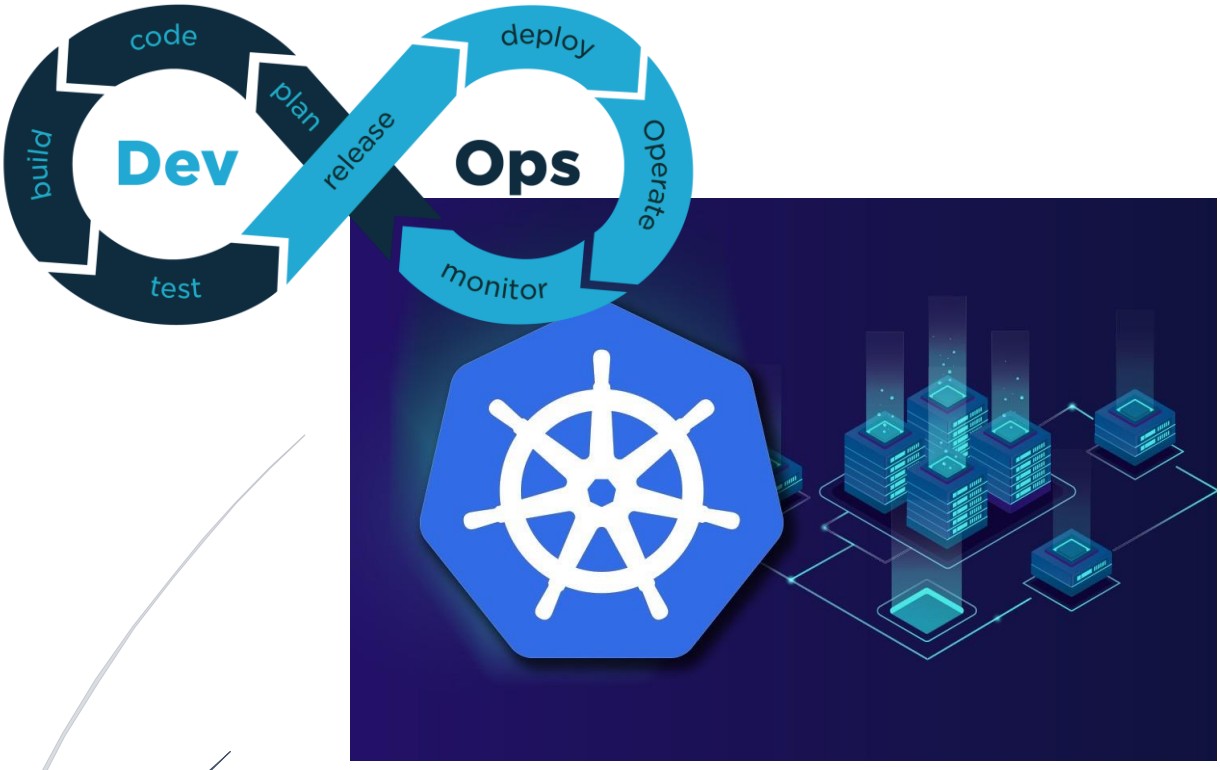
DF FORMATION

MINISTÈRE DU TRAVAIL, DE L'EMPLOI ET DE L'INSERTION

YYYOURS

EVALUATION EN COURS DE FORMATION

PARCOURS ADMINISTRATEUR SYSTEME DEVOPS



Automatiser la mise en production d'une application avec une plateforme

Evaluation : ECF8

Version : du 10/11/2025

Auteur : FT

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF8	Titre	**Automatiser la mise en production d'une application avec une plateforme**	Du	10/11/2025

Sommaire

Table des matières

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF8	Titre	**Automatiser la mise en production d'une application avec une plateforme**	Du	10/11/2025

1. PRESENTATION

1.1. DESCRIPTION DE LA COMPETENCE – PROCESSUS DE MISE EN ŒUVRE

- Lorsque les premiers tests de l'application à déployer sont satisfaisants, préparer un environnement de pré-production afin de tester l'application en situation réelle.
- Y déployer l'application, la tester et faire remonter aux développeurs les dysfonctionnements constatés.
- Lorsque les tests sont satisfaisants, déployer l'application sur l'environnement réel de production.

1.2. CONTEXTE(S) PROFESSIONNEL(S) DE MISE EN ŒUVRE

- Ces opérations concernent la mise en production d'une nouvelle application mais également la mise en production de ses évolutions successives.
- Cette compétence s'appuie sur l'utilisation d'une plateforme (orchestrateur) de type Kubernetes.

1.3. CRITERES DE PERFORMANCE

- L'environnement de pré-production est conforme à l'environnement de production
- Les dysfonctionnements sont remontés aux développeurs
- Les containers sont décrits sur la plateforme
- L'application ou sa mise à jour est disponible pour les utilisateurs

1.4. SAVOIR-FAIRE TECHNIQUES, ORGANISATIONNELS, RELATIONNELS

- Utiliser une plateforme de type Kubernetes
- Préparer un environnement de pré-production
- Déployer l'application dans l'environnement de pré-production
- Tester et faire remonter les erreurs aux développeurs
- Définir à l'aide de la plateforme les caractéristiques de chaque container nécessaire à l'application
- Lancer le déploiement à l'aide de la plateforme sur l'environnement de production
- Déployer une mise à jour de l'application à l'aide de la plateforme
- Echanger avec les développeurs
- Consulter de la documentation technique rédigée en anglais
- Effectuer une veille technologique
- Connaissance du processus de mise en production
- Connaissance des démarches Agile et Scrum

1.5. CAHIER DES CHARGES

a. INSTALLER VAGRANT ET METTRE EN PLACE UN CLUSTER KUBERNETES AVEC LE FICHIER d'installation vagrantfile

b. QUELLES SONT LES IP DU NODE ET DU MASTER ?

c. METTRE EN PLACE UN SCALING DE VOTRE APPLICATION NGINX

1. Mettre en place un pod **Nginx**
2. Créer un service nodeport tcp avec redirection sur le port 80 (8080:80)
   Vérifier que votre service a bien été créé avec la commande : kubectl get svc
3. Accéder au pod via le navigateur. Vous aurez besoin de l'IP du master:port du pod
   Vous pouvez lire le port avec un : kubectl get pods
4. Créer deux instances de réplica de votre pod
5. Modifier le contenu des pages html des deux réplicas, vous noterez réplica1 dans la première instance et réplica2 sur la page de la deuxième instance
6. Vérifier en rafraîchissant la page ipmaster:port que le scaling est bien en place
7. Exporter la sortie de la commande : "kubectl describe deploy"

2. QUESTIONS ECF8

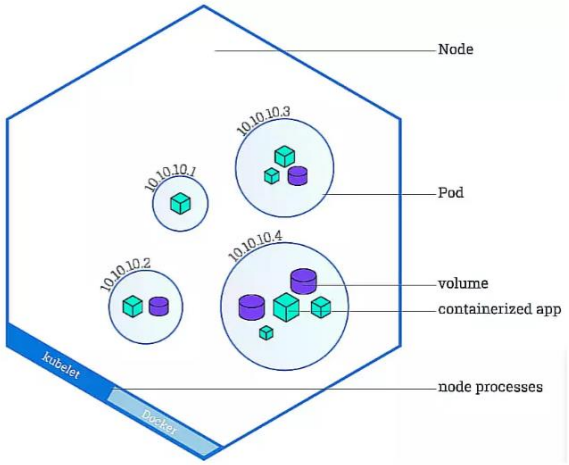
Issu du projet Borg de Google, Kubernetes pilote les avancées en termes d'orchestrations des conteneurs dans le monde open source depuis 2014.

Cet outil aux 7 rayons est utilisé en entreprise pour faciliter le déploiement, la mise à l'échelle et la maintenance des applications.

2.1. DEFINIR QU'EST-CE QU'UN NODE, UN POD, UN SERVICE ET UN DEPLOIEMENT

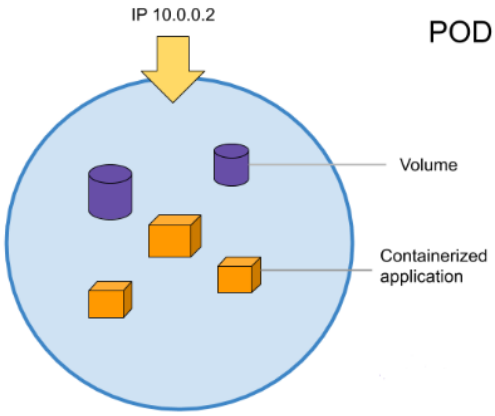
a. Node

- Un nœud Kubernetes est chacune des machines interconnectées, physiques ou virtuelles, qui fonctionnent ensemble en tant que cluster Kubernetes et contient chacune des charges de travail Kubernetes.
- Les deux types de nœuds dans Kubernetes sont les nœuds Master (nœud de plan de contrôle) et les nœuds Workers.
- Un cluster Kubernetes comprend généralement un ou plusieurs nœuds master et plusieurs nœuds workers.



b. Pod

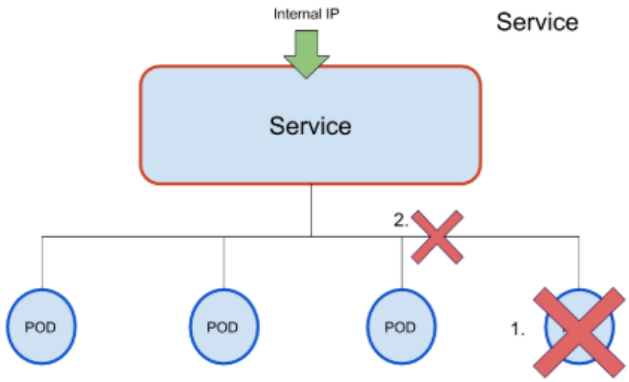
- Un pod est l'objet de base de Kubernetes qui est chargé d'encapsuler les conteneurs, les ressources de stockage et les adresses IP internes. Un pod représente une instance d'une application dans Kubernetes.
- Ces pods sont lancés dans un cluster Kubernetes composé de nœuds.
- Un nœud est une machine de travail (VM ou machine physique) dotée d'un environnement d'exécution de conteneur.
- Cela signifie qu'un pod s'exécute sur un nœud, mais peut facilement être instancié sur un autre nœud (pour des raisons de tolérance aux pannes, par exemple).



Les pods sont censés être éphémères, en fait Kubernetes peut faire évoluer le nombre de ces pods pour s'adapter au trafic entrant, créant ou supprimant ainsi des pods à la demande.

c. Service

- Un service est une abstraction qui maintient un ensemble logique de pods acceptant le trafic entrant et exposant un port de service pour accéder aux pods sous-jacents.
- Le service Kubernetes peut ainsi gérer les changements de trafic en modifiant le nombre de pods.
- Si l'application doit être accessible depuis l'extérieur du cluster, Kubernetes propose un composant appelé Ingress.
- Il s'appuie sur un service et gère le trafic externe avec un chiffrement SSL.



d. Déploiement

- Les déploiements sont des objets Kubernetes qui gèrent les pods.
- Ils permettent de déployer une version spécifique d'une application et de spécifier le nombre de pods nécessaires à son fonctionnement.
- Lorsqu'une nouvelle version est prête à être mise en production, le déploiement peut facilement gérer cette mise à niveau sans temps d'arrêt en appliquant deux règles de base :

  - maxSurge spécifie le nombre maximal de pods pouvant être créés au-delà du nombre de pods souhaité.
  - maxUnavailable spécifie le nombre maximal de pods qui peuvent être indisponibles pendant le déploiement.

2.2. QUELLE EST LA DIFFERENCE ENTRE DOCKER ET KUBERNETES ?

Docker et Kubernetes diffèrent par leurs fonctions d'écosystème de conteneurisation.

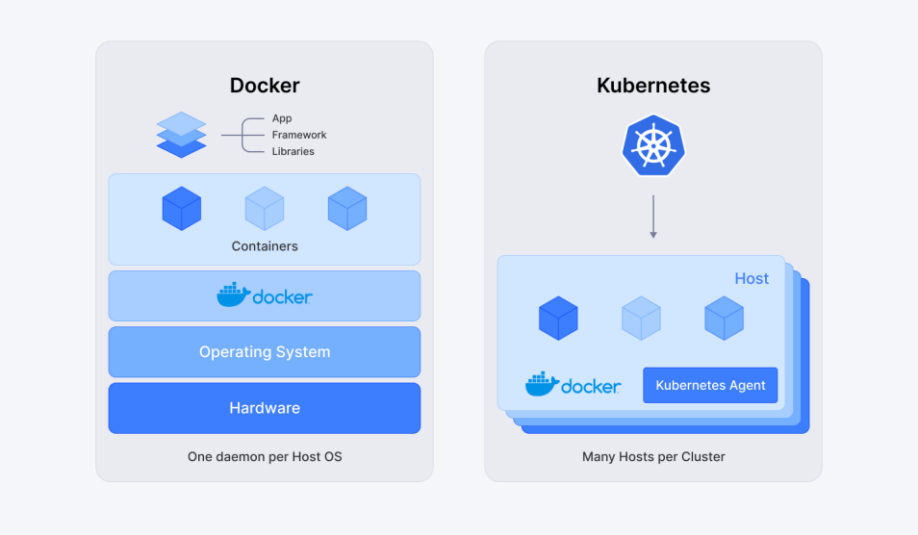
- Docker est un environnement d'exécution de conteneurs qui permet aux développeurs de conditionner des applications dans des conteneurs, garantissant ainsi la cohérence entre différents environnements informatiques.
- La plateforme d'orchestration de conteneurs de Kubernetes offre des fonctionnalités de gestion pour les conteneurs déployés sur des clusters, automatisant des tâches telles que le déploiement, la mise à l'échelle et l'auto-réparation des applications.

Tandis que Docker se concentre sur la conteneurisation, Kubernetes exécute les applications conteneurisées à grande échelle.

Ensemble, Docker et Kubernetes sont devenus fondamentaux pour le développement et l'orchestration d'applications dans des environnements cloud, permettant des paradigmes de développement logiciel plus agiles, évolutifs et efficaces.

Bien qu'ils gèrent des aspects différents de la gestion des conteneurs, ils sont souvent utilisés conjointement pour créer des environnements conteneurisés.

Docker fonctionne comme un environnement d'exécution de conteneurs axé sur l'automatisation du déploiement d'applications au sein des conteneurs ; Kubernetes va plus loin en gérant l'orchestration, la coordination et la planification des conteneurs sur un cluster de serveurs.



3. Mise en place Environnement Kubernetes avec Vagrant

3.1. Étape 1 : Pre-requis

La mise en place de la plateforme sera faite sur Windows en installant les 2 composants suivant :

- Vagrant : Version: 2.4.7
- VirtualBox : Version 7.1.10 platform packages
- Ubuntu/bionic64 : Ubuntu 20.04 64-bit operating system (chargé dans le vagrantfile)

3.2. Étape 2 : Vagrantfile pour Kubernetes

Création dossier vagrant/kubernetes dans lequel il y aura les 2 fichiers suivants :

- **vagrantfile** (**ANNEXE**)
- **kubernetes_install.sh** (**ANNEXE**)

a. Création vagrantfile (création de 2 VMs)

- **master** node
- **worker** node

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64" # "ubuntu/focal64" refers to Ubuntu 20.04 LTS (Focal Fossa) 64-bit.

  # Master Node
  config.vm.define "k8s-master" do |master|
    master.vm.hostname = "k8s-master"
    master.vm.network "private_network", type: "static" , ip: "192.168.56.10" # Configures a private network for the "k8s-master" VM.
    master.vm.hostname = "master"
    master.vm.provider "virtualbox" do |v|
      v.name = "master"
      v.memory = 2048
      v.cpus = 2
    end
    master.vm.provision "shell", path: "kubernetes_install.sh", args: ["master"] # Configures a shell script provisioner for the "master" VM.
  end

  # Worker Node
  config.vm.define "k8s-node" do |node|
    node.vm.hostname = "k8s-node"
    node.vm.network "private_network", type: "static" , ip: "192.168.56.11" # Configures a private network for the "k8s-node" VM
    node.vm.hostname = "node"
    node.vm.provider "virtualbox" do |v|
      v.name = "node"
      v.memory = 2048
      v.cpus = 2
    end
    node.vm.provision "shell", path: "kubernetes_install.sh", args: ["node"] # Configures a shell script provisioner for the "node" VM.
  end
end
```

Pour chaque VM un provisionnement va être effectué pour installer et configurer Kubernetes (script kubernetes_install.sh)

b. Lancement vagrantfile

vagrant_output.log

Lancement commande vagrant up 2>&1 | tee vagrant_output.log →

```
$ vagrant up 2>&1 | tee vagrant_output.log
Bringing machine 'k8s-master' up with 'virtualbox' provider...
Bringing machine 'k8s-node' up with 'virtualbox' provider...
==> k8s-master: Importing base box 'ubuntu/focal64'...
==> k8s-master: Matching MAC address for NAT networking...
==> k8s-master: Checking if box 'ubuntu/focal64' version '20240821.0.1' is up to date...
==> k8s-master: Setting the name of the VM: master
==> k8s-master: Clearing any previously set network interfaces...
==> k8s-master: Preparing network interfaces based on configuration...
    k8s-master: Adapter 1: nat
    k8s-master: Adapter 2: hostonly
==> k8s-master: Forwarding ports...
    k8s-master: 22 (guest) => 2222 (host) (adapter 1)
==> k8s-master: Running 'pre-boot' VM customizations...
==> k8s-master: Booting VM...
==> k8s-master: Waiting for machine to boot. This may take a few minutes...
    k8s-master: SSH address: 127.0.0.1:2222
    k8s-master: SSH username: vagrant
    k8s-master: SSH auth method: private key
    k8s-master: Warning: Connection reset. Retrying...
```

```
k8s-master: Your Kubernetes control-plane has initialized successfully!
k8s-master:
k8s-master: To start using your cluster, you need to run the following as a regular user:
k8s-master:
k8s-master:   mkdir -p $HOME/.kube
k8s-master:   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
k8s-master:   sudo chown $(id -u):$(id -g) $HOME/.kube/config
k8s-master:
k8s-master: Alternatively, if you are the root user, you can run:
k8s-master:
k8s-master:   export KUBECONFIG=/etc/kubernetes/admin.conf
k8s-master:
k8s-master: You should now deploy a pod network to the cluster.
k8s-master: Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
k8s-master:   https://kubernetes.io/docs/concepts/cluster-administration/addons/
k8s-master:
k8s-master: Then you can join any number of worker nodes by running the following on each as root:
k8s-master:
k8s-master: kubeadm join 192.168.56.10:6443 --token jilind.x8ezw1bu3jclcm7j \
k8s-master:         --discovery-token-ca-cert-hash sha256:af0aab0fbb7fea3e48bed1be491d36680e81a0637c923be8
k8s-master: namespace/kube-flannel created
k8s-master: clusterrole.rbac.authorization.k8s.io/flannel created
k8s-master: clusterrolebinding.rbac.authorization.k8s.io/flannel created
k8s-master: serviceaccount/flannel created
k8s-master: configmap/kube-flannel-cfg created
k8s-master: daemonset.apps/kube-flannel-ds created
k8s-node: Importing base box 'ubuntu/focal64'...
k8s-node: Matching MAC address for NAT networking...
k8s-node: Checking if box 'ubuntu/focal64' version '20240821.0.1' is up to date...
k8s-node: Setting the name of the VM: node
k8s-node: Fixed port collision for 22 => 2222. Now on port 2200.
k8s-node: Clearing any previously set network interfaces...
k8s-node: Preparing network interfaces based on configuration...
k8s-node: Adapter 1: nat
k8s-node: Adapter 2: hostonly
k8s-node: Forwarding ports...
k8s-node: 22 (guest) => 2200 (host) (adapter 1)
k8s-node: Running 'pre-boot' VM customizations...
k8s-node: Booting VM...
k8s-node: Waiting for machine to boot. This may take a few minutes...
k8s-node: SSH address: 127.0.0.1:2200
k8s-node: SSH username: vagrant
k8s-node: SSH auth method: private key
```

```
k8s-node: This node has joined the cluster:
k8s-node: * Certificate signing request was sent to apiserver and a response was received.
k8s-node: * The Kubelet was informed of the new secure connection details.
k8s-node:
k8s-node: Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

c. Vérification Création VM

```
PS C:\vagrant\Kubernetes> vagrant status
Current machine states:

k8s-master                running (virtualbox)
k8s-node                  running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

3.3. ÉTAPE 3 : INITIALISATION DE KUBERNETES

a. Création script kubernetes_install.sh

Ce script est lancé dans le vagrantfile par provisionnement des 2 VMs.

```bash
#!/bin/bash

ROLE=$1
HOSTNAME=$(hostname)

# Install Docker
sudo apt-get update && sudo apt-get install -y docker.io
sudo systemctl enable docker && sudo systemctl start docker

# Install Kubernetes tools
sudo apt-get install -y apt-transport-https curl gpg # apt-transport-https enables secure connections for the package manager
sudo mkdir -p /etc/apt/keyrings

sudo rm -f /etc/apt/keyrings/kubernetes-archive-keyring.gpg
# These two lines allow to trust the Kubernetes packages and know where to download them from.
    curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
    echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

# kubeadm: the command to bootstrap the cluster.
# kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.
# kubectl: the command line util to talk to your cluster.
sudo apt-get update && sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

# Disable swap (MUST be disable in order for the kubelet to work properly)
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab

# Master configuration
# 192.168.56.10 - IP address that the API Server will advertise to other nodes and clients
# 10.244.0.0 - IP address range for the Pod network, especially when using Flannel as the Container Network Interface (CNI)
if [ "$ROLE" = "master" ]; then
  # 10.244.0.0/16 range typically forms a network overlay on top of the existing host network (192.168.56.x)
  kubeadm init --apiserver-advertise-address=192.168.56.10 --pod-network-cidr=10.244.0.0/16

  mkdir -p /home/vagrant/.kube
  sudo cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config # Allowing the vagrant user to interact with the Kubernetes cluster
  sudo chown vagrant:vagrant /home/vagrant/.kube/config

  # Install Flannel CNI(Container Network Interface) plugin in Kubernetes cluster
  su - vagrant -c "kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml"

  # Save the join command (generate and securely share the command that worker nodes will use to join the cluster)
  kubeadm token create --print-join-command > /vagrant/join.sh

elif [ "$ROLE" = "node" ]; then
  while [ ! -f /vagrant/join.sh ]; do sleep 5; done
  cp /vagrant/join.sh /tmp/join.sh
  chmod +x /tmp/join.sh
  /tmp/join.sh # Allow to connect to and join the Kubernetes cluster
fi
```

Kubernetes a besoin d'un réseau de pods qui permet aux pods, répartis sur plusieurs nœuds, de communiquer entre eux comme s'ils étaient sur un même réseau local. C'est ce qu'on appelle le CNI (Container Network Interface).

b. Utilisation du réseau Flannel

- **Simplicité** : Flannel est souvent recommandé pour démarrer car il est simple à déployer et configurer. C'est un excellent choix pour les petits clusters et pour apprendre Kubernetes.
- **Compatibilité** : Fonctionne bien avec la plupart des distributions Kubernetes.
- **Overlay Network** : Utilise VXLAN (par défaut) ou d'autres backend (host-gw, AWS VPC) pour créer un réseau overlay.
- Alternatives à Flannel
  - **Calico** : Supporte réseau et sécurité (firewall par pod), très populaire en production.
  - **Weave Net** : Facile, bonne isolation, multi-cloud.
  - **Cilium** : Basé sur eBPF, sécurité et observabilité avancées.

c. Vérification de l'installation Kubernetes

- Accès au master kubernetes (vagrant ssh k8s-master)

```
PS C:\vagrant\Kubernetes> vagrant ssh k8s-master
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-216-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Wed Jun 18 14:24:36 UTC 2025

  System load:             1.02
  Usage of /:              8.6% of 38.70GB
  Memory usage:            37%
  Swap usage:              0%
  Processes:               151
  Users logged in:         0
  IPv4 address for enp0s3: 10.0.2.15
  IPv6 address for enp0s3: fd00::ca:62ff:feef:6a24


Expanded Security Maintenance for Infrastructure is not enabled.

3 updates can be applied immediately.
3 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

14 additional security updates can be applied with ESM Infra.
Learn more about enabling ESM Infra service for Ubuntu 20.04 at
https://ubuntu.com/20-04

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Jun 18 14:22:07 2025 from 10.0.2.2
vagrant@master:~$
```

- Vérification IP (**ANNEXE**)
  - Master node : 192.168.56.10
  - Worker node : 192.168.56.11

- Vérification cluster Kubernetes

```
vagrant@master:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.56.10:6443
CoreDNS is running at https://192.168.56.10:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

```
vagrant@master:~$ kubectl get nodes
NAME     STATUS   ROLES           AGE     VERSION
master   Ready    control-plane   9m2s    v1.28.15
node     Ready    <none>          3m44s   v1.28.15
```

4. Mise en place scaling application Nginx

4.1. Mise en place du pod NGINX

- Création du fichier nginx-pod.yml

```yaml
apiVersion: v1 # version of the Kubernetes API
kind: Pod # type of Kubernetes object to create (smallest deployable unit in Kubernetes)
metadata:
  name: nginx-pod
  labels:
    app: nginx # specific label applied to this Pod
spec:
  containers: # A Pod can run one or more containers
  - name: nginx
    image: nginx # Docker image
    ports:
    - containerPort: 80 # nginx container inside the Pod will listen for connections on port 80
```

- Déploiement du pod nginx : kubectl apply -f nginx-pod.yml

```
vagrant@master:~/kubernetes$ kubectl apply -f nginx-pod.yml
pod/nginx-pod created
```

- Vérification du pod nginx

```
vagrant@master:/home/nginx$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          2m49s
```

```
vagrant@master:/home/nginx$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP           NODE
nginx-pod   1/1     Running   0          10m   10.244.1.3   node
```

4.2. Creation du service NodePort TCP

- Création du fichier nginx-service.yml

```yaml
apiVersion: v1 # version of the Kubernetes API
kind: Service # type of Kubernetes object to create
metadata:
  name: nginx-service
spec:
  type: NodePort # Services expose the Service on a static port on each Node in the cluster
  selector: # how the Service finds the Pods it should route traffic to
    app: nginx # This selector tells the nginx-service to direct traffic to any Pods that have the label app: nginx
  ports:
  - port: 80 # This is the port on the Service
    targetPort: 80 # This is the port on the container
# Kubernetes reserves the port range 30000-32767 for NodePort Services. You must choose a port within this range
    nodePort: 30800  # NodePort must be in the range 30000-3276
```

- Déploiement du service nginx

Le service exposé sur tous les nodes est le `30800` (non 8080 car le NodePort doit être dans 30000-32767).

  - kubectl apply -f nginx-service.yml

```
vagrant@master:~/kubernetes$ kubectl apply -f nginx-service.yml
service/nginx-service created
```

- Vérification du service nginx : kubectl get svc

```
vagrant@master:/home/nginx$ kubectl get svc
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        2d5h
nginx-service   NodePort    10.101.180.187   <none>        80:30800/TCP   27s
```

- Vérification que Kubernetes expose le port 30800 à l'extérieur

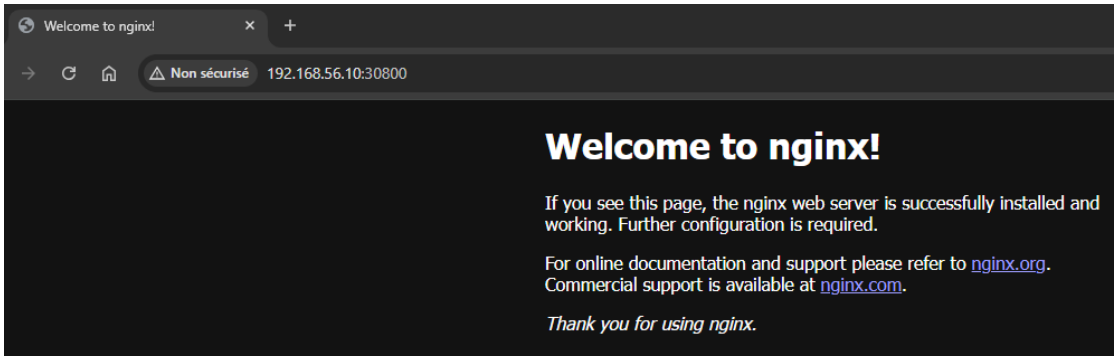
```
vagrant@master:/home/nginx$ kubectl get svc -A | grep 30800
default       nginx-service   NodePort    10.101.180.187   <none>        80:30800/TCP
```

- PORT(S)
  - **80** : port sur lequel le service écoute à l'intérieur du cluster
  - **30800** : port ouvert sur tous les nœuds du cluster (via iptables), redirigé vers le port 80 du pod cible

- CLUSTER-IP — IP interne attribuée par Kubernetes
  - 10.96.0.1 (pour le service kubernetes) et 10.101.180.187 (pour nginx-service) sont des adresses IP internes au réseau de service Kubernetes.
  - Par défaut, le réseau de service est dans la plage 10.96.0.0/12 (peut-être changé au moment de l'initialisation du cluster avec --service-cidr).
  - Ces IPs ne sont pas accessibles de l'extérieur, elles ne servent qu'au routage interne dans le cluster (pod ↔ service).

4.3. ACCES AU POD NGINX VIA LE NAVIGATEUR

http://192.168.56.10:30800 (IP du master + port NodePort)



4.4. CREATION DE DEUX REPLICAS DU POD NGINX

- Création du fichier nginx-replica.yml

```yaml
apiVersion: apps/v1 # Specifies the API version for this Kubernetes object
kind: Deployment # Defining a Kubernetes Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2 # Specifies the desired number of identical Pods
  selector:
    matchLabels: # Deployment will only manage Pods that match the labels specified in this selector
      app: nginx
  template: # This is the heart of the Deployment
            # Defines the "blueprint" for the Pods that the Deployment will create and manage
    metadata:
      labels:
        app: nginx # This label matches the selector
    spec:
      containers:
      - name: nginx
        image: nginx # The Docker image to use for the container
        ports:
        - containerPort: 80 # The port that the NGINX container will expose within the Pod
```

- Déploiement des réplicas nginx : kubectl apply -f nginx-replica.yml

```
vagrant@master:/home/nginx$ kubectl apply -f nginx-replica.yml
deployment.apps/nginx-deployment created
```

- Vérification du déploiement créé :

```
vagrant@master:/home/nginx$ kubectl get deployments -A
NAMESPACE     NAME               READY   UP-TO-DATE   AVAILABLE   AGE
default       nginx-deployment   2/2     2            2           54s
kube-system   coredns            2/2     2            2           2d6h
```

4.5. MODIFICATION DU CONTENU DES PAGES HTML DES 2 REPLICAS

Pour ça, il faut créer deux pods différents ou utiliser des configmaps et des pods séparés (en Prod)

- Accès aux pods et modifier le fichier index.html.
- Liste des pods : kubectl get pods -l app=nginx

```
vagrant@master:/home/nginx$ kubectl get pods -l app=nginx
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-7c5ddbdf54-h8qqb   1/1     Running   0          5m9s
nginx-deployment-7c5ddbdf54-vhmlz   1/1     Running   0          5m9s
nginx-pod                           1/1     Running   0          45m
```

- Connexion sur le **premier** pod : kubectl exec -it nginx-deployment-7c5ddbdf54-h8qqb -- /bin/bash

```
vagrant@master:/home/nginx$ kubectl exec -it nginx-deployment-7c5ddbdf54-h8qqb -- /bin/bash
root@nginx-deployment-7c5ddbdf54-h8qqb:/#
```

Modifier /usr/share/nginx/html/index.html avec echo "Replica 1" > /usr/share/nginx/html/index.html

```
root@nginx-deployment-7c5ddbdf54-h8qqb:/# echo "Replica 1" > /usr/share/nginx/html/index.html
root@nginx-deployment-7c5ddbdf54-h8qqb:/# cat /usr/share/nginx/html/index.html
Replica 1
```

- Connexion sur le **second** pod : kubectl exec -it nginx-deployment-7c5ddbdf54-vhmlz -- /bin/bash

```
vagrant@master:/home/nginx$ kubectl exec -it nginx-deployment-7c5ddbdf54-vhmlz -- /bin/bash
root@nginx-deployment-7c5ddbdf54-vhmlz:/#
```

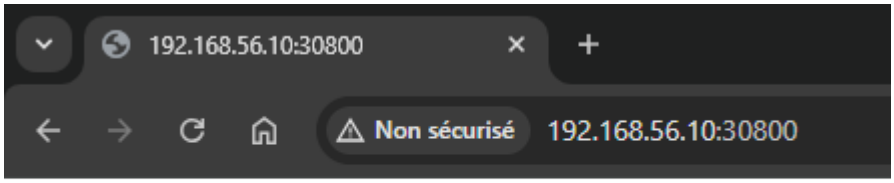
Modifier /usr/share/nginx/html/index.html avec echo "Replica 2" > /usr/share/nginx/html/index.html

```
root@nginx-deployment-7c5ddbdf54-vhmlz:/# echo "Replica 2" > /usr/share/nginx/html/index.html
root@nginx-deployment-7c5ddbdf54-vhmlz:/# cat /usr/share/nginx/html/index.html
Replica 2
```

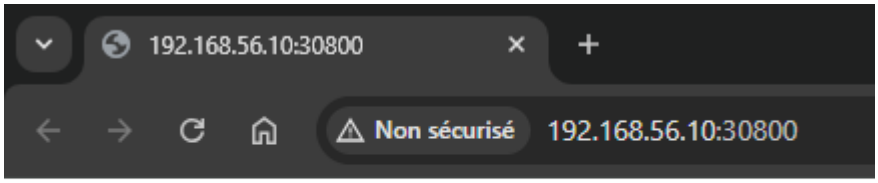
4.6. VERIFIER LE SCALING EN RAFRAICHISSANT

Le service NodePort load balance sur les deux réplicas.

En rafraîchissant, on alterne entre "**Replica 1**" et "**Replica 2**".



Replica 1



Replica 2

4.7. EXPORTER LA SORTIE DE KUBECTL DESCRIBE DEPLOY

Lancement de la commande : kubectl describe deploy nginx-deployment > nginx-deployment.txt

Ce déploiement Kubernetes appelé nginx-deployment gère 2 pods exécutant l'image nginx dans l'espace de noms default.

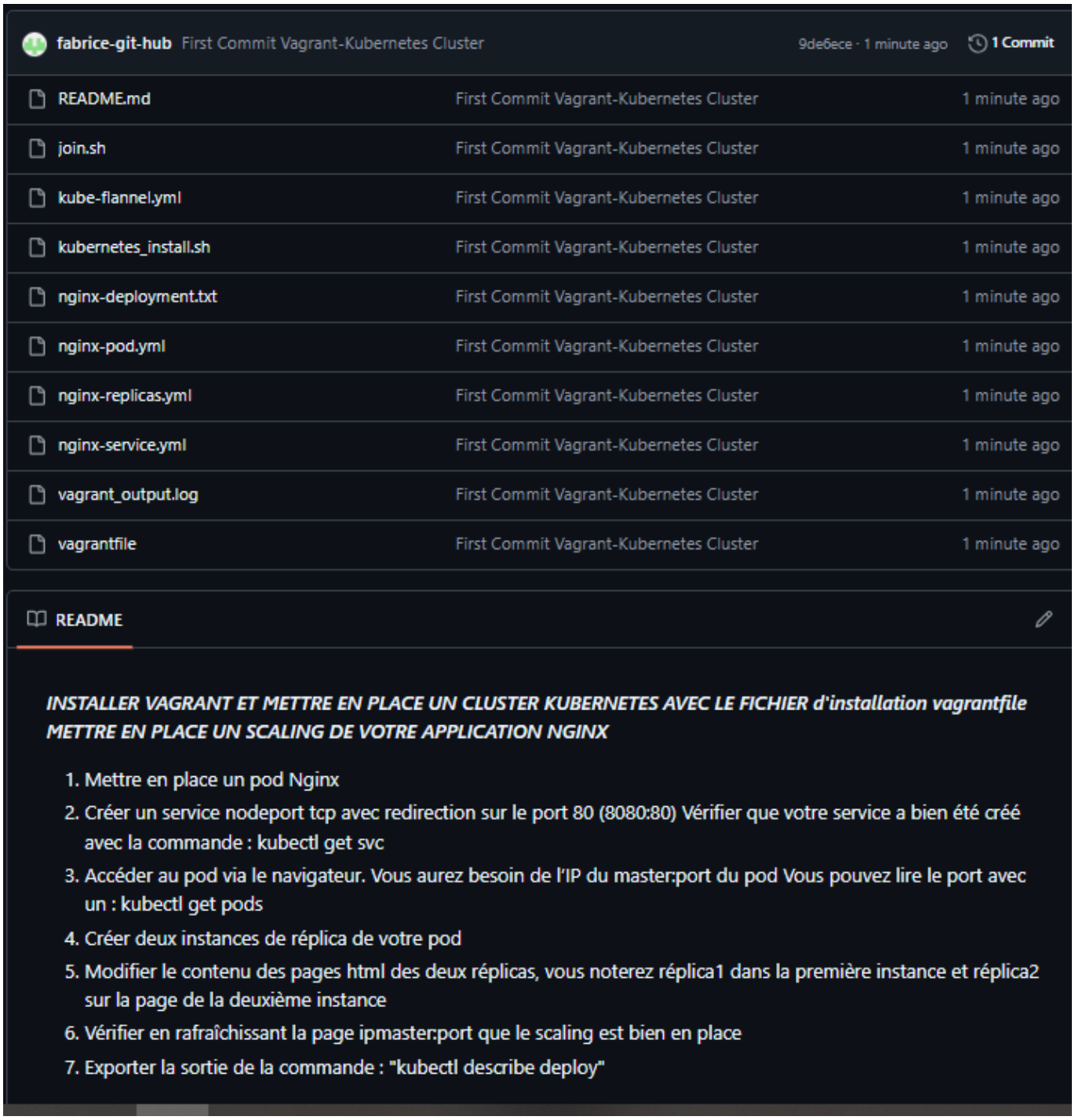
Il utilise une mise à jour progressive qui assure que l'application reste disponible pendant la mise à jour.

Le déploiement est stable, avec ses deux pods en ligne et prêts à servir du trafic HTTP sur le port 80.

```
vagrant@master:/tmp$ cat nginx-deployment.txt
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Fri, 20 Jun 2025 19:14:41 +0000
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-7c5ddbdf54 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  25m   deployment-controller  Scaled up replica set nginx-deployment-7c5ddbdf54 to 2
```

5. COMMIT VERS GITHUB

COMMIT vers https://github.com/fabrice-git-hub/ec8.git



6. ANNEXE

6.1. ENVIRONNEMENT KUBERNETES

a. vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64" # "ubuntu/focal64" refers to Ubuntu 20.04 LTS
(Focal Fossa) 64-bit.

# Master Node
  config.vm.define "k8s-master" do |master|
    master.vm.hostname = "k8s-master"
    master.vm.network "private_network", type: "static" , ip: "192.168.56.10" #
Configures a private network for the "k8s-master" VM.
      master.vm.hostname = "master"
    master.vm.provider "virtualbox" do |v|
        v.name = "master"
      v.memory = 2048
      v.cpus = 2
    end
    master.vm.provision "shell", path: "kubernetes_install.sh", args: ["master"]
# Configures a shell script provisioner for the "master" VM.
  end

# Worker Node
  config.vm.define "k8s-node" do |node|
    node.vm.hostname = "k8s-node"
    node.vm.network "private_network", type: "static" , ip: "192.168.56.11" #
Configures a private network for the "k8s-node" VM
      node.vm.hostname = "node"
    node.vm.provider "virtualbox" do |v|
        v.name = "node"
      v.memory = 2048
      v.cpus = 2
    end
    node.vm.provision "shell", path: "kubernetes_install.sh", args: ["node"] #
Configures a shell script provisioner for the "node" VM.
  end
end
```

b. kubernetes_install.sh

```
#!/bin/bash

ROLE=$1
HOSTNAME=$(hostname)

# Install Docker
sudo apt-get update && sudo apt-get install -y docker.io
sudo systemctl enable docker && sudo systemctl start docker

# Install Kubernetes tools
sudo apt-get install -y apt-transport-https curl gpg # apt-transport-https enables secure
connections for the package manager
sudo mkdir -p /etc/apt/keyrings

sudo rm -f /etc/apt/keyrings/kubernetes-archive-keyring.gpg

# These two lines allow to trust the Kubernetes packages and know where to download them from.
    curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
    echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

# kubeadm: the command to bootstrap the cluster.
# kubelet: the component that runs on all of the machines in your cluster and does things like
starting pods and containers.
# kubectl: the command line util to talk to your cluster.
sudo apt-get update && sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

# Disable swap (MUST be disable in order for the kubelet to work properly)
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab

# Master configuration
# 192.168.56.10 - IP address that the API Server will advertise to other nodes and clients
# 10.244.0.0 - IP address range for the Pod network, especially when using Flannel as the
Container Network Interface (CNI)

if [ "$ROLE" = "master" ]; then
# 10.244.0.0/16 range typically forms a network overlay on top of the existing host network
(192.168.56.x)
  kubeadm init --apiserver-advertise-address=192.168.56.10 --pod-network-cidr=10.244.0.0/16

  mkdir -p /home/vagrant/.kube
  sudo cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config # Allowing the vagrant user
to interact with the Kubernetes cluster
  sudo chown vagrant:vagrant /home/vagrant/.kube/config

# Install Flannel CNI(Container Network Interface) plugin in Kubernetes cluster
  su - vagrant -c "kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml"

# Save the join command (generate and securely share the command that worker nodes will use to
join the cluster)

  kubeadm token create --print-join-command > /vagrant/join.sh
```

```
elif [ "$ROLE" = "node" ]; then
  while [ ! -f /vagrant/join.sh ]; do sleep 5; done
  cp /vagrant/join.sh /tmp/join.sh
  chmod +x /tmp/join.sh
  /tmp/join.sh # Allow to connect to and join the Kubernetes cluster
fi
```

c. Configuration IP

- Master node

```
vagrant@master:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:ca:62:ef:6a:24 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 80430sec preferred_lft 80430sec
    inet6 fd00::ca:62ff:feef:6a24/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86399sec preferred_lft 14399sec
    inet6 fe80::ca:62ff:feef:6a24/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:9d:ab:8f brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe9d:ab8f/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:51:bb:45:e2 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
5: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 06:7c:93:e2:8f:fd brd ff:ff:ff:ff:ff:ff
    inet 10.244.0.0/32 scope global flannel.1
       valid_lft forever preferred_lft forever
    inet6 fe80::47c:93ff:fee2:8ffd/64 scope link
       valid_lft forever preferred_lft forever
6: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    link/ether 52:c9:60:0d:e1:c1 brd ff:ff:ff:ff:ff:ff
    inet 10.244.0.1/24 brd 10.244.0.255 scope global cni0
       valid_lft forever preferred_lft forever
    inet6 fe80::50c9:60ff:fe0d:e1c1/64 scope link
       valid_lft forever preferred_lft forever
7: veth1cd4dc33@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master cni0 state UP group default
    link/ether 3e:51:a1:bb:ee:a7 brd ff:ff:ff:ff:ff:ff link-netns cni-45ed3037-1cc5-a95a-18b6-40ec124b7721
    inet6 fe80::3c51:a1ff:febb:eea7/64 scope link
       valid_lft forever preferred_lft forever
8: vetheebb27e9@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master cni0 state UP group default
    link/ether da:6b:09:2f:e4:3b brd ff:ff:ff:ff:ff:ff link-netns cni-9f2a7f45-23d9-c18b-ace5-974b3693d357
    inet6 fe80::d86b:9ff:fe2f:e43b/64 scope link
       valid_lft forever preferred_lft forever
```

- Worker node

```
vagrant@node:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:ca:62:ef:6a:24 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 80495sec preferred_lft 80495sec
    inet6 fd00::ca:62ff:feef:6a24/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86347sec preferred_lft 14347sec
    inet6 fe80::ca:62ff:feef:6a24/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:6d:ef:0c brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.11/24 brd 192.168.56.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6d:ef0c/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:21:30:0c:a6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
5: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 9e:4c:22:b2:7d:89 brd ff:ff:ff:ff:ff:ff
    inet 10.244.1.0/32 scope global flannel.1
       valid_lft forever preferred_lft forever
    inet6 fe80::9c4c:22ff:feb2:7d89/64 scope link
       valid_lft forever preferred_lft forever
```

6.2. TROUBLESHOOTING

a. Dépôt Kubernetes

Lors du lancement du script de provisionnemment les URLs suivantes ont été renseignées pour configurer le dépôt kuberbetes sur le master et le worker mais au lancement du "vagrant up", l'erreur suivante est donné en retour :

curl -s **https://packages.cloud.google.com**/apt/doc/apt-key.gpg | apt-key add -

echo "deb **https://apt.kubernetes.io/ kubernetes-xenial main**" > /etc/apt/sources.list.d/kubernetes.list

```
k8s-master: Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
k8s-master: Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease
k8s-master: Hit:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease
k8s-master: Hit:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease
k8s-master: Ign:4 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
k8s-master: Err:6 https://packages.cloud.google.com/apt kubernetes-xenial Release
k8s-master:   404  Not Found [IP: 142.250.75.238 443]
k8s-master: Reading package lists...
k8s-master: E: The repository 'https://apt.kubernetes.io kubernetes-xenial Release' does not have a Release file.
k8s-master: E: Unable to locate package kubelet
k8s-master: E: Unable to locate package kubeadm
k8s-master: E: Unable to locate package kubectl
k8s-master: E: No packages found
```

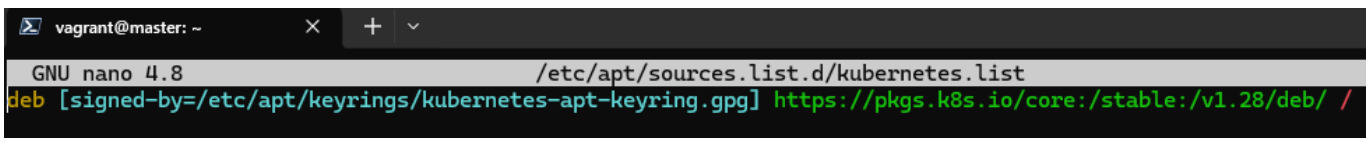
- Cela indique que le dépôt https://packages.cloud.google.com/apt avec la distribution kubernetes-xenial est introuvable (erreur 404).
- Ubuntu ne peut pas utiliser ce dépôt, car il n'y a pas de fichier Release, ce qui est requis pour la validation et la sécurité.
- Important : Le nom kubernetes-xenial est volontairement conservé par Google, même si Ubuntu Focal est utilisé. Ce n'est pas une erreur. Ce qui est important, c'est de récupérer la bonne clé GPG et que le dépôt soit accessible.

Il convient d'utiliser le dépôt suivant : https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key

curl -fsSL **https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key** | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] **https://pkgs.k8s.io/core:/stable:/v1.28/deb/** /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

```
vagrant@master:~$ ls /etc/apt/keyrings
kubernetes-apt-keyring.gpg
```



- Connexion au dépôt et récupération des packages :

```
k8s-master: deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /
k8s-master: Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
k8s-master: Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
k8s-master: Hit:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease
k8s-master: Hit:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease
k8s-master: Get:3 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.28/deb  InRelease [1192 B]
k8s-master: Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.28/deb  Packages [21.3 kB]
k8s-master: Fetched 22.5 kB in 2s (10.6 kB/s)
k8s-master: Reading package lists...
k8s-master: Reading package lists...
k8s-master: Building dependency tree...
k8s-master: Reading state information...
k8s-master: The following additional packages will be installed:
k8s-master:   conntrack cri-tools kubernetes-cni
k8s-master: Suggested packages:
k8s-master:   nftables
k8s-master: The following NEW packages will be installed:
k8s-master:   conntrack cri-tools kubeadm kubectl kubelet kubernetes-cni
k8s-master: 0 upgraded, 6 newly installed, 0 to remove and 3 not upgraded.
```

b. Internal-IP Kubernetes

Au lancement de la commande suivante : kubectl get nodes -o wide

```
vagrant@master:~$ kubectl get nodes -o wide
NAME     STATUS   ROLES           AGE   VERSION    INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
master   Ready    control-plane   2d    v1.28.15   10.0.2.15     <none>        Ubuntu 20.04.6 LTS   5.4.0-216-generic
node     Ready    <none>          2d    v1.28.15   10.0.2.15     <none>        Ubuntu 20.04.6 LTS   5.4.0-216-generic
```

- On observe un problème important : les deux nœuds (master et node) ont la même adresse IP interne (10.0.2.15), ce qui ne devrait pas arriver dans un cluster Kubernetes correctement configuré.
  - 10.0.2.15 est une IP fournie par VirtualBox en mode NAT, attribuée par défaut à toutes les machines.
  - Cela cause un conflit dans Kubernetes, car chaque nœud doit avoir une IP interne unique pour le réseau du cluster.
- Le vagrantfile a pourtant été configuré pour utiliser un réseau privé.

```
# Master Node
config.vm.define "k8s-master" do |master|
  master.vm.hostname = "k8s-master"
  master.vm.network "private_network", ip: "192.168.56.10"

# Worker Node
config.vm.define "k8s-node" do |node|
  node.vm.hostname = "k8s-node"
  node.vm.network "private_network", ip: "192.168.56.11"
```

- Par défaut, kubelet choisit la première interface réseau qu'il détecte.
  - Dans une VM VirtualBox, eth0 est souvent en NAT (IP: 10.0.2.15), et le réseau privé (eth1) n'est pas utilisé par kubelet à moins qu'on le lui précise.
  - Kubernetes a pourtant été configuré pour utiliser l'IP privée
    kubeadm init --apiserver-advertise-address=**192.168.56.10** --pod-network-cidr=10.244.0.0/16
  - L'interface réseau sur le master et le worker est la suivante : enp0s8

```
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_co
    link/ether 08:00:27:9d:ab:8f brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global enp0s8
```

```
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_co
    link/ether 08:00:27:6d:ef:0c brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.11/24 brd 192.168.56.255 scope global enp0s8
```

- **Etapes de correction**
  1. **<u>Configurer l'IP privée dans kubelet</u>**

✓ Sur chaque nœud (master, node), il faut ajouter dans le fichier : sudo nano /etc/default/kubelet

```
KUBELET_EXTRA_ARGS=--node-ip=192.168.56.10   # sur le master

KUBELET_EXTRA_ARGS=--node-ip=192.168.56.11   # sur le worker
```

✓ Puis redémarrer le service :

```
sudo systemctl daemon-reexec

sudo systemctl restart kubelet
```

2. **<u>Supprimer et réinstaller Flannel</u> (UNIQUEMENT sur le master)**

✓ Supprimession de l'ancien réseau flannel :

```
kubectl delete -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

```
vagrant@master:~$ kubectl delete -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace "kube-flannel" deleted
clusterrole.rbac.authorization.k8s.io "flannel" deleted
clusterrolebinding.rbac.authorization.k8s.io "flannel" deleted
serviceaccount "flannel" deleted
configmap "kube-flannel-cfg" deleted
daemonset.apps "kube-flannel-ds" deleted
```

- ✓ Modification du YAML de Flannel pour forcer l'interface enp0s8
  - Téléchargez localement le manifeste

wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```
vagrant@master:~$ wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
--2025-06-20 14:12:21--  https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4348 (4.2K) [text/plain]
Saving to: 'kube-flannel.yml'

kube-flannel.yml                    100%[=======================================================================

2025-06-20 14:12:22 (7.58 MB/s) - 'kube-flannel.yml' saved [4348/4348]
```

  - Dans ce fichier, il faut repérer la section du **kube-flannel container** et modifiez la ligne de commande en ajoutant : **- --iface=enp0s8**

```
containers:
- name: kube-flannel
  image: ghcr.io/flannel-io/flannel:v0.27.0
  command:
  - /opt/bin/flanneld
  args:
  - --ip-masq
  - --kube-subnet-mgr
  - --iface=enp0s8
```

3. **Lancer la commande :** kubectl apply -f kube-flannel.yml **(UNIQUEMENT sur le master)**

```
vagrant@master:~$ kubectl apply -f kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

4. **Vérification à nouveau des annotations** : kubectl describe node master

```
vagrant@master:~$ kubectl describe node master
Name:               master
Roles:              control-plane
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=master
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=
                    node.kubernetes.io/exclude-from-external-load-balance
Annotations:        flannel.alpha.coreos.com/backend-data: {"VNI":1,"Vtep
                    flannel.alpha.coreos.com/backend-type: vxlan
                    flannel.alpha.coreos.com/kube-subnet-manager: true
                    flannel.alpha.coreos.com/public-ip: 192.168.56.10
```

```
Addresses:
  InternalIP:  192.168.56.10
  Hostname:    master
```

Relance de la commande suivante : kubectl get nodes -o wide

```
vagrant@master:~$ kubectl get nodes -o wide
NAME     STATUS   ROLES           AGE    VERSION    INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
master   Ready    control-plane   2d2h   v1.28.15   192.168.56.10   <none>        Ubuntu 20.04.6 LTS
node     Ready    <none>          2d1h   v1.28.15   192.168.56.11   <none>        Ubuntu 20.04.6 LTS
```