
# Chapitre 7

## Orchestration des conteneurs avec Kubernetes

kubernetes

Enseignante: Dr-Ing. Amina JARRAYA
Email : amina.jarraya@ensi-uma.tn
Niveau: II3- GL

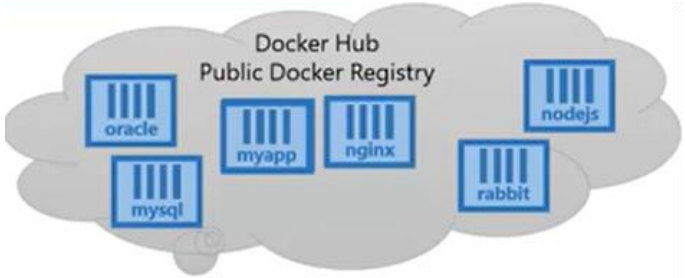
# Plan

1. Présentation du Kubernetes
2. Architecture de Kubernetes
3. Composants de Kubernetes
4. Fichier de déploiement YAML de Kubernetes
5. Mise en place de Kubernetes sous WSL
6. Création d’un service dans un cluster

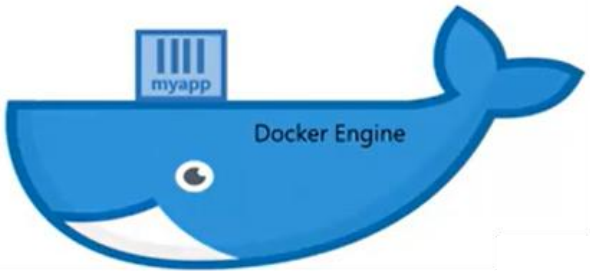
# 1. Présentation du Kubernetes

# Pourquoi l'orchestration des conteneurs ?

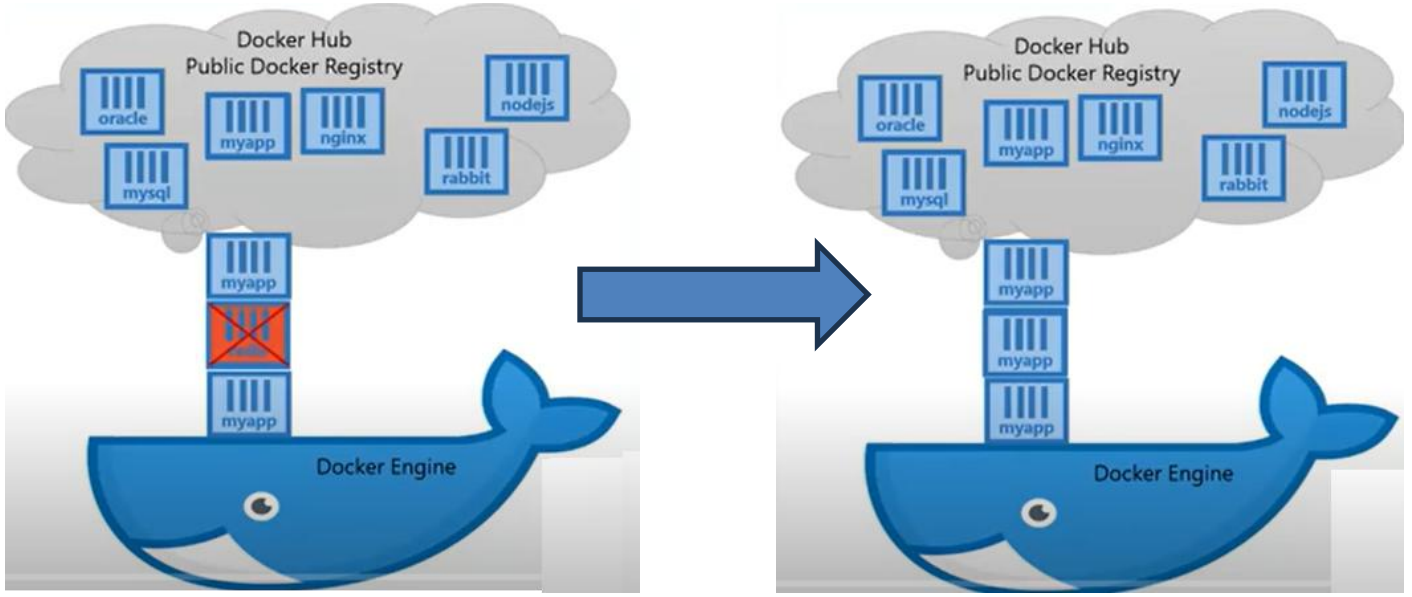
```
$ docker run myapp
```




- Si l'image n'existe pas en local, il va chercher dans Dockerhub ou Docker registry (Dockerhub, est dépôt publique alors que Docker registry est dépôt privé)

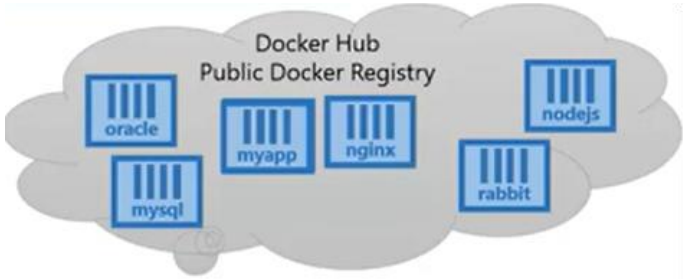



- L'administrateur doit veiller sur le bon déroulement de l'exécution des instances.

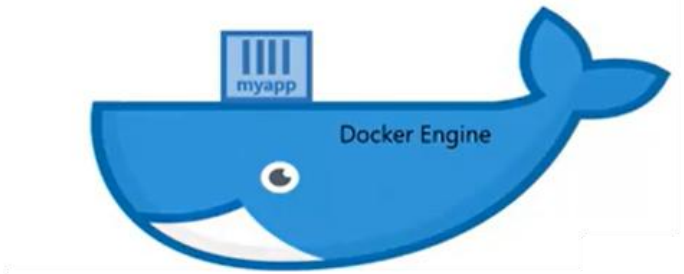


# Pourquoi l’orchestration des conteneurs ?

- C’est vrai que Docker facilite le packaging et le déploiement des applications Mais ne pas utiliser un outil d’orchestration entraine la gestion manuelle des conteneurs (exp conteneur tombe en panne), il faut surveiller l’état des instances.

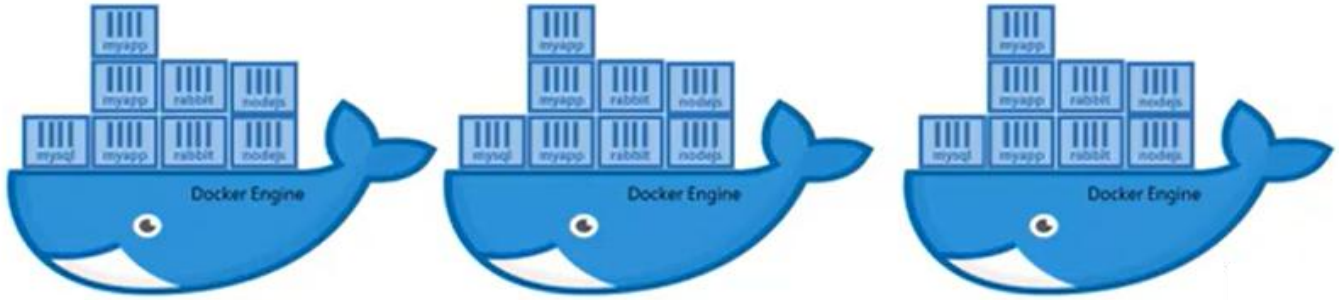



- Problème de disponibilité et de montée en charge :
  - Il faut démarrer plusieurs instances en fonction de la charge.
  - Si une instance tombe en panne, il faut la remplacer.
  - Si Docker Engine tombe en panne, tous les conteneurs vont cracher




- L’orchestration des conteneurs est une solution pour ce problème.

# Orchestration des conteneurs




- Consiste en un ensemble d'outils et de scripts qui permettent l'automatisation des opérations (scalabilité des conteneurs, Réplication des conteneurs, Haute disponibilité, état des conteneurs, etc).

- Exemple d'outils : Docker Swarm, Kubernetes, MESOS

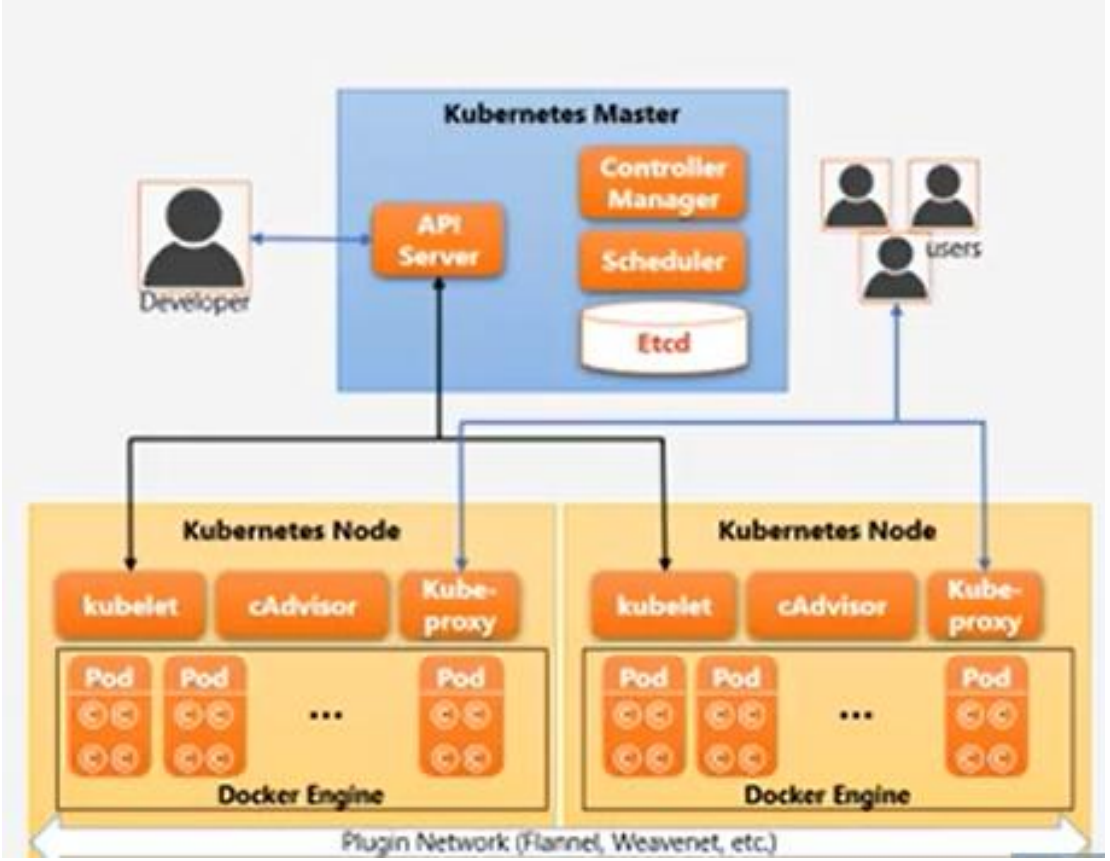


Apache MESOS

# Kubernetes

- Outils d'orchestration des conteneurs
- Développé par Google, écrit avec le langage GO
- Peut tourner en Local, Cloud (AWS, Azure, GCE, …) machines physiques

- Kubernetes privilégie la communication REST API pour exposer ses fonctionnalités
- L'objectif est de gérer directement des applications et non pas de machine

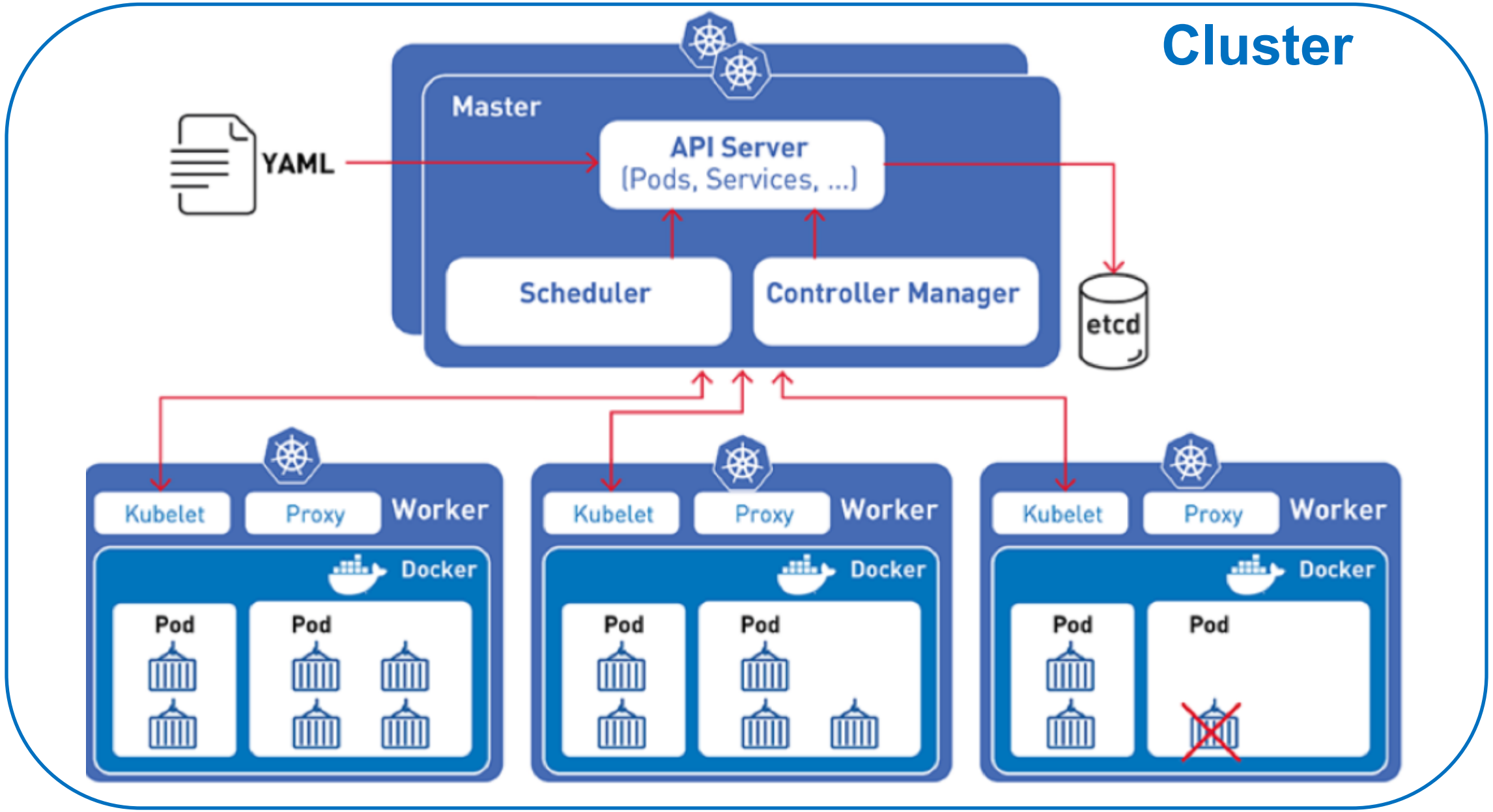


# Qu’est ce qu’on peut faire avec Kubernetes ?

- Déployer une application sous forme de conteneurs d’une manière très rapide et prévisible
- Automatiser le déploiement et la réplication des conteneurs
- Organiser les conteneurs en groupes et faire du Load balancing
- Déclarer l’architecture cible (Nombre de pods, contenu, stratégie de mise à jour, etc) et laisser le système atteindre et à maintenir cette cible.
- Faire des MAJ sans interrompre le service
- Séparer l’application de l’architecture sous-jacente
- Détecter les problèmes et les résoudre tout seul

# 2. Architecture de Kubernetes

# Architecture de kubernetes



# Architecture de kubernetes

- Kubernetes a une architecture master/slave composés d'un **control plane (master)** et de nœuds **workers (slaves)**.
- Les **pods** Kubernetes servent à grouper des conteneurs fortement couplés en unités d'application
- Les **deployments** sont une abstraction pour gérer les pods (créer, mettre à jour ou supprimer des pods).
- Enfin, les **services** exposent un ensemble de pods à l'intérieur ou à l'extérieur du cluster grâce à une adresse IP stable.

# Kubernetes master (control plane)

- Le **Kubernetes master** est responsable du maintien de l'état souhaité de votre cluster. Lorsque vous interagissez avec Kubernetes, par exemple en utilisant l'interface en ligne de commande **kubectl**, vous communiquez avec le master Kubernetes de votre cluster.

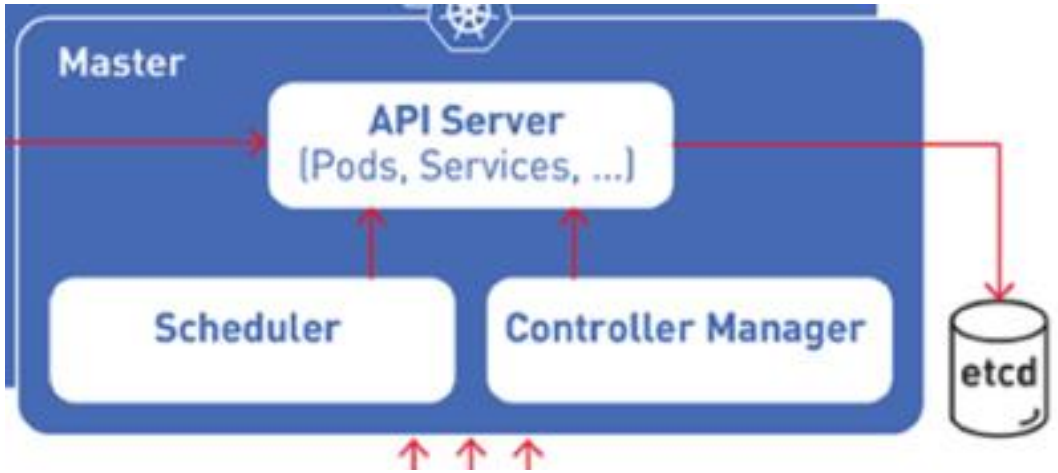
- Le "master" fait référence à un ensemble de processus gérant l'état du cluster. Le master peut également être répliqué pour la disponibilité et la redondance.

- Responsable de la gestion globale du cluster Kubernetes et à la planification des pods sur les nœuds esclaves.  Il surveille l'état du cluster et intervient en cas de défaillance (par exemple, en redémarrant des pods sur d'autres nœuds).

- Le **plan de contrôle** de Kubernetes consiste en plusieurs composants, chacun ayant son propre processus, qui peuvent s'exécuter sur un seul Node maitre ou sur plusieurs maitres permettant de créer des clusters haute disponibilité.

# Kubernetes master (control plane)

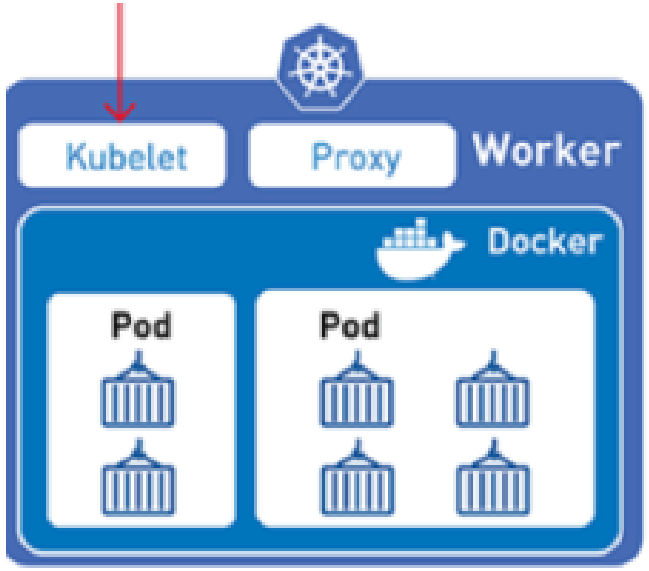
- Les différents composants du plan de contrôle de Kubernetes sont décrits ci-dessous :
- **Etcd** : Unité de stockage distribué persistante et légère
- **API serve**r : API REST de communication avec les composants internes et externes
- **Scheduler** : L'ordonnanceur qui permet de sélectionner quel Node devrait tourner un Pod
- **Controller manager** : processus dans lequel s'exécutent les controleurs principaux de Kubernetes tels que DaemonSet Controller et la Replication Controller. Il gère les réplicas et l'état des nœuds.



# Kubernetes esclave (les nœuds workers/nodes)

- Un node est **une machine physique ou virtuelle** sur laquelle s'exécutent les **Pods** (les unités contenant les conteneurs de vos applications).
- Chaque nœud possède un **kubelet**, qui interagit avec le maître pour gérer les pods et l'exécution des conteneurs.
- Chaque nœud peut héberger un ou plusieurs **pods**, en fonction de la configuration et des ressources disponibles.

- Voici les **éléments principaux** d'un Node :
  - Kubelet
  - Kube-Proxy
  - Container Runtime
  - Pods

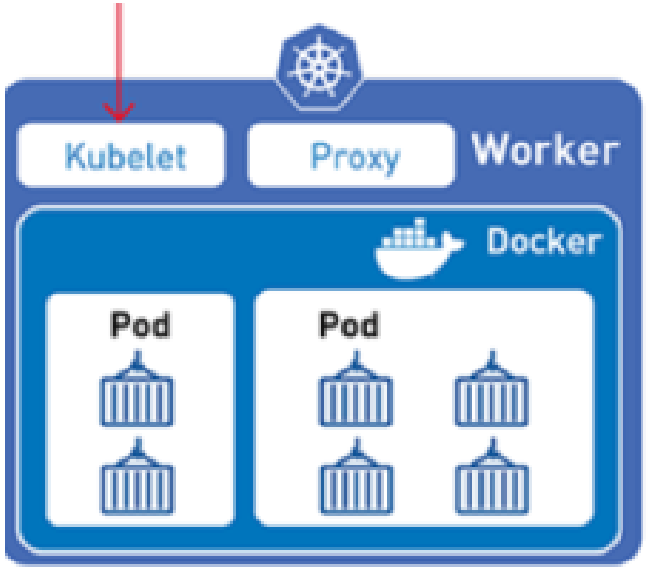


# Kubernetes esclave (les nœuds workers)

**Kubelet** : C'est l'agent principal qui fait le lien entre le Master (Control Plane) et le Node.

- Il **écoute** les instructions envoyées par l'API Server.
- Il **lance, surveille et gère** les conteneurs (Pods) sur le Node.
- Il **s'assure** que les Pods tournent comme décrit dans la configuration (Deployment, ReplicaSet, etc.).

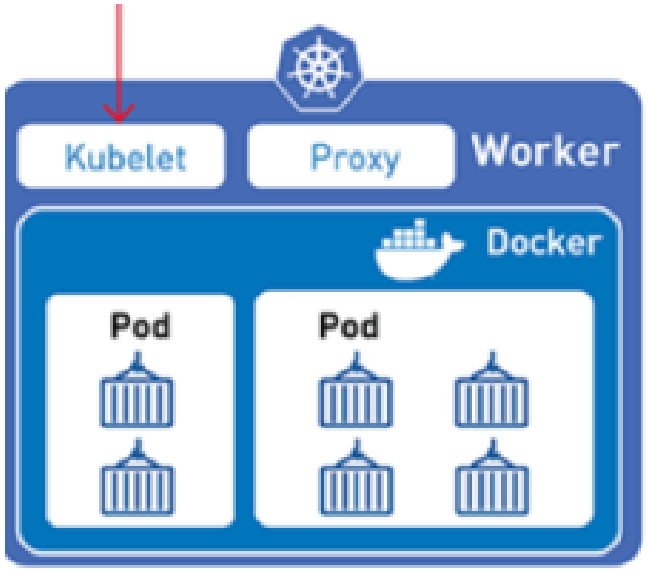
**Exemple** :
Le scheduler décide de placer un Pod sur un Node → le **kubelet** de ce Node reçoit l'ordre → il crée les conteneurs correspondants.



# Kubernetes esclave (les nœuds workers)

**Kube-Proxy** : Gère le **réseau** à l'intérieur du cluster.

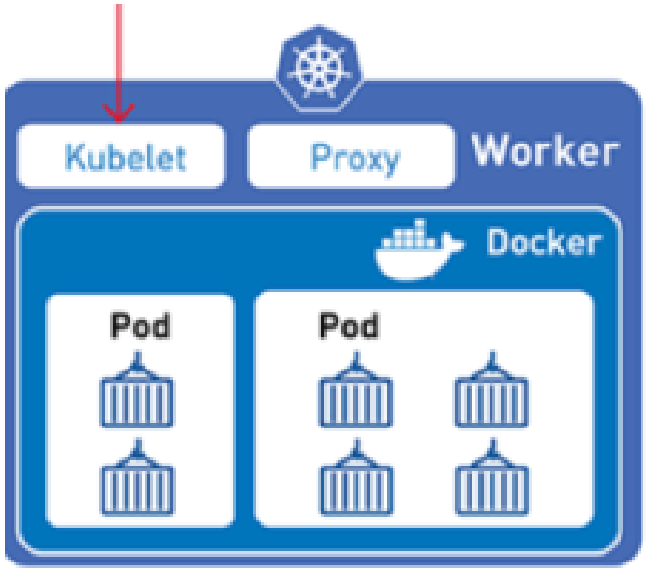
- Il met en place les **règles de routage** et **load balancing** entre les Pods et les Services.
- Il assure la **communication interne** et externe aux Pods.
- Il s'appuie sur **iptables** ou **ipvs** pour le trafic réseau.



# Kubernetes esclave (les nœuds workers)

**Container-runtime** : C’est le moteur qui **exécute réellement les conteneurs** dans les Pods.

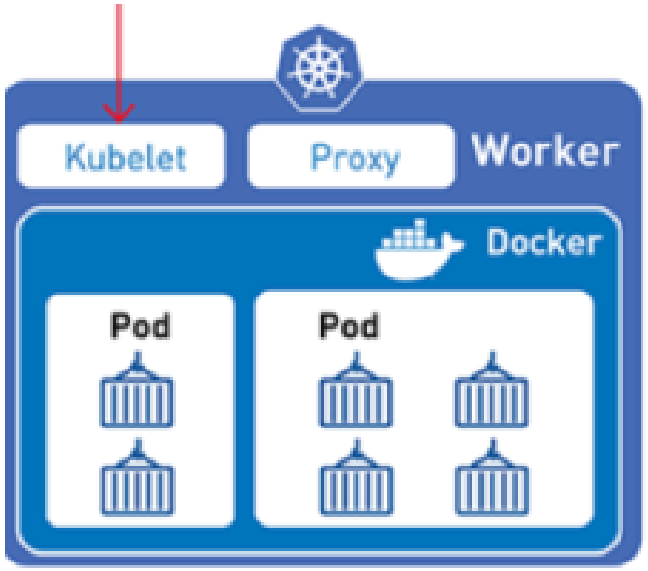
- Kubernetes peut fonctionner avec différents runtimes : dans notre cas, c’est Docker.
- Le runtime gère le **téléchargement des images**, leur **démarrage**, **exécution** et **arrêt**.



# Kubernetes esclave (les nœuds workers)

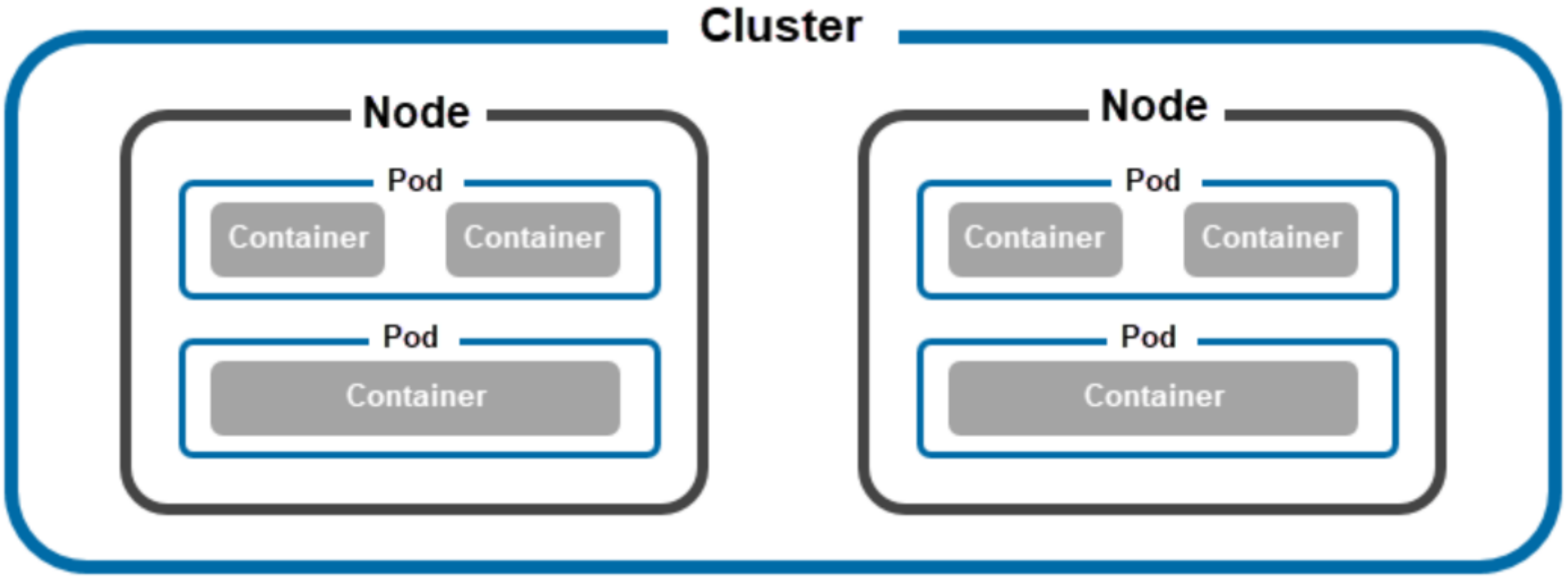
**Pods** : Ce sont les **unités d'exécution** hébergées par le Node.

- Chaque **Pod** contient un ou plusieurs **conteneurs** (Docker, containerd…).
- Le Pod partage un **même espace réseau** et des **volumes** entre ses conteneurs.



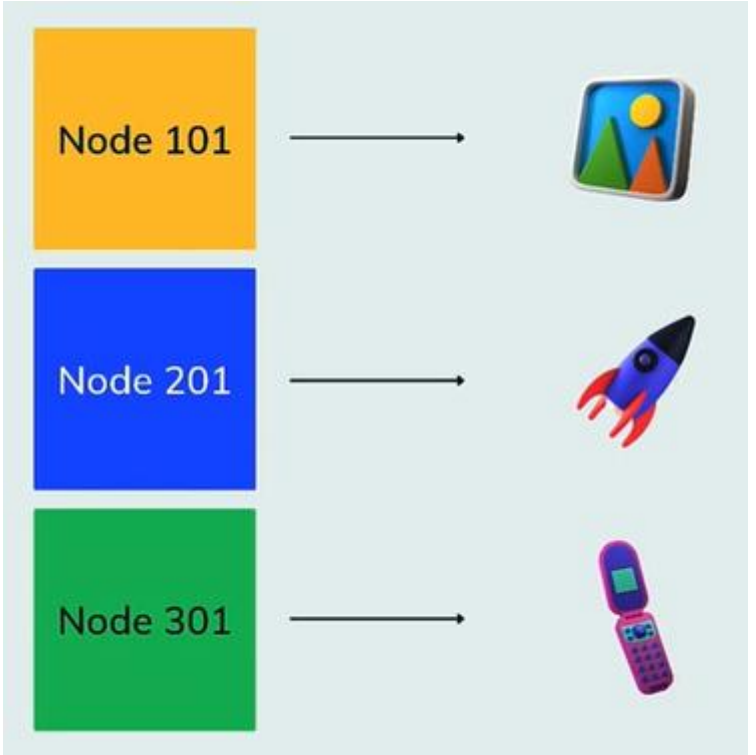
# 3. Composants de Kubernetes

# Les composants de Kubernetes



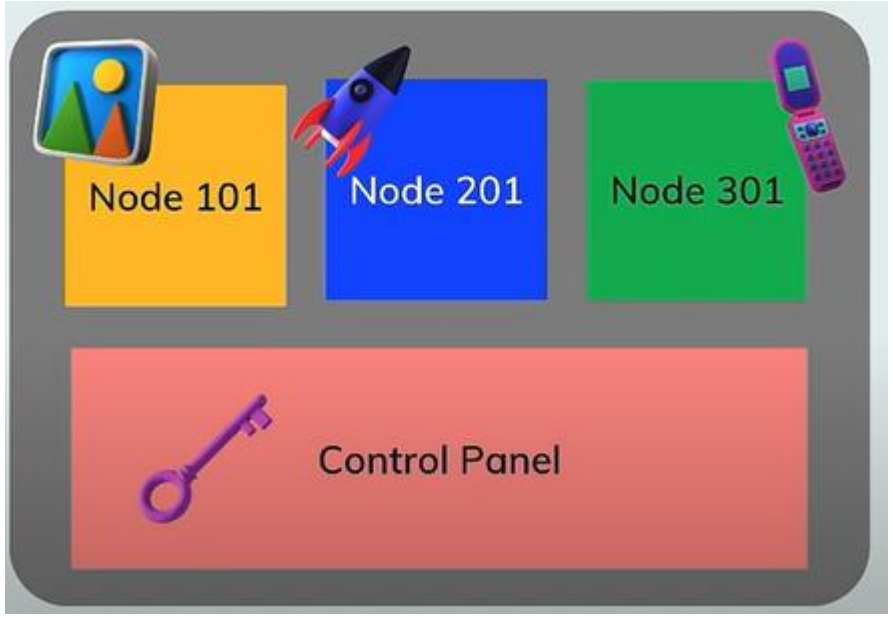
# C’est quoi un noeud

- Un nœud est une machine (physique ou virtuelle) dans un cluster Kubernetes.
- C'est l'endroit où les conteneurs (pods) sont exécutés.
- Les nœuds sont gérés par le plan de contrôle (master) du cluster.
- Ils peuvent être considérés comme les "travailleurs" du cluster.



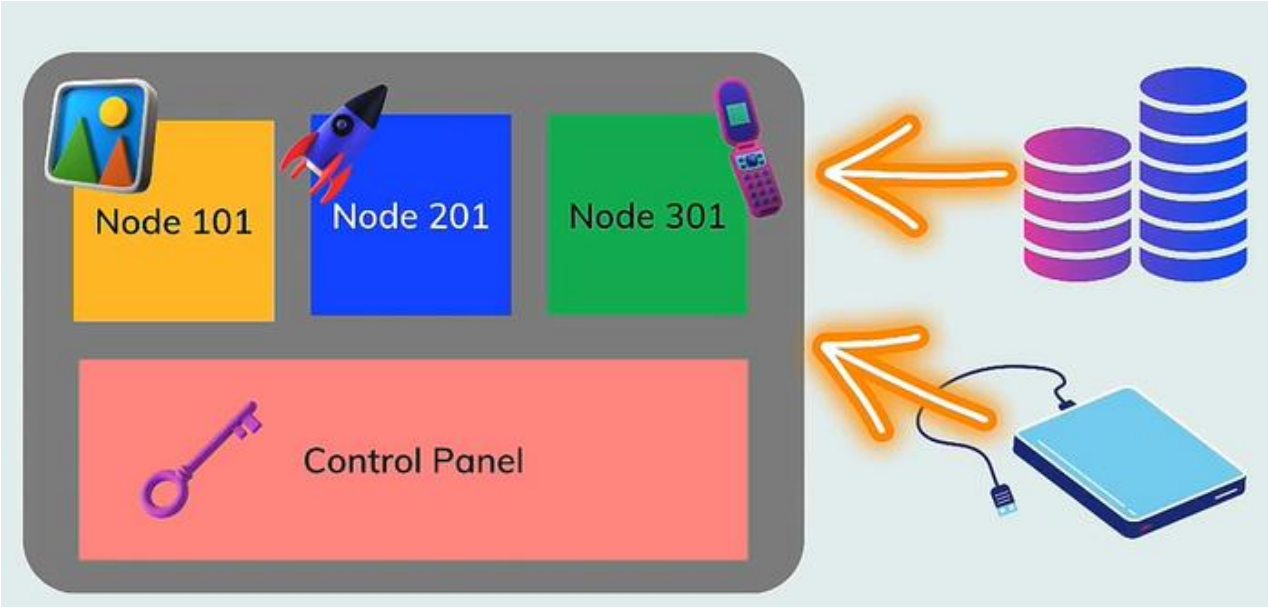
# C’est quoi un cluster ?

- Un cluster Kubernetes est un ensemble de machines, physiques ou virtuelles, qui fonctionnent ensemble pour exécuter des applications conteneurisées. Il est composé d'un plan de contrôle (qui gère le cluster) et de nœuds de travail (qui exécutent les applications).



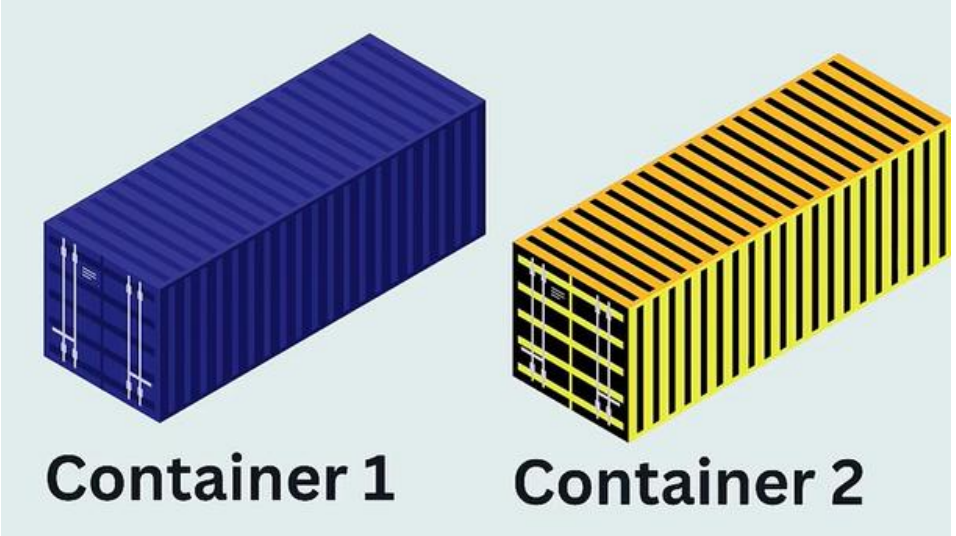
# C’est quoi un volume (etcd) ?

- Les conteneurs dans un pod peuvent partager un volume
- Un volume : espace de stockage qui peut être attaché à un conteneur ou à un groupe de conteneurs dans un même pod.
- Le volume peut être persisté indépendamment du conteneur
- Par exemple, il permet aux conteneurs de partager un même répertoire dans un même Pod.



# C’est quoi un conteneur ?

- Un conteneur est une unité d'exécution légère et portable qui encapsule une application et ses dépendances, permettant ainsi une exécution cohérente sur différents environnements.
- Les conteneurs sont créés à partir d'images de conteneurs, qui sont des modèles immuables contenant tout le nécessaire pour exécuter une application.
- Les conteneurs sont isolés les uns des autres, ce qui signifie qu'ils n'interfèrent pas avec le fonctionnement des autres conteneurs s'exécutant sur la même machine.

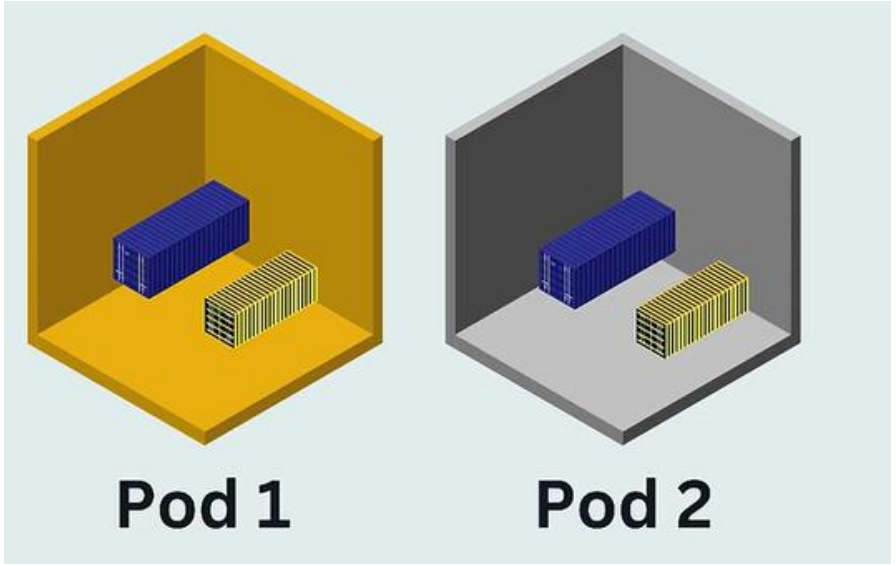


# C’est quoi un POD ?

- Groupe de conteneurs déployés ensemble

- Les conteneurs du Pod :
- Sont toujours démarrés, arrêtés et répliqués en groupe
- Partageant le même réseau, namesapce, IP et Ports
- Peuvent communiquer en utilisant localhost
- Peuvent partager des données via des shared volumes

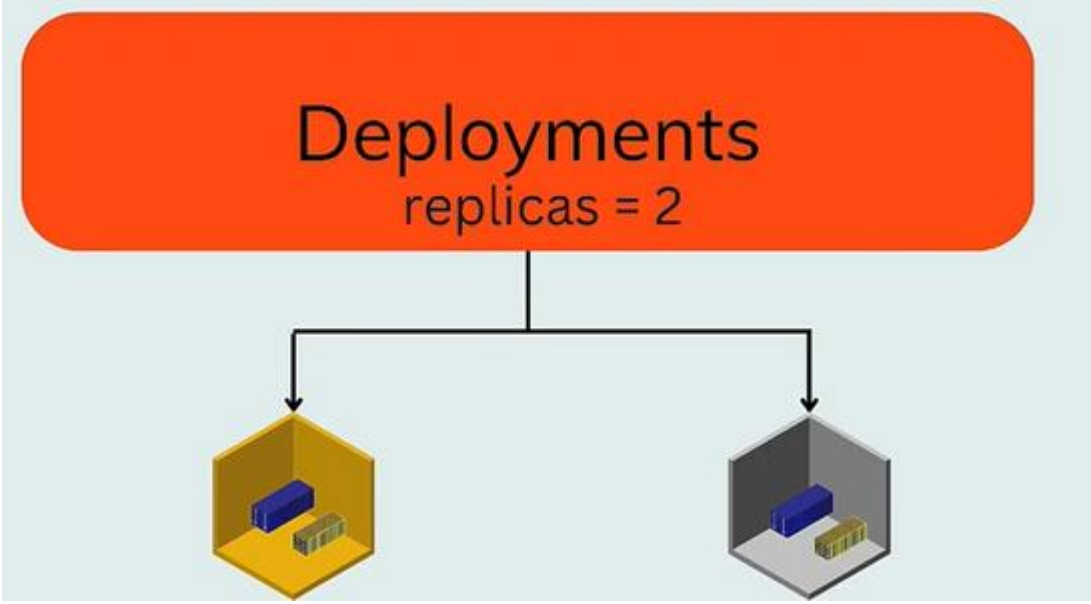
- Le pod est l’entité de base qu’on va répliquer et qu’on va mettre sur différents nœuds du cluster.



# C’est quoi un déploiement

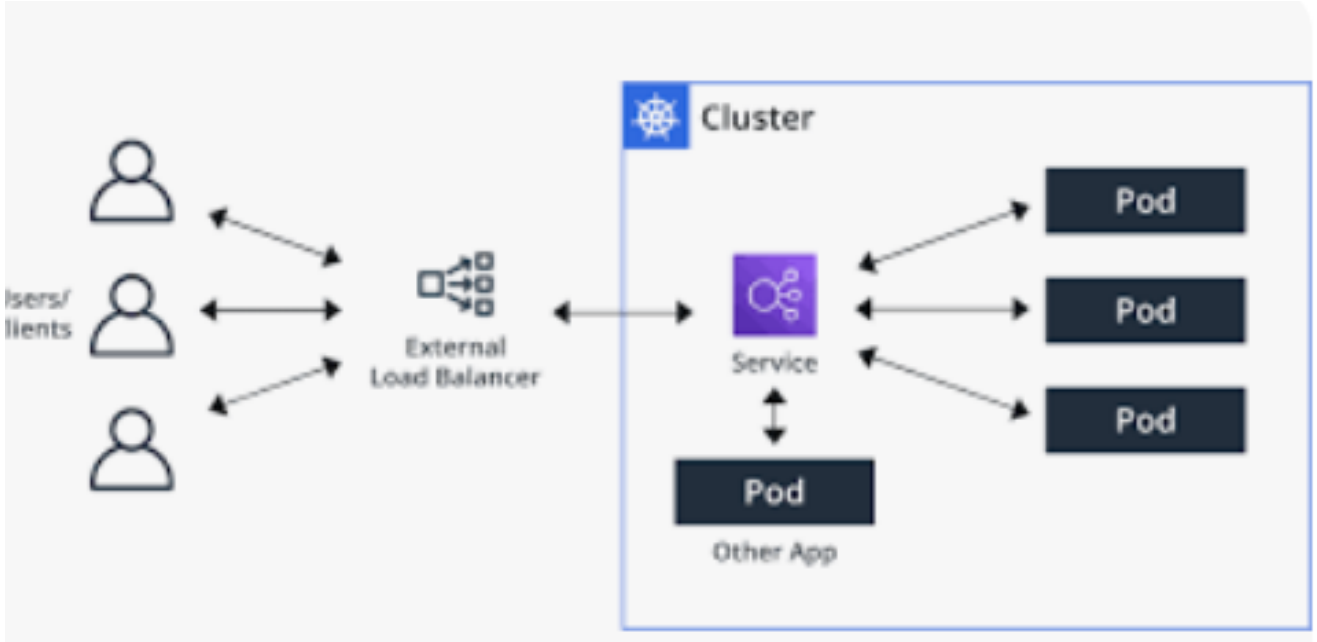
- Un déploiement Kubernetes gère la création et la mise à jour des pods. Il assure la disponibilité de l'application en garantissant qu'un certain nombre de réplicas (instances) de l'application soient toujours en cours d'exécution.

- Les déploiements permettent également de gérer les mises à jour progressives et les retours en arrière en cas de problème.



# C’est quoi un service ?

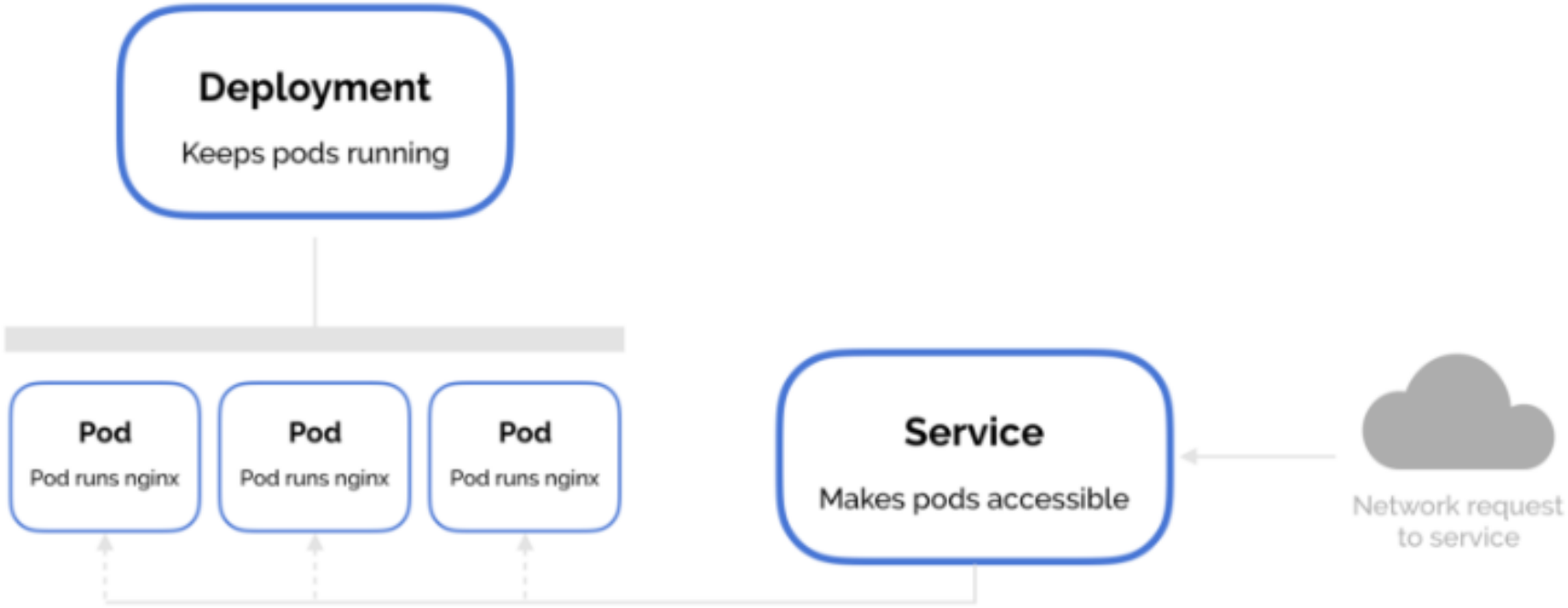
- Un service Kubernetes est une abstraction qui définit un ensemble de pods et leur fournit un point d'accès réseau stable.
- Les services permettent aux applications de communiquer entre elles ou avec l'extérieur du cluster, sans se soucier de l'emplacement des pods.
- Ils utilisent des sélecteurs d'étiquettes pour identifier les pods auxquels ils se réfèrent et peuvent être exposés via des adresses IP virtuelles ou des noms de domaine.



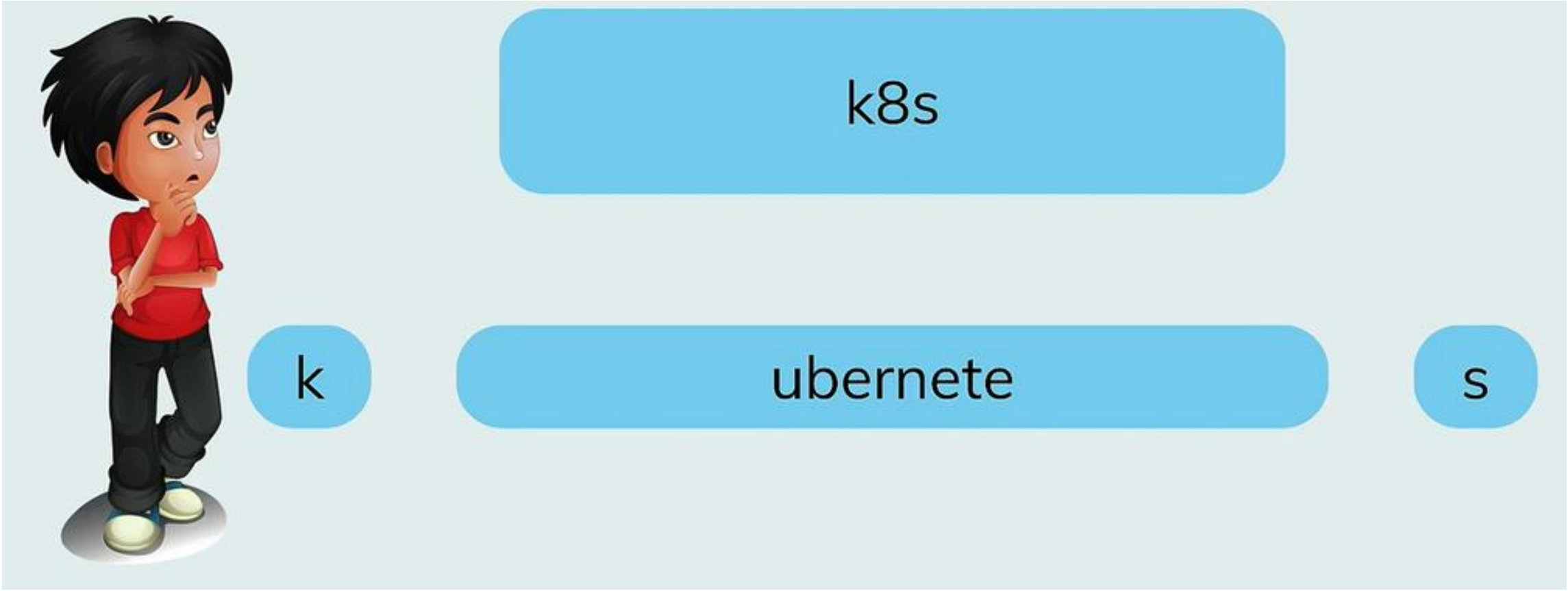
# Deployment vs service

- Un déploiement permet de maintenir un ensemble de pods en activité en créant des pods à partir d'un modèle.

- Un service permet d'autoriser l'accès réseau à un ensemble de pods.



# C’est quoi K8S ?



# 4. Fichier de déploiement YAML de Kubernetes

# Le fichier de configuration YAML

- Les fichiers de configuration semblent donc être la meilleure solution afin de gérer un cluster Kubernetes.
- Ces fichiers au format YAML doivent respecter un schéma bien particulier afin d'être compris par Kubernetes.
- Il est important de rappeler que YAML est une version dérivée de JSON. Cela veut dire que vous pouvez aussi très bien utiliser des fichiers au format JSON pour piloter le cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
  labels:
    app: web
spec:
  containers:
    - name: front-end
      image: nginx
      ports:
        - containerPort: 80
    - name: my-site
      image: myRegistry/mySite:v1
      ports:
        - containerPort: 88
```

# Exemple de création d’un pod

- Ce fichier décrit un pod nommé « mon-pod » qui exécute un conteneur unique basé sur l'image nginx:latest et qui expose le port 80.

- **apiVersion**: v1: Spécifie la version de l'API Kubernetes utilisée pour créer cet objet (ici, la version de base pour les objets de type Pod).
- **kind**: **Pod**: Indique qu'il s'agit d'un objet de type Pod.
- **metadata.name**: **mon-pod**: Définit le nom du Pod, qui doit être unique dans son namespace.
- **spec.containers**: Une liste de conteneurs à exécuter dans le Pod. Dans cet exemple, il y a un seul conteneur.
- **spec.containers.name**: Le nom du conteneur.
- **spec.containers.image: nginx:latest**: L'image Docker du conteneur, ici, la dernière version de l'image Nginx.
- **spec.containers.ports**: Une liste de ports exposés par le conteneur.
- **spec.containers.ports.containerPort**: 80: Le port 80 du conteneur est rendu accessible.

Code

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
spec:
  containers:
  - name: ma-premiere-container
    image: nginx:latest
    ports:
    - containerPort: 80
```

# 5. Mise en place de Kubernetes

# Installer Kubernetes

- Deux méthodes d'installation mais ces deux méthodes installent un master et un worker.

Minikube

https://minikube.sigs.k8s.io/docs/start/


Docker Desktop

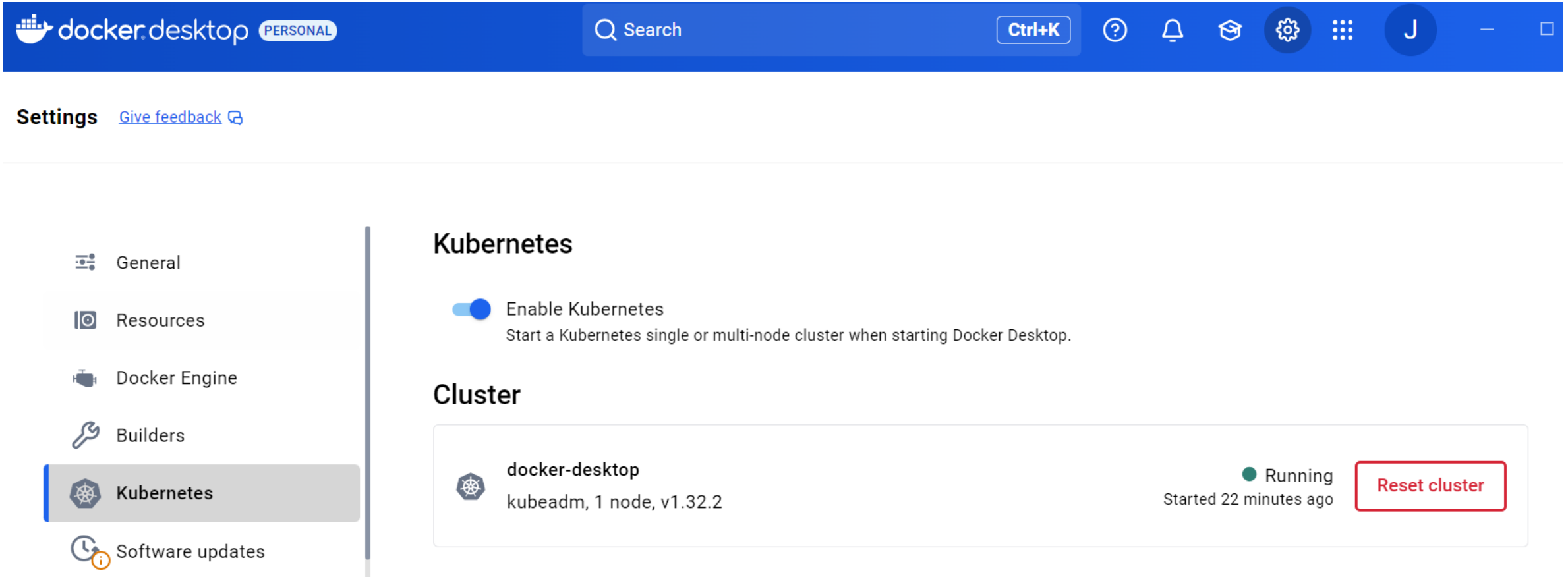
https://docs.docker.com/desktop/

# Installer Kubernetes avec Docker Desktop

Voici les étapes pour activer Kubernetes dans Docker Desktop :

- **Ouvrez Docker Desktop:** Lancez l'application Docker Desktop sur votre machine.
- **Accédez aux paramètres:** Cliquez sur l'icône Docker en haut de la barre des tâches, puis sélectionnez "Settings" (ou "Préférences" sur macOS).
- **Activez Kubernetes:** Dans les paramètres, allez dans l'onglet "Kubernetes" et cochez la case "Enable Kubernetes".
- **Appliquez les modifications:** Cliquez sur "Apply & Restart" pour démarrer le cluster Kubernetes.

# Résultat d’installation sous Docker Desktop



# Résultat d’installation sous Docker Desktop

- Tapez la commande « kubectl » pour vérifier la mise en place de Kubernetes

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
  create          Create a resource from a file or from stdin
  expose          Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service
  run             Run a particular image on the cluster
  set             Set specific features on objects

Basic Commands (Intermediate):
  explain         Get documentation for a resource
  get             Display one or many resources
  edit            Edit a resource on the server
  delete          Delete resources by file names, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout         Manage the rollout of a resource
  scale           Set a new size for a deployment, replica set, or replication controller
  autoscale       Auto-scale a deployment, replica set, stateful set, or replication controller
```

# 6. Création d’un déploiement et d’un service dans un cluster

# Comment créer le déploiement/le service sous Kubernetes ?

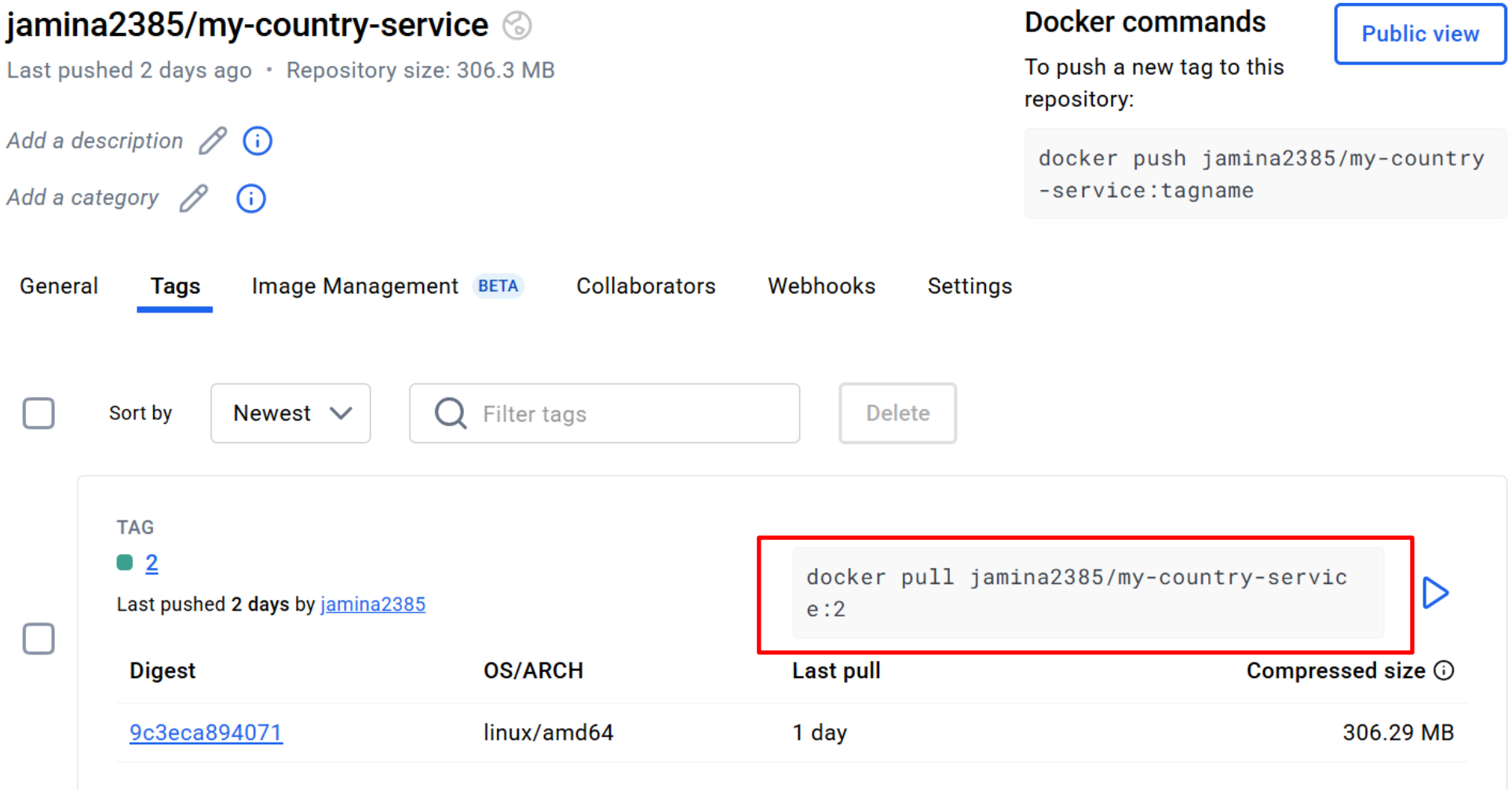
- **1ère méthode : kubectl expose**

•C'est une commande interactive qui permet de créer un service à partir d'une ressource existante dans Kubernetes.
•Elle peut être utilisée pour exposer des pods, des déploiements, des réplicasets, etc.
•Elle génère automatiquement une configuration de service basée sur la ressource spécifiée, notamment le type de service, les sélecteurs, les ports, etc.
•Elle est utile pour des configurations simples et rapides.

- **2ème méthode : les fichiers yaml (service.yaml et deployment.yaml)**

•C'est un fichier qui contient la définition complète d'un service Kubernetes au format YAML.
•Il permet de définir tous les aspects du service : son nom, son type (ClusterIP, NodePort, LoadBalancer, etc.), les sélecteurs pour identifier les pods à exposer, les ports à exposer, et d'autres options avancées.
•Il permet une grande flexibilité et un contrôle précis sur la configuration du service.

# Vérification de l’image sous Dockerhub



# 6. 1 . 1ère méthode : kubectl expose

# Créer le déploiement dans un cluster

- Un déploiement est un objet Kubernetes qui gère la création, la mise à jour et la suppression des pods, qui sont les unités de base exécutant votre application.

**A partir d'une image existante sur Dockerhub**

```
kubectl create deployment my-nginx-app --image=nginx --replicas=3
```

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl create deployment my-country-service --image=jamina2385/my-country-service:2
deployment.apps/my-country-service created
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl get all
NAME                                      READY   STATUS    RESTARTS   AGE
pod/my-country-service-7578f6bcc-s5j6r    1/1     Running   0          7s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP   21h

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-country-service   1/1     1            1           7s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/my-country-service-7578f6bcc    1         1         1       7s
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
my-country-service   1/1     1            1           31s
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
my-country-service-7578f6bcc-s5j6r    1/1     Running   0          35s
```

# Créer un **Service** pour exposer le déploiement

- Créer un **Service** pour exposer votre déploiement, Exposer le port de votre application pour un accès au pod de l'extérieur. --port est le port du service

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl expose deployment my-country-service --type=NodePort --port=8082
service/my-country-service exposed
```

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl describe service my-country-service
Name:                     my-country-service
Namespace:                default
Labels:                   app=my-country-service
Annotations:              <none>
Selector:                 app=my-country-service
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.106.241.213
IPs:                      10.106.241.213
Port:                     <unset>  8082/TCP
TargetPort:               8082/TCP
NodePort:                 <unset>  30343/TCP
Endpoints:                10.1.0.8:8082
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

**Port : port du service**

**TargetPort : port du container**

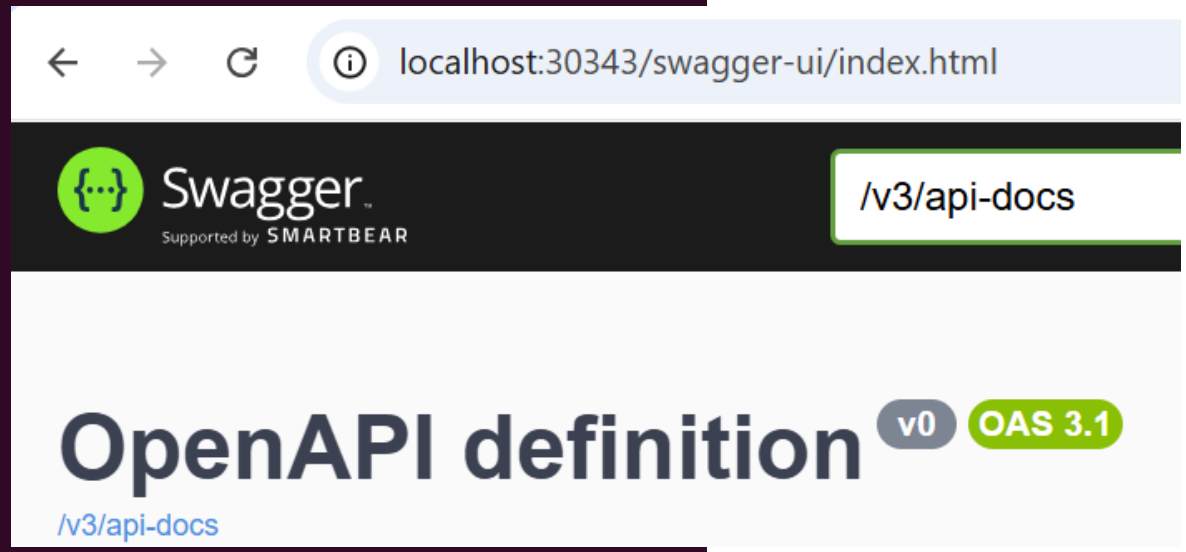
**NodePort : Le port du nœud (utilisé seulement avec type: NodePort). Permet d'accéder au service depuis l'extérieur.**

# Exposer le déploiement

- Exposer le port de votre application pour un accès au pod de l'extérieur

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl expose deployment my-country-service --type=NodePort --port=8082
service/my-country-service exposed
```

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl describe service my-country-service
Name:                     my-country-service
Namespace:                default
Labels:                   app=my-country-service
Annotations:              <none>
Selector:                 app=my-country-service
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.106.241.213
IPs:                      10.106.241.213
Port:                     <unset>  8082/TCP
TargetPort:               8082/TCP
NodePort:                 <unset>  30343/TCP
Endpoints:                10.1.0.8:8082
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```



# Supprimer le service et le deploiement

- Pour supprimer le service et le déploiement :

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl delete services my-country-service
service "my-country-service" deleted
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl delete deployment my-country-service
deployment.apps "my-country-service" deleted
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   22h
jamina_dev@DESKTOP-SKOOCDK:~$
```

# 6. 1 . 2ème méthode : utilisant des fichiers yaml

# Créer le déploiement depuis deployment.yaml

## A partir d'un fichier yaml

- Créer votre fichier yaml puis exécutez la commande kubectl apply -f

```
kubectl apply -f my-deployment.yaml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-country-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: country-service
  template:
    metadata:
      labels:
        app: country-service
    spec:
      containers:
      - name: country-service-container
        image: my-country-service:v1
        ports:
        - containerPort: 8082
```

```
C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl apply -f deployment.yaml
deployment.apps/my-country-service created

C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
my-country-service-555b445bd-7kczz    1/1     Running   0          14s
my-country-service-555b445bd-gsxpn    1/1     Running   0          14s
my-country-service-555b445bd-tjtn9    1/1     Running   0          14s
```

# Créer le déploiement depuis deployment.yaml

**Explication du fichier deployement.yaml**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment            # Nom du déploiement
  labels:
    app: myapp                      # Label qui identifie l'application
spec:
  replicas: 3                       # Nombre de pods à créer
  selector:
    matchLabels:
      app: myapp                    # Le sélecteur correspond aux labels des pods gérés
  template:                         # Modèle des pods créés par le déploiement
    metadata:
      labels:
        app: myapp                  # Label appliqué aux pods (doit correspondre au selector)
    spec:
      containers:
        - name: myapp-container     # Nom du conteneur
          image: nginx:latest       # Image Docker utilisée
          ports:
            - containerPort: 80     # Port sur lequel le conteneur écoute
```

# Créer le service depuis service.yaml

**A partir d'un fichier yaml**

- Créer votre fichier yaml puis exécutez la commande kubectl apply –f service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-country-service
spec:
  type: NodePort
  selector:
    app: country-service
  ports:
    - port: 8082
      targetPort: 8082
      nodePort: 30007
      protocol: TCP
```

**kubectl apply -f service.yaml**

# Démarche pour créer un déploiement /un service

C:\WINDOWS\system32\cmd.exe

```
C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl delete deployment my-country-service
deployment.apps "my-country-service" deleted

C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl delete service  my-country-service
service "my-country-service" deleted

C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl get pods
No resources found in default namespace.

C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl apply -f deployment.yaml
deployment.apps/my-country-service created

C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl apply -f service.yaml
service/my-country-service created
```

# Démarche pour créer un déploiement /un service

```
C:\WINDOWS\system32\cmd.exe
C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl describe service my-country-service
Name:                     my-country-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=country-service
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.99.50.58
IPs:                      10.99.50.58
Port:                     <unset>  8082/TCP
TargetPort:               8082/TCP
NodePort:                 <unset>  30007/TCP
```

- Extraire le numéro de port du nœud :

```
C:\Users\HP-EliteBook\Documents\eclipse-workspace\country-service>kubectl get service my-country-service -o jsonpath='{.spec.ports[0].nodePort}'
'30007'
```

**kubectl get service my-country-service -o jsonpath='{.spec.ports[0].nodePort}'**

# Kubernetes les types d’exposition d’un service

En Kubernetes, le service est exposé selon un type : clusterip, nodeport et loadbalancer permettant de rendre accessibles des services (pods, déploiements, etc.) à l'extérieur du cluster ou au sein de celui-ci.

**1. ClusterIP:**

- **Fonctionnement**: Crée un service qui expose un ensemble de pods à l'intérieur du cluster, sur une adresse IP interne et un port.
- **Accessibilité**: Accessible uniquement depuis l'intérieur du cluster Kubernetes.
- **Type par défaut**: Si aucun type n'est spécifié, le service est créé avec le type ClusterIP.

```
kubectl expose deployment <nom-du-déploiement> --port=<port>
--target-port=<port-cible> --type=ClusterIP
```

```
kubectl expose deployment myapp --port=80 --target-port=8080 --type=ClusterIP
```

# Kubernetes les types d’exposition

En Kubernetes, les commandes **kubectl expose** avec les options clusterip, nodeport et loadbalancer permettent de rendre accessibles des services (pods, déploiements, etc.) à l'extérieur du cluster ou au sein de celui-ci.

## 2. NodePort:

- **Fonctionnement**: Ouvre un port sur chaque nœud du cluster et redirige le trafic vers le service. Permet d’accéder au service depuis l’extérieur du cluster via NodeIP:NodePort.
- **Accessibilité**: Accessible depuis l'extérieur du cluster en utilisant l'adresse IP de n'importe quel nœud suivie du port NodePort.
- **Utilité**: Utile pour les tests et le développement, ou pour des applications qui n'ont pas besoin d'un équilibreur de charge externe.

```
kubectl expose deployment <nom-du-déploiement> --port=<port>
--target-port=<port-cible> --type=NodePort
```

```
kubectl expose deployment myapp --type=NodePort --port=80 --target-port=8080
```

# Kubernetes les types d’exposition

En Kubernetes, les commandes kubectl expose avec les options clusterip, nodeport et loadbalancer permettent de rendre accessibles des services (pods, déploiements, etc.) à l'extérieur du cluster ou au sein de celui-ci.

### 3. LoadBalancer:

- **Fonctionnement**: Demande à un fournisseur de cloud de créer un load balancer externe (par exemple, Google Cloud Load Balancer, AWS Elastic Load Balancer) qui gère le trafic vers le service. Kubernetes crée automatiquement un IP externe (ou DNS) pour accéder au service
- **Accessibilité**: Accessible depuis l'extérieur via l'adresse IP publique du load balancer.
- **Avantages**: Fournit une haute disponibilité et une scalabilité, gère le trafic entrant et peut être utilisé pour des applications de production.

```
kubectl expose deployment <nom-du-déploiement> --port=<port>
--target-port=<port-cible> --type=LoadBalancer
```

```
kubectl expose deployment myapp --type=LoadBalancer --port=80 --target-port=8080
```

# Questions

## 1. Peut on accéder à un pod bien donnée avec une adresse IP ?

➢ Chaque pod a une IP propre

```
kubectl get pods -o wide
```

```
NAME          READY   STATUS    IP            NODE
myapp-abc123  1/1     Running   10.244.0.12   worker-node1
myapp-def456  1/1     Running   10.244.1.7    worker-node2
```

➢ **on n'accède pas directement aux Pods**

Parce que :

- Les pods sont **éphémères** (ils peuvent mourir et être recréés).
- Leur **IP change** à chaque recréation.
- Kubernetes crée plusieurs pods identiques derrière un **Service**, qui agit comme un **load balancer interne**.

➢ Vous pouvez accéder à un Pod **directement via son IP**, mais uniquement depuis un **autre pod dans le cluster** ;

# Questions

**2. Est ce que qu'on peut accéder à l'application avec le port du service et non pas le nodeport ?**

➢ Ça va dépendre du **type de Service** que tu utilises.

**1. Service de type ClusterIP (par défaut)**

✅ **Accès avec le port du Service**

❌ **Pas d'accès depuis l'extérieur du cluster**

```yaml
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8080
```

- Le **port du service (80)** est accessible **uniquement depuis à l'intérieur du cluster** (ex. depuis un autre pod avec : curl http://myapp-service:80)
- Le trafic est redirigé automatiquement vers le **targetPort (8080)** dans les pods.

⚠️ Vous **ne pouvez pas accéder** à l'application depuis votre PC ou navigateur, sauf si vous êtes **dans le cluster.**

# Questions

**2. Est ce que qu'on peut accéder à l'application avec le port du service et non pas le nodeport ?**

➢ Ça va dépendre du **type de Service** que tu utilises.

**2. Service de type NodePort**

✅ **Accès depuis l'extérieur via NodeIP:NodePort**

✅ **Accès interne via le port du Service**

```
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30080
```

- Accès possibles :

**- Interne (dans le cluster)** → http://myapp-service:80

**- Externe (hors cluster)** → http://<NodeIP>:30080

👉 On peut toujours utiliser le **port du Service (80)** pour accéder à l'application **depuis un autre pod**, mais **pas depuis l'extérieur** (votre navigateur ou machine locale) — là, il faut le **NodePort**.

# Questions

**2. Est ce que qu'on peut accéder à l'application avec le port du service et non pas le nodeport ?**

➢ Ça va dépendre du **type de Service** que tu utilises.

**3. Service de type LoadBalancer**

✅ **Accès depuis l'extérieur via une IP publique**

Le LoadBalancer redirige le trafic vers le **port du service**, qui lui-même redirige vers le **targetPort**

```yaml
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
```

- Accès possibles :

**-** Vous accèdez à votre app via l'**adresse IP publique** (fournie par le cloud provider) :

`http://<ExternalIP>:80`

# Questions

**3. Dans un node, peut on avoir plusieurs pods qui sont différents ?**

- Un **nœud (Node)** peut héberger **plusieurs pods différents**, appartenant à **différentes applications, namespaces, déploiements, ou services**.
- Imaginons un cluster avec **un seul node**, et tu veux y déployer :
  - une **application web (nginx)**,
  - et une **base de données (redis)**.

Les deux pods peuvent coexister sur le même node.

Il suffit de créer deux fichiers deployement.yaml et exécutez les commandes sur le même node :

kubectl apply -f nginx-deployment.yaml

kubectl apply -f redis-deployment.yaml

# Questions

## 4. Dans Kubernetes, peut on avoir plusieurs cluster ?

- Dans Kubernetes, il est possible d'avoir **plusieurs clusters**. Chaque cluster est **indépendant** et contient son propre ensemble de **nœuds (Nodes), Pods, Services, namespaces, etc.**.

- **Isolation des environnements** : dev, test, production.
- **Haute disponibilité / résilience géographique** : déployer des clusters dans différentes régions.
- **Gestion de charges différentes** : un cluster pour des applications légères et un autre pour des workloads lourds.
- **Sécurité** : séparer des équipes ou projets sensibles.

# Questions

## 5. Quel est le rôle de Kubernetes dans l'environnement cloud ?

- Dans Kubernetes, il est possible d’avoir **plusieurs clusters**. Chaque cluster est **indépendant** et contient son propre ensemble de **nœuds (Nodes), Pods, Services, namespaces, etc.**.

- Le cloud fournit les ressources, Kubernetes fournit la **gestion et l’orchestration des applications conteneurisées** dans ces ressources.

# Questions

**5. Quand est ce qu'on crée le service avec le type clusterIP ?**

On crée un **Service de type ClusterIP** lorsque **l'application doit être accessible uniquement à l'intérieur du cluster Kubernetes**, **pas depuis l'extérieur**.

ClusterIP est le **type de Service par défaut** dans Kubernetes.
Il crée une **adresse IP virtuelle interne** (Cluster IP) accessible seulement depuis :

- les autres **Pods** du même cluster,
- ou d'autres **Services**.

Aucune ouverture vers l'extérieur (Internet ou machine cliente) n'est faite.

# Questions

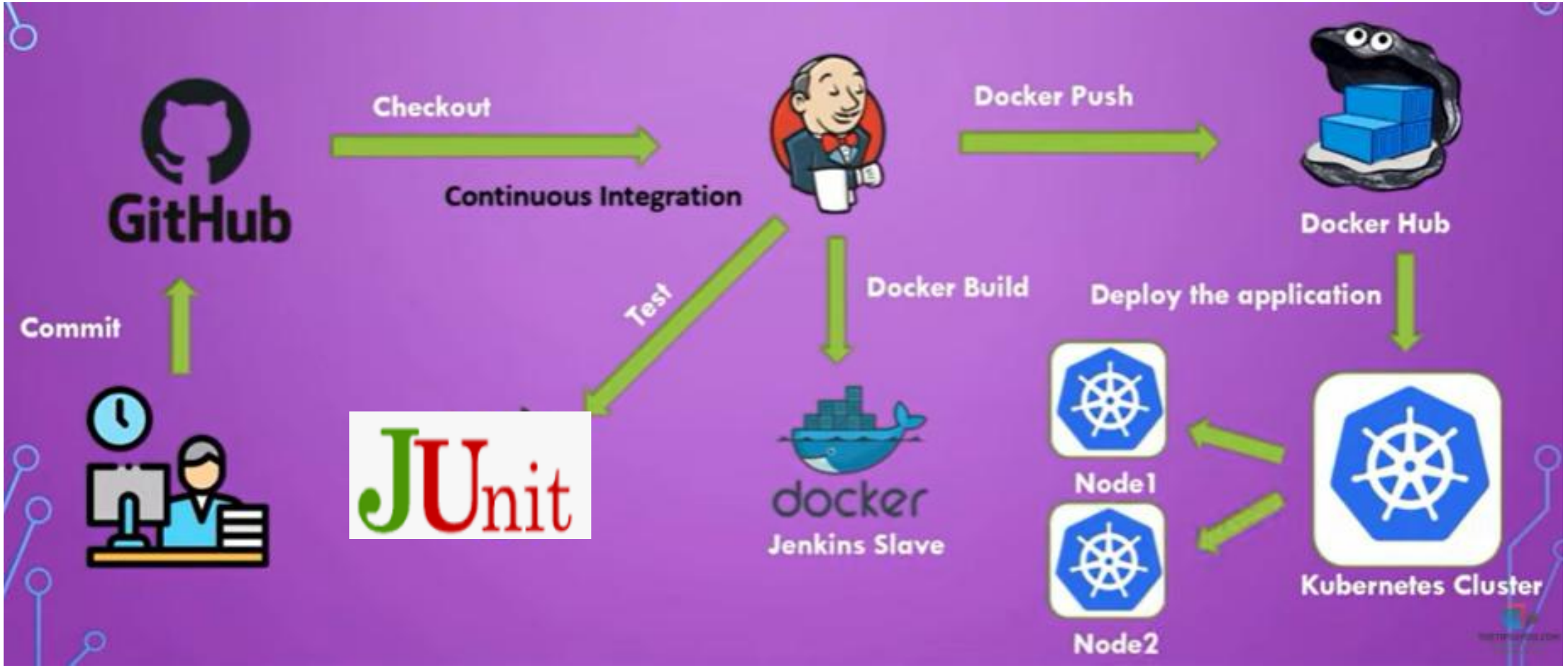
## 6. Quand est ce qu'on crée le service avec le type LoadBalancer ?

On crée un **Service de type LoadBalancer** lorsque **l'application doit être accessible depuis l'extérieur du cluster (Internet)**, **dans un environnement cloud** (comme AWS, GCP, Azure, etc.).
En d'autres termes :

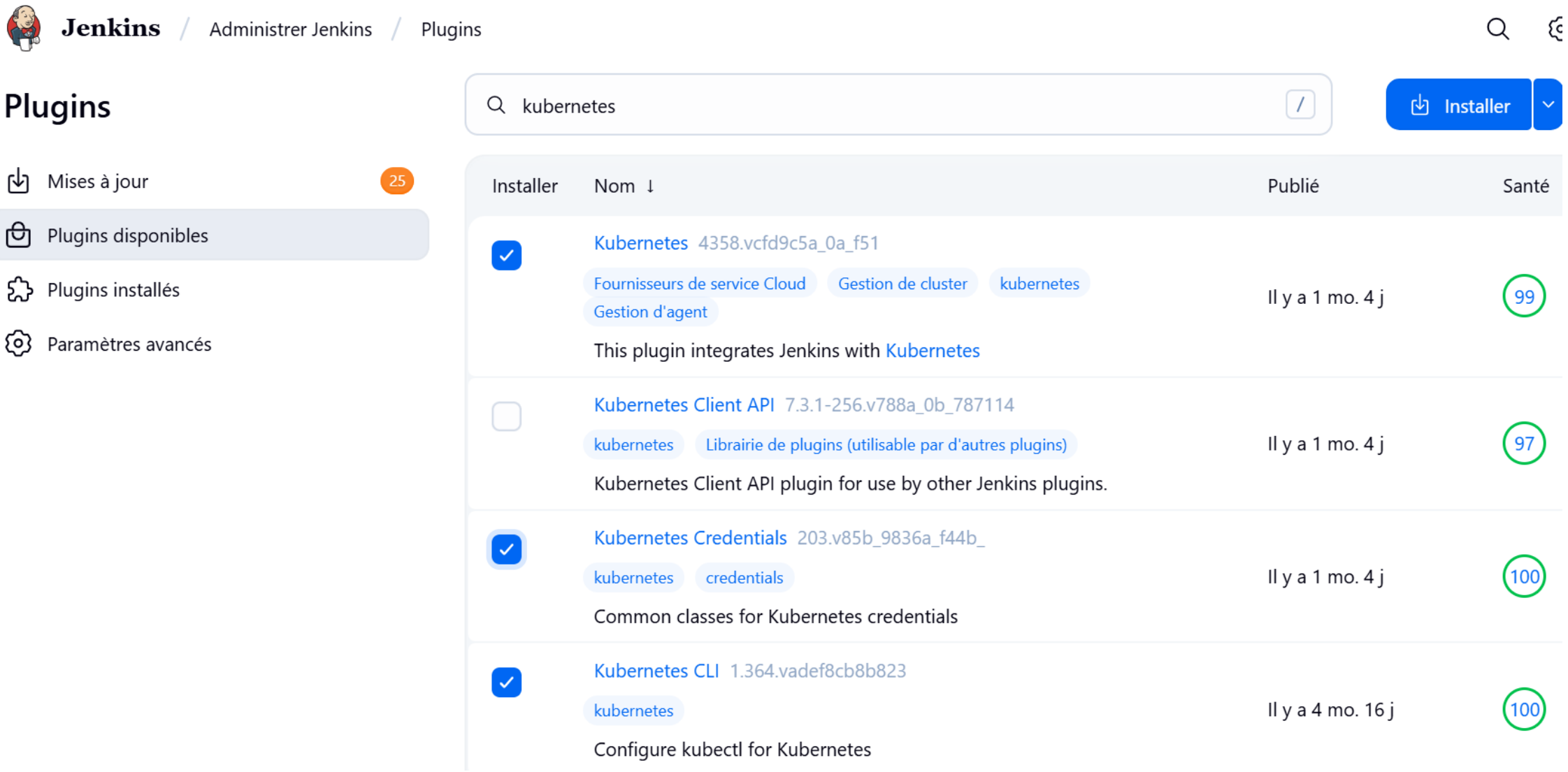
- **ClusterIP** → accès interne
- **NodePort** → accès externe basique
- **LoadBalancer** → accès externe **avec IP publique et équilibrage automatique du trafic**

# 6. Création d’un pipeline CI/CD avec Kubernetes

# Pipeline CI/CD avec Kubernetes



# Plugins à installer pour Kubernetes



# Création du token pour Credential

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl create namespace jenkins
namespace/jenkins created
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   43h
jenkins           Active   6s
kube-node-lease   Active   43h
kube-public       Active   43h
kube-system       Active   43h
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl create sa jenkins -n jenkins
serviceaccount/jenkins created
```

# Création du token pour Credential

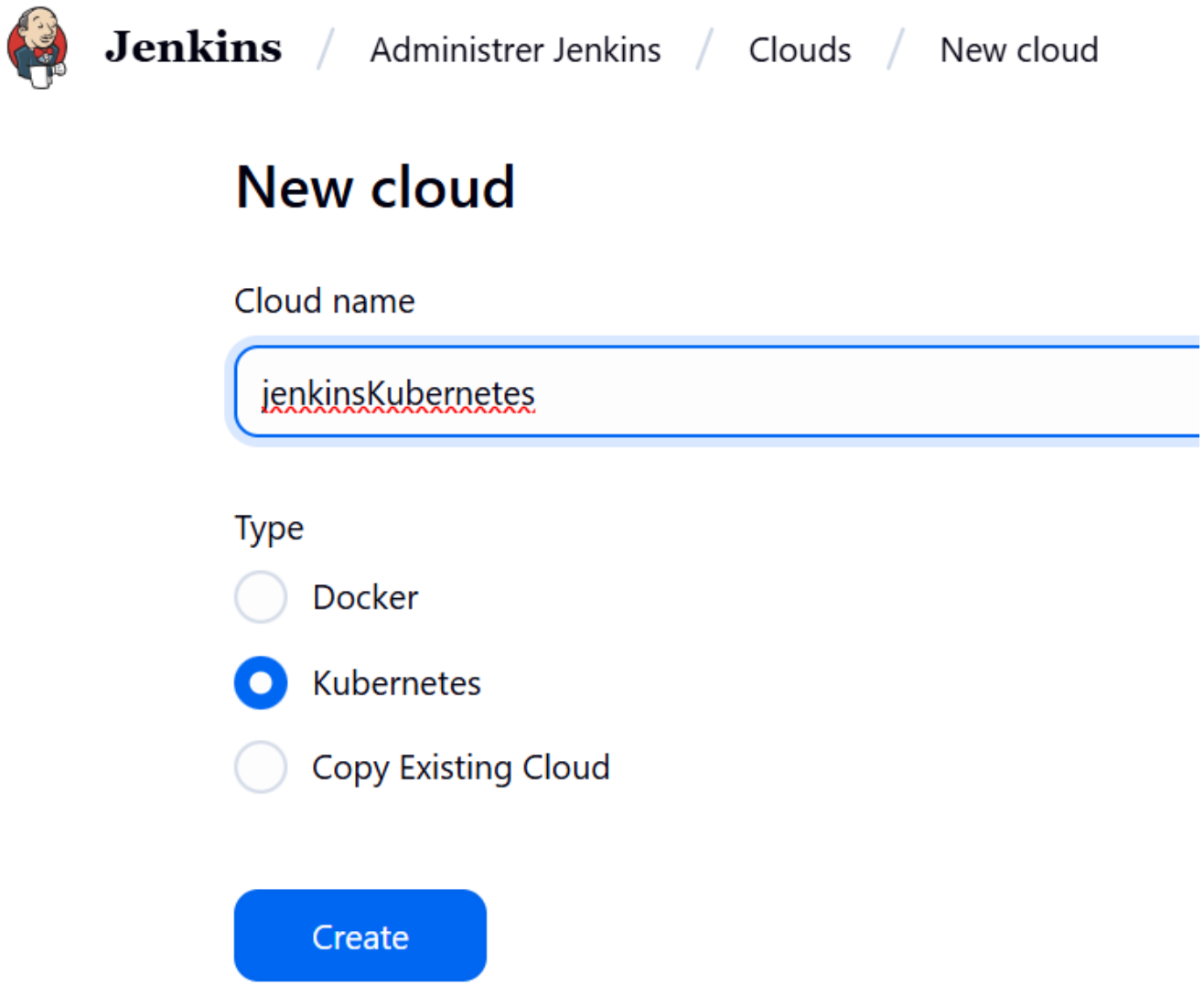
- Génération du token pour un an (duration=8760)

```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl create token jenkins -n jenkins --duration=8760h
eyJhbGciOiJSUzI1NiIsImtpZCI6IjJsM3R5N2ZiU0tEdHgyam5GME9wd1ZlMlhBNGlGX0RhdWNFY2tVcTVWSzQifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzg1OTI4OTUwLCJpYXQiOjE3NTQzOTI5NTAsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwianRpIjoiOTllZGUxYjYtMTZlNC00MGZmLWIyNDctYmRiYjA3MDQyODlkIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJqZW5raW5zIiwic2VydmljZWFjY291bnQiOnsibmFtZSI6Implbmtpbnmi LCJ1aWQiOiJiMTNiMDliYi01ZDYxLTRmY2MtYWJhMC1iMjgwNDZiZjFhY2EifX0sIm5iZiI6MTc1NDM5Mjk1MCwic3ViIjoic3lzdGVtOnNlcnZpY2VhY2NvdW50Omplbmtpbnm6amVua2lucyJ9.wiQTPwWBu3H7S_pSiR53_YHbp9hviMkWxJhuBm3A1PzBcUQa3gBRw-BjkqN1GbJcR_XlM00EawYpsGUcUp5gjDv_PZMiWZzFQ6Ih1mvHAfzAKjiNv1tXMvXd748r9isYPBTj_7QX5AXCw_D_O91DG7618mlQNIOU7JBNh1EAKxocxUKr8lRbmAhwK3LD_TmPoL3S88JKu8pa2cgwAPrI1Zek_m3py9F47c3Erja_jyoSD4PSPpSBHFsHBCjCjZeVJiPJiO5_GDPaLOAL5FznKQ1eHDeADGmB7DA9hHuAhTaMvVXetKd20nvV5TXP5N93HBHYuPYMj1ofxv30hDD59Q
```

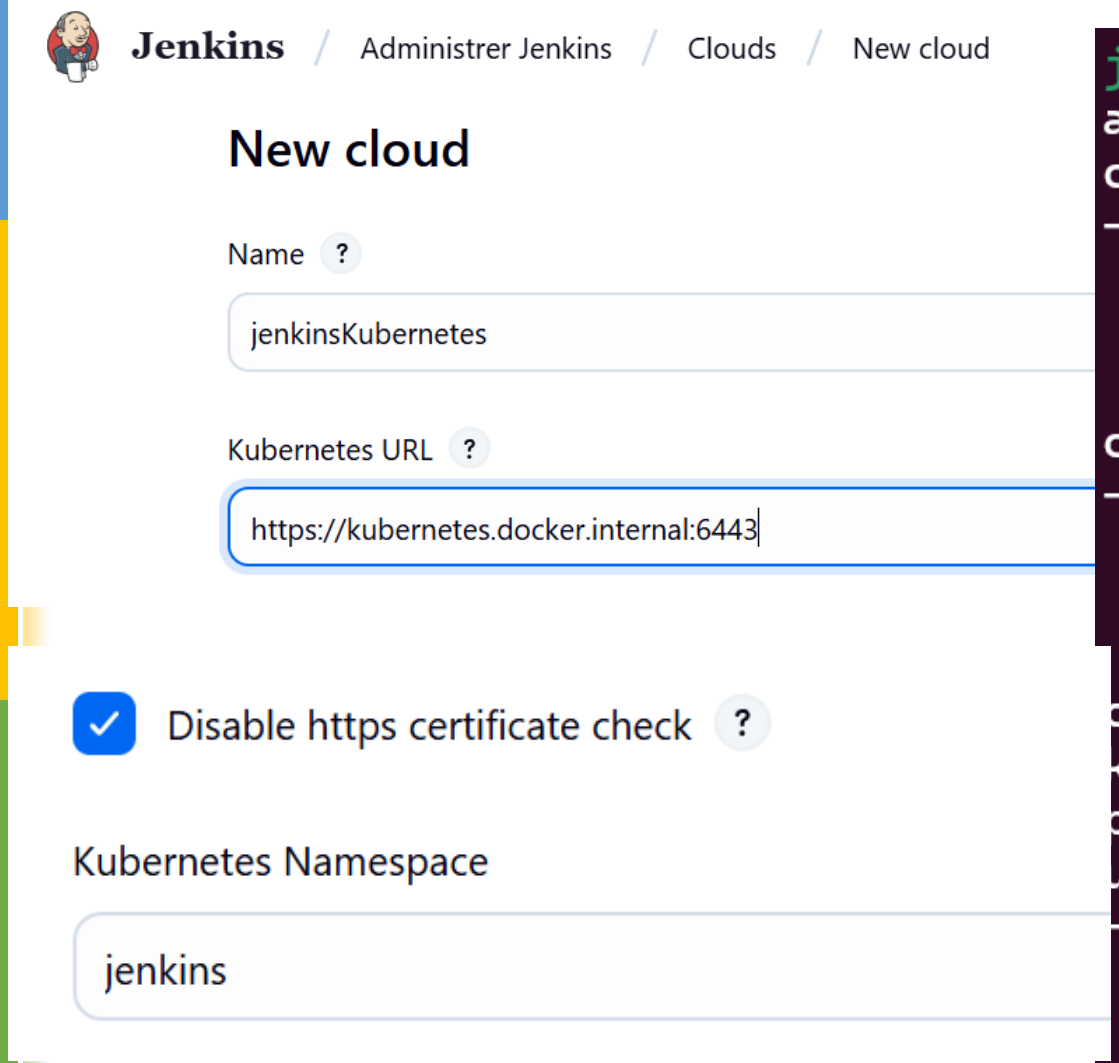
```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl create rolebinding jenkins-admin-binding --clusterrole=admin --serviceaccount=jenkins:jenkins --namespace=jenkins
rolebinding.rbac.authorization.k8s.io/jenkins-admin-binding created
```

# Création d’un nouveau cloud kubernetes

# Création d’un nouveau cloud kubernetes



# Création d'un nouveau cloud kubernetes

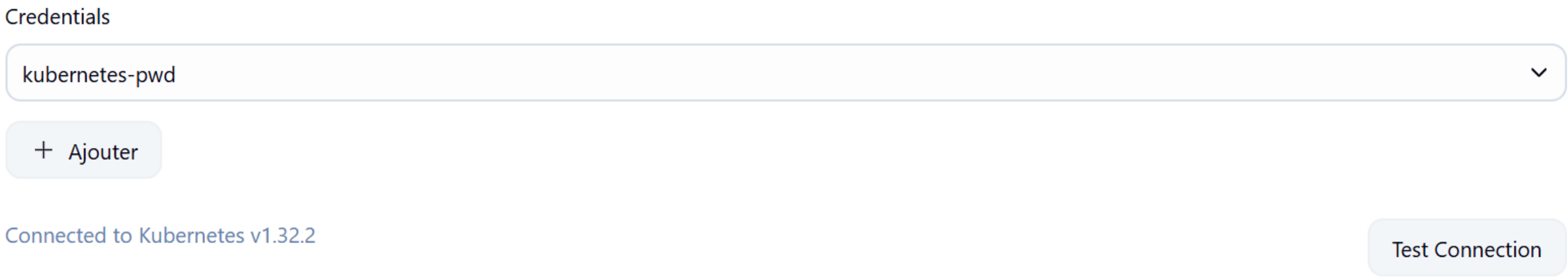



```
jamina_dev@DESKTOP-SKOOCDK:~$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://kubernetes.docker.internal:6443
  name: docker-desktop
contexts:
- context:
    cluster: docker-desktop
    user: docker-desktop
  name: docker-desktop
current-context: docker-desktop
kind: Config
preferences: {}
users:
- name: docker-desktop
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
```

# Création d’un nouveau cloud kubernetes


**Mette dans Secret le token généré**

# Tester la connexion avec Kubernetes



# Création du pipeline CI/CD

```
pipeline {
        agent any
        tools {
            maven 'mymaven'
        }
        stages {
            stage('Checkout code')
            {
                steps {
                    git branch: 'master', url: 'https://github.com/Jamina-ENSI/Country-service
                }
            stage('Build maven')
            {
                steps {
                    sh 'mvn clean install'
                }
            }
```

# Création du pipeline CI/CD

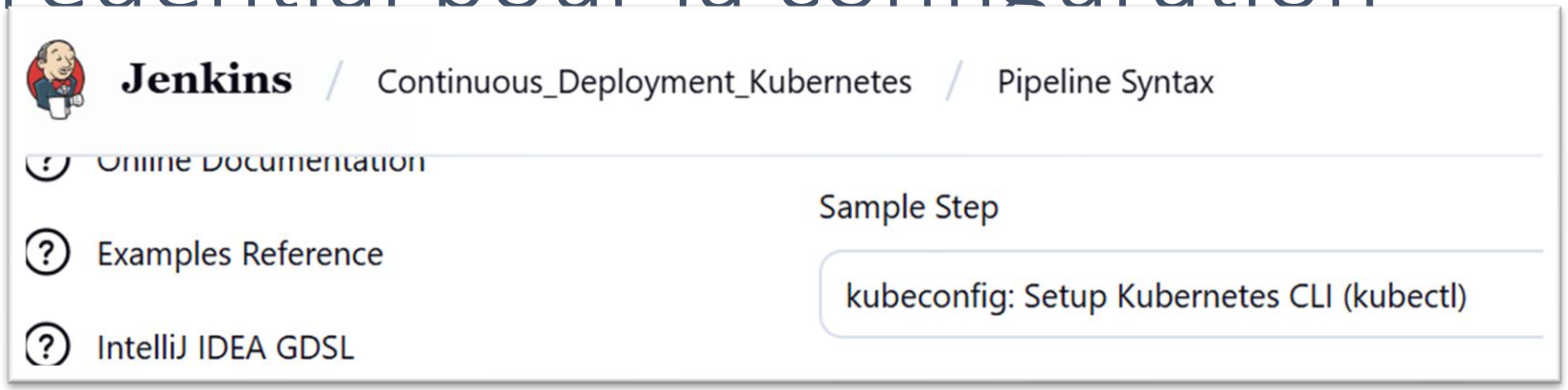
```
19  stage('Build Dockerfile ')
20  {
21      steps {
22          sh 'docker build . -t my-country-service:$BUILD_NUMBER '
23          withCredentials([string(credentialsId: 'dockerhub-pwd', variable: 'dockerhubpwd')]) {
24          sh 'docker login -u jamina2385 -p ${dockerhubpwd}'
25          }
26          sh 'docker tag my-country-service:$BUILD_NUMBER jamina2385/my-country-service:$BUILD_NU
27          sh 'docker push jamina2385/my-country-service:$BUILD_NUMBER'
28      }
29  }
```

# Création du pipeline CI/CD

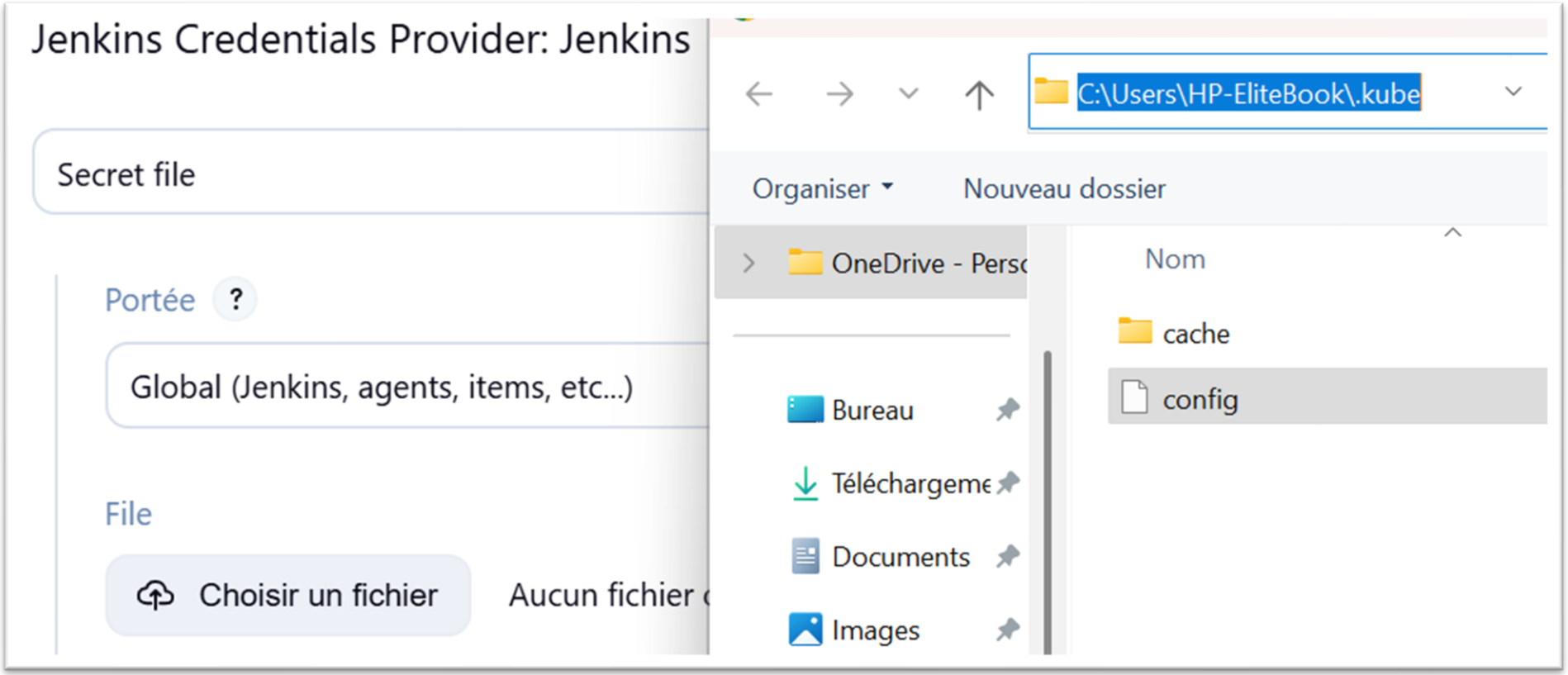
```
31            stage('Deploy to Kubernetes') {
32            steps {
33                script {
34                kubeconfig(credentialsId: 'kubeconfig-file', serverUrl: '') {
35                    sh 'kubectl apply -f deployment.yaml'
36                    sh 'kubectl apply -f service.yaml'
37                     }
38
39                }
40            }
41        }
42    }
43    }
```

# Création du crédential pour la configuration

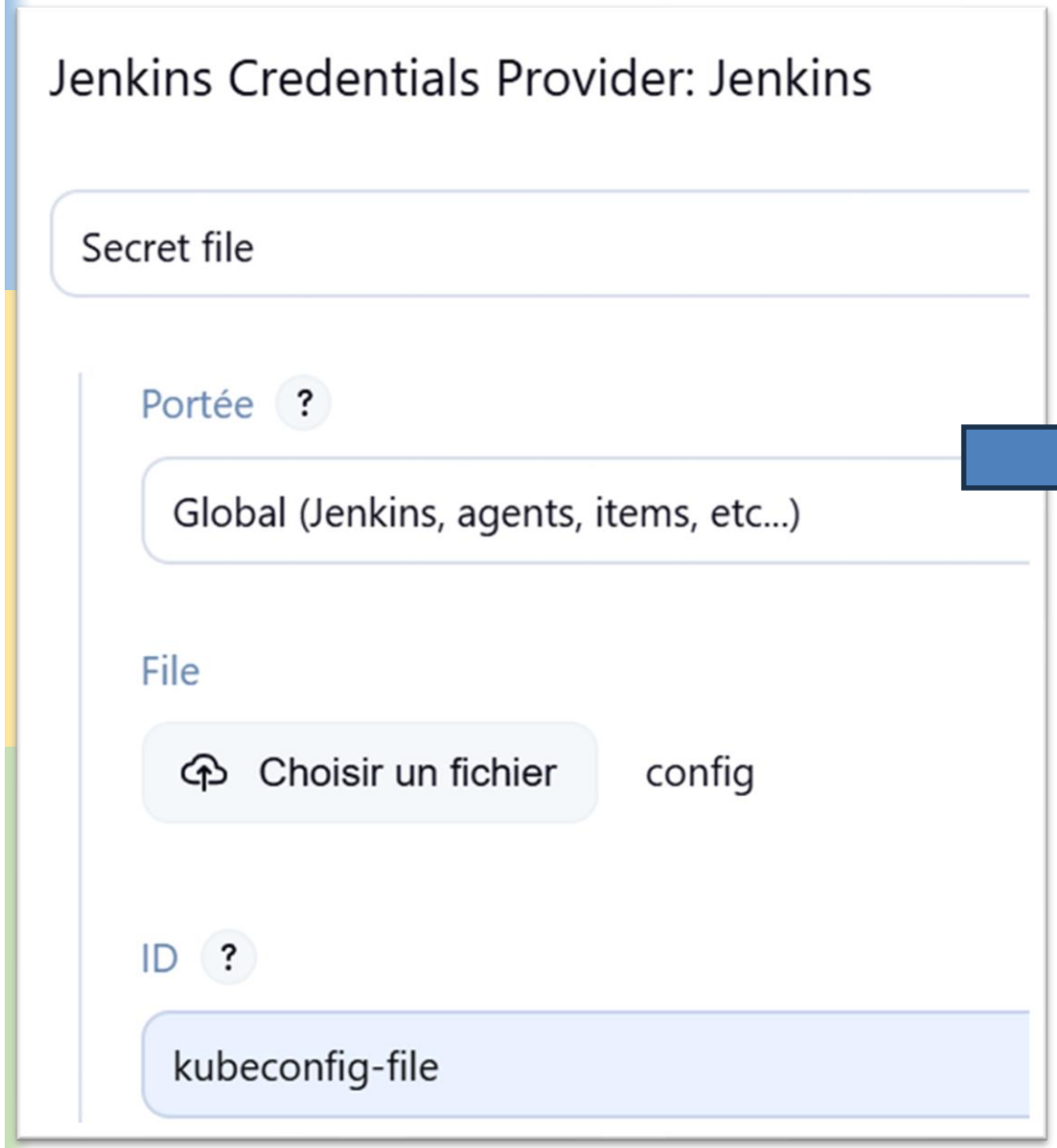
- Pour générer le credential, allez sur pipeline syntax :

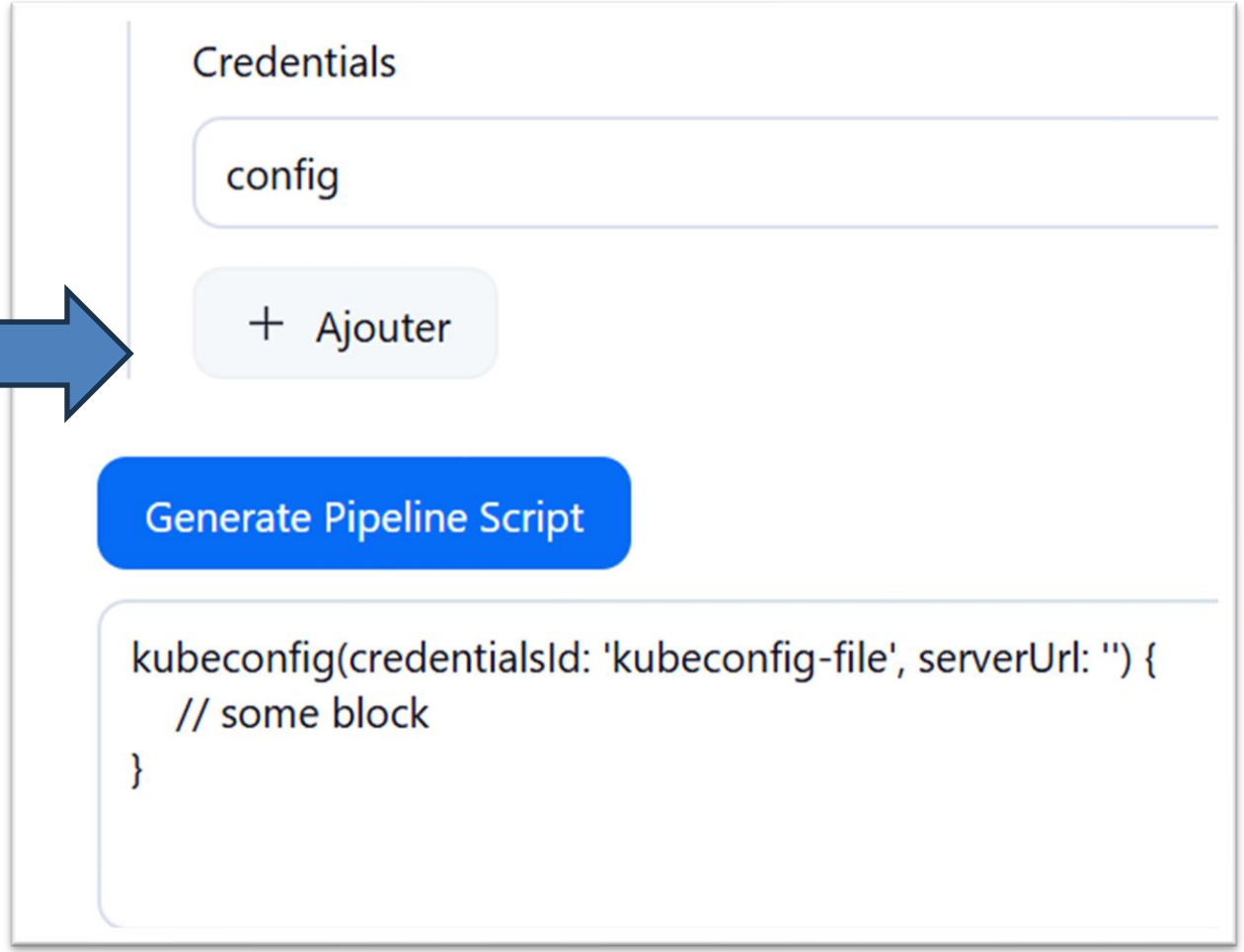



- Cliquez sur ajouter, choisissez Secret file puis choisissez le fichier config qui se trouve sous .Kube :



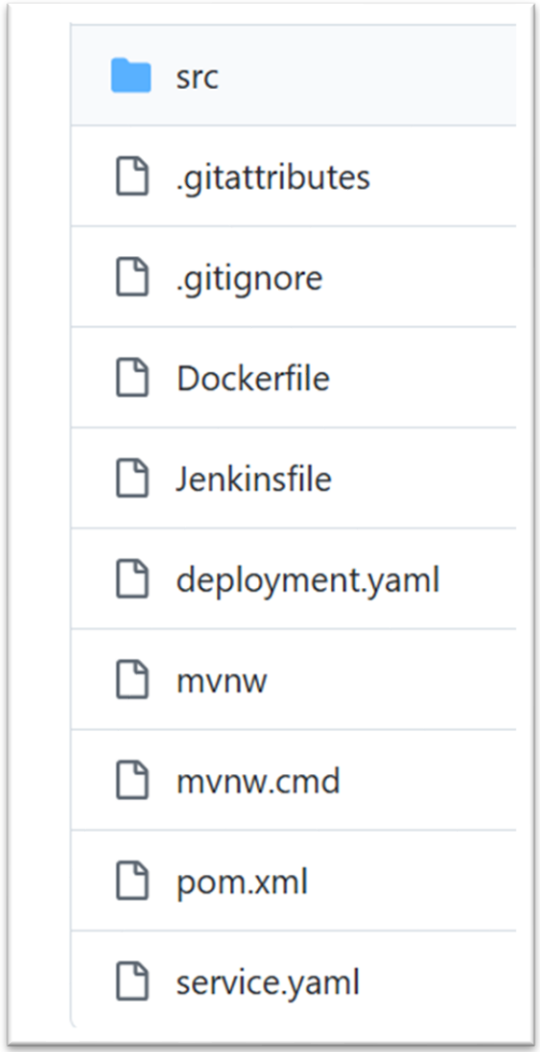
# Création du credential pour la configuration



# deployment.yaml et service.yaml

- Vérifiez que les deux fichiers existent dans votre Github.




master Country-service / service.yam

Jamina-ENSI Add files via upload

Code Blame 13 lines (13 loc) · 232 Bytes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-country-service
spec:
  type: NodePort
  selector:
    app: country-service
  ports:
    - port: 8082
      targetPort: 8082
      nodePort: 30007
      protocol: TCP
```

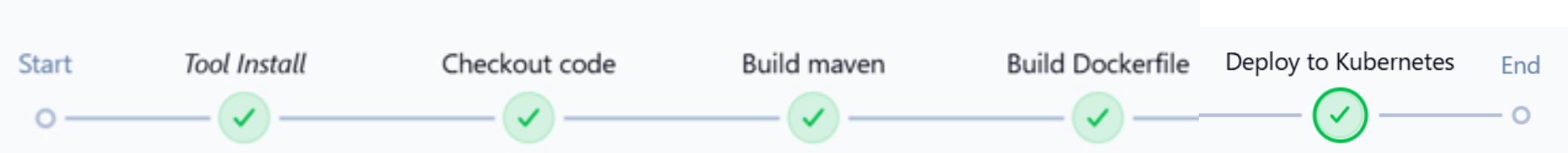
Jamina-ENSI Update deployment.yaml

Code Blame 20 lines (19 loc) · 400 Bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-country-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: country-service
  template:
    metadata:
      labels:
        app: country-service
    spec:
      containers:
      - name: country-service-container
        image: jamina2385/my-country-service:2
        ports:

        - containerPort: 8082
```

# Déploiement réussi




```
+ kubectl apply -f deployment.yaml
deployment.apps/my-country-service created
[Pipeline] sh
+ kubectl apply -f service.yaml
service/my-country-service created
[Pipeline] }
[Pipeline] // kubeconfig
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

- *"Apprendre par le projet, c'est découvrir par l'action, créer par la compréhension, et réussir par la persévérance."*

amina.jarraya@ensi-uma.tn

**ENSI Manouba**