

# CKA : Certified Kubernetes Administrator

## I. Les concepts de base

### 1. Architecture de Kubernetes

- a. **Le nœud Master** : nœud central chargé de piloter les workers
  - i. ETCD : base de données
  - ii. Kube API Server : permet de répondre aux demandes faites
  - iii. Kube Controller manager : Surveiller ou manager les workers
  - iv. Kube Scheduler : Planifier l'ensemble des actions à exécuter
- b. **Les nœuds workers** : nœud permettant d'exécuter les conteneurs ou applications
  - i. Kubelet : Client Kubernetes
  - ii. Kube-proxy : se chargeant de la partie réseau
  - iii. Container-runtime : docker et rkt



### 1.1. ETCD

L'ETCD c'est la base de données qui va stocker toutes informations liées à un cluster Kubernetes, il va stocker les infos sur les nodes, sur leur systèmes d'exploitation, la version de Kubernetes installée dessus, il aura les informations sur les pods, le nom du Node sur lequel il est déployé, l'image que ce pod utilise, les infos sur les secrets, le bindings, configMap, ... toutes les infos du cluster seront stockées dans la base de données ETCD

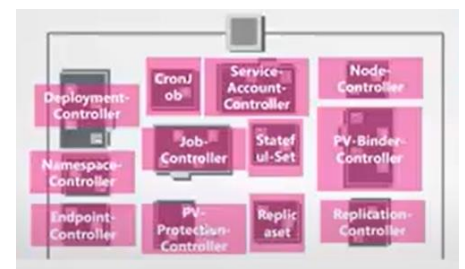
- `kubectl exec etcd_master -n namespace` : affiche les infos liées à etcd.

### 1.2. KUBE-APISERVER

Composant essentiel pour nous en tant qu'administrateur du cluster, car c'est via ce composant qu'on pourra fournir des informations à Kubernetes. Lorsqu'une demande est faite, kube-apiserver commence par vérifier les autorisations de la personne qui effectue la demande et si elle a les autorisations suffisantes, alors il écrit les modifications demandées dans la base de données ETCD. Kube-scheduler qui réalise des vérifications périodiques de cette base de données ETCD se rendra compte qu'il y'a eu de récentes modifications, les analysera, planifiera leur déploiement et va retourner vers Kube-Apiserver pour lui demander d'exécuter les modifications sur les workers suivant la demande initiale faite. Kube-apiserver contactera les clients kubelets des workers pour leur demander d'exécuter les modifications. Des modifications exécutées, kube-apiserver retournera écrire les modifications ou créations réalisées pour archivage dans la base de données ETCD.

### 1.3. KUBE CONTROLLER MANAGER

Le kube Controller manager a pour rôle de surveiller le cluster, il va regarder le statut de l'ensemble des nœuds, et si jamais il y'a un problème (genre perte d'un nœud), il va donc redéployer le contenu de ce nœud sur les autres nœuds disponibles. Il va tester chaque 5s si l'application est toujours disponible ou pas, il va lui donner un période de grâce de 40s et il va l'évincer complètement au bout de 5min car jugera qu'il y'a un problème si l'application est indisponible plus de 5 min. il est constitué de plusieurs contrôleurs internes qui permettent de contrôler le statuts de l'ensemble des composants du cluster. Le Kube Controller manager permet de s'assurer que l'on ait une certaine intégrité au sein de notre cluster et également de la résilience.



### 1.4. KUBE - SCHEDULER

Le kube scheduler est en effet le module kubernetes qui sera chargé de décider quelle ressource sera créée où, il va utiliser un principe de notation pour pouvoir déterminer quel est le nœud le plus approprié pour recevoir la ressource à créer. Il sera capable de filtrer un certain nombre de nœuds, de noter les nœuds (note 20/20) fonction des ressources existantes sur les nœuds. Il a pour rôle de préparer le plan d'action en se basant sur la notion de filtre et de rang pour pouvoir déterminer où est ce qu'il doit envoyer la ressource qu'il faut créer

## 1.5. KUBELET

C'est le client de kubernetes (alter ego du kube-apiserver mais pour les nodes worker). Il va enregistrer le nœud au niveau du kube master ( En disant oui je suis un de tes workers et je suis là pour suivre tes ordres). C'est lui qui va appliquer l'ensemble des opérations qui viendront du kube-scheduler et dont la demande sera exprimée par le kube-apiserver. Il va créer les conteneurs en pilotant docker ou rkt, il va permettre également de monitorer les pods et les nodes en envoyant régulièrement leur état au master.

## 1.6. KUBE - PROXY

Il a pour rôle de permettre la connectivité à l'intérieur et à l'extérieur de notre cluster en créant des règles sur l'ensemble des nœuds qui se basent sur les informations que nous aurons fournies.

## 1.7. POD

Le Pod c'est l'entité qui va contenir notre application qui sera embarquée dans un conteneur, un pod peut avoir plusieurs conteneurs, un POD permettra d'avoir un environnement où l'ensemble des conteneurs partageront le même réseau et éventuellement le même espace de stockage.

## 1.8. REPLICASET

Le replicaset est un objet qui permettra de s'assurer qu'on ait toujours un nombre minimal défini de pods dans notre infrastructure Kubernetes. Donc si on se rend compte qu'un pod est tombé, alors le replicaset va ordonner la création d'un nouveau.

## 1.9. DEPLOYMENT

Il permet de piloter les replicaset, utile lors des upgrade des versions de nos applications. En cas upgrade, le déploiement créera un autre replicaset avec la nouvelle version de l'application, puis basculera ce nouveau replicaset en prod et désactivera le replicaset ayant l'ancienne version.

## 1.10. NAMESPACE

Permettent de créer des environnements isolés dans lesquels les ressources seront créées. Par défaut il existe 03 namespace

- Kube-system : regroupant les ressources nécessaires au fonctionnement de Kubernetes
- Default : regroupant toutes les ressources créées sans précision de namespace
- Public : regroupant les ressources visibles par tous mais ne pouvant être modifiées que par leur créateur.

Sur kubernetes le nom de domaine par défaut qui est créé est « cluster.local »

Pour atteindre un composant dans kubernetes, on précise juste le nom de ce composant si et seulement si le composant avec lequel on souhaite l'atteindre se trouve dans le même namespace que lui, sinon il faudrait préciser le nom de ce composant suivi du nom du « .namespace.type\_composant.cluster.local »

```
mysql.connect("db-service")
```

Ici il s'agit d'un service appartenant au namespace « dev »

```
mysql.connect("db-service.dev.svc.cluster.local")
```

Les namespace permettent également de gérer les quotas, ils permettent de définir le nombre de ressources maximum pouvant être utilisés par un namespace (CPU, RAM, Nbre POD, ...)

## 1.11. SERVICE

Ils permettront d'exposer les pods en interne ou à l'extérieur

Il existe 03 principaux types de services :

- Nodeport : exposer nos applications depuis l'extérieur
- Clusterip : Exposer nos applications en interne
- LoadBalancer : délégation de l'accès au cloud Provider

## II. Scheduling

Il s'agit en effet des différentes stratégies pouvant être mises en œuvre afin de déterminer où est ce que notre application sera déployée au sein du cluster. Il existe plusieurs méthodes pour mettre en place ces stratégies parmi lesquelles :

## 1. La méthode manuelle (Manual Scheduling)

Ici le but est de préciser le paramètre **nodeName** qui obligera le scheduler à envoyer la demande sur un nœud particulier. Ce paramètre doit être précisé dans les specs du node au même niveau d'indentation que celui du container

## 2. Scheduling via labels et node selectors

Cette méthode permet de sélectionner manuellement le ou les nodes sur lesquels déployer nos container dans un premier temps en leur attribuant des labels permettant leur sélection, et puis dans le manifest de création du pod, spécifier au niveau de ses specs le label du node sur lequel ce pod sera déployé.

Pour attribuer un label à un node , on utilise la commande :

- `kubectl label nodes <node-name> <label-key>=<label-value>`. Ex. `kubectl label nodes node01 size=Large`

après avoir attribué un label à nos ou à notre node, on doit préciser ce critère de sélection du node dans la déclaration de notre manifest servant à créer le pod.

**Exemple.**

**nodeSelector :**  
size : Large

- `kubectl get po -l env=prod -l bu=finance -l tier=frontend`

**NB :** le nodeSelector permet de choisir le nœud fonction du label défini

```
pod-definition.yml
apiVersion:
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
    - name: data-processor
      image: data-processor
  nodeSelector:
    size: Large

➤ kubectl create -f pod-definition.yml
```

## 3. Scheduling by TAINTS AND TOLERATION

Il s'agit en effet d'une autre stratégie de déploiement de notre pod ou de notre application. Il s'agit d'une autre sorte de label mais plutôt appelé « taint »

```
kubectl taint nodes node-name key=value:taint-effect
```

NoSchedule | PreferNoSchedule | NoExecute

```
kubectl taint nodes node1 app=myapp:NoSchedule
```

```
pod-definition.yml
apiVersion:
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
  tolerations:
    - key: "app"
      operator: "Equal"
      value: "myapp"
      effect: "NoSchedule"
```

A l'aide du **kubectl taint**, on définit le taint (teinture) à utiliser sur le node et éventuellement les paramètres de cette teinture ; dans notre cas la teinture est app et sa valeur est « myapp ». pour ce qui est paramètres de la teinture (taint-effet) on a :

- **NoSchedule** : qui dit en effet au serveur kubernetes de ne plus planifier de nouveau déploiement sur ce node teinté avec cet effet la ; par contre tous les pods déjà existants sur ce nœud même non teinté au même effet seront conservé sur le nœud.
- **PreferNoSchedule** : qui demande de planifier un déploiement vraiment si ce nœud est le préféré et qu'il n'y ait plus d'autre nœud pouvant accueillir de nouveaux déploiements.
- **NoExecute** : De ne plus planifier sur ce nœud et tous les pods existants n'ayant pas la teinte appropriée sont délogés

Après avoir appliqué la teinte et ses paramètres sur le ou les nœuds, il faut les préciser dans le manifest des pods que l'on souhaite déployer dans ces nœuds-là.

Cela se fait à l'aide du paramètre « tolerations : » pour lequel on précise la key et on dit fonction d'un opérateur la position de cette key par rapport à la valeur qu'on définit plus bas. Dans notre exemple l'opérateur est Equal (donc la clé doit être égale à la valeur définie plus bas) si la clé est égale et que l'effet défini aussi est le même que celui défini lors de la teinte du nœud, alors ce pod sera déployé sur le nœud teinté correspondant. Les opérateurs nous permettent une grande flexibilité car fonction de l'opérateur par exemple s'il avait été NoEqual, alors ce pod aurait été déployé sur tout autre nœud dont le key précisé n'est pas égal à la valeur entrée.

**NB :** On teinte le nœud et on lui donne les caractéristiques de la tolérance qui doit pouvoir appliquer sur les pods voulant se déployer ou déjà déployé sur lui. Pour enlever une teinte, on saisit la même commande en ajoutant – à la fin. Exemple. `Kubectl taint nodes node01 app=myapp:NoSchedule-`

## 4. Scheduling by NODE AFFINITY

Le but ici est de définir des critères ou relations d'affinité entre un pod et un nœud.

```
pod-definition.yml
apiVersion:
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
    - name: data-processor
      image: data-processor
  nodeSelector:
    size: Large
```

```
pod-definition.yml
apiVersion:
kind: Pod
metadata:
  name: myapp-pod
spec:
  containers:
    - name: data-processor
      image: data-processor
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: size
              operator: In
              values:
                - Large
```

Available:  
`requiredDuringSchedulingIgnoredDuringExecution`  
`preferredDuringSchedulingIgnoredDuringExecution`  
  
Planned:  
`requiredDuringSchedulingRequiredDuringExecution`

Cette méthode nous donne la possibilité d'avoir toute une liste de label ou key de nodes pour le choix du nœud sur lequel déployer le pod. Pour cela on définit dans un premier temps les paramètres d'affinité et ensuite la liste de label suivi de l'opérateur de vérification. Dans notre cas la clé size doit appartenir à la liste de valeurs définie plus bas.

Concernant les paramètres d'affinité, on distingue 03 :

- **requiredDuringSchedulingIgnoredDuringExecution** : ici on Oblige pendant la planification du déploiement d'un pod qu'il soit fait dans l'un des nœuds respectant les critères définis plus bas. La deuxième variante de ce paramètre IgnoredDuringExecution est en effet pour ignorer le fait qu'un pod ne respectant pas les critères s'exécute déjà sur ce nœuds, dans ce cas-là, ces pods là ne seront pas éjectés desdits nœuds.
- **preferredDuringSchedulingIgnoredDuringExecution** : ici on n'oblige pas que la planification du déploiement d'un nouveau pod ayant ce paramètre d'affinité soit obligatoirement fait dans les nœuds respectant lesdits critères. On souhaite juste qu'il soit de préférence déployé dans ces nœuds mais si par contre ces nœuds sont saturés ou autre, il pourront être déployés ailleurs. La deuxième variante est similaire à celle du 1<sup>er</sup> param
- **requiredDuringSchedulingRequiredDuringExecution** : obligé de respecter les nœuds définis dans les critères lors de nouveau déploiements et également tous les nœuds ne respectant pas ces critères déjà en cours d'exécution sont éjectés.

**NB:** les Labels Selectors Operator sont:

- **In** : Permet de rechercher si la Key du label fourni appartient à une liste de valeur obligatoire à définir
- **NotIn** : Permet de rechercher si la Key du label fourni n'appartient pas à une liste de valeur obligatoire à définir
- **Exists** : permet juste de vérifier si ce key label existe sur un nœud, ici on ne fournit pas de liste de valeur possible de ce key, car on vérifie juste s'il existe sur un nœud.
- **DoesNotExists** : Key label précisé n'existe pas (le pod est déployé sur les nœuds n'ayant pas ce label)

## 5. Scheduling by TAINT AND AFFINITY

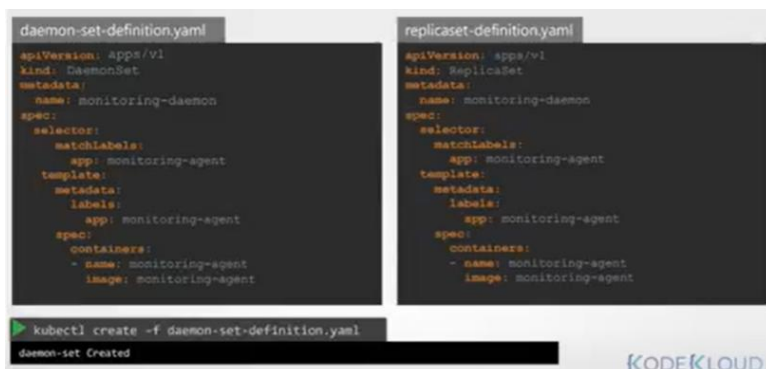
Cette méthode permet en effet de coupler les 02 notions de TAINT et AFFINITY précédemment vue dans le but de forcer un pod à se déployer sur le nœud ayant son taint grâce à l'ajout des paramètres d'affinités.

En effet le paramètre de taint tolérance n'oblige pas un pod à se déployer sur un nœud particulier, mais donne juste un pass à ce pod là à pouvoir s'il le souhaite se déployer sur le nœud ayant cette teinte-là. Les autres pods n'ayant pas cette tolérance seront bloqué par le nœud et ne pourront pas se déployer sur lui. Mais bien que le pod ayant la tolérance il n'est pas obligé de vouloir se déployer sur ce nœud précis. Si on veut ajouter à cela une obligation, alors il va falloir rajouter les paramètres affinity avec le critère required...

## 6. Scheduling by DAEMON SETS

Les Daemon sets permettent d'obliger un pod à être déployé sur l'ensemble de nœuds. Exemple les pods contenant des applications de monitoring devant être obligatoirement déployés sur tous les nœuds afin d'y collecter les logs. Et pour s'assurer que ce pod est bien déployé sur cet ensemble de nœuds, alors on utilise un objet de type daemon sets.

La définition d'un daemon-set se rapproche en effet de celle d'un replicaset, mais les replicaset permettent juste de déployer des pods, mais ne nous garantissent pas que chacun de ces pods seront déployés sur l'ensemble des nœuds comme les daemon sets. Tout comme pour les replicaset, il est important de préciser le matchLabels Selector qui permet de définir les pods qui seront fédérés par ce daemon set là.



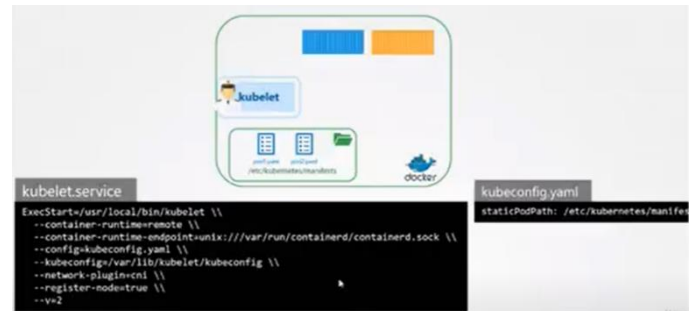
## 7. Scheduling by resources limits

```
spec:
  containers:
  - name: elephant
    image: polinux/stress
  resources:
    limits:
      memory: "20Mi"
    requests:
      memory: "5Mi"
```

## 8. Scheduling by Static pods

Comment planifier le déploiement d'un pods sans utiliser le scheduler. Les static pods sont déployés via kubelet si et seulement si leurs manifests se trouvent dans un dossier particulier.

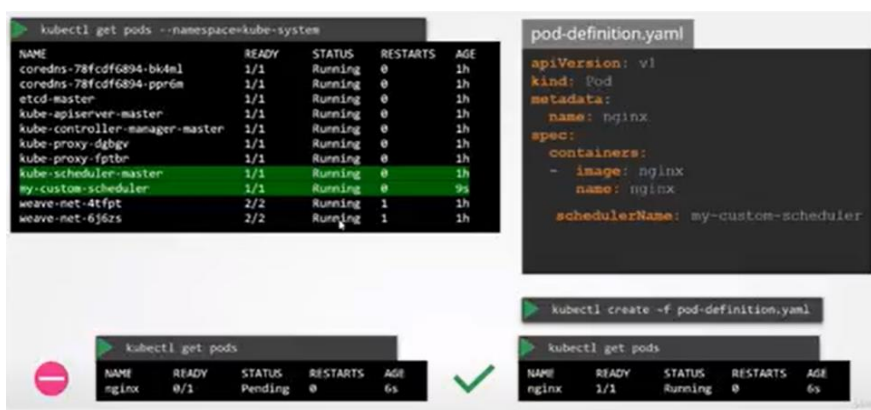
NB : Kubelet étant le client kubernetes à installer sur les workers n'est en effet pas déployé par kubeadm, c'est à nous les administrateurs de l'installer sur les différents workers, et dans sa configuration, il intègre un fichier qui s'appelle kubeconfig.yaml qui lui permet de définir où se trouvent les manifests. Ce manifest contiendra l'ensemble des pods à déployer et ces pods sont appelés pods statiques et c'est pour ça que le nom de ces pods sont toujours suffixés du nom du node sur lequel ils sont déployés. Kubelet au démarrage va aller lancer tous les manifest se trouvant dans le répertoire précisé dans la variable staticPodPath de son fichier de configuration kubeconfig.yaml.



Le répertoire par défaut contenant le fichier de configuration de kubernetes est : `/var/lib/kubelet` et après avoir ouvert le fichier de configuration, rechercher la variable « `staticPodPath` » qui précise où aller chercher les manifest des static pods. Après avoir créé votre nouveau manifest dans ledit répertoire, il faut redémarrer kubelet « `systemctl restart kubelet` »

## 9. Scheduling by Multiple SCHEDULER

Il est possible de déployer son propre scheduler ou alors de le paramétrer à notre guise. Il s'agit en effet ici d'utiliser la précédente méthode de création de static pods pour créer notre propre scheduler qui se lancera automatiquement au lancement de kubelet. Une fois ce scheduler créé, il faudra spécifier à nos pods lors de leurs création qu'ils devront plutôt utiliser notre nouveau scheduler au lieu de celui par défaut afin de déterminer dans quel nœud ils seront déployés. Pour cela on utilise le paramètre : `schedulerName : my-custom- scheduler`



Lors de la création de notre propre scheduler, nous devons appliquer les paramètres suivants dans les commandes

- scheduler-name=my-scheduler
- port=10282
- secure-port=0
- leader-elect=false

Après cela modifier le corps du code en remplaçant le nom du container, le numéro de port par 10282 et se mettre en http au lieu de HTTPS

Sauvegarder le fichier dans un autre répertoire que celui des manifests et faire la commande **Kubectl create -f my-scheduler.yml**

## III. Imperative Commands

### 1) Création d'un pod

- `kubectl run --image=nginx:alpine nginx-pod`

### 2) Création d'un pod avec label

- `kubectl run --image=redis:alpine -l tier=db redis`

### 3) Création d'un service pour exposer un pod

- `kubectl expose pod redis --port=6379 --name=redis-service`

### 4) Créer un deployment avec 3 replicas

- `kubectl create deployment webapp --image=kodekloud/webapp-color --replicas=3`

### 5) Créer un pod en exposant son container à un port

- `kubectl run --image=redis:nginx --port=8080 custom-nginx`

### 6) Créer un nouveau namespace

- `kubectl create namespace dev-ns`

### 7) Créer un deployment dans un namespace

`kubectl create deployment redis-deploy --image=redis --replicas=2 -n dev-ns`

### 8) Créer un service de type cluster IP

- `kubectl create svc clusterip httpd --tcp=80`

### 9) Créer un service de type clusterIp pour exposer un pod à travers le port 80

- `Kubectl expose pod httpd --port=80 --name=httpd`

<https://blog.zenika.com/2018/12/18/certification-kubernetes-ils-passent-la-cka-et-vous-disent-tout/>

## IV. Logging and Monitoring

Il est important de réfléchir à une stratégie pour les logs et le monitoring lorsqu'on a une infra en production. Nous avons pour cela :

- Docker logs : `docker logs -f ecf` (l'option `-f` permet d'avoir en temps réels les logs de l'objet)
- Kubernetes logs
- `kubectl logs <pod-name> <Container-name> -ff`

Avec kubernetes, lorsqu'on a un pod ayant plusieurs containers, pour avoir les logs d'un container, il faut mettre en plus du nom du pod, le container sur lequel on souhaite réellement avoir les logs

Le monitoring permet de remonter un certain nombre de métriques liées à la consommation de notre infrastructure, dans le but de l'améliorer, ou optimiser l'utilisation des ressources. Plusieurs solutions sont disponibles :

- Metrics Server
- Prometheus
- Elastic stack
- Datadog

La solution la plus utilisée pour l'examen CKA est metrics server qui se télécharge à partir d'un lien git avec application de la commande `kubectl create -f` pour création des pods ou ressources décrites dans les manifests de ce répertoire Git.

Après installation de Metric serveurs, on peut avoir accès aux commandes de monitoring suivantes :

- Commande pour avoir les informations cpu et RAM des nodes : `kubectl top node`
- Pour voir la consommation des pods c'est `kubectl top pod`

## V. Application Lifecycle Management

Comment gérer le cycle de vie de notre application, la mettre à jour (Rolling Update), faire du rollback.

Kubernetes propose 02 stratégies pour permettre les mises à jour ou les rollBack :

- Recreate : ici kubernetes supprime tout ce qu'il y'avait et il déploie les nouveaux pods dans la nouvelle version souhaitée (inconvenient, temps d'indisponibilité de l'application)
- Rolling Update : ici l'administrateur définit certains paramètres pour obliger kubernetes à ne pas tous supprimer les pods d'un seul coup il définit le nombre de pods max pouvant être supprimés en simultanés et le nombre maximum de pods que l'on peut créer à la fois

NB : si aucune stratégie n'est précisée lors de la création du pod, alors c'est celle du Rolling update qui est choisie permettant ainsi de réduire les down time.

La commande pour update la version de l'image d'un conteneur sur un pod est :

- `kubectl set image deployment/<nom-deployment> <container-name>=<image_name>:version`

Une fois l'application mise à jour, s'il faille faire un Rollback vers la version précédente, alors on utilise la cde :

- `kubectl rollout undo deployment/<nom-deployment>`

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      name: webapp
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        name: webapp
    spec:
      containers:
        - image: kodekloud/webapp-color:v2
          name: simple-webapp
          ports:
            - containerPort: 8080
              protocol: TCP
```



## 5.1. Application Lifecycle Management

```
kubectl run --image=kodekloud/webapp-color --env APP_COLOR=green -l name=webapp-color webapp-color
```

**ConfigMap**  
APP\_COLOR: blue  
APP\_MODE: prod

**Imperative**  
1  
Create ConfigMap

```
kubectl create configmap  
<config-name> --from-literal=<key>=<value>
```

```
kubectl create configmap \
```

```
  app-config --from-literal=APP_COLOR=blue \
```

```
  --from-literal=APP_MODE=prod
```

```
kubectl create configmap  
<config-name> --from-file=<path-to-file>
```

```
kubectl create configmap \
```

```
  app-config --from-file=app_config.properties
```

**Secret**  
DB\_Host: mysql  
DB\_User: root  
DB\_Password: paswr

**Imperative**  
1  
Create Secret

```
kubectl create secret generic  
<secret-name> --from-literal=<key>=<value>
```

```
kubectl create secret generic \
```

```
  app-secret --from-literal=DB_Host=mysql \
```

```
  --from-literal=DB_User=root
```

```
  --from-literal=DB_Password=paswr
```

```
kubectl create secret generic  
<secret-name> --from-file=<path-to-file>
```

```
kubectl create secret generic \
```

```
  app-secret --from-file=app_secret.properties
```

**Secret**  
DB\_Host: mysql  
DB\_User: root  
DB\_Password: paswr

**Declarative**  
1  
Create Secret

```
kubectl create -f
```

**secret-data.yaml**

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: app-secret  
data:  
  DB_Host: bXlzcWw=  
  DB_User: cm9vdA==  
  DB_Password: cGFzd3Jk
```

```
kubectl create -f secret-data.yaml
```

```
DB_Host: mysql
DB_User: root
DB_Password: paswr
```

```
DB_Host: bXlzcWw=
DB_User: cm9vdA==
DB_Password: cGFzd3Jk
```

```
> echo -n 'mysql' | base64
bXlzcWw=
```

```
> echo -n 'root' | base64
cm9vdA==
```

```
> echo -n 'paswr' | base64
cGFzd3Jk
```

```
DB_Host: mysql
DB_User: root
DB_Password: paswr
```

```
DB_Host: bXlzcWw=
DB_User: cm9vdA==
DB_Password: cGFzd3Jk
```

```
> echo -n 'bXlzcWw=' | base64 --decode
mysql
```

```
> echo -n 'cm9vdA==' | base64 --decode
root
```

```
> echo -n 'cGFzd3Jk' | base64 --decode
paswr
```

## pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
    - name: simple-webapp-color
      image: simple-webapp-color
      ports:
        - containerPort: 8080
      envFrom:
        - secretRef:
            name: app-secret
```

## secret-data.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  DB_Host: bXlzcWw=
  DB_User: cm9vdA==
  DB_Password: cGFzd3Jk
```



```
> kubectl create -f pod-definition.yaml
```

```
envFrom:
  - secretRef:
      name: app-config
```

ENV

SINGLE ENV

```
env:
  - name: DB_Password
    valueFrom:
      secretKeyRef:
        name: app-secret
        key: DB_Password
```

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

VOLUME

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

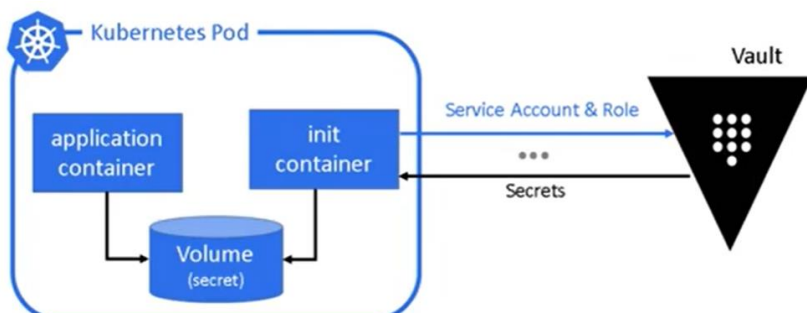
VOLUME

```
> ls /opt/app-secret-volumes
```

```
DB_Host  DB_Password  DB_User
```

```
> cat /opt/app-secret-volumes/DB_Password
paswr
```

Inside the Container



```
kubectl get pod orange -o yaml > /root/orange.yaml : Permet de récupérer un objet kubernetes dans un fichier yaml
```

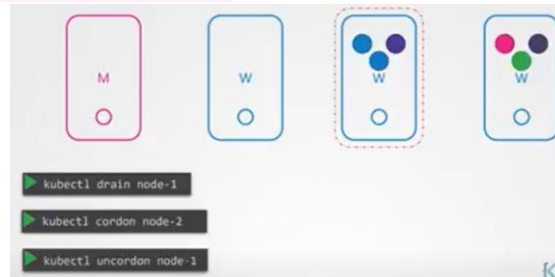


## 5.2. Maintenance du cluster

### a) Node update

Pour mettre à jour l'OS d'un node de notre cluster, on doit se rassurer que notre upgrade dure moins de 5 mins ; si le temps nécessaire est plus de 5 min, alors il faudra

- Evincer l'ensemble des pods sur le node en question : `kubectl drain node01`
- Il faut dire à kube de ne plus envoyer de pod sur ce node-là : `kubectl cordon node01`
- Pour enlever le cordon : `kubectl uncordon node01`

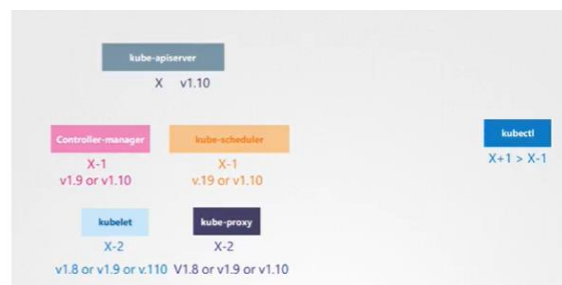


```
kubectl drain node01 --ignore-daemonsets
```

```
kubectl drain node01 --ignore-daemonsets --force
```

#Supprimer tous les pods même ceux qui ne sont pas gérés par des replicaset, donc ne pourront pas être déployés sur les autres nodes

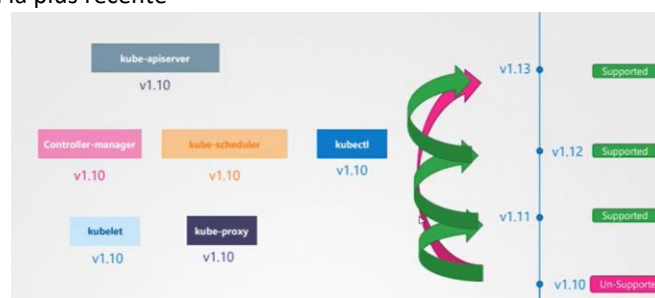
### b) Kubernetes Upgrade



**NB** : si le Kube-Apiserver est dans une version X, alors le controller-manager et le kube-scheduler doivent être au moins en version X-1 et kubelet et kube-proxy quant à eux devront être également au moins en version X-2

Le Kubectl quant à lui peut être au plus à une version X+1 ou au moins à une version X-1

Les bonnes pratiques d'upgrade Kubernetes demandent de faire l'upgrade progressivement d'une version à une autre jusqu'à arriver à la cible qui est la version la plus récente



Il existe plusieurs méthodes d'upgrade de kubernetes, dépendant de la façon dont il a été initialement installé :

- Si supporté par un cloud provider : juste cliquer sur le bouton update available pour lancer
- Si déployé à l'aide de kubeadm : saisir les commandes `[kubeadm upgrade plan]` et `[kubeadm upgrade apply]`
- Pour le faire à la mains : installer les package à la main avec les apt et en récupérant les binaires

Il est également important de bien choisir la méthode à utiliser pour l'upgrade de kubernetes sur l'ensemble des nodes :

- Soit upgrader en simultanément tous les worker : créera de l'indisponibilité au niveau des services
- Soit upgrader chaque nœud à son tour (drain, cordon pour le rendre indisponible)
- Déployer un nouveau nœud avec la nouvelle version, basculer les pods sur ce nouveau nœud là et après rendre indisponible tous les autres nœuds et procéder à leur upgrade.

```
kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade/versions] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.11.3
[upgrade/versions] Latest stable version: v1.13.4
[upgrade/versions] Latest version in the v1.11 series: v1.11.8

Components that must be upgraded manually after you have
upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT  CURRENT  AVAILABLE
Kubelet    3 x v1.11.3  v1.13.4

Upgrade to the latest stable version:
COMPONENT  CURRENT  AVAILABLE
API Server  v1.11.8  v1.13.4
Controller Manager  v1.11.8  v1.13.4
Scheduler   v1.11.8  v1.13.4
Kube Proxy  v1.11.8  v1.13.4
CoreDNS     1.1.3    1.1.3
Etcd        3.2.18   N/A

You can now apply the upgrade by executing the following command:
kubeadm upgrade apply v1.13.4

Note: Before you can perform this upgrade, you have to update kubeadm to v1.11.4.
```

```
apt-get upgrade -y kubeadm=1.12.0-00
kubeadm upgrade apply v1.12.0
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.0". Enjoy!
[upgrade/kubelet] Now that your control plane is upgraded, please proceed with
upgrading your kubelets if you haven't already done so.

kubectl get nodes
NAME      STATUS    ROLES    AGE    VERSION
master    Ready     master   1d     v1.11.3
node-1    Ready     <none>   1d     v1.11.3
node-2    Ready     <none>   1d     v1.11.3

apt-get upgrade -y kubelet=1.12.0-00
systemctl restart kubelet
```

KODECLOUD

Sur un cluster installé à partir de kubeadm, lorsqu'on entre la cde [kubeadm upgrade plan], ce dernier nous affiche les possibilité d'upgrade comme nous pouvons le voir ci-dessus, toutefois il nous met en garde qu'il faudra déjà upgrade kubeadm vers cette nouvelle version avant de pouvoir upgrade ses composants et pour upgrade kubeadm :

- `Apt-get upgrade -y kubeadm=1.12.0-00` (le -00 de la fin est indispensable) après avoir terminé l'upgrade de kubeadm, on procède à l'upgrade de ses composants avec :
- `Kubeadm upgrade apply v1.12.0`

Il faut également par la suite mettre à jour le kubelet qui est l'agent kubelet indispensable au fonctionnement de kubernetes.

NB : répéter les mêmes actions sur les workers l'un après les autres en décommissionnant les pods et en mettant à jour le kubeadm (kube-proxy) et le kubelet des workers.

On the controlplane node, run the command run the following commands:

1. `apt update`  
This will update the package lists from the software repository.
2. `apt install kubeadm=1.20.0-00`  
This will install the kubeadm version 1.20
3. `kubeadm upgrade apply v1.20.0`  
This will upgrade kubernetes controlplane. Note that this can take a few minutes.
4. `apt install kubelet=1.20.0-00` This will update the kubelet with the version 1.20.
5. You may need to restart kubelet after it has been upgraded.  
Run: `systemctl restart kubelet`

On the node01 node, run the command run the following commands:

1. If you are on the master node, run `ssh node01` to go to node01
2. `apt update`  
This will update the package lists from the software repository.
3. `apt install kubeadm=1.20.0-00`  
This will install the kubeadm version 1.20
4. `kubeadm upgrade node`  
This will upgrade the node01 configuration.
5. `apt install kubelet=1.20.0-00` This will update the kubelet with the version 1.20.
6. You may need to restart kubelet after it has been upgraded.  
Run: `systemctl restart kubelet`

## 5.3. Backup et restauration

Les principales ressources ou données à sauvegarder dans un cluster kubernetes sont :

- Les fichier de configuration du cluster et des ressources (config et Yaml...)
- La base de données ETCD Cluster
- Les différents volumes persistants existants

### a) Fichiers de configuration des ressources

Concernant les fichiers de configuration des ressources, il peuvent être sauvegardés des manière suivantes :

- sur Git ou sur un autre SCM si les ressources ont été créées à l'aide de manifest
- On peut également créer un manifest de toutes les ressources à l'aide de la commande : `Kubectl get all --all-namespaces -o > all-deploy-services.yaml`
- On peut également utiliser l'outil VELERO qui permet de réaliser les backup et la restauration des ressources du cluster.

`etcdctl snapshot save snapshot.db`

### b) La base de données ETCD

```
ETCDCTL_API=3 etcdctl \
  snapshot save snapshot.db

ls
snapshot.db

ETCDCTL_API=3 etcdctl \
  snapshot status snapshot.db

+-----+-----+-----+-----+
| HASH | REVISION | TOTAL KEYS | TOTAL SIZE |
+-----+-----+-----+-----+
| e63b3fc5 | 473353 | 875 | 4.1 MB |
+-----+-----+-----+-----+

ETCDCTL_API=3 etcdctl \
  snapshot restore snapshot.db \
  --data-dir /var/lib/etcd-from-backup \
  --initial-cluster master-
1=https://192.168.5.11:2380,master-
2=https://192.168.5.12:2380 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-advertise-peer-urls
https://${INTERNAL_IP}:2380

I | mvcc: restore compact to 475629
I | etcdserver/membership: added member 5e89ccdf3
[https://192.168.5.12:2380] to cluster 894c7131f5165a78
I | etcdserver/membership: added member c8246cee7c
[https://192.168.5.11:2380] to cluster 894c7131f5165a78

systemctl daemon-reload

service etcd restart
Service etcd restarted

etcd.service
ExecStart=/usr/local/bin/etcd \
  --name ${ETCD_NAME} \
  --cert-file=/etc/etcd/kubernetes.pem \
  --key-file=/etc/etcd/kubernetes-key.pem \
  --peer-cert-file=/etc/etcd/kubernetes.pem \
  --peer-key-file=/etc/etcd/kubernetes-key.pem \
  --trusted-ca-file=/etc/etcd/ca.pem \
  --peer-trusted-ca-file=/etc/etcd/ca.pem \
  --peer-client-cert-auth \
  --client-cert-auth \
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \
  --listen-peer-urls https://${INTERNAL_IP}:2380 \
  --listen-client-urls https://${INTERNAL_IP}:2379,https://${INTERNAL_IP}:2380 \
  --advertise-client-urls https://${INTERNAL_IP}:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster controller-0=https://${CONTROLLER_IP}:2380 \
  --initial-cluster-state new \
  --data-dir=/var/lib/etcd-from-backup
```

NB : pour sauvegarder, il faut préciser les paramètres suivants

- 1) La version de ETCDCTL\_API à utiliser
- 2) Le Endpoint (l'url de l'etcd en question) : `--endpoints= ""`
- 3) Les paramètres pour se connecter à cet etcd ca certificat (`--cacert=""`)
- 4) Certificat de connexion (`--cert=""`)
- 5) La clé de connexion (`--key=""`)

```
ETCDCTL_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot save /opt/snapshot-pre-boot.db
```

et pour la restauration :

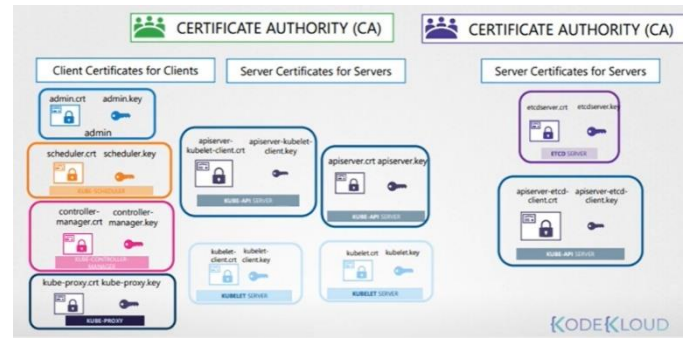
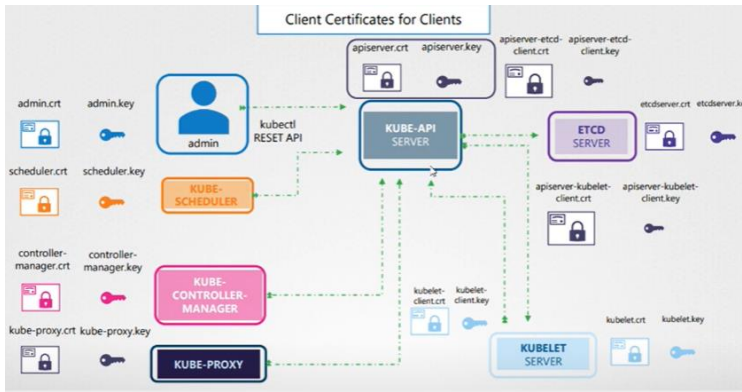
```
ETCDCTL_API=3 etcdctl --data-dir /var/lib/etcd-from-backup \
snapshot restore /opt/snapshot-pre-boot.db
```

après il faut modifier le fichier de création du static pod etcd en modifiant l'emplacement (path) du volume etcd-data.

- ➔ `Systemctl daemon-reload`
- ➔ `Systemctl restart kubelet`

# VI. Sécurité

## 6.1. TLS CERTIFICATE



Sur Kubernetes, il est nécessaire d'avoir un certificat + key pour pouvoir communiquer avec ses différentes ressources.

Il existe 02 types de certificats sur Kubernetes :

- Les certificats + key de type serveur que possèdent les ressources de types serveurs (Kube-Apiserver, ETCD et Kubelet) sur lesquelles les clients peuvent avoir besoin de s'y connecter pour demander un service
- Les certificats + key de type client que les clients devront présenter aux serveurs afin qu'ils leur donnent l'accès.

Sur kubernetes installé à l'aide de kubeadm, les certificats serveurs et admin principal sont générés automatiquement à l'installation et se trouvent dans le dossier /etc/kubernetes/pki/

Les certificats génériques existants sont :

### - KUBE-APISERVER

- Certificat + key de type serveur : /etc/kubernetes/pki/apiserver.crt et \*.key
- Certificat + key de type client pour se connecter à ETCD : --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt et \*.key
- Certificat + key de type client pour se connecter à Kubelet : --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt et \*.key
- Certificat + key de l'autorité de certificat kubernetes-ca : /etc/kubernetes/pki/ca.crt et \*.key
- Certificat + key de l'autorité de certificat etcd-ca : --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt et \*.key

### - ETCD

- Certificat + key ETCD server : --cert-file=/etc/kubernetes/pki/etcd/server.crt et --key-file=\*.key
- Certificat de l'autorité de certificats etcd-ca : --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt et \*.key

### - KUBELET

- Certificat + key kubelet server : /etc/kubernetes/pki/kubelet.crt et kubelet.key
- Certificat de l'autorité de certificat kubernetes-ca : /etc/kubernetes/pki/ca.crt et \*.key

### - LES ADMINIS

- Certificat + key client lui permettant de se connecter à API-server : <nom-admin>.crt et \*.key
- Certificat de l'autorité de certificat kubernetes-ca : /etc/kubernetes/pki/ca.crt et \*.key

**NB** : comme vous avez pu les remarquer, il existe 02 autorité de certificats sur kubernetes :

- L'autorité de certificat Kubernetes-ca pour api-server et kubelet
- L'autorité de certificat etcd-ca pour générer les certificat nécessaire pour ETCD

Tous ces certificats doivent être bien définis dans les manifests yaml permettant de créer les ressources static pods de l'api-server, kubelet et etcd se trouvant dans /etc/kubernetes/manifests

**NB** : la commande pour voir le contenu détaillé d'un certificat est : `openssl x509 -in <file-path>.crt -text`

## 6.2. CERTIFICAT API

Il s'agit ici de créer un nouvel utilisateur à kubernetes. Le processus de création d'utilisateurs sur kubernetes se fait en plusieurs étapes :

1. Création de la demande de certificats (CSN ou CertificateSigningRequest)
2. Après la création de la CSR qui est en effet un objet kubernetes, un admin existant devra vérifier ce CSR
3. Après vérification l'admin existant devra approuver ou refuser la demande de certificat



- 
- The diagram illustrates the process of creating a Certificate Signing Request (CSR) and managing certificates using the Certificates API. It is divided into three main sections:
- Left Section (Users):** Shows two user icons, one blue and one red, each with a small square icon next to it, representing users who can request or receive certificates.
  - Middle Section (Steps):** A vertical list of four steps, each in a rounded rectangle:
    1. Create CertificateSigningRequest Object
    2. Review Requests
    3. Approve Requests
    4. Share Certs to Users
  - Right Section (Certificates API):** Shows a large rounded rectangle labeled "CERTIFICATES API" containing a grid of colored squares (green, orange, blue, red) and a large circle below it. To the right of this are two smaller rounded rectangles, each containing two small blue squares and a circle, representing individual certificates or user profiles.

```

>openssl gensrsa -out jane.key2048
jane.key

>openssl req -new -key jane.key -subj "/CN=jane" -out jane.csr
jane.csr

-----BEGIN CERTIFICATE REQUEST-----
MIIODCCAAQAwEiZRMABGAlUEAwIB3VXZlVwggEIMARCSG5SId3Q0EB
AQUAAIAIB0wggEwEiBAQ000KjW+DKsAJS1r-jpNo5vR18pInzg+6x6c94Uwkk18
LFc27+1eEnONSjMu99NvnmPEdnrDUJ/thyVq2X2XNIDRKhYyF48Fbm+5zcyK
9wBBAQsFAAOCAQEA591S6C1uxTuFs8BY5U70FQilzaiNAdYs+DRRQhw4ZwHg6I4
hOK4a2zyNy144001jYaD6t1w8DSxkrBBLK8K3s+rEt3q15LZy91RVrs+Jghd4gV
P9NL+adRSxROVSqBa82mWpYmScJ5TF51s1sNSNLQ2+++RMnJ0Q37jvP1cB/dk1
Wz2UM6Uawzykrd1m72mYm9rDNtVY1v+eH0H/YE1+FSGjH5LSYU1Idq1y
413E/y3qL71mfAcfH305vPlUnQj1SmQ9sBqMCS66CCSDHGPZ1pbnKlUpAwka+8E
vWQ87jg+hpkmxfAeXgUwodALa37uJ/TD1Cw==
-----END CERTIFICATE REQUEST-----

```

- [illegible]

```
kubectl get csr
```

NAME	AGE	REQUESTOR	CONDITION
jane	10m	admin@example.com	Pending

```
kubectl certificate approve jane
```

```
jane approved!
```

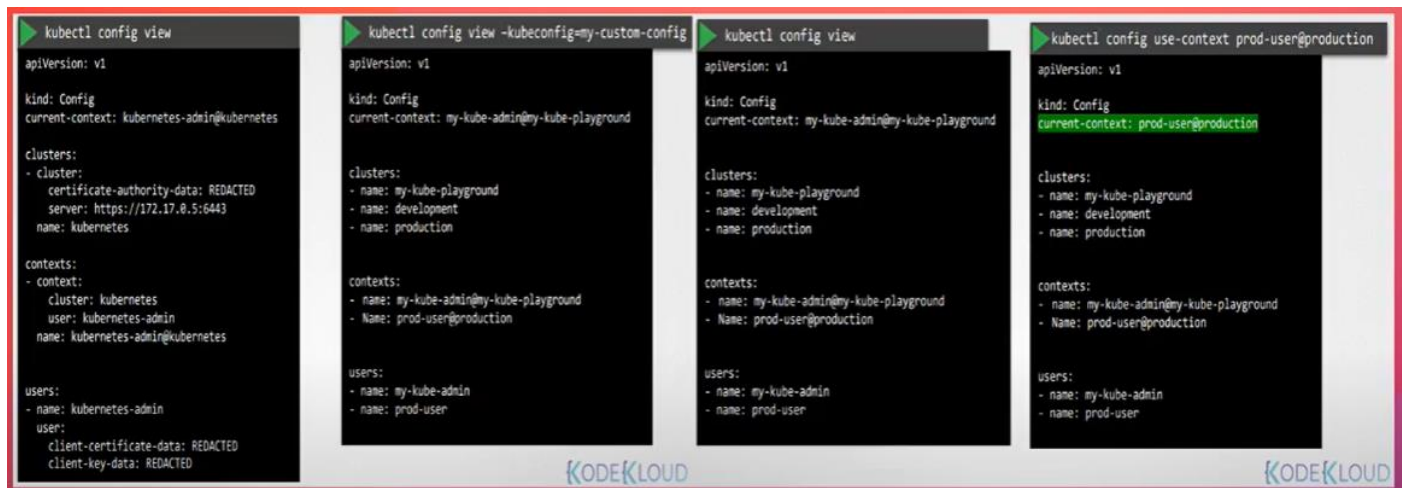
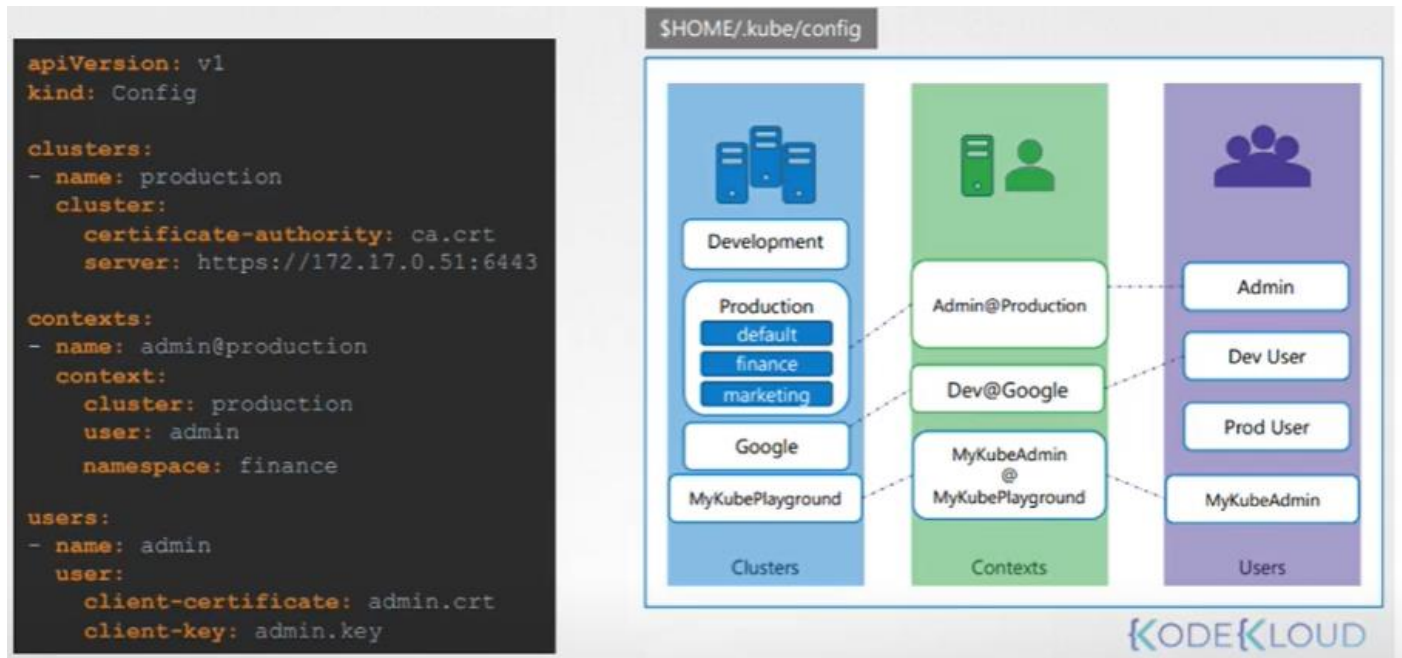
- Kubect1 certificate approve/deny <csr-name>
- Kubect1 get csr # la liste des csr
- Kubect1 get csr <csr-name> -o yaml #Détails objet
- Kubect1 delete csr <csr-name>
- Kubect1 describe csr <csr-name>

**NB :** le fichier de config kubernetes par défaut se trouve dans la variable d'environnement KUBECONFIG



Ce fichier de configuration `/home/user/.kube/config` nous permet en effet de configurer et déterminer ainsi les accès qui seront utilisés sur les différents clusters kubernetes existants. Il est généralement divisé en plusieurs sections :

- **Clusters** : permettant de définir la liste de clusters existants (leur nom, le certificat de l'autorité de certificat à utiliser et le nom ou l'adresse IP dudit cluster)
- **Users** : permettant de définir la liste des user en spécifiant pour chaque user sa clé et son certificat reçu du cluster kubernetes vers lequel il souhaitera se connecter
- **Contexts** : contenant la liste des contextes qui permettent en effet de faire l'association entre un user et un cluster et dans l'interface de kubernetes, quand on souhaite utiliser un contexte, cela veut dire qu'on souhaite se connecter au cluster défini dans ce cluster avec le user également défini.



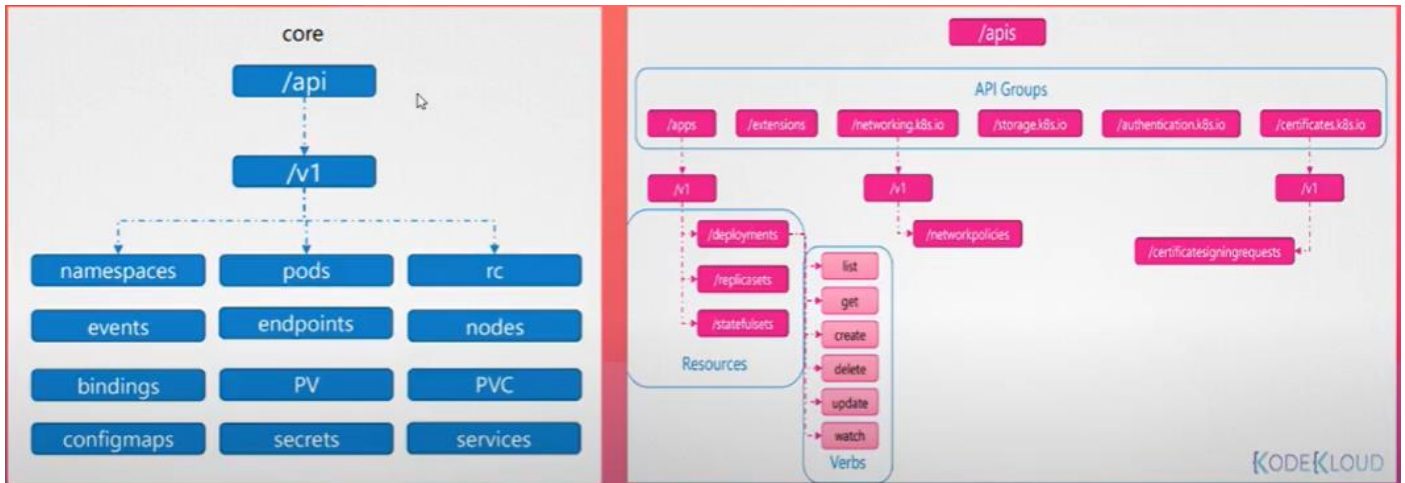
On peut également appliquer un contexte se trouvant dans un fichier de configuration autre que celui par défaut :

`Kubectl config use-context -kubeconfig=<my-custom-config-file> <context-name>`

`Kubectl config view -kubeconfig=<my-custom-config-file> # pour voir le contenu du custom config-file`

`Kubectl config use-context <context-name> #utilise le config-file par défaut et bascule le contexte`

## 6.4. API GROUPS



Lorsqu'on interagit avec kubernetes, il est intéressant de savoir exactement sur quel objet on travaille. Il existe à cet effet les api group Core et les autres apis. (NB : cf. fichier manifest de création des ressources). Il est important de connaître la structure de ces différents api group car il sont utilisés pour remplir les champs apiVersion et kind lors de la création des manifests.

**kubectl proxy** : permet d'exposer les api group d'un cluster kubernetes en local sur le port 8001 afin qu'il puisse être accessibles sans avoir besoin de fournir les informations liées aux certificats.

```
curl http://localhost:6443 -k
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {
  },
  "status": "Failure",
  "message": "Forbidden: User \"system:anonymous\" cannot get path \"/\".",
  "reason": "Forbidden",
  "details": {
  },
  "code": 403
}

curl http://localhost:6443 -k --key admin.key --cert admin.crt --cacert ca.crt
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/healthz",
    "/logs",
    "/metrics",
    "/openapi/v2",
    "/swagger-2.0.0.json",
    "/swagger-2.0.0.json"
  ]
}
```

`curl http://localhost:6443 -k` #permet d'afficher la liste des api groups utilisables par l'utilisateur kubernetes connectés. L'attribut « -k » permet de ne pas fournir les infos d'authentifications, on remarque dans ce cas que la requête s'affiche mais kubernetes refuse d'afficher la liste d'api car il n'y a pas d'infos d'autorisation.

## 6.5. RBAC (Role Based Access Controls)

Cette notion permet lors de l'administration d'un cluster kubernetes, de définir de façon détaillée les droits d'accès aux ressources des différents users existants.

Cela permet de créer un certains nombres de droits et de les assigner à des users spécifiques.

Pour vérifier le type d'autorisations définis sur un pod, il faut taper la commande `kubectl describe` sur ce pod et regarder la valeur de son paramètre `authorization`

```
kubectl get role --all-namespaces : Affiche la liste des rôles existants dans l'ensemble des namespaces
kubectl get role -n kube-system kube-proxy -o yaml: affiche les details sur creation du role kube proxy
kubectl get rolebindings.rbac.authorization.k8s.io --all-namespaces
kubectl get rolebindings.rbac.authorization.k8s.io -n kube-system kube-proxy -o yaml
kubectl --as dev-user get po
```

**NB:** utiliser des manifests yaml pour créer des roles et des roleBinding pour associer ces rôles là aux users.

```
kubectl create -f devuser-developer-binding.yaml
```

```

developer-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list", "get", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["ConfigMap"]
  verbs: ["create"]

devuser-developer-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devuser-developer-binding
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer
  apiGroup: rbac.authorization.k8s.io

```

```

kubectl auth can-i create deployments
yes

kubectl auth can-i delete nodes
no

kubectl auth can-i create deployments --as dev-user
no

kubectl auth can-i create pods --as dev-user
yes

kubectl auth can-i create pods --as dev-user --namespace test
no

```

**NB** : les clusters Rôles permettent de manipuler ou de donner des autorisations aux objets n'ayant pas de namespace ou n'étant pas cloisonnés à un namespace, donc s'appliquant à tout le cluster.

Use the command `kubectl create` to create a new `ClusterRole` and `ClusterRoleBinding`. Assign it correct resources and verbs.

After that test the access using the command `kubect1 auth can-i list storageclasses --as michelle`.

`Kubect1 api-resources` : permet de voir la liste de toutes les ressources existantes dans kubernetes.

```

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: storage-admin
rules:
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["get", "watch", "list", "create", "delete"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "watch", "list", "create", "delete"]

```

```

---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: michelle-storage-admin
subjects:
- kind: User
  name: michelle
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: storage-admin
  apiGroup: rbac.authorization.k8s.io

```

## 6.6. Image Security

Il peut arriver de vouloir déployer un container dont l'image provient d'un registre privé d'entreprise pour lequel il faut s'authentifier avant de pouvoir télécharger ladite image. A cette fin kubernetes prévoit le mécanisme suivants :

```

docker login private-registry.io

docker run private-registry.io/apps/internal-app:1.0

```

```

nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image:
      imagePullSecrets:
      - name: regcred

```

```

kubectl create secret docker-registry regcred \
  --docker-server= private-registry.io \
  --docker-username= registry-user \
  --docker-password= registry-password \
  --docker-email= registry-user@org.com


```

- 1) Créer un type de secret particulier de type `docker-registry` auquel on fournira les informations d'authentification
  - a. Nom du server ou adresse du registre (`--docker-server=`)
  - b. Username (`--docker-username=`)
  - c. Password (`--docker-password=`)
  - d. Adresse mail (`--docker-email=`)
- 2) Ensuite il faudra dans le manifest de création du pod utilisant l'image de ce registre privé utiliser l'attribut « `imagePullSecrets` » pour lequel on fournira le paramètre `name` = au nom du secret de type `docker-registry` initialement crée.

## 6.7. Context Security

La notion de Security context permet lors du déploiement du container sur kubernetes de définir un context dans lequel ce

```
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
    - name: ubuntu
      image: ubuntu
      command: ["sleep", "3600"]
      securityContext:
        runAsUser: 1000
        capabilities:
          add: ["MAC_ADMIN"]
```




**Note:** Capabilities are only supported at the container level and not at the POD level

```
# Example_context_Def

securityContext:
  runAsUser: 1000
  runAsGroup: 3000
  fsGroup: 2000
  fsGroupChangePolicy: "OnRootMismatch"
```

conteneur doit s'exécuter et également de fournir les autorisation ou les capacités qui pourront être utilisés par ce container à travers le context. Le context comme dans notre exemple ici permet de lancer le container comme l'utilisateur 1000 (UserID ou groupID) et lui ajoute un certain nombre de capacités. La notion de capacité est interne à un container et ne peut être définie qu'à l'intérieur d'un container pourtant celle de context peut être mise dans la section spec d'un manifest au même niveau que « containers » et là ce context s'appliquera à l'ensemble des containers définis plus bas (conteneurs pour lesquels le contexte n'est pas défini)

## 6.8. Network Policy



Rule	Type	Port
1	Ingress	80
2	Egress	5000
3	Ingress	5000
4	Egress	3306
5	Ingress	3306

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              name: api-pod
      ports:
        - protocol: TCP
          port: 3306
```

Les networks Policy sont des sortes de pare-feu ou Security groupe qu'on peut mettre sur les différents pods afin de filtrer le trafic réseau entrant (ingress) ou sortant (egress) vers ce pods via des ports particuliers.

**NB :** toutes les fonctions communes aux objets kubernetes s'appliquent également aux networkpolicy.

## VII. Stockage

### 7.1. Les Volumes / PersistentVolumes

Permettent de rendre les données des containers persistantes

```
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
    - image: alpine
      name: alpine
      command: ["/bin/sh", "-c"]
      args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
      volumeMounts:
        - mountPath: /opt
          name: data-volume
  volumes:
    - name: data-volume
      hostPath:
        path: /data
        type: Directory
```

```
persistentvolumeclaim-definition.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-vol1
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

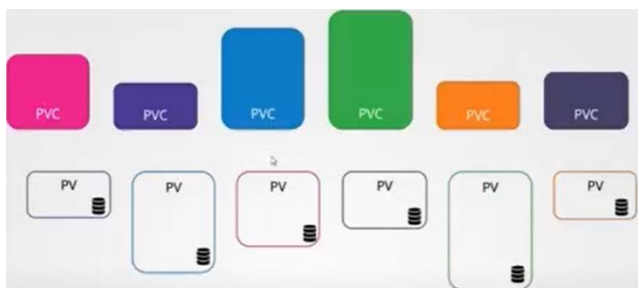
```
kubectl create -f pv-definition.yaml
```

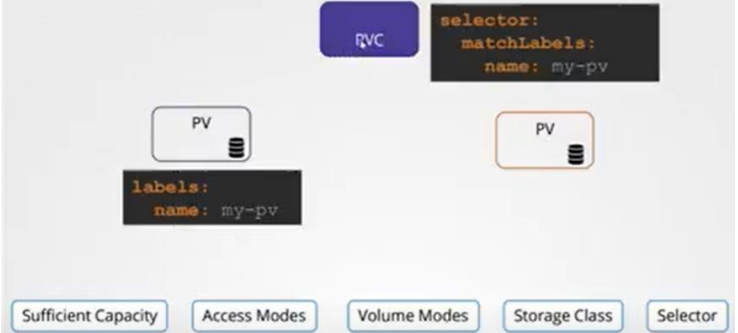
```
kubectl get persistentvolume
```



Après avoir créé le PV, il faut créer un PVC qui sera lié à ce PV et pourra être consommé par nos pods. La liaison entre PV et PVC se fait fonction de certains critères définis par kubernetes suivants :

- Critère de capacité** : est-ce que la capacité demandée par le PVC correspond à tel ou tel PV
- Le type d'accès mode configuré** sur le PVC : le PVC choisira comme PV, celui qui est libre et qui correspond le mieux à son access mode
- Volume Modes** : Permet de définir un paramètre kubernetes liés aux volumes
- Storage class** : Permet de définir le type de stockage ou le driver qui permettra de provisionner le PVC(le PVC choisit le PV dont le storage Class est équivalent)
- Selector** : Ne cherche à te lier à un PV uniquement si son label match avec ce qui est entré dans le selector





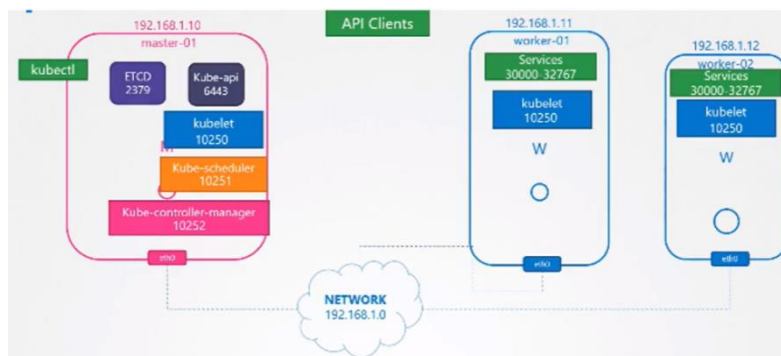
```
pvc-definition.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

kubectl create -f pvc-definition.yaml

```
pv-definition.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-vol1
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

## VIII. Reseau

### 8.1. Les ports



Dans un cluster kubernetes, il y'a un certain nombre de ports par défaut qui doivent être absolument ouverts pour permettre le bon fonctionnement des composants. Ces ports permettent en effet l'interconnexion entre les différents modules et garantissent ainsi le bon fonctionnement de nos services

Cf. image ci-après.

### 8.2. CNI dans KUBERNETES

Le CNI (Container Network Interface) est en effet un plugin utilisé pour donner une interface réseau (permettant la communication interne entre l'hôte et le container) à un container. Il permet en effet de lier une interface réseau à un container afin que ce dernier puisse avoir une adresse IP dans le réseau Kubernetes. La liste des plugins ou CNI existants se trouvent dans le répertoire /opt/cni/bin.

/etc/cni/net.d/ : permet de voir le network plugin utilisé pour le cluster ; L'agent weave ou le plugin doit être installé sur tous les nœuds du cluster.



**NB** : les CNI sont gérés par le server Kubelet, donc si jamais le service kubelet n'est pas opérationnel, alors les containers ne pourront être déployés convenablement et pas accessibles.

`ip link show docker0` : permet de voir l'état d'une interface réseau

`arp node01` : permet d'avoir l'adresse Ip et l'adresse mac du node01

`ip route show default` : permet d'avoir le DNS sur une machine

`netstat -nplt` : permet la liste de ports ouverts et leurs processus respectifs

`netstat -laputen | grep etcd` : permet d'avoir le détail sur le nombre de processus et de ports ouverts par etcd

`ps -aux | grep kubelet | grep network` : permet d'avoir la liste complète de processus en cours et de filtrer pour avoir le network plugin utilisé pour le service kubelet

### 8.3. Le CNI (Network plugin) WEAVE

S'il y'a pas de plugin dans un cluster, alors il y'a pas de réseau et alors les nœuds ne seront pas Ready. Donc si un `kubectl get nodes` est fait et qu'on se rend compte que les nodes ne sont pas à l'état Ready, alors ce cluster n'a pas de plugin défini. WEAVE fait partir de la liste de plugin que l'on peut avoir sur kubernetes ; ci-dessous la méthode d'installation de WEAVE

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

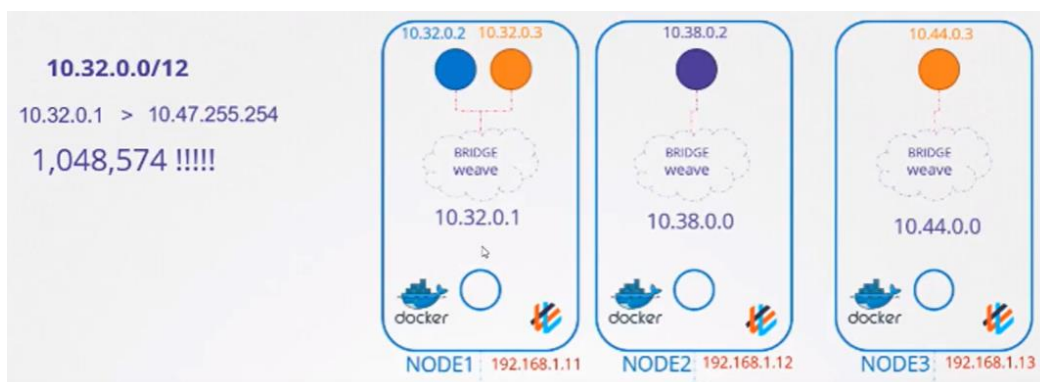
```
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.extensions/weave-net created
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE						
coredns-78fcdf89b4-99lhw	2/2	Running	0	10m	10.44.0.2	master <none>
coredns-78fcdf89b4-p7dpj	2/2	Running	0	10m	10.44.0.11	master <none>
etcd-master	1/1	Running	0	18m	172.17.0.11	master <none>
kube-apiserver-master	1/1	Running	0	18m	172.17.0.11	master <none>
kube-scheduler-master	1/1	Running	0	17m	172.17.0.11	master <none>
weave-net-5gcnb	2/2	Running	1	19m	172.17.0.20	node02 <none>
weave-net-frojd	2/2	Running	1	19m	172.17.0.11	master <none>
weave-net-m6sz	2/2	Running	1	19m	172.17.0.23	node01 <none>
weave-net-tbzyz	2/2	Running	1	10m	172.17.0.52	node03 <none>

```
kubectl logs weave-net-5gcnb weave --n kube-system
```

#### Installation de Weave

- `weave reset`
- `rm /opt/cni/bin/weave-*`
- `ls /opt/cni/bin`
- `kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"`
- `ls /opt/cni/bin`
- `kubectl get po`

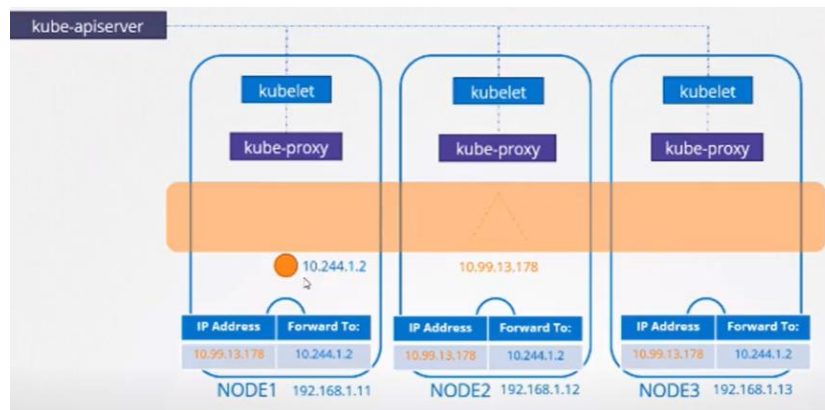


Weave utilise une plage d'adresse qui va du 10.32.0.1 au 10.47.255.254 lui permettant d'adresser les différents composants de kubernetes. Afin de garantir une unicité des adresses par pod dans le réseau, Weave attribut généralement une plage d'adresse unique par nœud, histoire que les pod soient dans le même réseau certes, mais dans des plages d'adresses différentes afin d'éviter d'éventuels conflits d'adresses. Le plugin Weave devra être installé sur chaque nœud et il permettra de jouer les agent de liaison entre les différents nœuds

Le plugin Weave une fois installé sur chaque node à l'aide des daemon sets, crée une carte réseau (eni) appelée « weave » qui sera en effet la passerelle pour les différents pods présents sur ce nœud-là.

## 8.4. Le Service Cluster IP & NODEPORT

Pour que les services de type cluster IP et NodePort fonctionnent convenablement, il est important que Weave et kube-proxy soient également installés et bien fonctionnels.

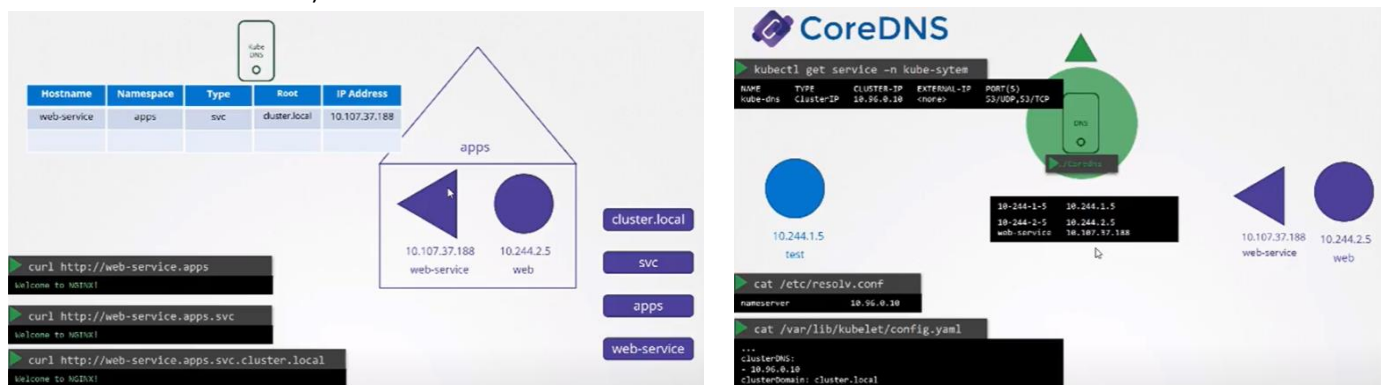


Kube-proxy permettra de rendre l'adresse Ip du service disponible sur tous les nœuds de notre cluster de telle sorte que si on interroge n'importe quel nœud à travers ce service, il soit capable de traduire la demande vers le pod ou service exact. Si le pod se trouve sur le nœud questionné, alors Kube-proxy fera la translation de port directement vers le pod demandé. Par contre si le pod demandé à travers le service se trouve sur un nœud différent de celui questionné, alors Weave ou le plugin CNI utilisé entre en jeu afin de permettre d'atteindre le pod demandé se trouvant sur un autre nœud. Donc en résumé, Kube-proxy permet d'exposer le service crée sur l'ensemble des nœuds existants afin qu'il soit connu de tous et puisse être appelé de n'importe quel nœud de notre cluster. Si le nœud questionné pour avoir accès a notre service ou pod correspond à celui dans lequel se trouve notre pod concerné, alors kube-proxy seule suffira dans ce cas pour atteindre notre pod. Par contre s'il se trouve dans un autre nœud, alors on aura besoin du plugin CNI pour atteindre notre pod se trouvant dans cet autre nœud.

`kubectl logs -n kube-system kube-proxy` : permet de voir les logs du pod kube-proxy afin de déterminer quel type de proxy est utilisé.

## 8.5. CoreDNS

CoreDNS est le composant que kubernetes utilise pour lier les adresses IP des composants à leur noms (noms qui leur sont donnés lors de leur création).



CoreDNS permet également d'avoir un nom DNS pour les POD. C'est le service Kubelet qui configure l'adresse DNS sur tous nos pods, et donne par défaut comme adresse DNS à tous les pods créés, l'adresse du service Kube-dns (10.96.0.10, qui est en fait un service de type clusterIP) que nous pouvons trouver dans le fichier de kubelet /var/lib/kubelet/config.yaml.

`kubectl get pods -n kube-system` : avoir la liste des pods appartenant au namespace kube-system afin de voir le type de DNS utilisé à travers le nom du pod.

`kubectl -n kube-system describe deploy coredns` : permet de décrire le deployment coredns permettant d'avoir accès au repertoire du fichier de configuration de coredns

`kubectl exec -it hr -- nslookup mysql.payroll > /root/CKA/nslookup.out`

```
kubectl get deploy --all-namespaces
kubectl create namespace ingress-space
kubectl get namespaces
kubectl -n ingress-space create configmap nginx-configuration
kubectl -n ingress-space create serviceaccount ingress-serviceaccount
kubectl get roles -n ingress-space
kubectl get rolebindings.rbac.authorization.k8s.io -n ingress-space
kubectl describe roles ingress-role -n ingress-space
```

## IX. Installation Kubernetes with k8s

Run theses commands

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

**NB** : Bien vérifier que l'ensemble des ports nécessaires au bon fonctionnement de kubernetes soient activés (cf. doc k8s)

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
\${basearch}
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF

sudo mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF

# Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config

sudo yum install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes

sudo systemctl enable --now kubelet
swapoff -a
echo 1 > /proc/sys/net/ipv4/ip_forward
kubeadm init --apiserver-advertise-address=<ip-address>

#Installation du cni plugin weave
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl
version | base64 | tr -d '\n')"
```

## X. Json path

DATA	QUERY	RESULT
<pre>[   12,   43,   23,   12,   56,   43,   93,   32,   45,   63,   27,   8,   78 ]</pre>	Get all numbers greater than 40	<pre>[   43,   56,   43,   93,   45,   63,   78 ]</pre>
	<code>\$[ Check if each item in the array &gt; 40 ]</code>	
	Check if => ? ( )	
	<code>\$[?( each item in the list &gt; 40 )]</code>	
	each item in the list => @	
	<code>\$[?( @ &gt; 40 )]</code>	

NB : le symbole \$ représente le rootElement qui est en effet l'élément principal d'un fichier Json, qui dans notre cas ici est une liste sans nom []. Ca aurait également pu être un dictionnaire sans nom {} dans lequel sont définis l'ensemble des éléments de notre fichier Json.

Le symbole ?(critère) précise qu'un test de ce critère sera fait

`$[ ?(@ ==40)]` : test si chaque élément

de la liste est égal à 40 et n'affichera en sortie que la liste des éléments de cette liste égale à 40 ou `$[ ?(@ !=40)]`. On peut aussi avoir comme opérateur de comparaison (`@ in [val1, val1,...,valn]`) ou (`@ nin [val1, val1,...,valn]`)

<pre>{   "color": "blue",   "price": "\$20,000",   "wheels": [     {       "model": "X345ERT",       "location": "front-right"     },     {       "model": "X236DEM",       "location": "rear-right"     },     {       "model": "X346GRX",       "location": "front-left"     },     {       "model": "X987XNV",       "location": "rear-left"     }   ] }</pre>	Get the model of the rear-right wheel	<pre>"X236DEM"</pre>
	<code>\$.car.wheels[2].model</code>	
	<code>\$.car.wheels[?(@.location == "rear-right")].model</code>	

`$.prizes[?(@.laureates[1].firstname == "Malala")].laureates[?(@.firstname=="Malala")]`

<pre>{   "car": {     "color": "blue",     "price": "\$20,000"   },   "bus": {     "color": "white",     "price": "\$120,000"   } }</pre>	Get car's color	<code>\$.car.color</code>	<pre>[ "blue" ]</pre>
	Get bus's color	<code>\$.bus.color</code>	<pre>[ "white" ]</pre>
	Get all colors	<code>\$.*.color</code>	<pre>[ "blue", "white" ]</pre>
	Get all prices	<code>\$.*.price</code>	<pre>[ "\$20,000", "\$120,000" ]</pre>

le terme \* permet de sélectionner tous les éléments d'un dictionnaire

Get all wheels' model	<code>\$.[*].model</code>	<pre>[ "X345ERT", "X346ERT", "X347ERT", "X348ERT" ]</pre>
-----------------------	---------------------------	---

Identique pour les listes [\*] permet de sélectionner l'ensemble des éléments de la liste.

`$.prizes[?(@.year=="2014")].laureates[*].firstname`

DATA	QUERY	RESULT
<pre>[   0 "Apple", -10   1 "Google", -9   2 "Microsoft", -8   3 "Amazon", -7   4 "Facebook", -6   5 "Coca-Cola", -5   6 "Samsung", -4   7 "Disney", -3   8 "Toyota", -2   9 "McDonald's", -1 ]</pre>	Get the last element	<pre>[   "McDonald's" ]</pre>
	<code>\$[9]</code>	
	Get the last element	
	<code>\$[-1]</code>	
	<code>\$[-1:0]</code>	<pre>[   "Disney",   "Toyota",   "McDonald's" ]</pre>
	<code>\$[-1:]</code>	
	Get the last 3 elements	
	<code>\$[-3:]</code>	
<pre>[   "Apple",   "Microsoft",   "Facebook",   "Samsung" ]</pre>		

Does not work in certain implementations

# JSON PATH Examples

```
kubectl get nodes -o=jsonpath='{.items[*].metadata.name}'
```

```
master node01
```

```
{ "\n" }
```

New line

```
kubectl get nodes -o=jsonpath='{.items[*].status.nodeInfo.architecture}'
```

```
amd64 amd64
```

```
{ "\t" }
```

Tab

```
kubectl get nodes -o=jsonpath='{.items[*].status.capacity.cpu}'
```

```
4 4
```

```
kubectl get nodes -o=jsonpath='{.items[*].metadata.name} {.items[*].status.capacity.cpu}'
```

```
master node01 4 4
```

```
kubectl get nodes -o=jsonpath='{.items[*].metadata.name} {"\n"} {.items[*].status.capacity.cpu}'
```

```
master node01
```

```
4 4
```

## Loops - Range

```
kubectl get nodes -o=jsonpath='{.items[*].metadata.name}{ "\n" } {.items[*].status.capacity.cpu}'
```

```
master node01  
4 4
```

```
master 4  
node01 4
```

```
FOR EACH NODE
```

```
PRINT NODE NAME \t PRINT CPU COUNT \n
```

```
END FOR
```

```
{range .items[*]}
```

```
{.metadata.name} {"\t"} {.status.capacity.cpu} {"\n"}
```

```
{end}'
```

```
{range .items[*]} {.metadata.name} {"\t"} {.status.capacity.cpu} {"\n"} {end}'
```

## JSON PATH for Custom Columns

```
kubectl get nodes -o=jsonpath='{.items[*].metadata.name}{ "\n" } {.items[*].status.capacity.cpu}'
```

```
master node01  
4 4
```

NODE	CPU
master	4
node01	4

```
kubectl get nodes -o=custom-columns=<COLUMN NAME>:<JSON PATH>
```

```
kubectl get nodes -o=custom-columns=NODE:.metadata.name ,CPU:.status.capacity.cpu
```

NODE	CPU
master	4
node01	4

```
kubectl get pv --sort-by=.spec.capacity.storage -o=custom-columns=NAME:.metadata.name,CAPACITY:.spec.capacity.storage > /opt/outputs/pv-and-capacity-sorted.txt
```

```
kubectl get nodes -o=custom-columns=NODE:.metadata.name ,CPU:.status.capacity.cpu
```

NODE	CPU
master	4
node01	4

```
kubectl get nodes --sort-by=.metadata.name
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	5m	v1.11.3
node01	Ready	<none>	5m	v1.11.3

```
kubectl get nodes --sort-by=.status.capacity.cpu
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	5m	v1.11.3
node01	Ready	<none>	5m	v1.11.3



## Kube-api Server

1. Authenticate User

2. Validate Request

3. Retrieve data

4. Update ETCD

5. Scheduler

6. Kubelet

```
> kubectl replace -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 replicaset myapp-replicaset
```

TYPE NAME

```
kubectl scale replicaset my-replicaset --replicas=3
```

```
kubectl create deployment --image=nginx nginx --dry-run=client -o yaml > nginx-deployment.yaml
kubectl run nginx --image=nginx --dry-run=client -o yaml
kubectl expose pod redis --port=6379 --name redis-service --dry-run=client -o yaml
kubectl create service clusterip redis --tcp=6379:6379 --dry-run=client -o yaml
kubectl expose pod nginx --type=NodePort --port=80 --name=nginx-service --dry-run=client -o yaml
kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml
kubectl taint node node01 spray=mortein:NoSchedule
kubectl taint node controlplane node-role.kubernetes.io/master:NoSchedule-
kubectl label node node01 color=blue

git clone https://github.com/kodekloudhub/kubernetes-metrics-server.git
kubectl set image deployment/frontend simple-webapp=kodekloud/webapp-color:v2
```

kube-scheduler.yaml from the directory /etc/kubernetes/manifests/

- --leader-elect=false
- --port=10282
- --scheduler-name=my-scheduler
- --secure-port=0

Atteindre un service étant dans un autre namespace. `nom_service.namespace.svc.cluster.local`

The default name of kubernetes cluster is: `cluster.local`

```
> kubectl config set-context $(kubectl config current-context) --namespace=dev
```

Imperative

```
> kubectl run --image=nginx nginx
```

```
> kubectl create deployment --image=nginx nginx
```

```
> kubectl expose deployment nginx --port 80
```

```
> kubectl edit deployment nginx
```

```
> kubectl scale deployment nginx --replicas=5
```

```
> kubectl set image deployment nginx nginx=nginx:1.18
```

```
> kubectl create -f nginx.yaml
```

```
> kubectl replace -f nginx.yaml
```

```
> kubectl delete -f nginx.yaml
```

Create

Get

Update

Status

Rollback

```
> kubectl create -f deployment-definition.yml
```

```
> kubectl get deployments
```

```
> kubectl apply -f deployment-definition.yml
```

```
> kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1
```

```
> kubectl rollout status deployment/myapp-deployment
```

```
> kubectl rollout history deployment/myapp-deployment
```

```
> kubectl rollout undo deployment/myapp-deploy
```

```
kubectl get all --all-namespaces -o yaml > all-deploy-services.yaml
```

NB

```
etcdctl snapshot save /opt/snapshot-pre-boot.db --endpoints=127.0.0.1:2379 --
cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --
key=/etc/kubernetes/pki/etcd/server.key
```

```
etcdctl snapshot restore /opt/snapshot-pre-boot.db --endpoints=127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key --data-dir=/var/lib/etcd-from-backup
```

**NB :** Toujours modifier le fichier manifests etcd pour que le hostpath du volume pointe vers le nouveau répertoire crée lors de la restauration /var/lib/etcd-from-backup

ca.key

ca.crt

ADMIN USER

Generate Keys

admin.key

```
openssl genrsa -out admin.key 2048
```

Certificate Signing Request

admin.csr

```
openssl req -new -key admin.key -subj \
"/CN=kube-admin" -out admin.csr
```

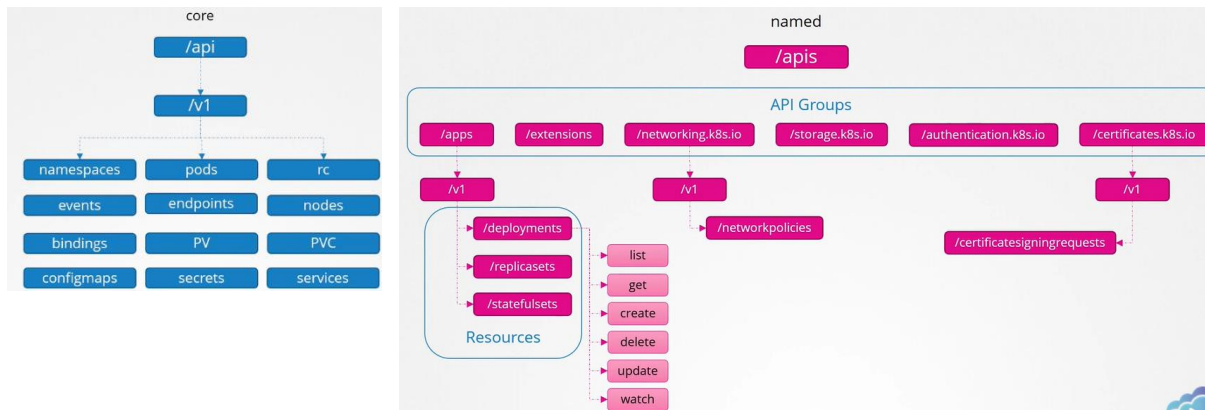
Sign Certificates

admin.crt

```
openssl x509 -req -in admin.csr -CA ca.crt -CAkey ca.key -out admin.crt
```

**openssl genrsa -out <nom.key> 2048**  
**openssl req**

- -new : nouvelle requette
- - key <nom-clé>
- - subj "CN=<nom-user>"
- - out <fichier de sortie.csr>



```
curl http://localhost:6443 -k
```

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/healthz",
    "/logs",
    "/metrics",
    "/openapi/v2",
    "/swagger-2.0.0.json",

```

```
curl http://localhost:6443/apis -k | grep "name"
```

```
"name": "extensions",
"name": "apps",
"name": "events.k8s.io",
"name": "authentication.k8s.io",
"name": "authorization.k8s.io",
"name": "autoscaling",
"name": "batch",
"name": "certificates.k8s.io",
"name": "networking.k8s.io",
"name": "policy",
"name": "rbac.authorization.k8s.io",
```

```
kubectyl proxy
```

```
Starting to serve on 127.0.0.1:8001
```

```
curl http://localhost:8001 -k
```

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/healthz",
    "/logs",
    "/metrics",
    "/openapi/v2",
    "/swagger-2.0.0.json",

```

```
developer-role.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list", "get", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["ConfigMap"]
  verbs: ["create"]
```

les RBAC ont 04 sections

- apiVersion
- kind
- metadata
- rules: qui doit avoir 03 sections sous forme de liste
  - apiGroups : [ " " ]
  - resources : [ ]
  - verbs : [ ]

```
developer-role.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "create", "update"]
  resourceName: ["blue", "orange"]
```

### devuser-developer-binding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devuser-developer-binding
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl auth can-i create deployments
yes

kubectl auth can-i delete nodes
no

kubectl auth can-i create deployments --as dev-user
no

kubectl auth can-i create pods --as dev-user
yes

kubectl auth can-i create pods --as dev-user --namespace test
no
```

kubectl create role <name-role> --verb=get,create,list -resource=pods, deployments, replicaset

kubectl create rolebinding <name-rb> --clusterrole=<role-name> --user=<user-name> --group=<group-name>

NB: Kubectl api-resources: to view the list of api resources that can be use to create role or cluster role

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

## Private Repository

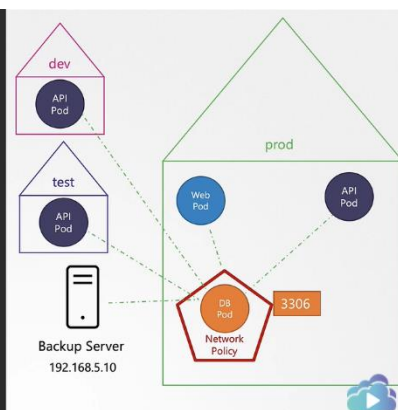
```
docker login private-registry.io

docker run private-registry.io/apps/internal-app

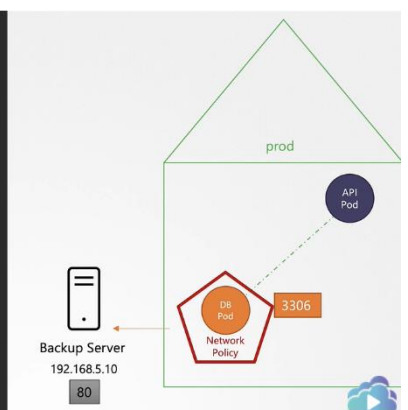
kubectl create secret docker-registry regcred \
  --docker-server=private-registry.io \
  --docker-username=registry-user \
  --docker-password=registry-password \
  --docker-email=registry-user@org.com
```

```
nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: private-registry.io/apps/internal-app
    imagePullSecrets:
    - name: regcred
```

```
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: api-pod
        namespaceSelector:
          matchLabels:
            name: prod
    - ipBlock:
        cidr: 192.168.5.10/32
    ports:
    - protocol: TCP
      port: 3306
```



```
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: api-pod
    ports:
    - protocol: TCP
      port: 3306
  egress:
  - to:
    - ipBlock:
        cidr: 192.168.5.10/32
    ports:
    - protocol: TCP
      port: 80
```



### pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-vol1
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 500Mi
  gcePersistentDisk:
    pdName: pd-disk
    fsType: ext4
```

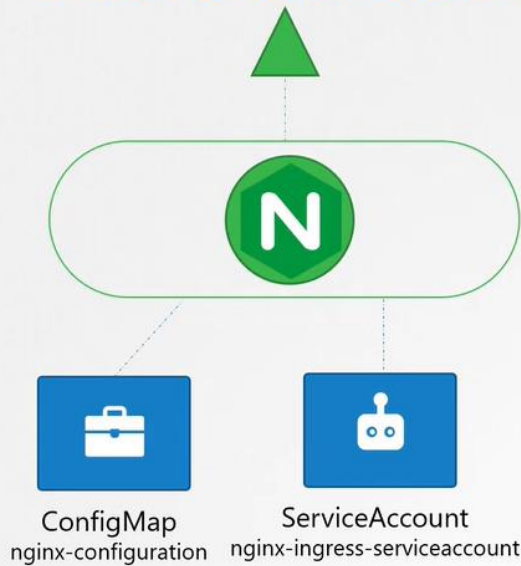
### pvc-definition.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

### pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh", "-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
    volumeMounts:
    - mountPath: /opt
      name: data-volume
  volumes:
  - name: data-volume
    persistentVolumeClaim:
      claimName: myclaim
```

# INGRESS CONTROLLER



## Deployment

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      name: nginx-ingress
  template:
    metadata:
      labels:
        name: nginx-ingress
    spec:
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          ports:
            - name: http
              containerPort: 80
            - name: https
              containerPort: 443
```

## Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: http
    - port: 443
      targetPort: 443
      protocol: TCP
      name: https
  selector:
    name: nginx-ingress
```

## ConfigMap

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
```

## Auth

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
```

Roles

ClusterRoles

RoleBindings

<https://trainingportal.linuxfoundation.org/learn/dashboard>