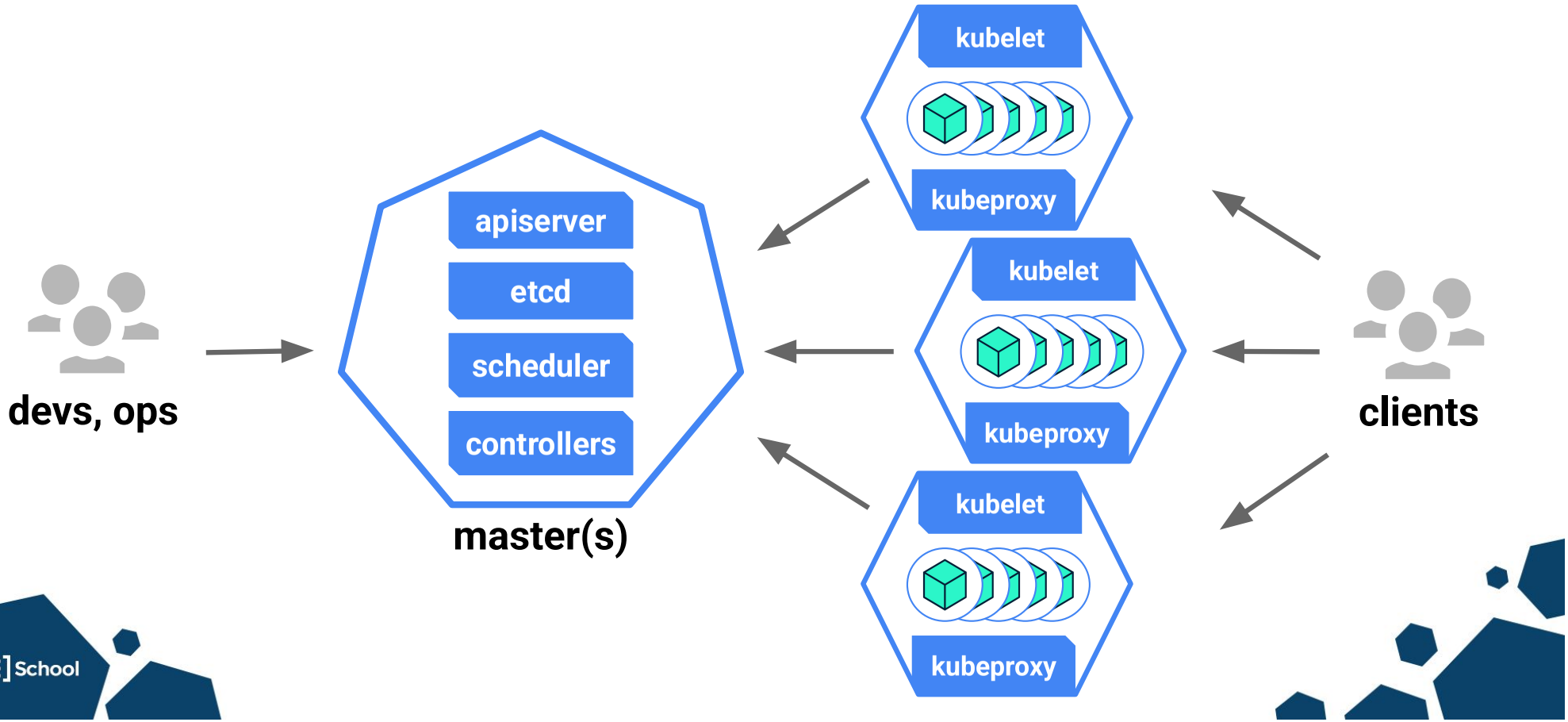


Orchestrer les pods

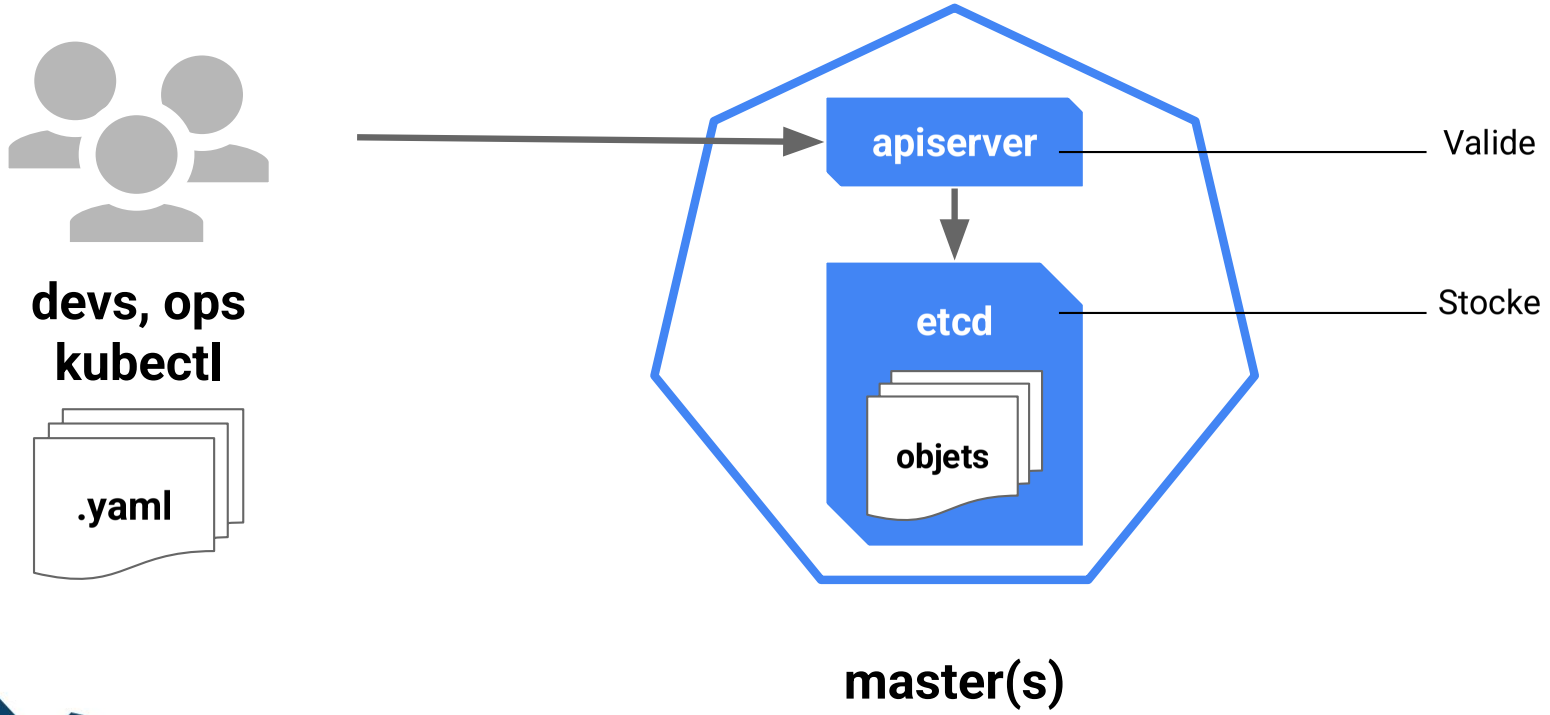


Architecture interne

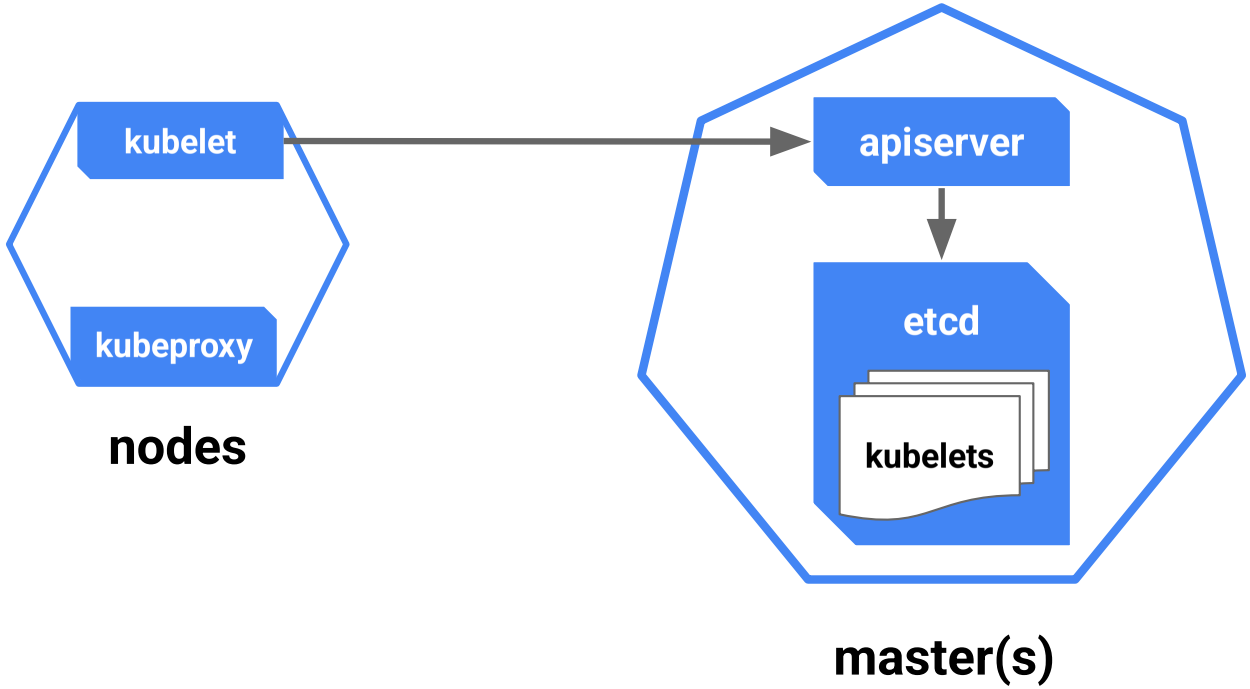
Master et Nodes



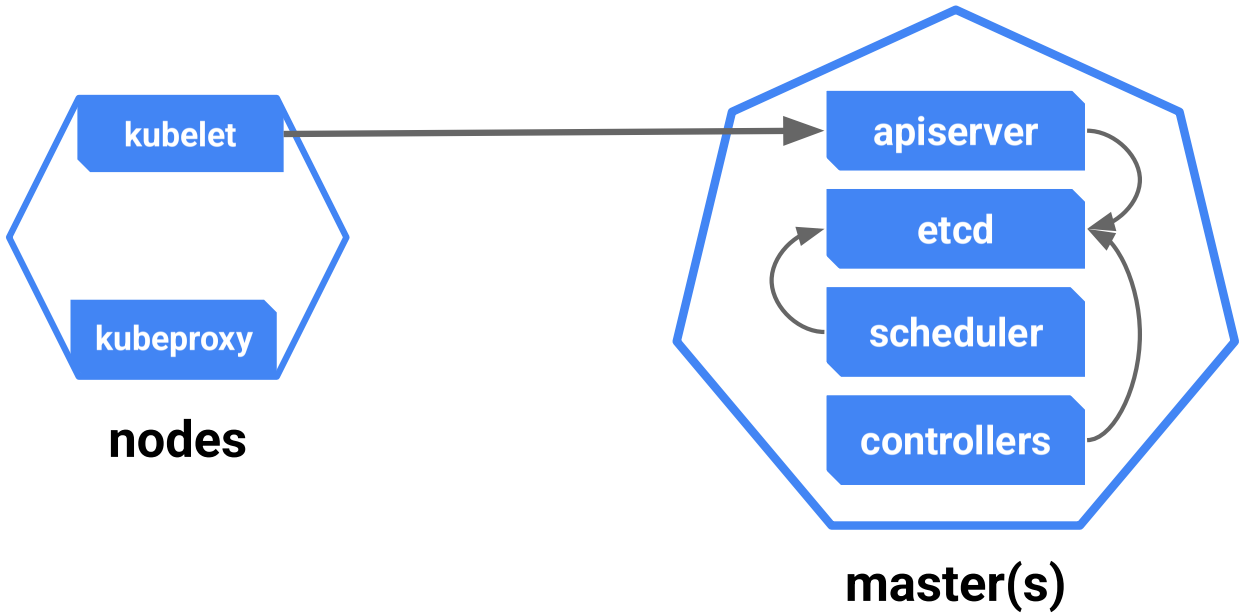
ApiServer et etcd



Kubelet

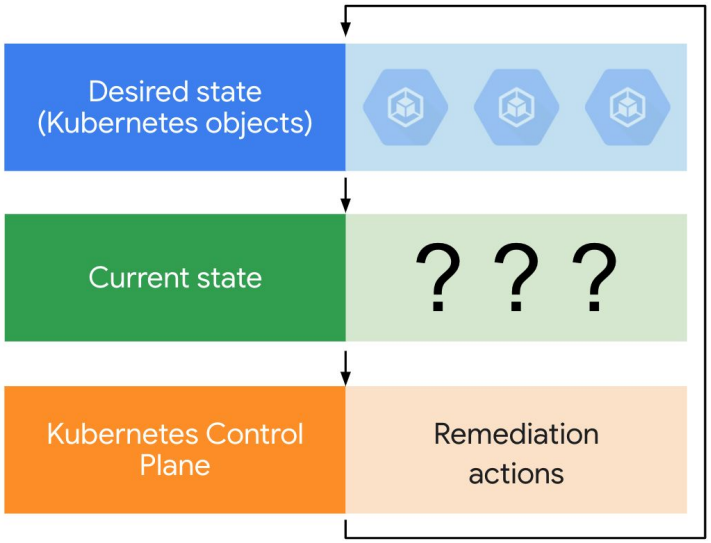


Controllers

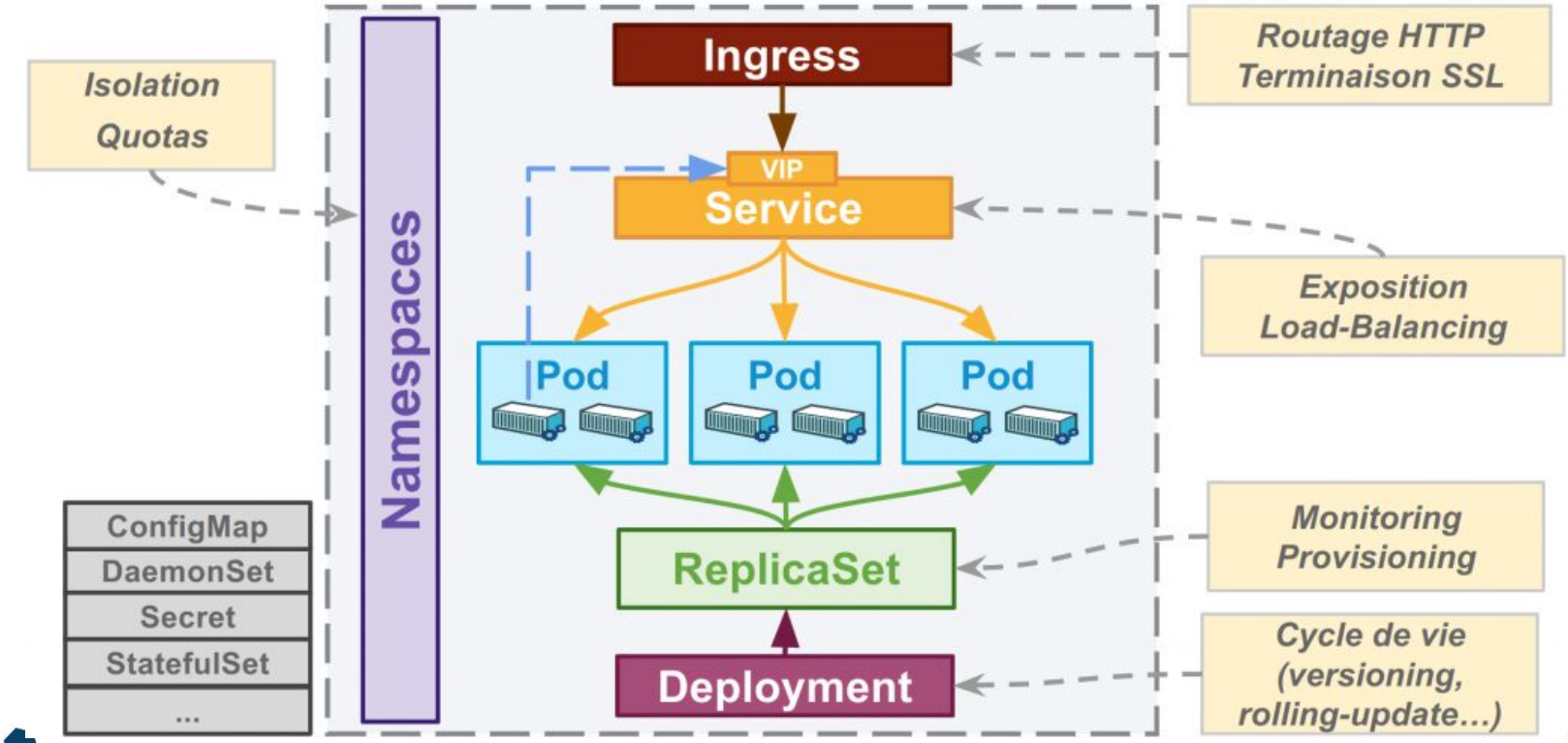


Récapitulatif

- Le Kubelet s'enregistre dans la config
- L'utilisateur envoi l'état voulu sur etcd via l'apiserver
- Les contrôleurs et le Kubelet réagissent aux changements d'états pour obtenir et maintenir l'état voulu



À venir...


Isolation
Quotas
Namespaces
Ingress
VIP
Service
Pod
Pod
Pod
ReplicaSet
Deployment
ConfigMap
DaemonSet
Secret
StatefulSet
...
Routage HTTP
Terminaison SSL
Exposition
Load-Balancing
Monitoring
Provisioning
Cycle de vie
(versioning,
rolling-update...)

Getting started

Création de cluster
Configuration de kubectl

Créer un cluster

GKE : Google **K**ubernetes **E**ngine

Minikube

Utilitaires autour de kubectl

- https://kubernetes.io/docs/tasks/tools/install-kubectl/#optional-kubectl-configurations

```
$ kubectl <TAB>    # autocompletion
```

- https://github.com/ahmetb/kubectx

```
$ kubectx gke-dev      # changer de contexte
$ kubens kube-system   # modifier le ns du contexte courant
```

Utilitaires autour de kubectl

- ~800 kubectl aliases (bash/zsh)
  https://github.com/ahmetb/kubectl-aliases

```
$ kgpo     # kubectl get pod
```

- Shell prompt
  https://github.com/jonmosco/kube-ps1

```
(⎈ |minikube:default)$
```

Récupérer les exercices

- Clonez le dépôt github suivant :

```
git clone https://github.com/raab-d/ESGI-K8S-5IABD1-2026
```

- et placez-vous dans le dépôt cloné :

```
cd ESKI-K8S-5IABD1-2026/terraform/gke
```

- Démarrer un cluster K8S et configurer kubectl

Namespace

Namespace

- Espace de nom pour isoler les déploiements
- Peut être utilisé pour séparer les environnements (soft multi-tenant)
- Le nom d'une ressource est unique au sein d'un namespace.
- Par défaut:
  - default
  - kube-system
  - kube-public

Manifest

Manifest

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:alpine
status:
  ...
```

Créer et intéragir avec un pod

kubectl get pods

```
$ kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
nginx-5dbbff858d-qfb7f      1/1       Running   0          2m

$ kubectl get pod -o yaml nginx-5dbbff858d-qfb7f
                apiVersion: v1
                kind: Pod
                metadata:
                  name: nginx-5dbbff858d-qfb7f
                ...
```

Pod yaml

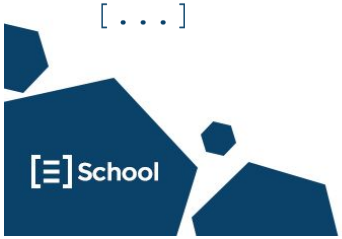
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:alpine
status:
  ...
```

Pods explained

```
$ 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>
   kind   <string>
   metadata    <Object>
   spec   <Object>
[...]
```



Pods explained... deeper

```
$ kubectl explain pod.spec.containers.image
KIND:     Pod
VERSION:  v1

FIELD:    image <string>

DESCRIPTION:
     Docker image name. More info:
     https://kubernetes.io/docs/concepts/containers/images This field is
     optional to allow higher level config management to default or override
     container images in workload controllers like Deployments and
     StatefulSets.
```

Pod status (live)

```
...
status:
  containerStatuses:
  - name: nginx
    containerID: docker://8db3af41eb87[...]3d103255
    image: nginx:alpine
    imageID: docker-pullable://nginx@sha256:1aed1[...]a0a5440
    state:
      running:
        startedAt: 2018-08-13T21:08:10Z
  hostIP: 192.168.65.3
  podIP: 10.1.0.227
```

TP : Pod

Configuration d’une application

Configuration

- Configurer l'URI d'une base de données
- Injecter un fichier application-prod.yml
- Spécifier les credentials d'une api

Variables d’environnement

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
spec:
  containers:
  - name: envar-demo-container
    image: debian
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

ConfigMap : création

$ kubectl create configmap <map-name> <data-source>

$ kubectl create configmap my-app-conf
--from-file=my-app-conf/configmap/application-dev.properties

$ kubectl create configmap my-app-conf
--from-literal=db-server=mydbserver.mycompany.com

ConfigMap: manifests

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
data:
  application.properties: |
    db-server=mydbserver.dev.mycompany.com
    username=my-rw-dbuser
```

```
$ kubectl apply -f configmap.yml
```

ConfigMap : utilisation

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
    - name: test
      image: busybox
      volumeMounts:
        - name: config-vol
          mountPath: /etc/config
```

```
volumes:
  - name: config-vol
    configMap:
      name: log-config
      items:
        - key: log_level
          path: log_level
```

Point de montage du fichier : **/etc/config/log_level**

Secret : type

```
$ kubectl create secret --help
```

```
Available Commands:
  docker-registry Create a secret for use with a Docker registry
  generic         Create a secret from a local file, directory or literal
value
  tls             Create a TLS secret
```

Secret : manifest

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Secret : création

- Créer un secret depuis un fichier

$ kubectl create secret generic db-user-pass --from-file=./password.txt

- Créer un secret depuis une clé/valeur

$ kubectl create secret generic prod-db-secret
--from-literal=username=produser

- Créer un secret “docker-registry ”

$ kubectl create secret docker-registry regcred
--docker-server=<your-registry-server> --docker-username=<your-name>
docker-password=<your-pword> --docker-email=<your-email>

Secret : env vars

```yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: envar-demo-container
    image: debian
    env:
    - name: DEMO_GREETING
      valueFrom:
        secretKeyRef:
            name: demo        # le nom du secret
            key: password     # la clé dans le secret
```

Secret : volume

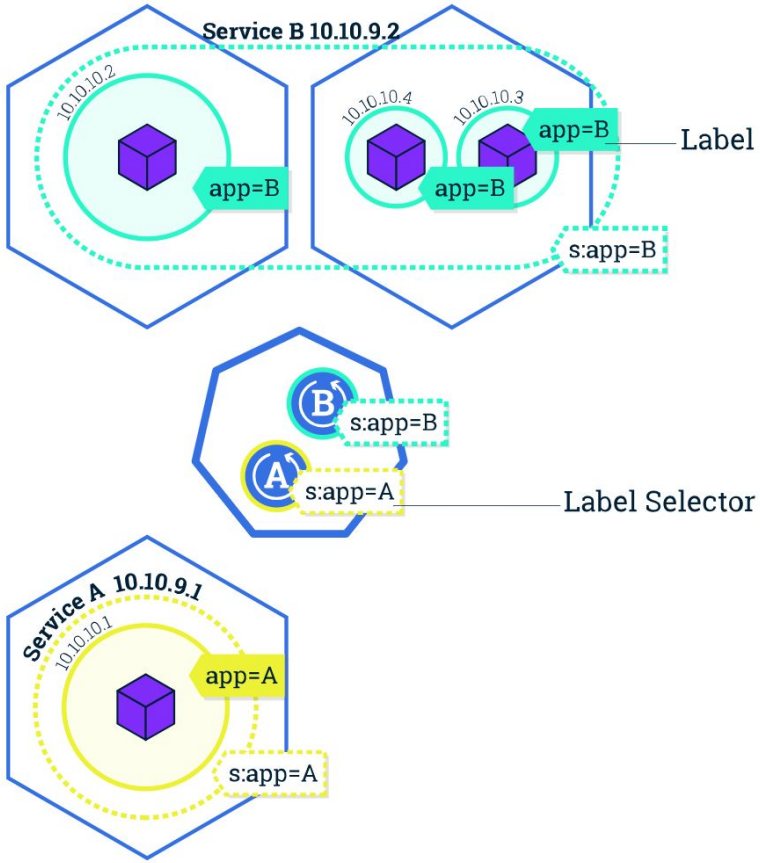
```
...
kind: Pod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: secret-volume
      mountPath: "/etc/foo"
      readOnly: true
```

```
volumes:
- name: secret-volume
  secret:
    secretName: mysecret
```

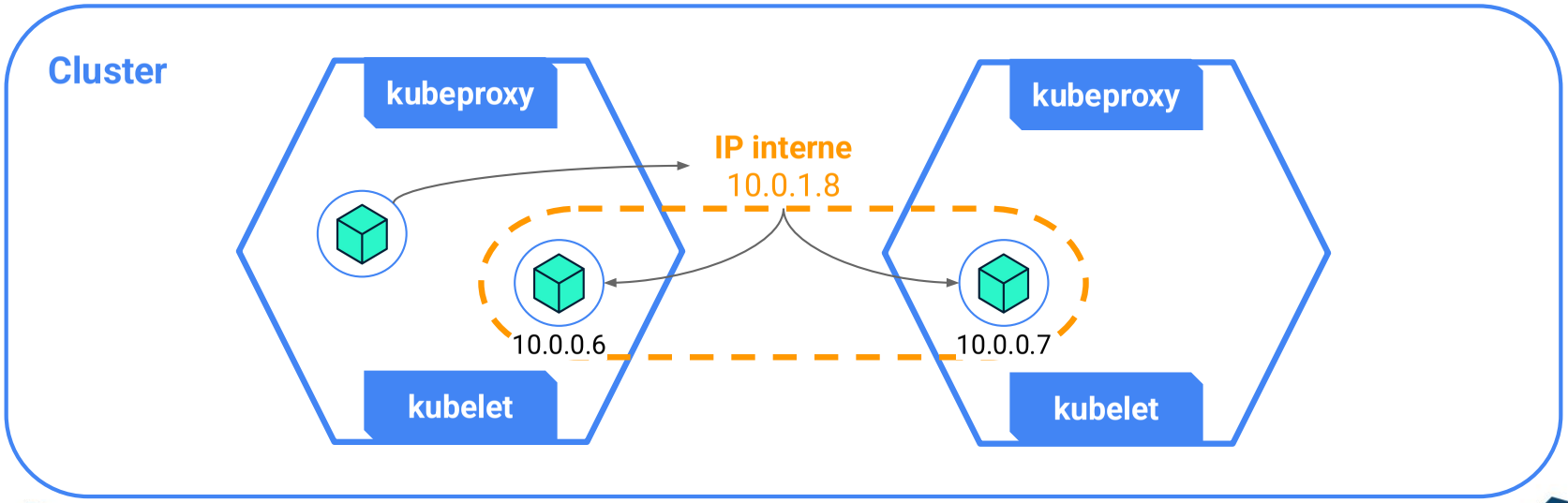
TP : Configuration

Service

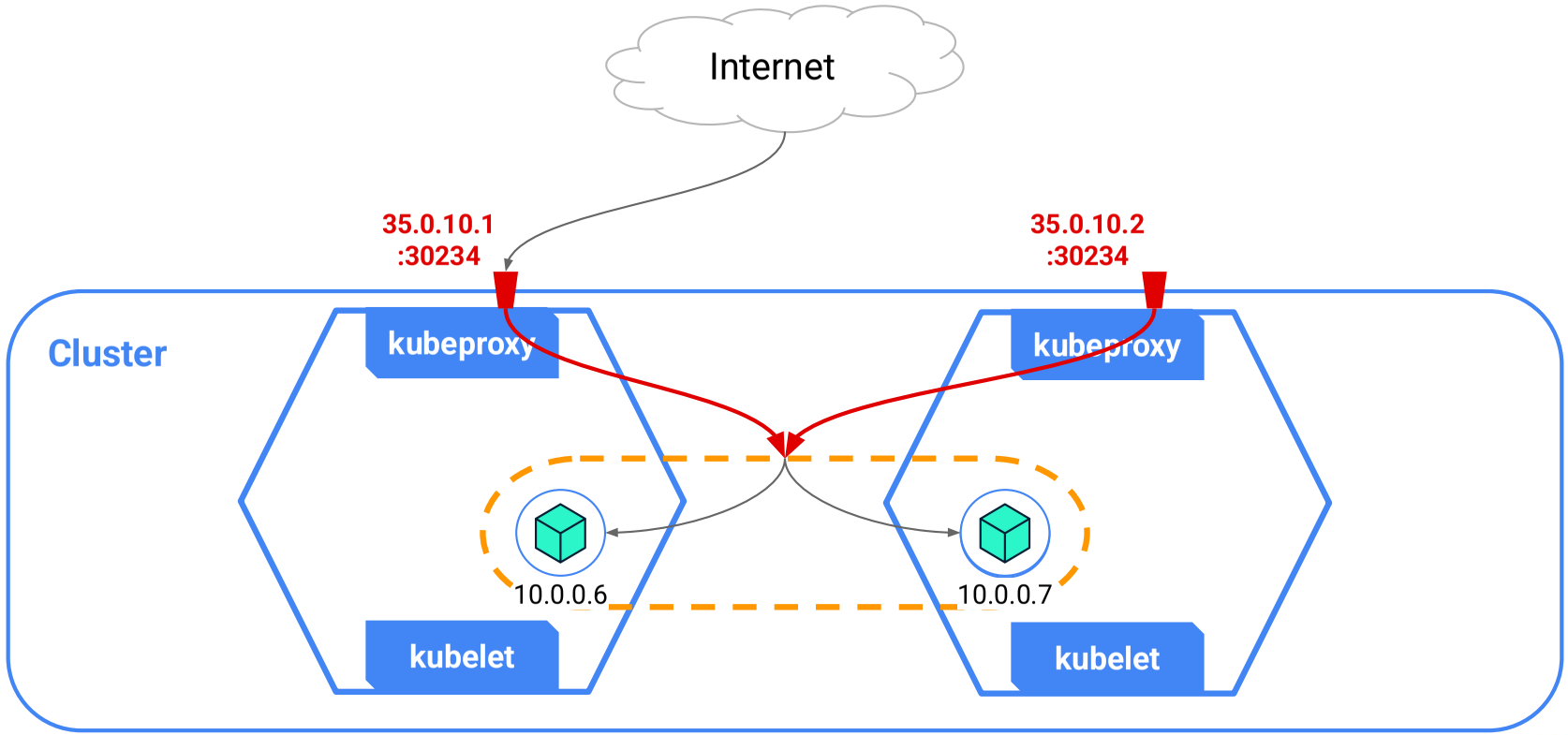
Exposer des pods



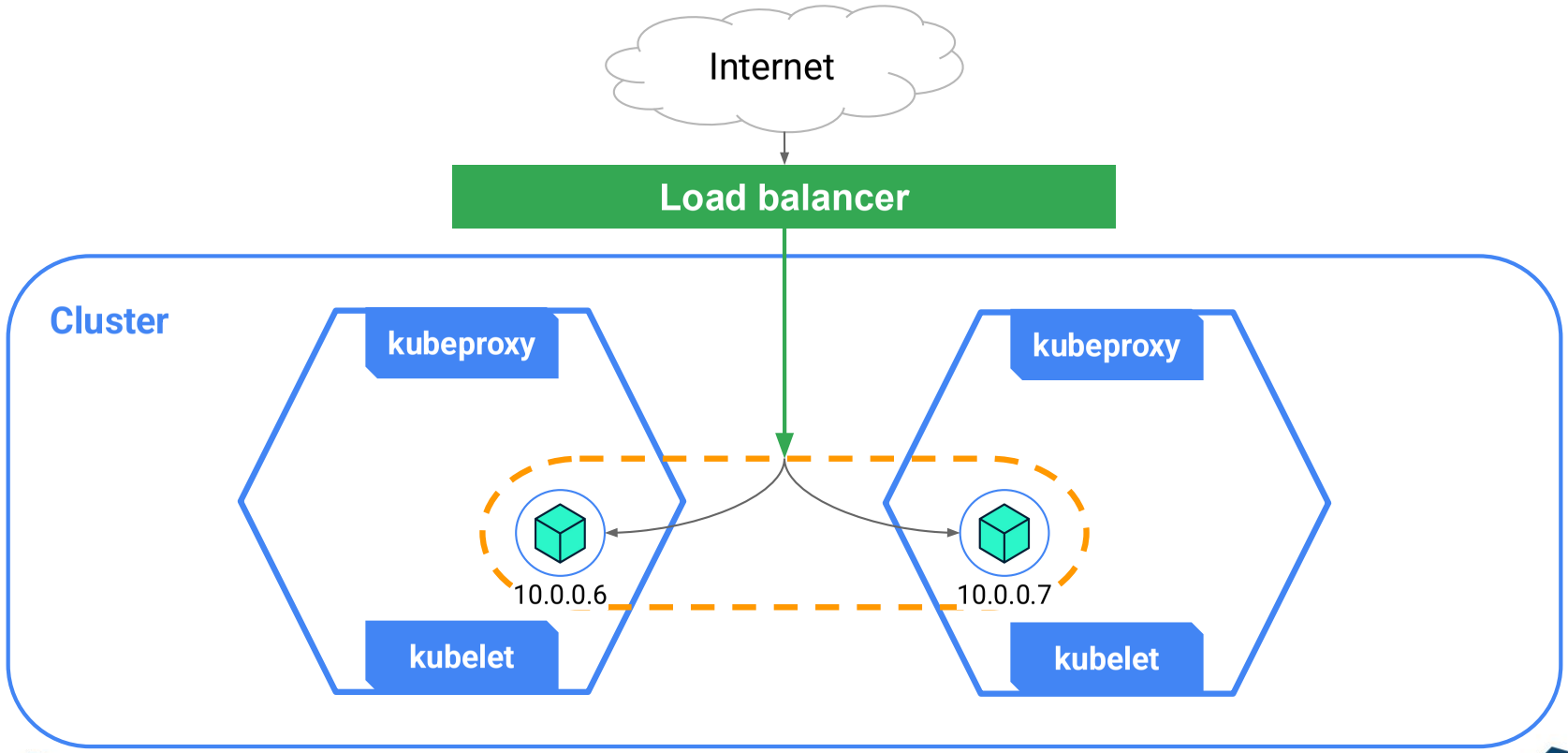
Service type ClusterIP



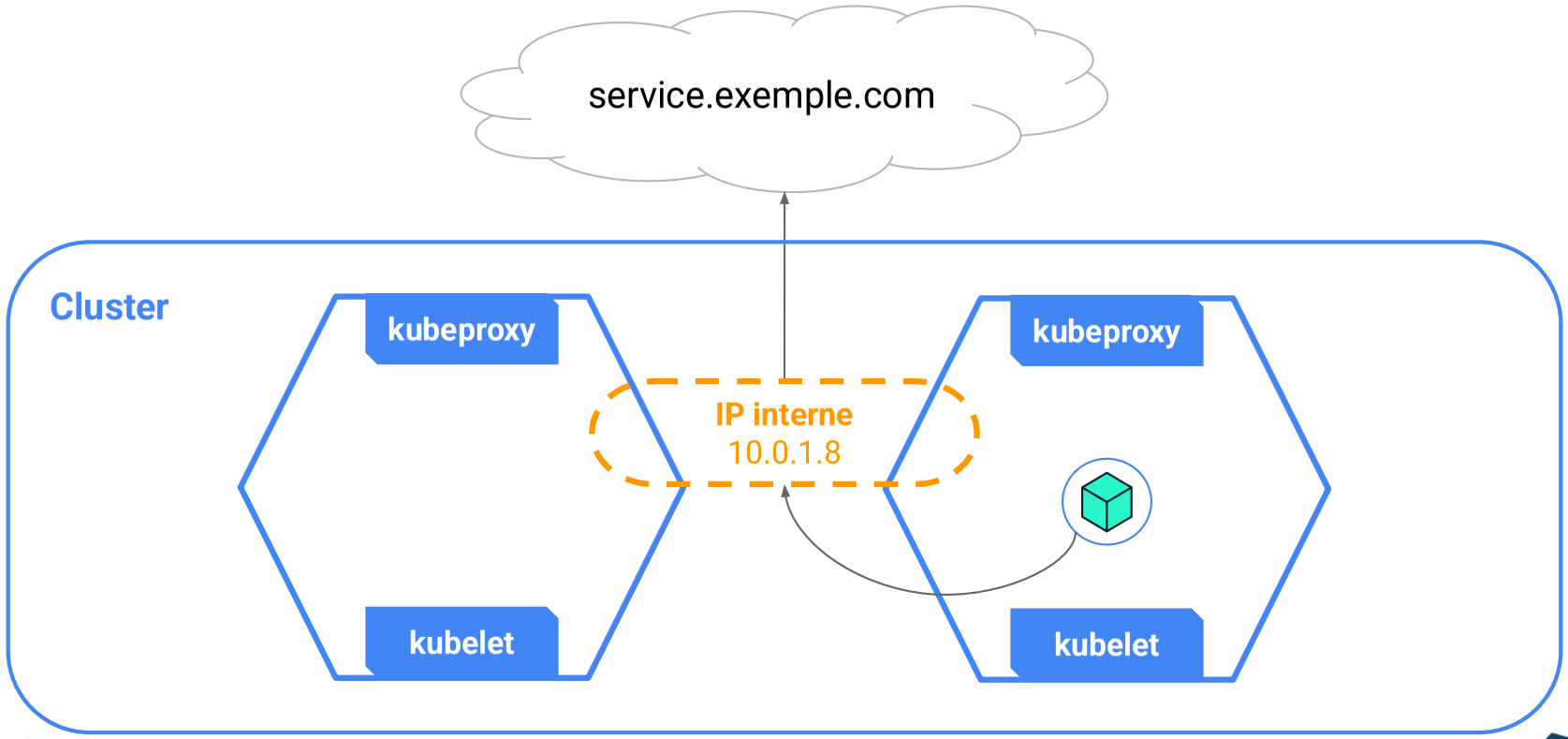
Service type NodePort



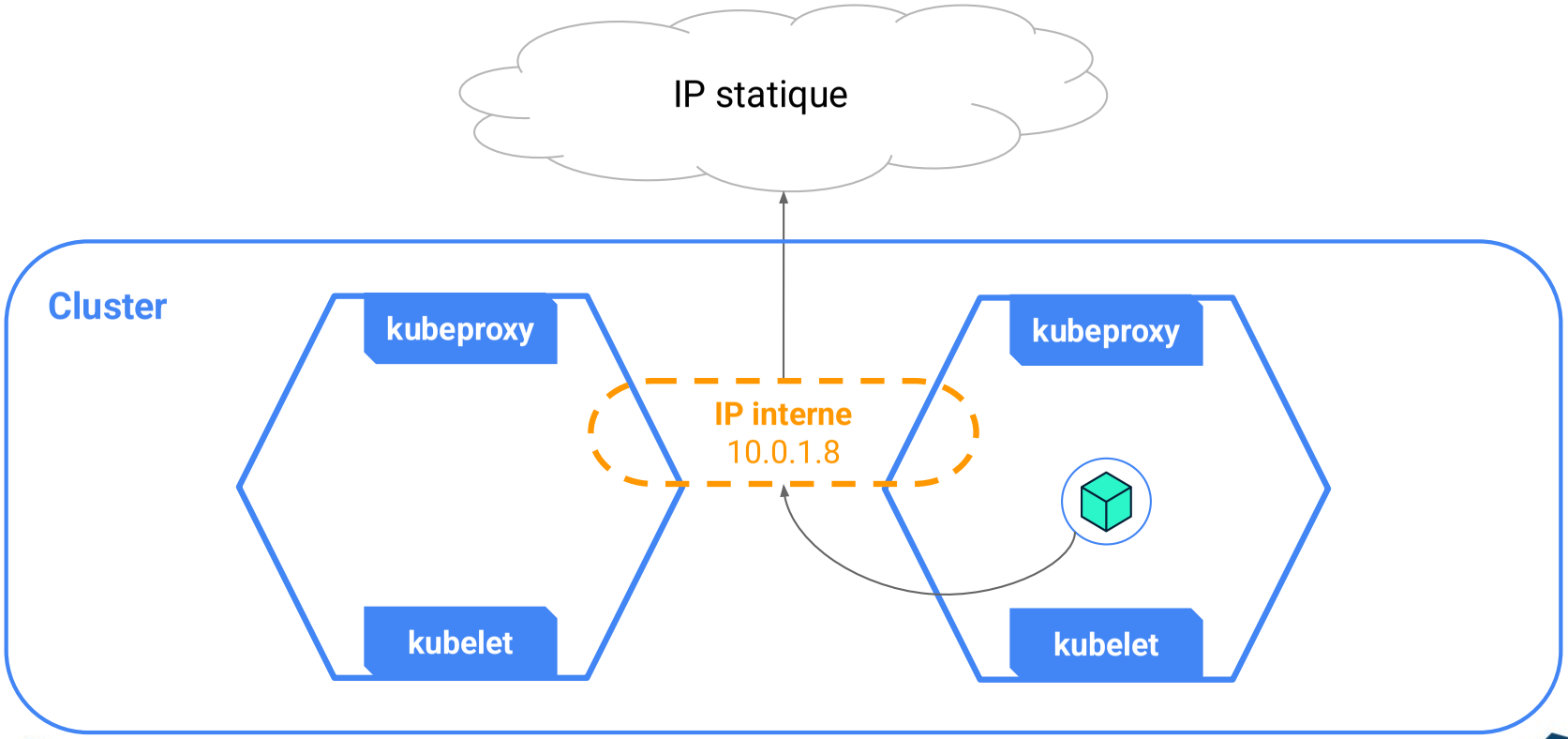
Service type LoadBalancer



Service type ExternalName



Service avec Endpoint explicite



Service yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
```

Créer un service

```
$ kubectl apply -f service.yaml
```

- La ligne de commande suivante donne un résultat équivalent au fichier yaml :

```
$ kubectl expose nginx --port=80 --type=NodePort
```

Services

```
$ kubectl get services
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP PORT(S)         AGE
kubernetes   ClusterIP   10.96.0.1     <none>      443/TCP         3d
nginx        NodePort    10.97.47.155  <none>      80:31450/TCP    2s
```

```
$ kubectl get svc -o yaml nginx
                apiVersion: v1
                kind: Service
                metadata:
                  name: nginx
                ...
```

Service : DNS interne

- Entrée DNS : <service>.<namespace>.svc.cluster.local
- Et dans le namespace : <service>

TP : Service

ReplicaSet

ReplicaSet.yaml

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    <... pod definition ...>
```

kubectl get replicaset

```
$ kubectl get ReplicaSet
NAME                  DESIRED   CURRENT   READY     AGE
nginx-5dbbff858d      1         1         1         14m

$ kubectl get rs -o yaml nginx-5dbbff858d
              apiVersion: extensions/v1beta1
              kind: ReplicaSet
              ...
```

ReplicaSet status (live)

```
...
status:
  availableReplicas: 1
  fullyLabeledReplicas: 1
  readyReplicas: 1
  replicas: 1
```

Deployment

Deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
  ...
```

```
...
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:alpine
```

kubectl get deployment

```
$ kubectl get deployment
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     1         1         1            1           1h

$ kubectl get deploy -o yaml nginx
    apiVersion: extensions/v1beta1
    kind: Deployment
    ...
```

Mise à l'échelle manuelle

$ kubectl scale deployment/nginx --replicas=10

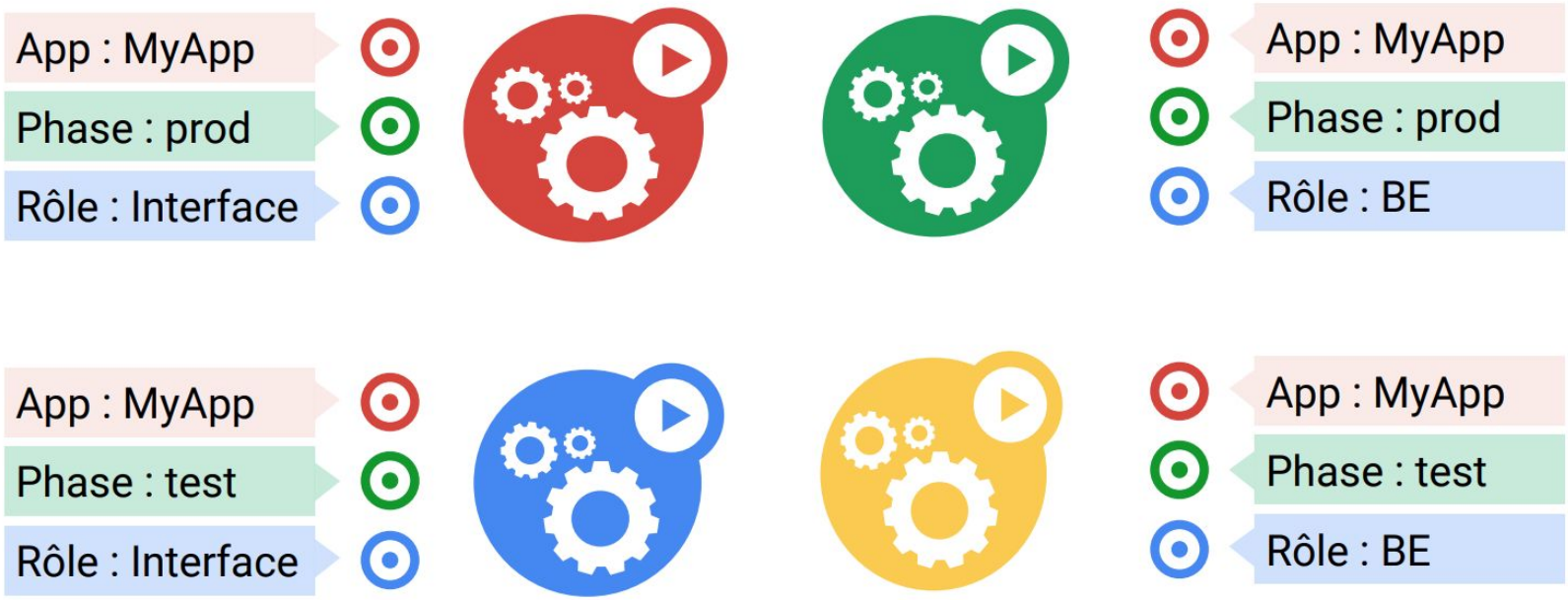
$ kubectl get all

The Label Game

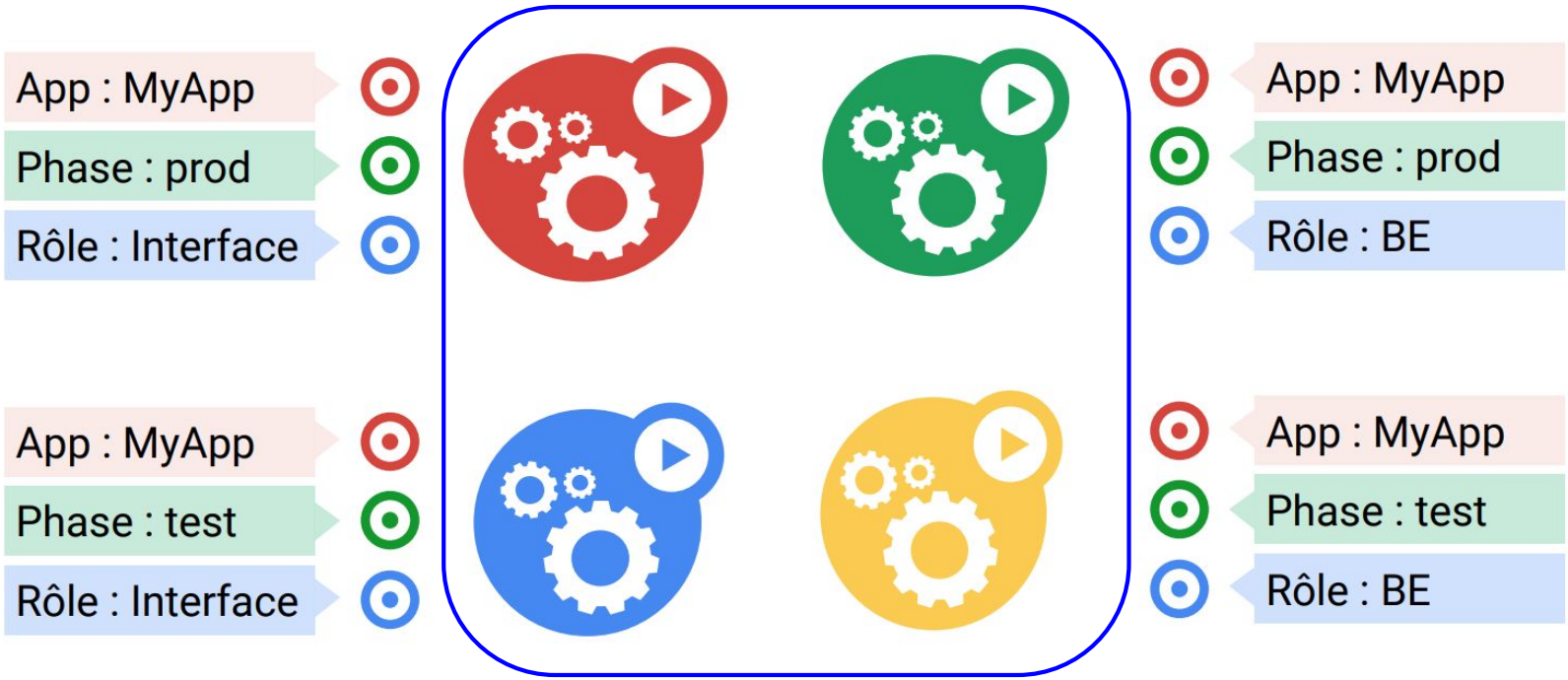
Les labels Kubernetes

- Pour le fonctionnement interne de Kubernetes
  - ReplicaSet && Deployments ⇒ Pods
  - Services ⇒ Pods

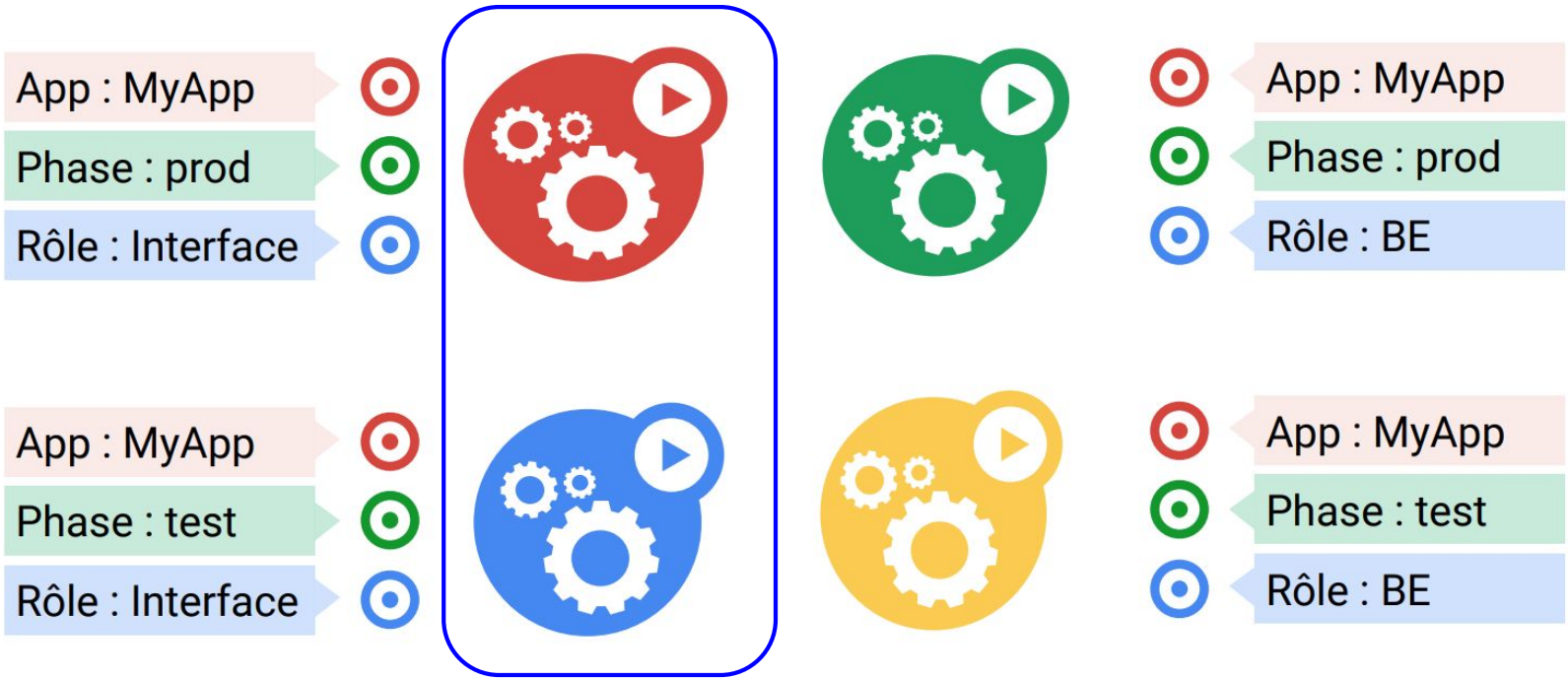
“The label game”



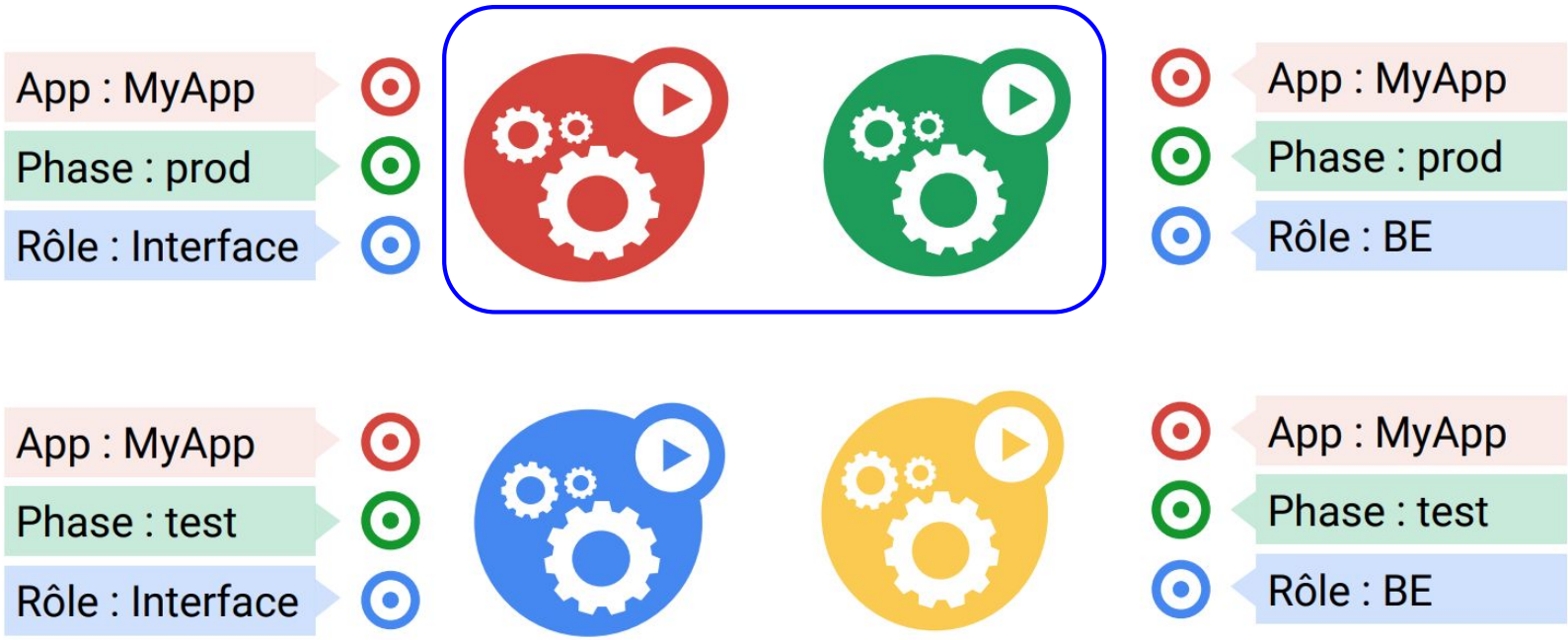
“The label game”



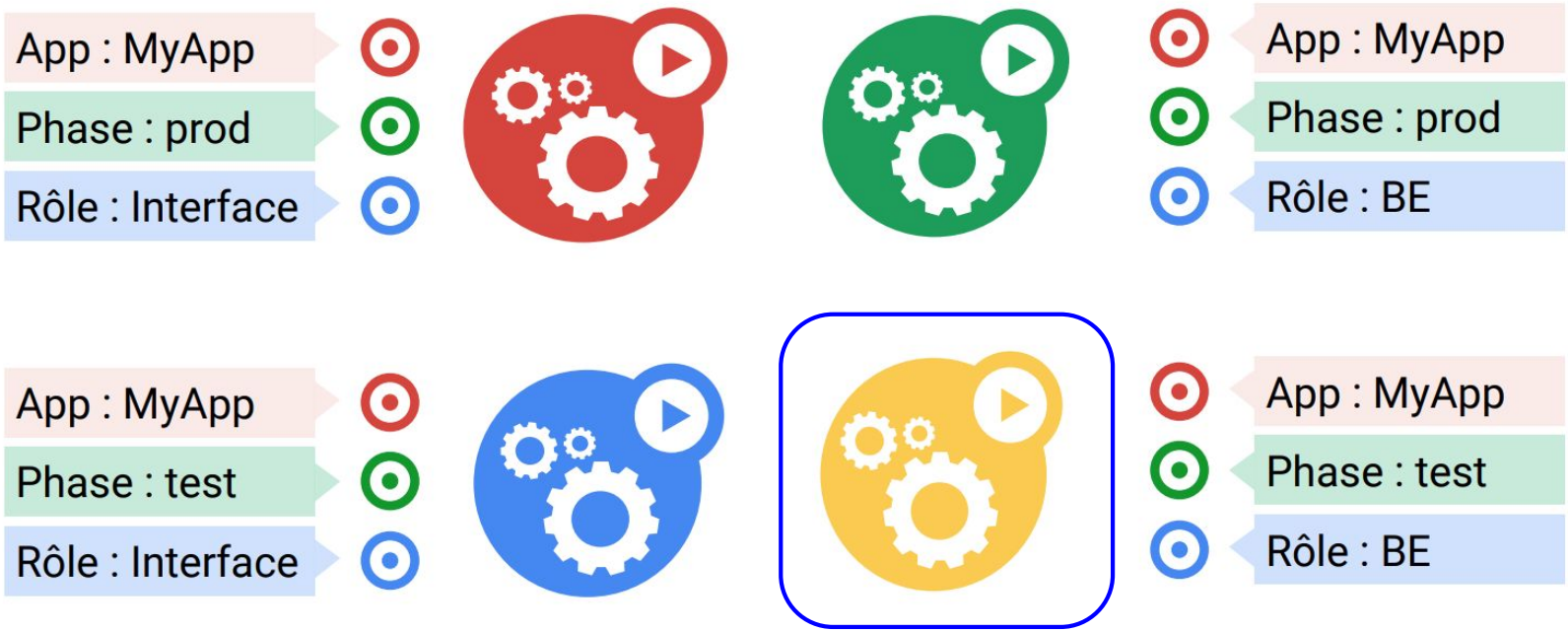
“The label game”



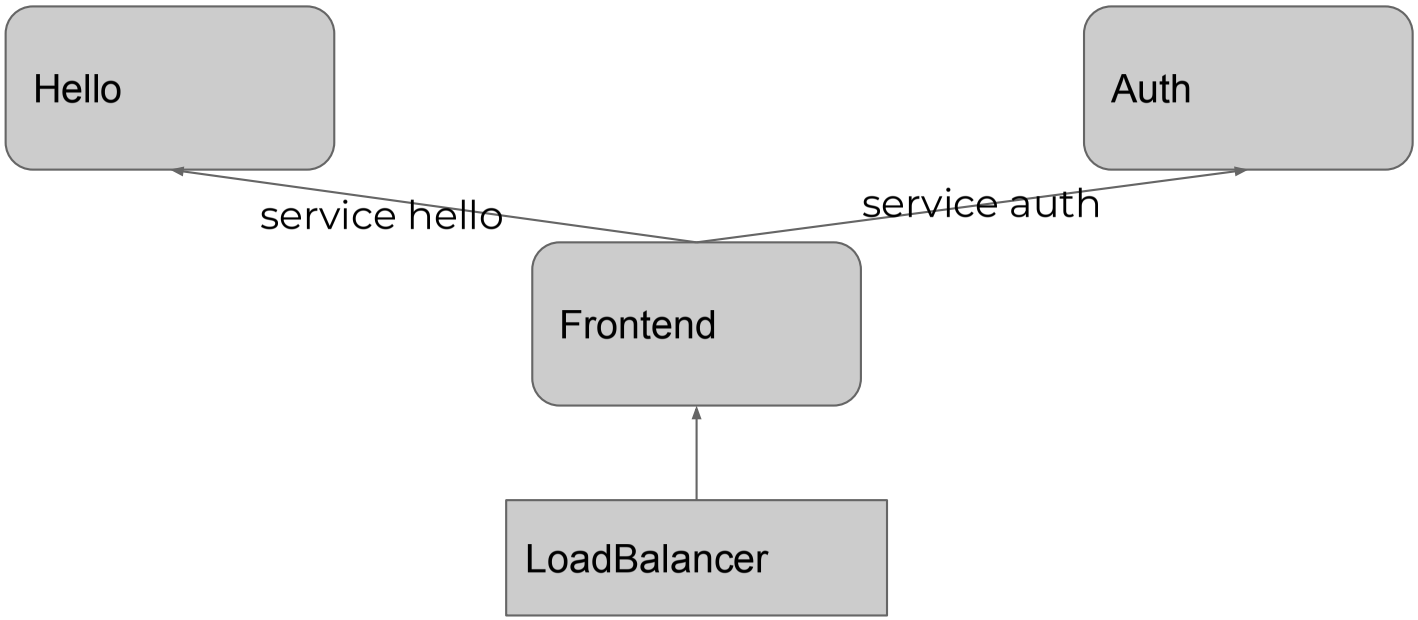
“The label game”



“The label game”



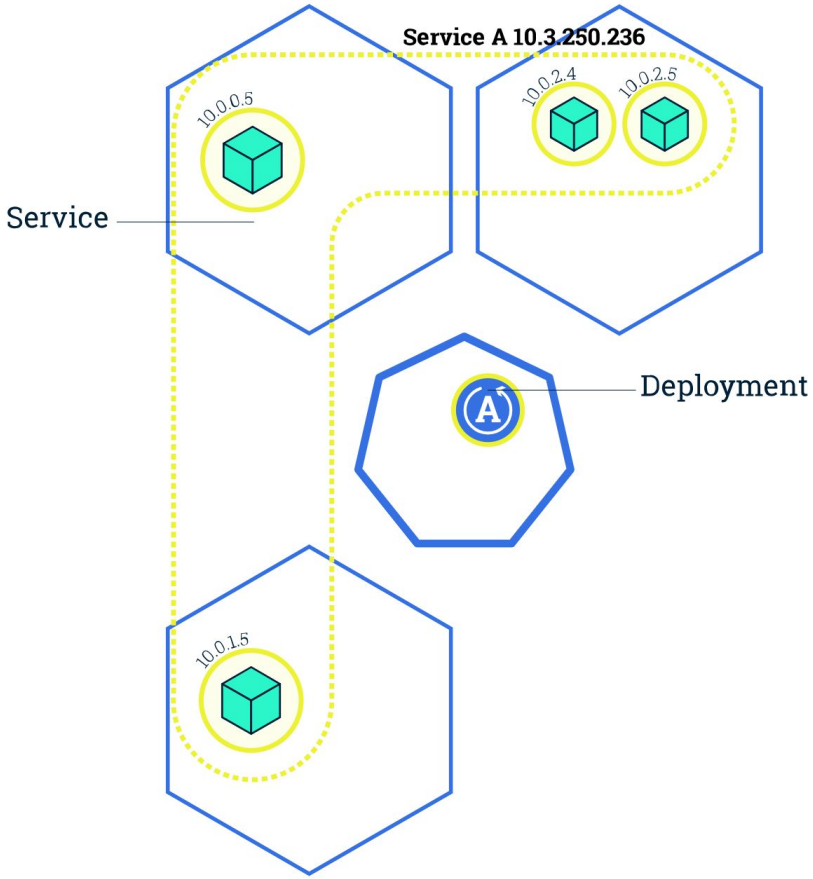
TP : Deployment


Hello
Auth
service hello
service auth
Frontend
LoadBalancer

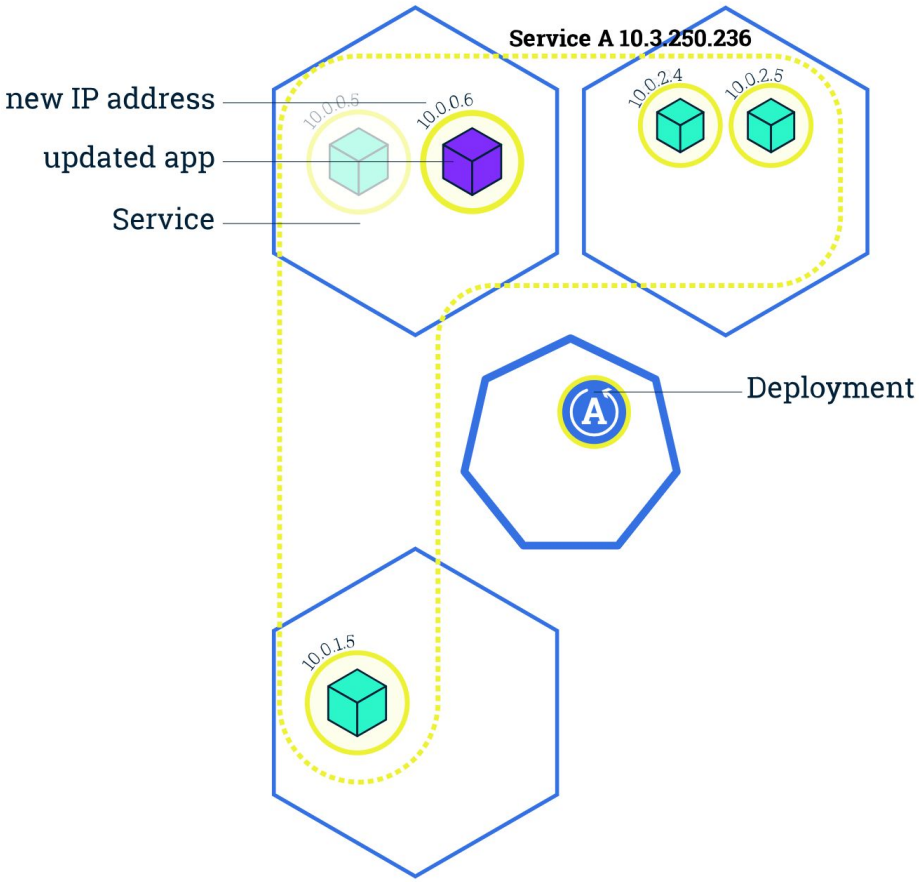

School

Rolling upgrade

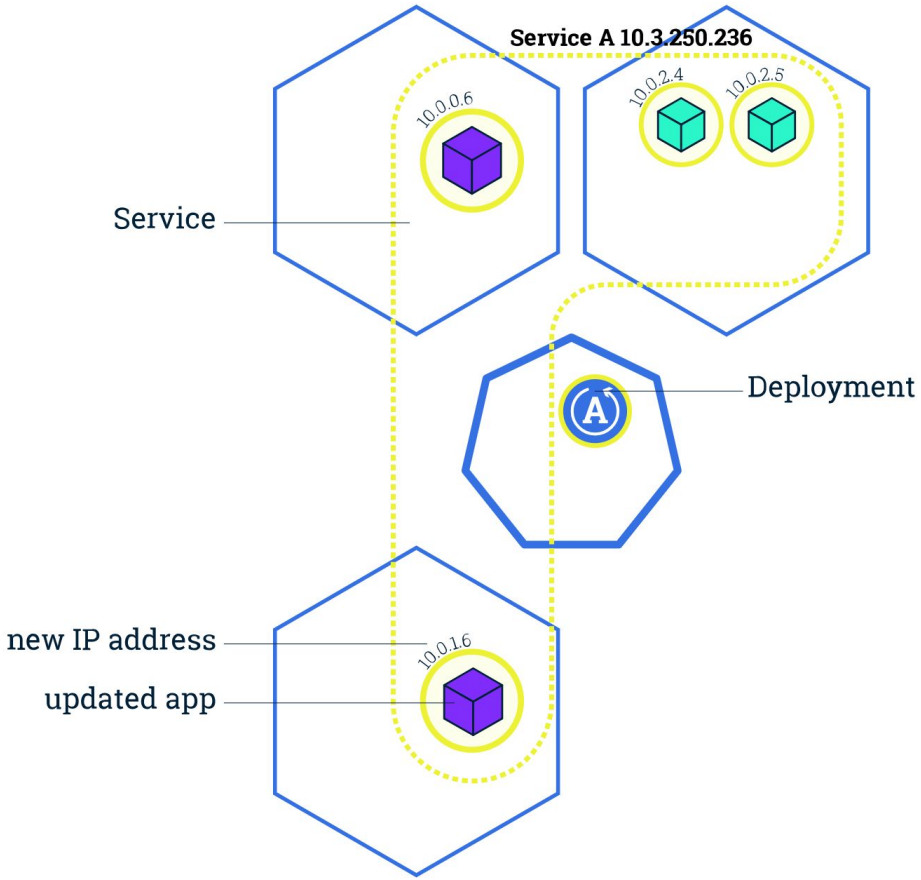
Rolling upgrade 1/4



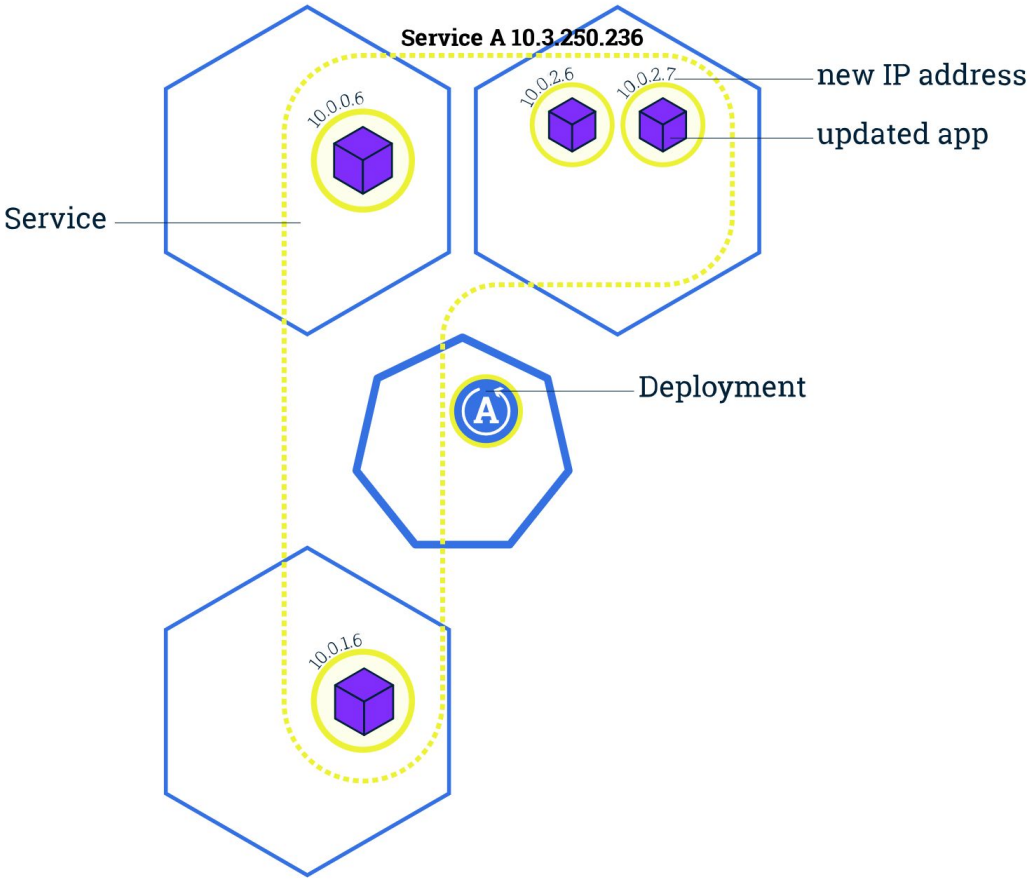
Rolling upgrade 2/4



Rolling upgrade 3/4



Rolling upgrade 4/4



Mise à jour progressive

- Mise à jour progressive (rolling-upgrade) sans interruption de service
- Gérée par le Deployment

Remplacer nginx par Apache httpd :

```
$ kubectl edit deployment nginx --record
```

“image: **nginx**:alpine” ⇒ “image: **httpd**:alpine”

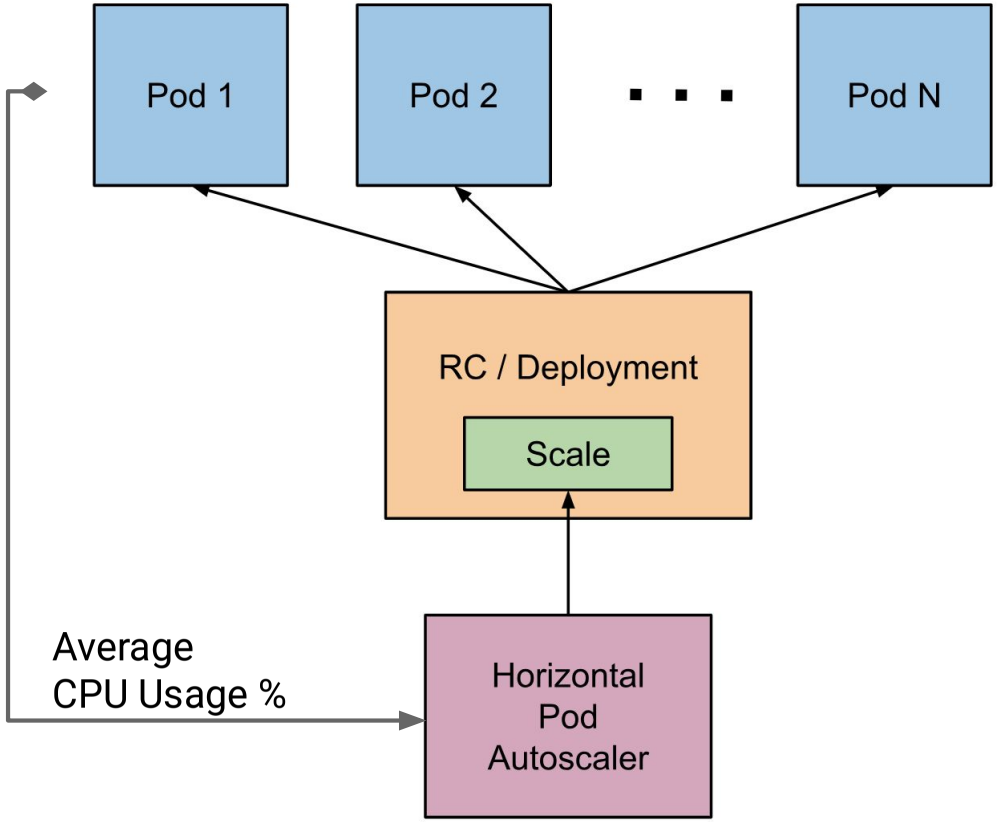
Annuler une mise à jour progressive

```
$ kubectl rollout history deployment nginx
REV  CHANGE-CAUSE
1    kubectl apply --filename=deployment.yaml
2    kubectl edit deployment nginx
3    kubectl set image deploy nginx nginx=nginx:stable-alpine

$ kubectl rollout undo deployment --to-revision=1
```

Mise à l’échelle automatique

Horizontal Pod Autoscaler



Horizontal Pod Autoscaler

- Déclarer un Horizontal Pod scaler en CLI :

```
$ kubectl autoscale deployment nginx \
    --cpu-percent=50 --min=1 --max=10
```

Horizontal Pod Autoscaler

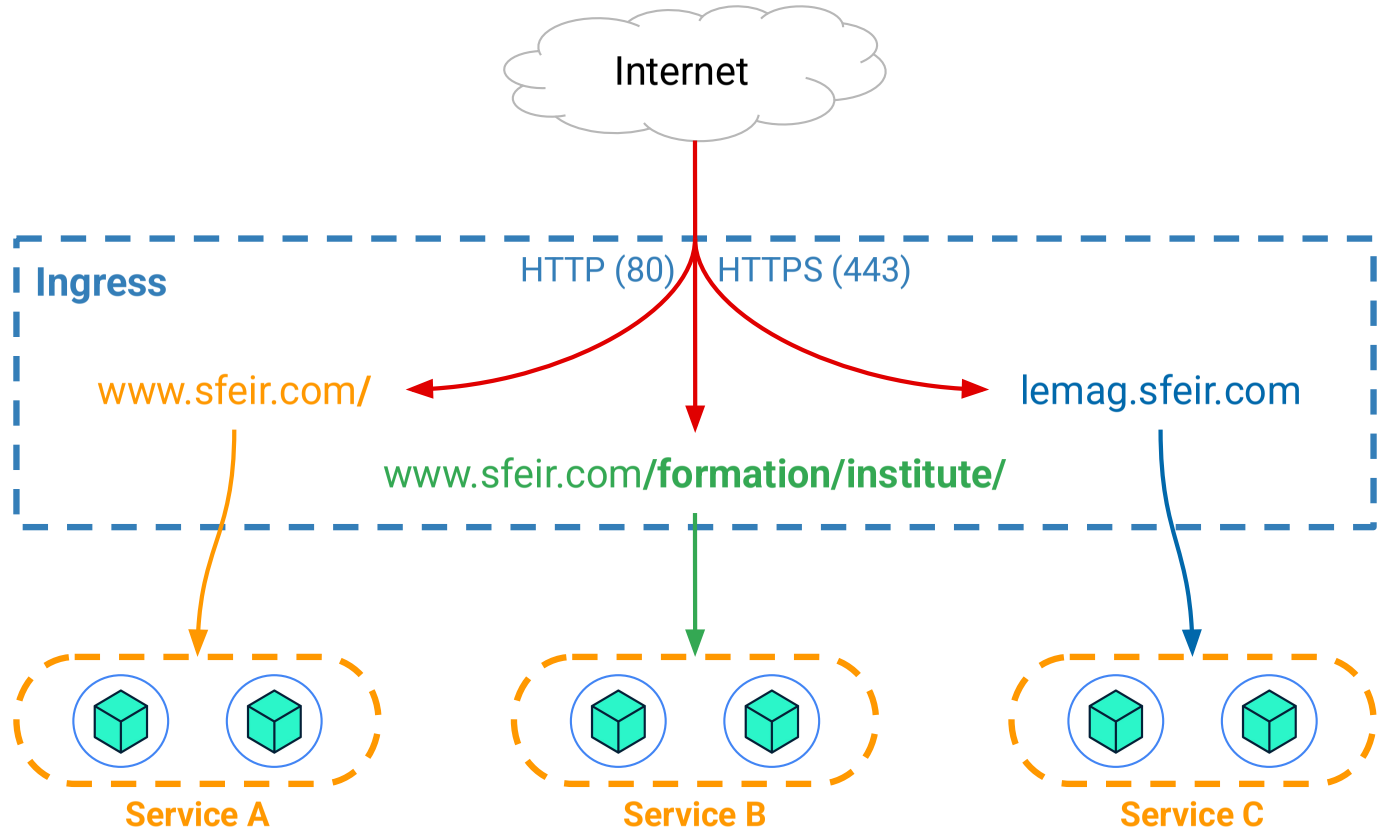
```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: helloweb
spec:
  maxReplicas: 10
  minReplicas: 1
  targetCPUUtilizationPercentage: 50
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: helloweb
```

Ingress

Ingress

- Point d’entrée unique du cluster
- HTTP (port 80) et HTTPS (port 443)
- Gestion des certificats SSL

Ingress


Internet
Ingress
HTTP (80)
HTTPS (443)
www.sfeir.com/
lemag.sfeir.com
www.sfeir.com/formation/institute/
Service A
Service B
Service C

Ingress yaml

```
apiVersion: v1
kind: Ingress
metadata:
  name: nginx
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: http
```

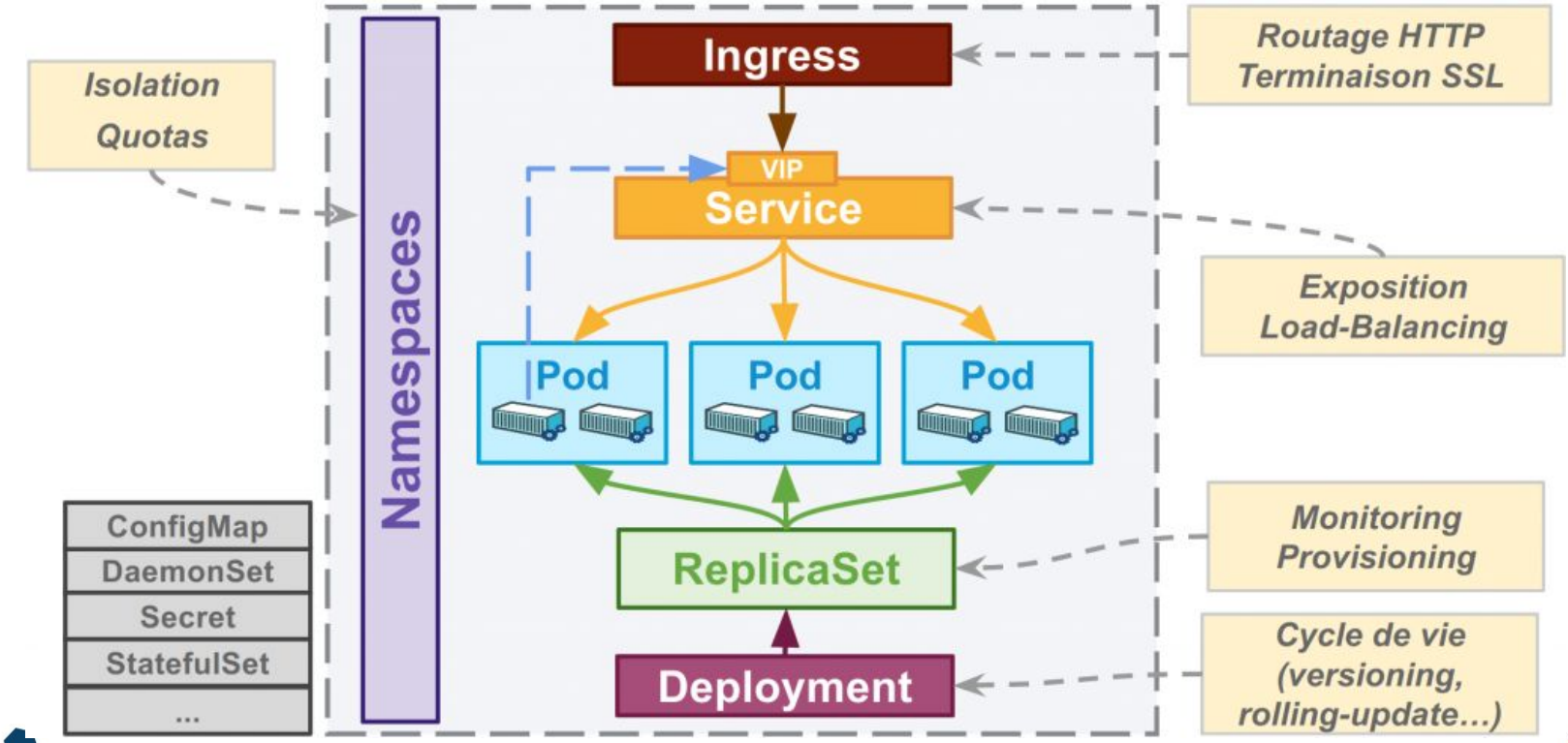
Ingress

```
$ kubectl get ingress
NAME      HOSTS     ADDRESS     PORTS     AGE
nginx     *         localhost   80        3m

$ kubectl get ingress -o yaml nginx
              apiVersion: extensions/v1beta1
              kind: Ingress
              metadata:
                name: nginx
              ...
```

TP : Autoscaling

Récapitulatif



Stocker des données

Volumes

- Juste un dossier monté dans un/des containers
- Associé à la vie du Pod, survit au restart des containers

- Nombreuses implémentations :
  - emptyDir
  - gcePersistentDisk
  - hostPath
  - persistentVolumeClaim
  - configMap / secret

emptyDir

- Volume vide, créé au démarrage d'un pod, supprimé avec la suppression du Pod.

```
volumes:
  - name: cache-volume
    emptyDir: {}
```

gcePersistentDisk

- PersistentDisk GCE
- Il doit être dans le même projet et la même zone que les VM du cluster
- Il n’est pas supprimé à la suppression du pod

gcePersistentDisk

```
volumes:
  - name: test-volume
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

hostPath

- Monte dans le container un dossier du noeud sur lequel le Pod s'exécute

```
volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

persistentVolume & persistentVolumeClaim

- L’admin crée une ressource PersistentVolume associée à une espace de stockage
- Le pod réclame du disque avec un PersistentVolumeClaim

persistentVolume

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

persistentVolumeClaim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Pod

```
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: task-pv-claim
```

```
containers:
   - name: task-pv-container
     image: nginx
     ports:
       - containerPort: 80
         name: "http-server"
     volumeMounts:
       - mountPath:
"/usr/share/nginx/html"
         name: task-pv-storage
```

TP : Volumes