# Kubernetes

## Installation de Kubernetes

### Client

* Télécharger le "kubectl" fichier par le commande:

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

* Télécharger le checksum fichier "kubectl.sha256":

curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

* Valider le fichier d'installation par le commande:

echo "$(<kubectl.sha256) kubectl" | sha256sum --check

**-->** l'output doit être: kubectl: OK

* Install kubectl:

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

* Installation test:

kubectl version –client

### Server

* Se réferer à: [minikube start | minikube (k8s.io)](https://minikube.sigs.k8s.io/docs/start/)
* Pour Ubuntu 20.10 (AMD64):

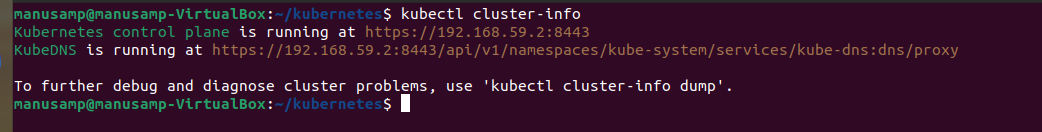
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

* Pour lancer minikube la première fois, utiliser le commande suivant:

minikube start --driver=docker

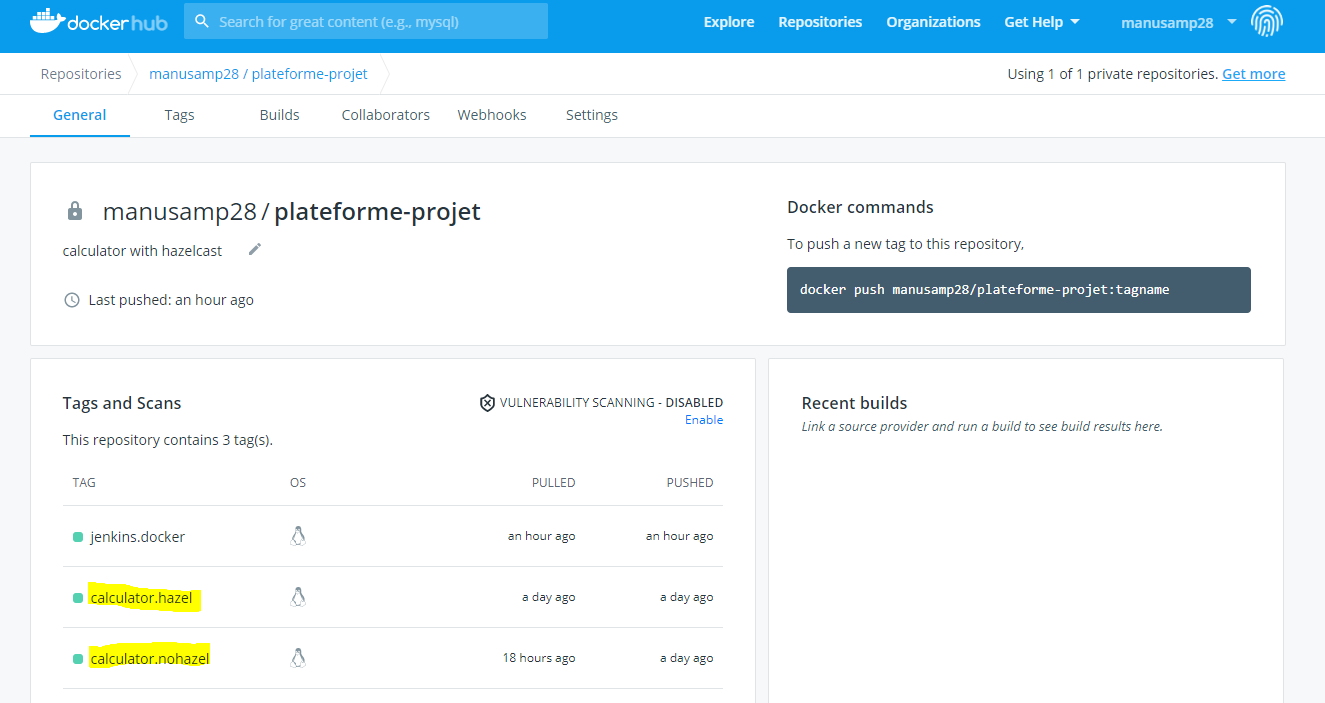
Maintenant le commande kubectl cluster-info permet de vérifier les détails du cluster:



## Déployer une application à partir d’une image privée disponible sur Docker Hub

Commençons par le déploiement d’une application à partir d’une image Docker disponible sur Docker Hub.

Pour simplifier les choses (et pour montrer comment utiliser Docker Hub), on utilise une version de l’application “calculator” qui n’utilise pas le service de caching Hazelcast. C'est la première version de l’application que j’ai préparé et pour lequel j’ai sauvegardé l’image crée en sein du stage “Docker build” de la pipeline et que j’ai déposé sur Docker Hub, où j’ai une compte:



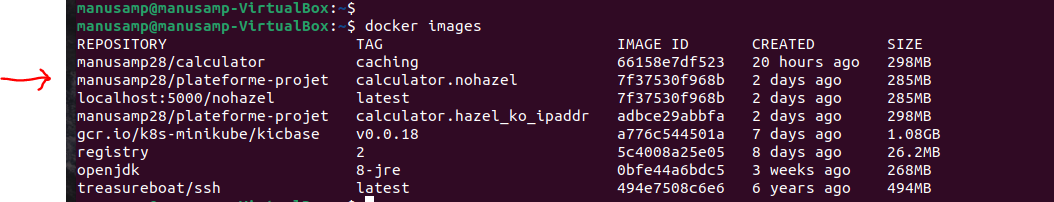
L'image à utiliser est “calculator-nohazel”.

Pour télécharger une image de propre compte Docker Hub, faurt se authentifier en utilisant le commande “docker login” et en fournissant ses crédentials.

Maintenant est possible télécharger l’image via le command docker pull:

docker pull manusamp28/plateforme-projet:calculator.nohazel

...et l’image est téléchargé et disponible comme on peut vérifier par le commande docker images:



Le problème est que Kubernetes par défaut peux faire le pull seulement à partir d'un registre Docker public; l'image récuperé par le Docker hub de cette façon est une image privée.

Pour pouvoir utliser cette image pour déployer l’application par kubernetes, est possible utiliser un **Secret**.

Un Secret est un objet Kubernetes utilisé pour stocker des informations sensibles, comme des mots de passe, des tokens, des clés SSH...

Ci-dessous la procédure pour créer un secret nommé **regcred** avec les identifiants pour utiliser notre image privé:

* Les identifiants dont nous avons besoin sont les mêmes à fournir quand on utilise le commande docker login: les identifiants Docker Hub, que sont stocké dans le fichier */home/manusamp/.docker/config.json* (sous forme cryptéé)
* Le suivant commande permet la création d’un secret “regcred” contenant ces identifiants:

*kubectl create secret generic regcred --from-file=.dockerconfigjson=/home/manusamp/.docker/config.json --type=kubernetes.io/dockerconfigjson*

* Le commande:

*kubectl get secret regcred --output=yaml*

Permet de voir le contenu du secret

* Maintenant est possible utiliser notre image pour déployer l’application.

Ci-dessous le contenu du fichier “deployment.yaml”:

apiVersion: apps/v1

kind: Deployment

metadata:

name: calculator-deployment

labels:

app: calculator

spec:

replicas: 3

selector:

matchLabels:

app: calculator

template:

metadata:

labels:

app: calculator

spec:

containers:

- name: calculator

image: manusamp28/plateforme-projet:calculator.nohazel

ports:

- containerPort: 8080

imagePullSecrets:

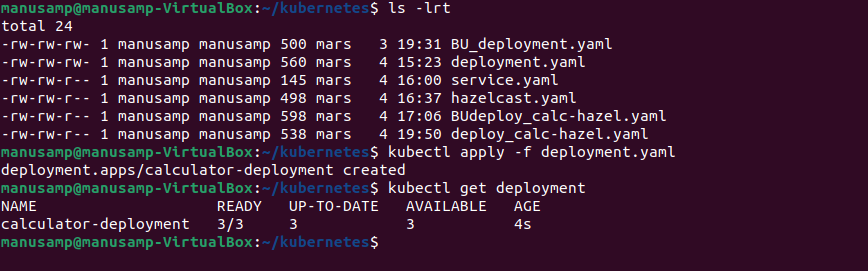
- name: regcred

À noter que on utilise l’image privé par le secret regcred.

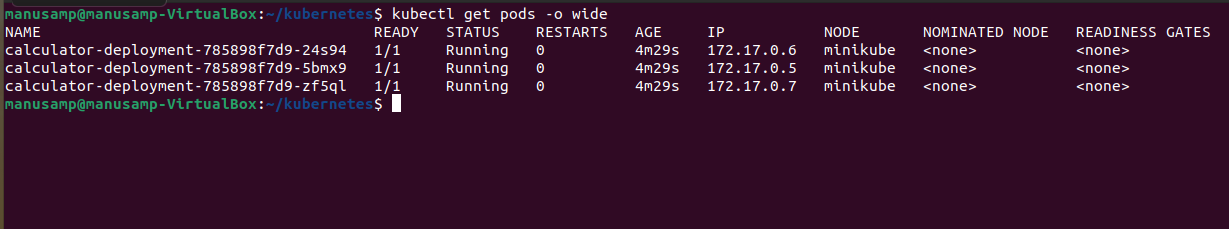
Ce fichier permet de déployer 3 replicas de l’application calculator.

Maintenant nous pouvons déployer l’application par le commande:

kubectl apply -f deployment.yaml:

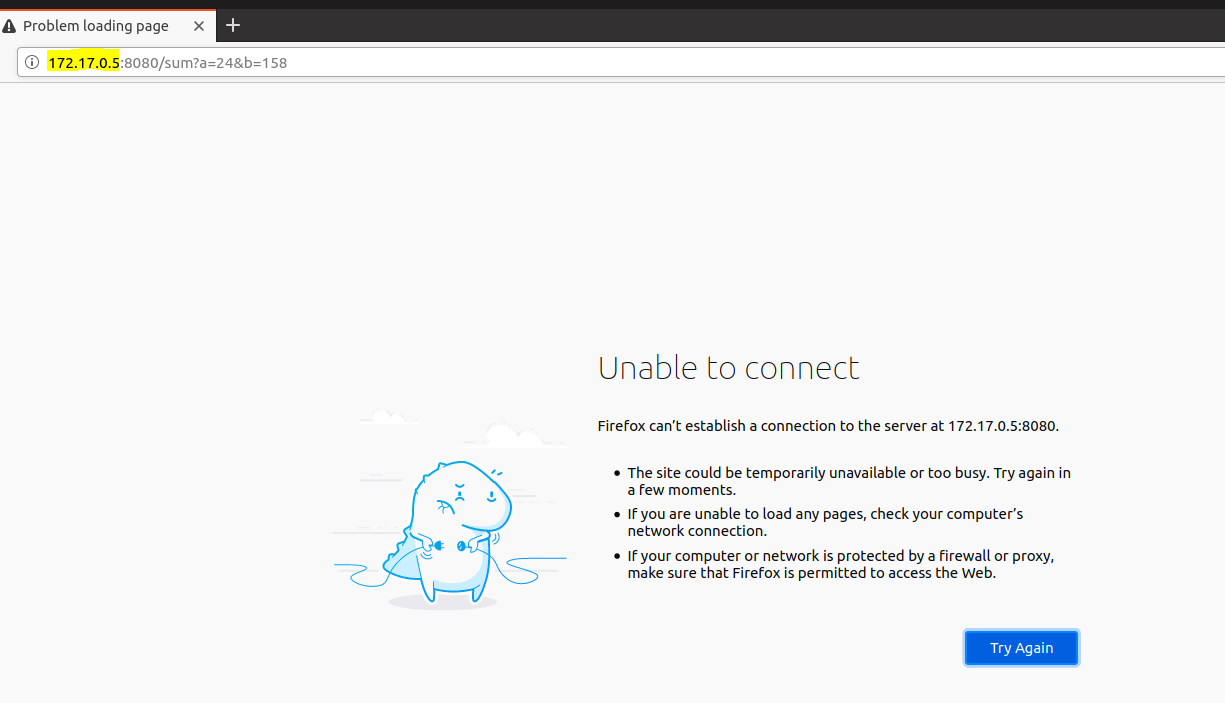


Le commande kubectl get pods permet de voir les pods, correspondants aux replicas de notre application:



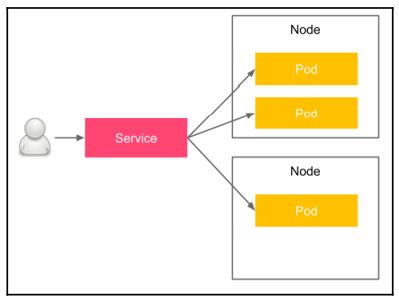
Par l’option “-o wide” il est possible voir aussi l’adresse ip des pods.

Attention: depuis que kubernetes est un orchestrator, le but est d’assurer que l’application soit toujours accessible et qu'il y a toujours autant de pods actifs que de répliques spécifiées dans le fichier de déploiement (3 dans ce cas). Donc les informations concernants l’identité du pods ont peu d’importance et il ne sont pas utilisé: l’application n’est pas accessible par l’adresse ip du pods:



Pour accéder à l’application, faut créer un **Service.**

Un Service est un objet qui permet l’abstraction des Pods sur lesquels notre application est déployé:



Ci dessous le service utilisé dans notre cas (service.yaml):

apiVersion: v1

kind: Service

metadata:

name: calculator-service

spec:

type: NodePort

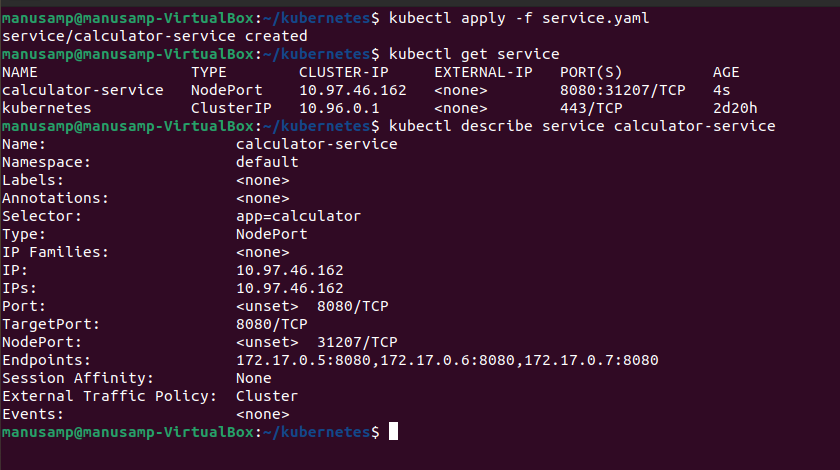
selector:

app: calculator

ports:

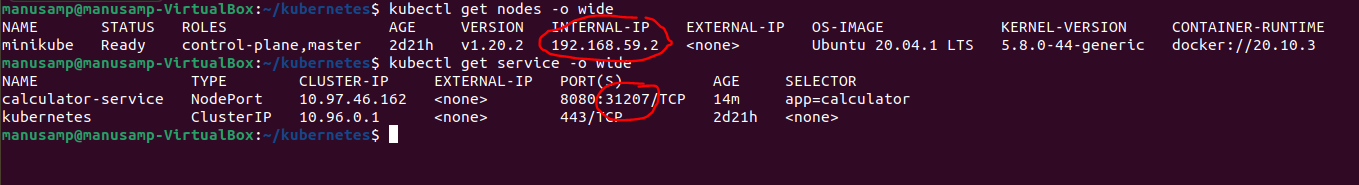
- port: 8080

Le commande kubectl apply -f service.yaml permet d’utiliser notre service:

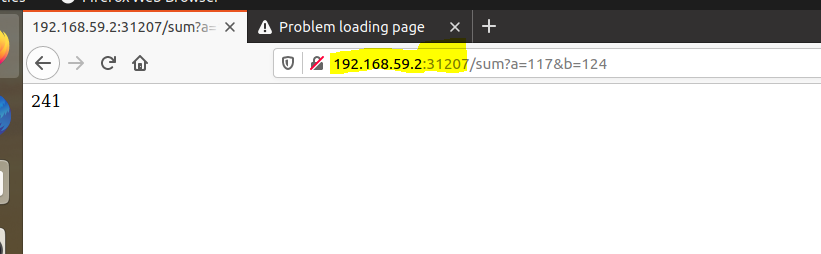


Même l’adresse du service (10.97.46.162) est pour usage interne. Le service peut exposer notre application au externe de différentes façons.

Dans ce cas on a utilisé un Service de type **NodePort:** c’est à-dire que notre application sera exposé sur le même porte du notre cluster. Pour accéder l’application faut utiliser <NODE-IP>:<NODE-PORT>, que nous pouvons découvrir par les commandes get nodes et get service:



Et maintenant est possible accéder au application:



Dans le cas où une des pods (ou touts) devrait-il devenir inaccessible, kubernetes assure que des autres pods seront deployés (avec différents IPs...): l’application sera toujours accessible avec le même IP.