Rapport de Projet : VM & Kubernetes Réalisé par : SALMA EZZAIMI

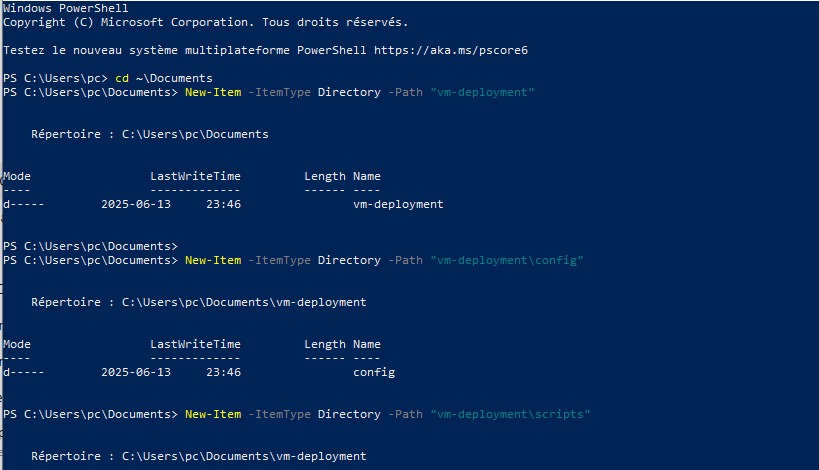
IMANE HADI

# Introduction

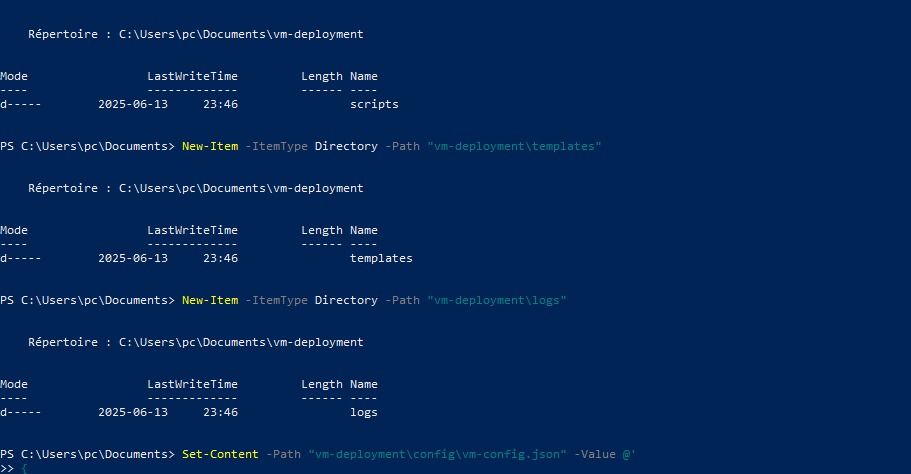
Ce document décrit deux initiatives d'automatisation destinées à optimiser la gestion d'une infrastructure IT. Le premier porte sur l'automatisation du déploiement de machines virtuelles (VM) via PowerShell et PowerCLI, en se servant de fichiers de configuration pour adapter les ressources. Le second projet concerne la mise en place d'un cluster Kubernetes (1 maître, 2 nœuds de travail) sur Ubuntu 20.04, utilisant kubeadm et un réseau CNI (Flannel), puis le déploiement d'une application NGINX pour vérification.  
  
Ces initiatives favorisent la compréhension des fondamentaux de l'automatisation des systèmes, la conception de scripts réutilisables et la validation du bon fonctionnement d'un environnement virtuel à la pointe de la technologie.

# Etape 1 : Preparation du projet PowerShell pour le deploiement de VM

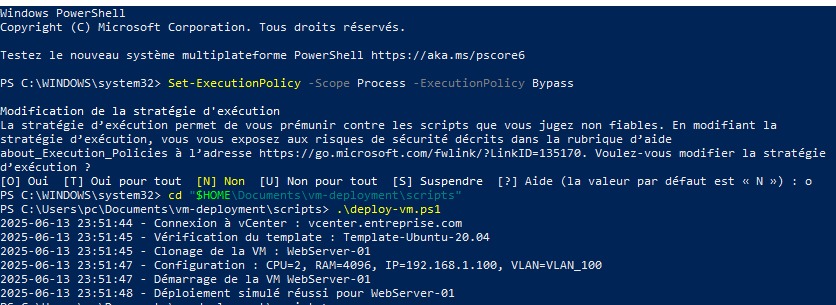
La mise en place du répertoire principal vm-deployment permet de centraliser tous les fichiers du projet en un lieu unique. Plusieurs sous-dossiers sont générés à l'intérieur :  
  
**Scripts** pour la conservation des scripts PowerShell d'automatisation,  
  
**Configuration** pour les fichiers JSON de configuration qui décrivent les spécifications des machines virtuelles à mettre en place.  
  
**logs** pour consigner les journaux d'exécution afin de faciliter le suivi et le débogage.



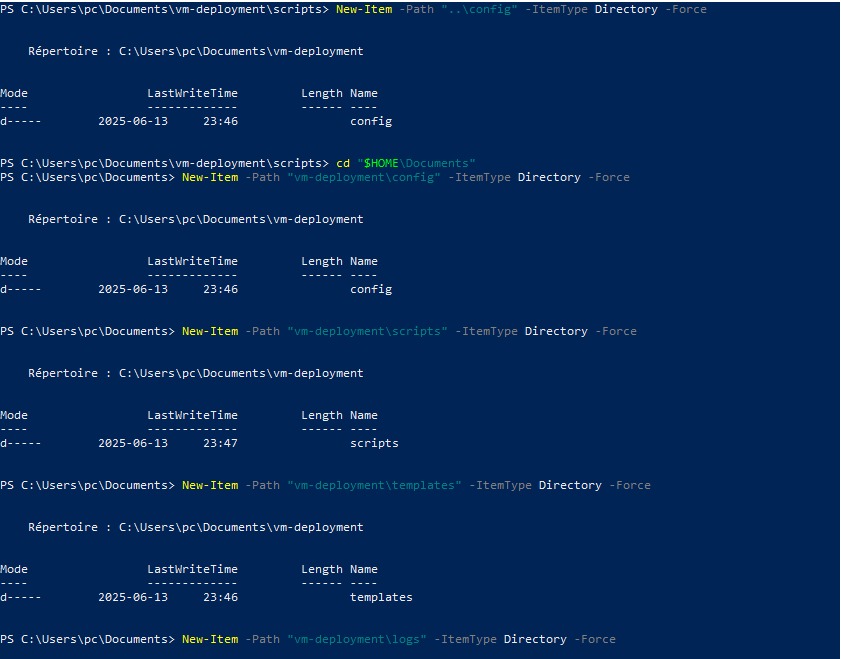
Le fichier *vm-config.json* renferme l'ensemble des configurations requises pour la mise en place d'une machine virtuelle. Il précise en particulier :  
  
Le nom de la machine virtuelle (*vm\_name*),  
L'IP fixe à assigner,  
La quantité de processeurs (CPU),  
le volume de mémoire vive,  
la dimension du disque,  
Le réseau à employer est le VLAN.  
et le modèle de base à copier.



Le fichier *vm-config2.json* offre la possibilité de mettre en place une seconde VM utilisant des configurations sur mesure. Il permet de créer plusieurs machines virtuelles en utilisant le même script, simplement en modifiant le fichier de configuration.

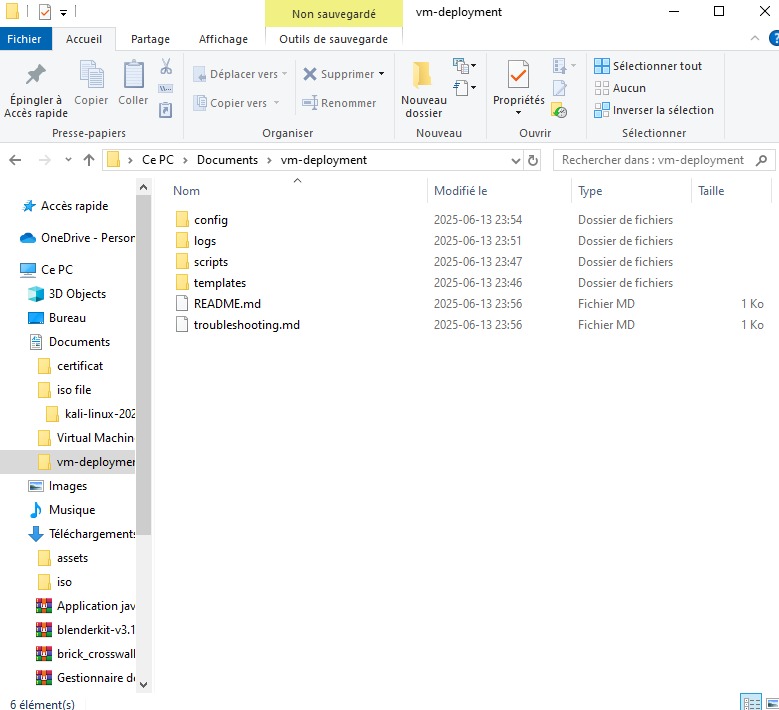


Le fichier de modèle prédéfini est utilisé comme référence pour la création des machines virtuelles. Ce dernier inclut une image système prête à l'utilisation (comme Ubuntu configuré) que le script se chargera de répliquer, dans le but d'assurer une base uniforme et normalisée pour toutes les machines virtuelles mises en place.

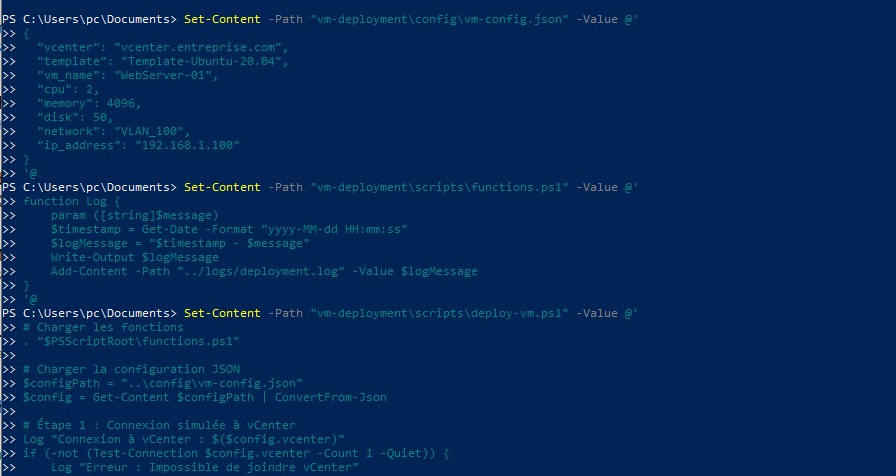


# Etape 2 : Simulation du deploiement d'une VM

L’exécution du script deploy-vm.ps1 dans le terminal lance le processus d’automatisation du déploiement. Le script lit les paramètres du fichier JSON, se connecte à vCenter, vérifie le template, puis simule ou effectue la création de la VM selon les instructions définies.

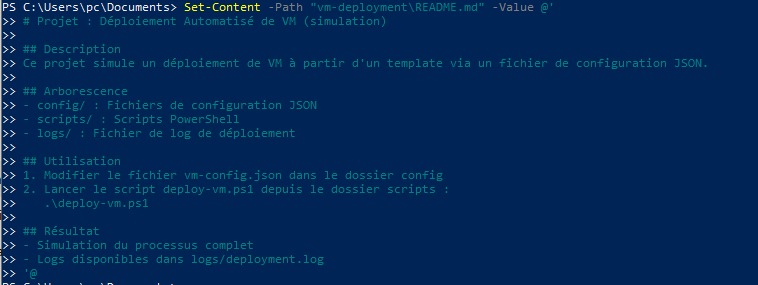


Le fichier functions.ps1 contient des fonctions réutilisables, notamment une fonction de **log** qui enregistre chaque étape du déploiement dans un fichier journal (deployment.log). Cela permet de tracer les actions effectuées (ex : connexion à vCenter, lecture du fichier JSON, clonage), ce qui facilite le suivi, le débogage et la maintenance du script.



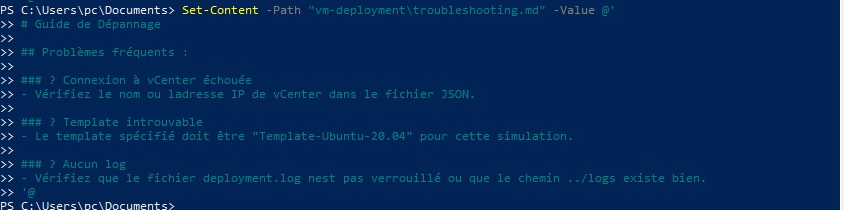
# Etape 3 : Structure du dossier final dans l'explorateur Windows

L'image illustre la structure bien organisée du dossier `*vm-deployment*`, qui inclut des sous-dossiers pour les scripts, les configurations, les journaux et les modèles, simplifiant ainsi la gestion du projet.

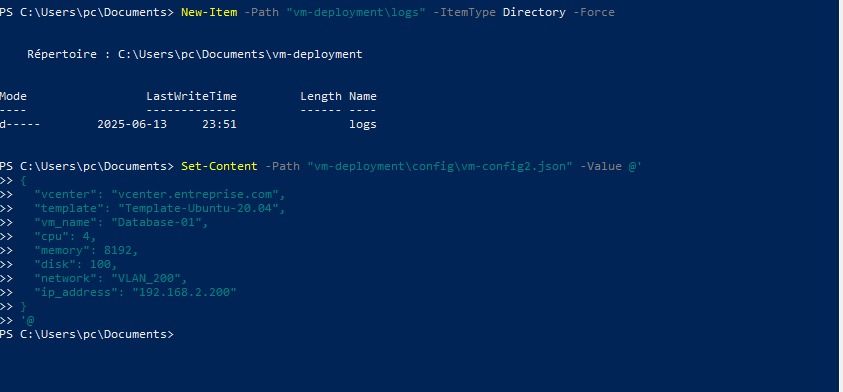


# Etape 4 : Documentation du projet VM

Le fichier `README.md` offre des directives précises pour l'emploi du script de déploiement. Il détaille comment élaborer le fichier de configuration, mettre en marche le script `deploy-vm.ps1`, et où trouver les journaux. Ce manuel accompagne l'utilisateur dans la compréhension et l'utilisation indépendante du projet.

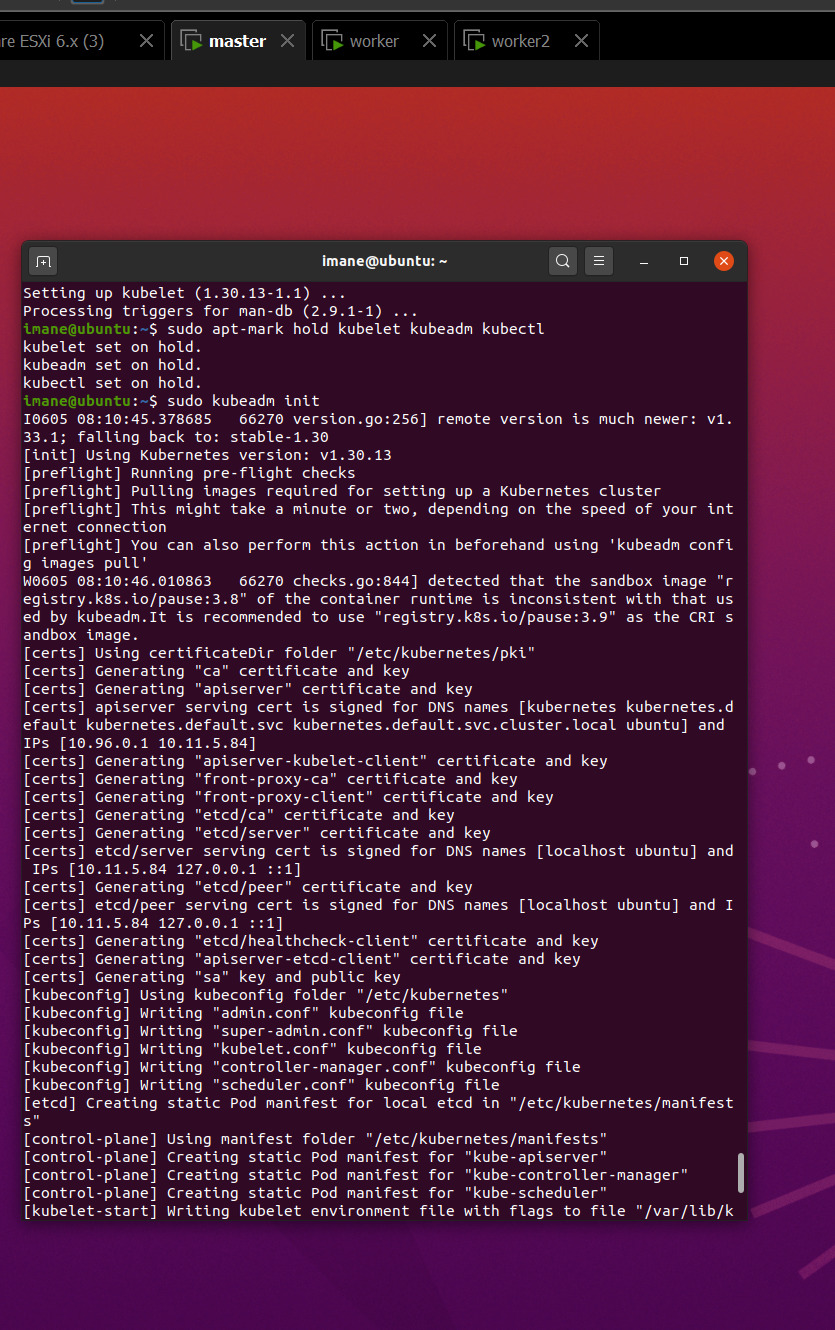


Le document `troubleshooting.md` énumère les problèmes fréquents qui surviennent lors de l'exécution du script (par exemple : échec de la connexion à vCenter, modèle non trouvé) et suggère des résolutions ou des contrôles à réaliser. C'est un manuel rapide pour identifier et rectifier les problèmes sans nécessiter de modification du code.

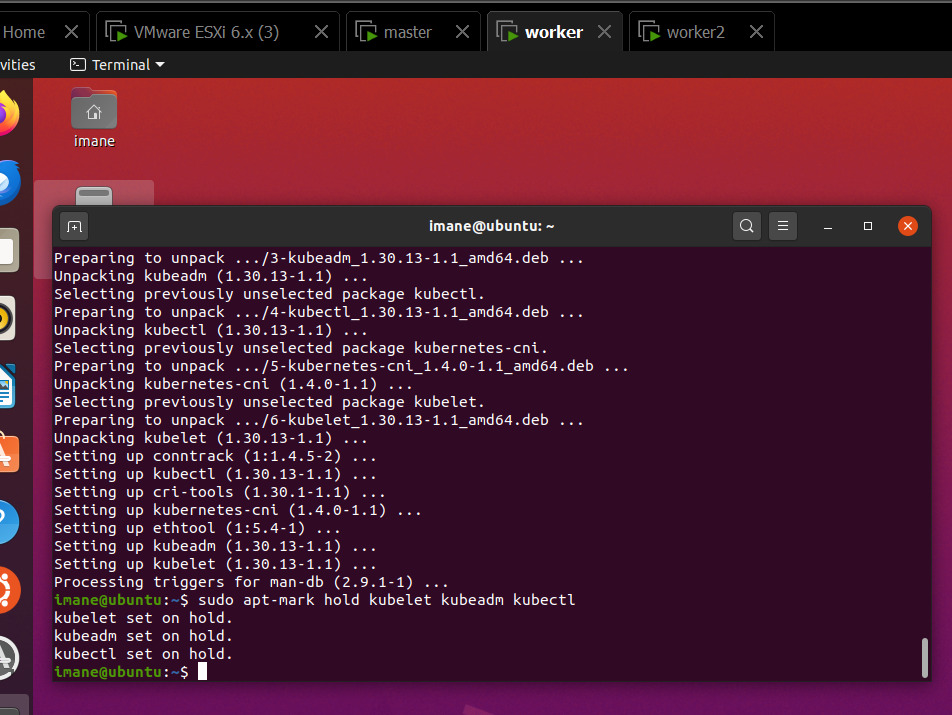


# Etape 5 : Installation du cluster Kubernetes – Master Node

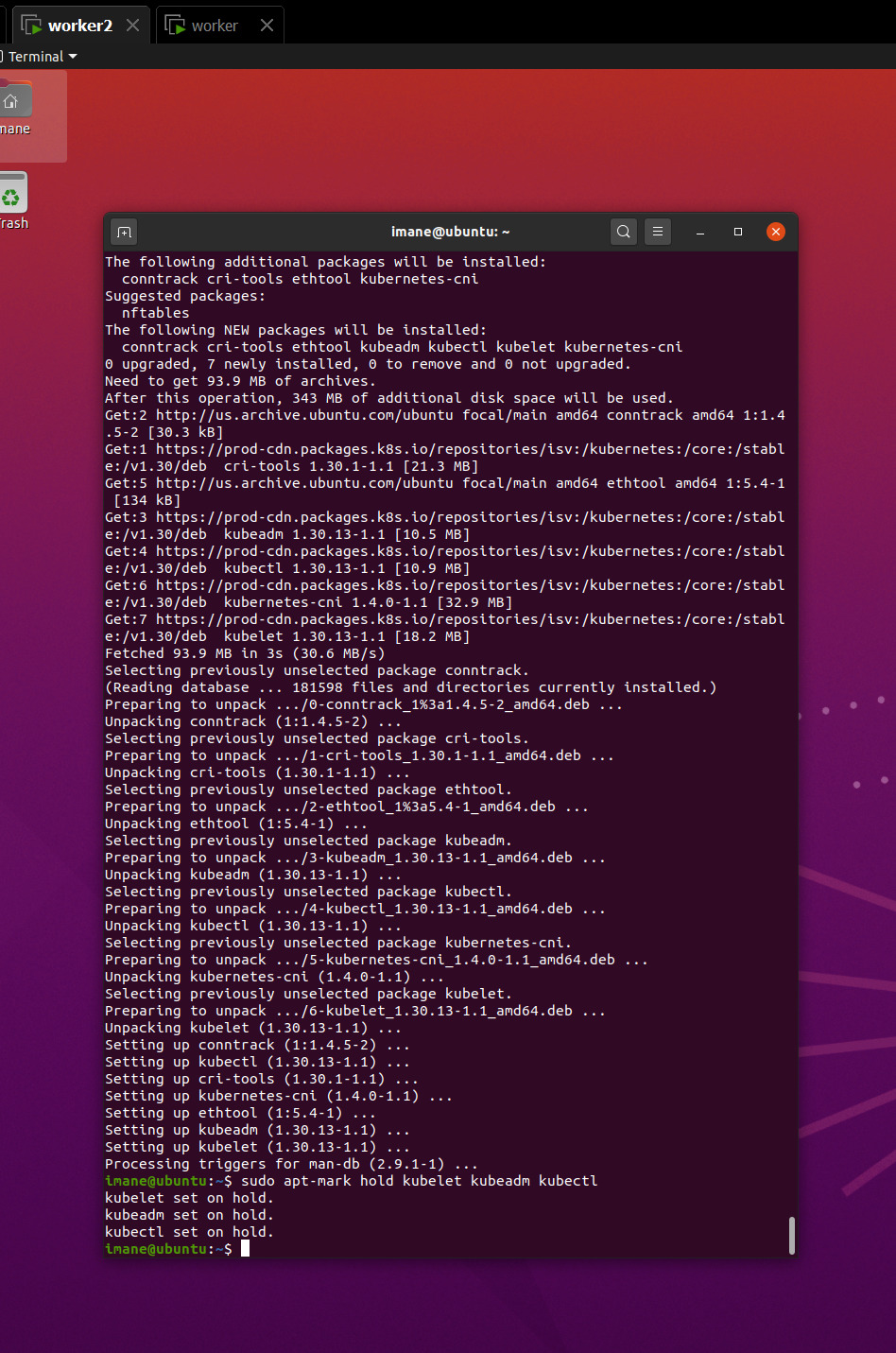
Le script `*install-master.sh*` est lancé sur le nœud principal afin d'automatiser l'installation des éléments clés de Kubernetes (`*kubeadm*`, `*kubelet*`, `*kubectl*`). Il commence par initialiser le cluster avec `kubeadm init`, désactive le swap, met en place l'accès à `kubectl`, et déploie Flannel pour faciliter la communication entre les pods.



Cette étape lance la commande kubeadm init sur le nœud master pour créer le plan de contrôle du cluster Kubernetes. Elle génère les fichiers de configuration nécessaires et fournit une commande kubeadm join que les nœuds workers utiliseront pour rejoindre le cluster.

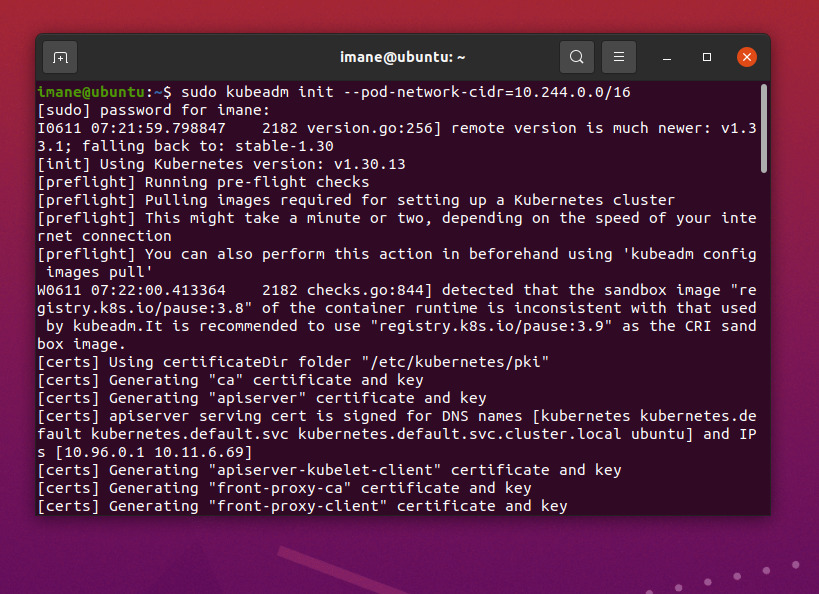


Une fois le master initialisé, la configuration du réseau du cluster se fait en utilisant un plugin CNI (tel que Flannel) via `kubectl apply`. Cette opération facilite la communication entre les pods. Après l'installation du réseau, le cluster devient fonctionnel et se trouve en condition d'accueillir les nœuds de travail.

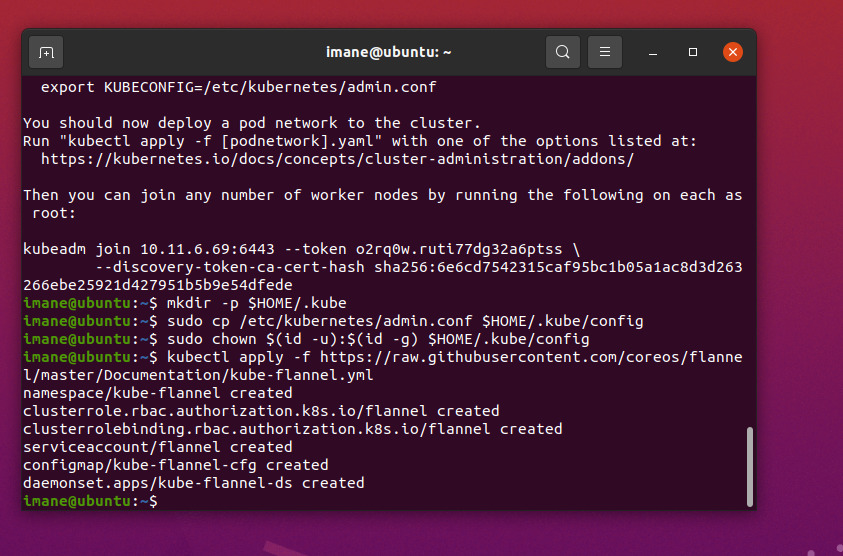


# Etape 6 : Integration des Workers Nodes au cluster

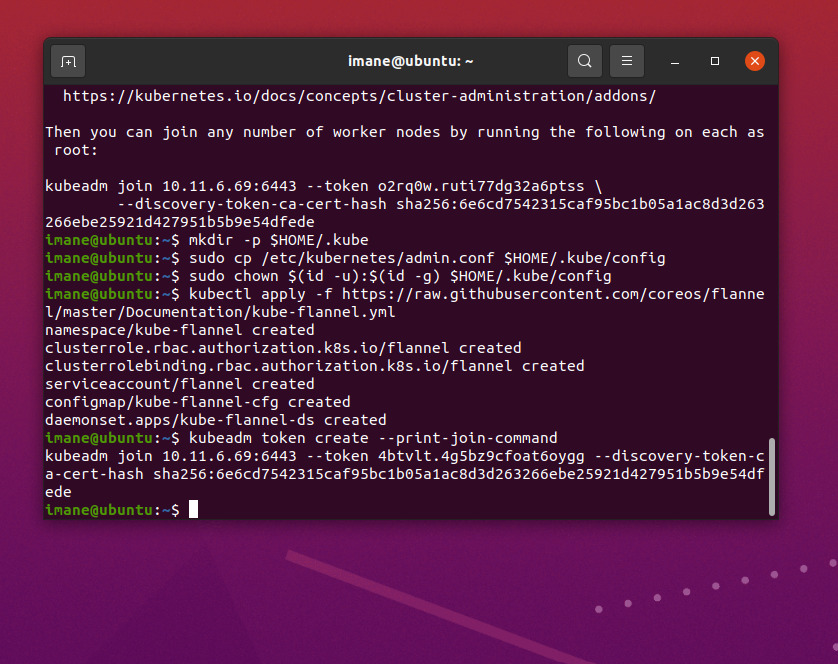
La commande `kubeadm join`, qui est créée sur le master suite à l'initialisation, est ensuite copiée et exécutée sur le nœud worker 1. Elle autorise l'ouvrier à se connecter au cluster en s'identifiant et en s'inscrivant auprès du nœud principal.



On exécute la même commande `*kubeadm join*` sur le deuxième nœud worker, ce qui permet à celui-ci d'intégrer également le cluster Kubernetes. Cela établit une connexion entre le worker et le plan de contrôle, préparant ainsi les deux nœuds pour l'exécution de pods.

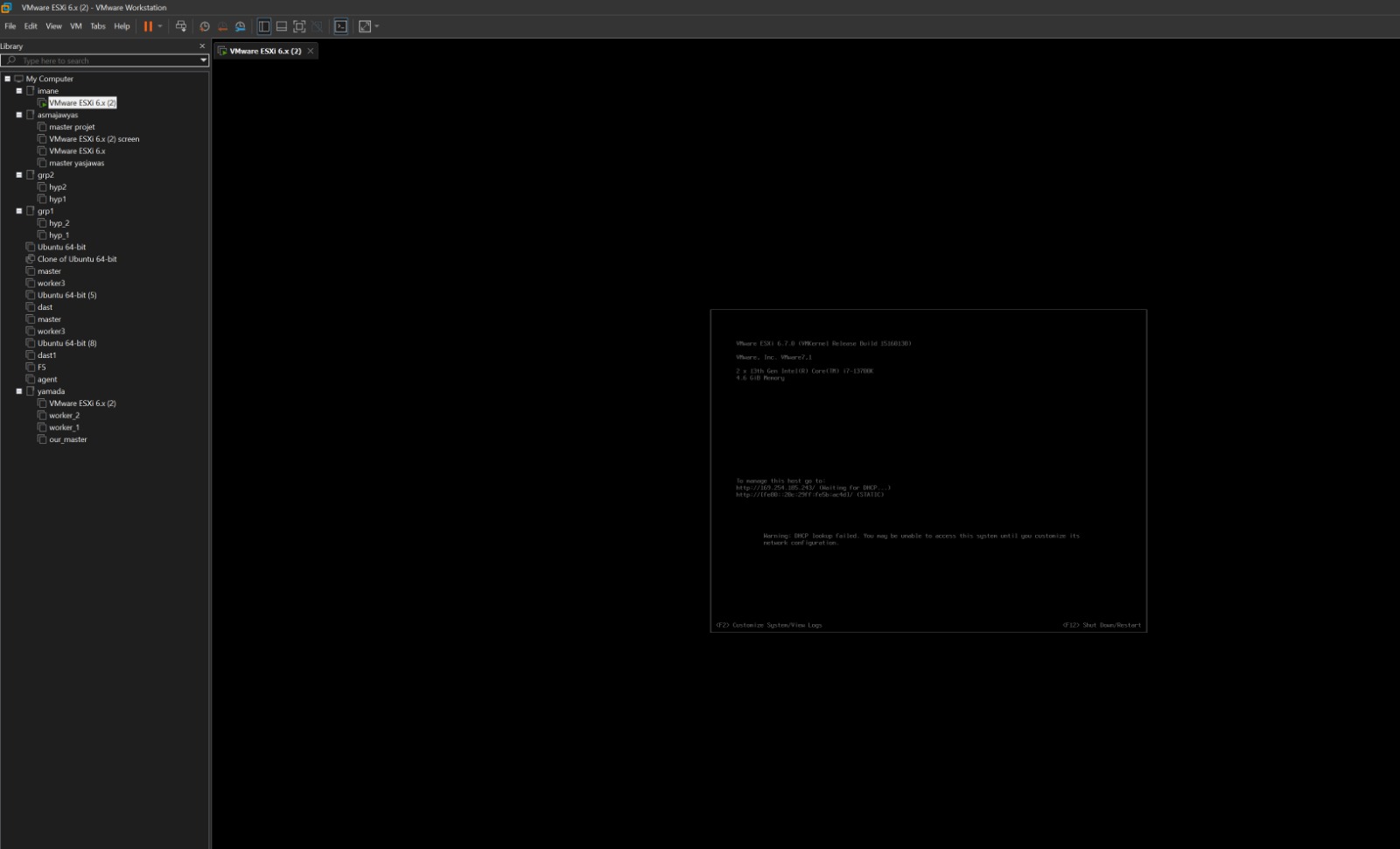


Après avoir lancé la commande `*kubeadm join`* sans rencontrer d'erreurs, le worker 2 a été correctement intégré au cluster. Lorsqu'on exécute la commande `*kubectl get nodes*` depuis le maître, il se manifeste désormais en tant que nœud actif et dans l'état « Ready ».



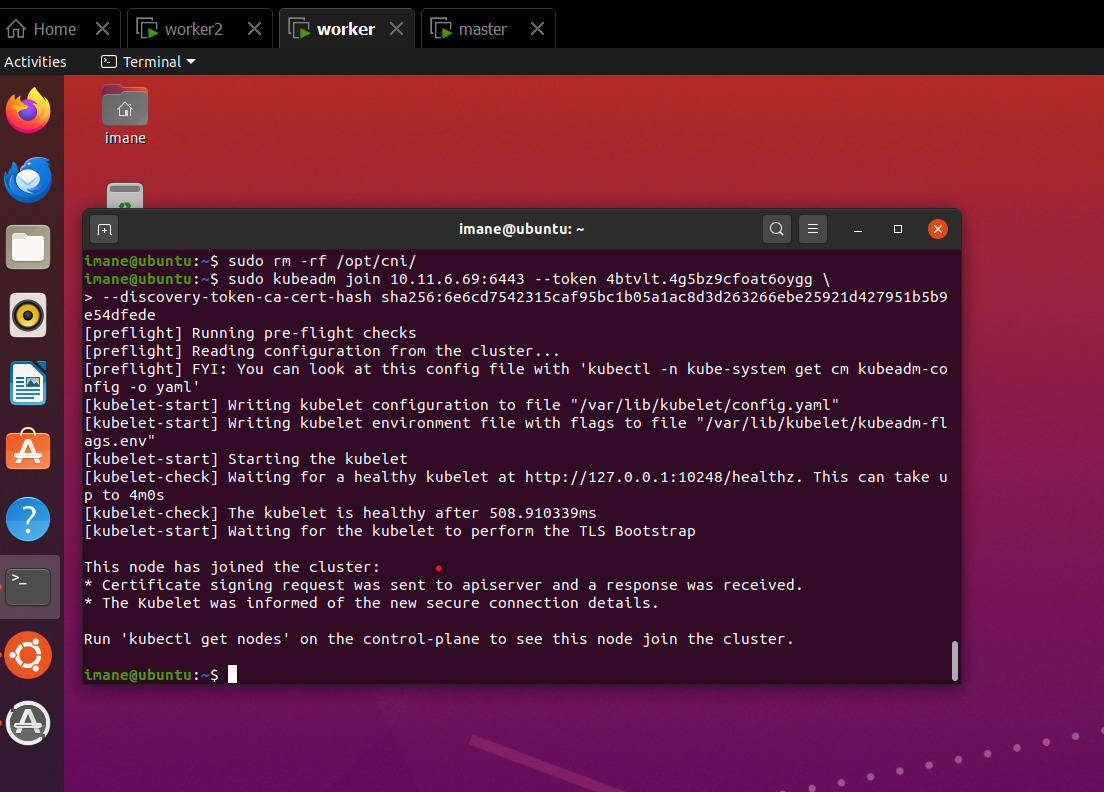
# Etape 7 : Deploiement de l'application de test NGINX

On emploie deux commandes pour vérifier le bon fonctionnement du cluster :   
  
1. `**kubectl create deployment nginx --image=nginx**` pour mettre en place un déploiement NGINX.  
2. `**kubectl expose deployment nginx --port=80 --type=NodePort**` permet de rendre ce service accessible sur un port externe.   
  
Cela permet de s'assurer que la répartition des pods est correcte et qu'ils sont accessibles sur le cluster.



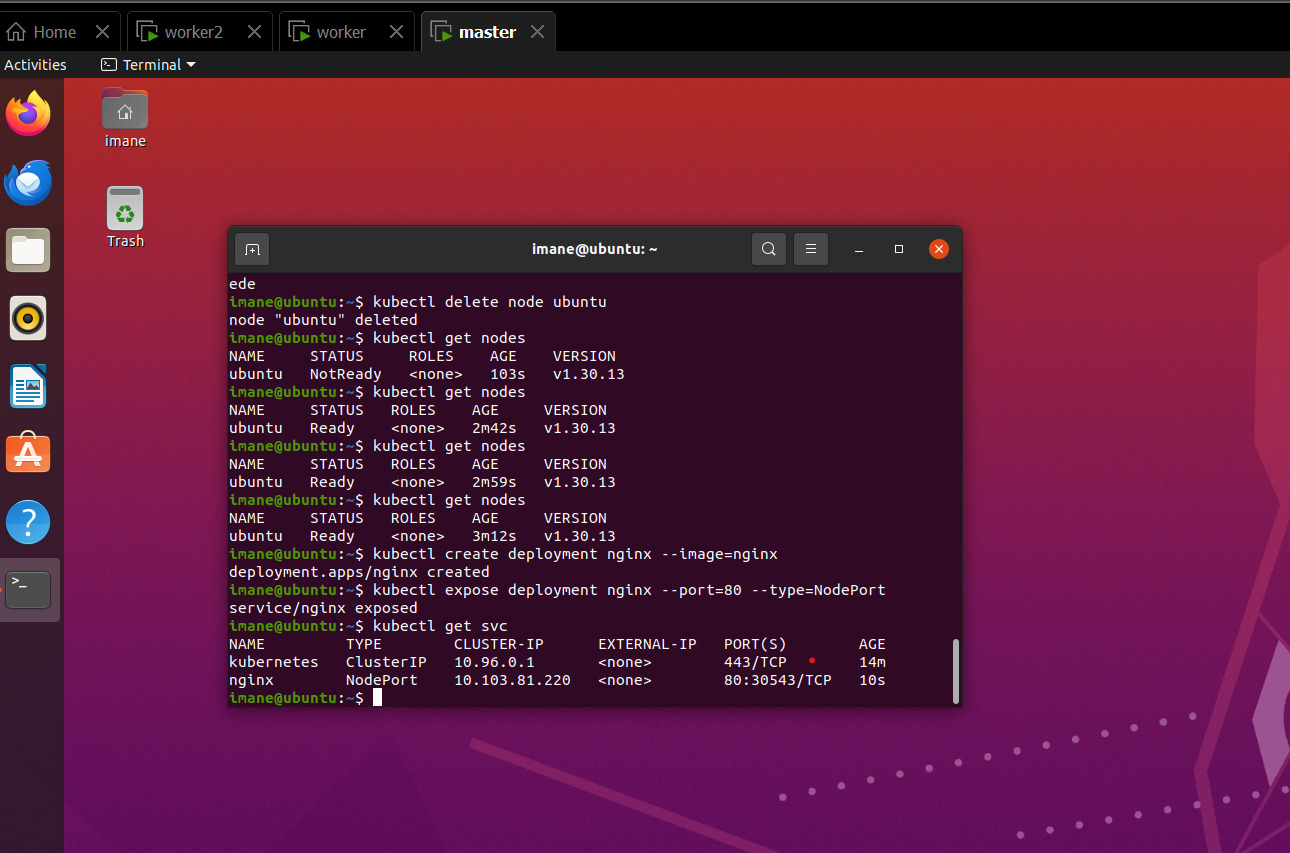
# Etape 8 : Resume des noeuds Kubernetes et validation du cluster

La commande `*kubectl get nodes*` sert à afficher tous les nœuds présents dans le cluster. Quand ils sont marqués avec l'état \*\*« Prêt »\*\*, cela signifie qu'ils sont bien intégrés, opérationnels et disposés à gérer des charges de travail (pods) au sein du cluster Kubernetes.



# Etape 9 : Vue d’ensemble des machines utilisees (VMware)

Cette phase présente toutes les machines virtuelles mises en œuvre pour constituer le cluster Kubernetes : une VM dédiée au nœud maître et deux autres pour les nœuds de travail. Toutes ces machines virtuelles utilisent Ubuntu 20.04 et sont paramétrées pour interagir les unes avec les autres à travers le réseau interne, ce qui offre la possibilité de reproduire un environnement de production.



# Conclusion

Ce projet a facilité l'acquisition des fondamentaux de l'automatisation dans les contextes VMware et Kubernetes. Le déploiement de machines virtuelles par le biais de PowerShell a été réalisé avec succès. Par la suite, la mise en place d'un cluster Kubernetes a confirmé l'efficacité de l'orchestration ainsi que le déploiement d'une application web basique (NGINX). Ce travail représente un socle robuste pour l'automatisation avancée de l'infrastructure informatique.