Projet DevOps : Déploiement WordPress sur Kubernetes

Titre professionnel visé : Administrateur Systèmes DevOps (ASD)

Nom : VARELA DURAN David Hugo

Centre : M2i Formation Date : Juin 2025

Remerciements

Je tiens à **remercier sincèrement toute l'équipe pédagogique de M2i Formation**, et en particulier mes **formateurs**, qui ont été présents à chaque étape de mon parcours. Leur patience, leur pédagogie et leur engagement m'ont profondément marqué. Ils ont su transmettre leur passion pour le métier avec une réelle bienveillance, et c'est grâce à eux que j'ai pu avancer,

gagner en confiance, et surmonter les nombreux défis techniques que j'ai rencontrés, notamment autour de **Kubernetes**, **Ansible**, **Jenkins** et des **outils de supervision**.

Je n'oublie pas mes **camarades de promotion**, avec qui j'ai partagé des moments forts, parfois intenses, mais toujours enrichissants. Leur entraide, leur bonne humeur et l'esprit d'équipe que nous avons su construire ensemble ont rendu cette aventure humaine encore plus belle.

Ce projet final représente bien plus qu'un exercice technique pour moi : c'est **l'aboutissement d'un vrai choix de vie**. Après plusieurs années passées dans un tout autre domaine, j'ai pris la décision de me reconvertir dans l'**administration systèmes et DevOps**. J'ai fait ce choix avec passion et détermination, en acceptant de tout reprendre à zéro, de sortir de ma zone de confort, et d'apprendre un nouveau métier. Cela n'a pas toujours été facile, mais **je n'ai jamais baissé les bras**.

Aujourd'hui, **je suis fier de ce que j'ai accompli**. Fier d'avoir mené ce projet jusqu'au bout, d'avoir mis en pratique tout ce que j'ai appris, et surtout, d'avoir prouvé que c'était possible. Si j'en suis là aujourd'hui, c'est aussi grâce au **soutien de mes proches**, et à l'**accompagnement bienveillant** de ceux qui ont cru en moi.

Dédicace

Je dédie ce travail à **ma famille**, à **ma fille**, et à **toutes les personnes qui m'ont soutenu**, de près ou de loin, dans cette aventure. Leur **patience**, leurs **encouragements** et leur **confiance** ont été essentiels tout au long de ma reconversion professionnelle.

Ils m'ont porté dans les moments de **doute**, de **fatigue** et de **remise en question**, et m'ont permis de garder le cap, même quand la charge de travail était importante.

À **ma fille**, je veux être un exemple de **persévérance** et de **résilience**, pour lui montrer qu'il est possible de repartir de zéro et de construire un avenir meilleur, avec passion et volonté.

À mes **proches**, je suis profondément reconnaissant pour leur **présence constante**, malgré mes absences et les sacrifices demandés par cette formation exigeante.

Ce projet est aussi un **hommage à ceux qui ont cru en moi**, alors que je me lançais dans un tout nouveau domaine, avec l'envie de progresser, de ne rien lâcher, et de m'ouvrir à de nouvelles perspectives.

Introduction

Ce projet s'inscrit dans le cadre de la **validation du Titre Professionnel Administrateur Systèmes DevOps (ASD)** que j'ai suivi au sein de **M2i Formation**. Il marque l'**aboutissement de plusieurs mois de formation intensive**, rythmés par des défis techniques, des découvertes technologiques et de nombreuses mises en pratique. Ce travail représente à la fois une **synthèse pédagogique**, un **enjeu d'évaluation**, mais aussi – et surtout – un **accomplissement personnel majeur** dans mon parcours de reconversion.

Issu d’un domaine très éloigné de l’informatique, j’ai décidé de changer de voie avec **détermination** et **courage**. Repartir de zéro, apprendre un métier aussi technique et exigeant que celui d’**administrateur DevOps**, comprendre les mécanismes des conteneurs, du cloud, de l’automatisation… cela n’a pas été un chemin facile. Mais c’est justement cette difficulté qui donne tout son **sens** et toute sa **valeur** à ce projet.

Pour ce travail de fin de formation, j’ai choisi de **déployer une application e-commerce WordPress** sur un **cluster Kubernetes auto-hébergé**, en mobilisant l’ensemble des compétences que j’ai acquises. J’ai mis en œuvre :

Ansible pour l’automatisation,

K3s pour l’orchestration des conteneurs,

Prometheus et **Grafana** pour la supervision,

Jenkins et **DockerHub** pour le déploiement continu,

Ainsi que **NFS** pour la persistance des données.

Ce projet va bien au-delà d’un simple TP. Il s’agit d’une **infrastructure DevOps complète**, pensée, construite, testée et documentée comme dans un **environnement réel en entreprise**.

Mais au-delà des outils, ce projet m’a surtout appris à **travailler de manière structurée**, à **persévérer malgré les blocages**, à **chercher des solutions par moi-même**, à **documenter proprement**, et à **valoriser chaque étape** franchie.

Aujourd’hui, je ne me considère plus simplement comme un apprenant. Je me sens prêt à **intégrer une équipe DevOps**, à contribuer à des projets concrets avec une **vision globale** et des **compétences solides**. Ce projet est pour moi **le symbole d’un nouveau départ**, construit avec passion, rigueur et résilience

Objectifs pédagogiques

À travers ce projet, **j’ai souhaité concrétiser l’ensemble des savoirs et compétences acquis** durant ma formation d’**Administrateur Systèmes DevOps** chez M2i Formation. Mon ambition était de démontrer que je suis capable de **concevoir, déployer et automatiser une infrastructure complète**, en respectant les **bonnes pratiques du métier**, avec une approche moderne et professionnelle.

Voici les **objectifs principaux** que je m’étais fixés :

Déployer une application conteneurisée complète : installer **WordPress** et **MySQL** sur Kubernetes, avec une base de données **persistante via un serveur NFS**, comme dans un environnement réel de production.

Orchestrer l'infrastructure avec Kubernetes (K3s) : gérer les **déploiements**, assurer la **résilience**, la **scalabilité** (HPA) et le **bon fonctionnement global** du cluster.

Automatiser les opérations avec Ansible : déploiement des composants, configuration des services, création de **rôles réutilisables** pour structurer le code de manière modulaire.

Mettre en place une chaîne CI/CD professionnelle : utiliser **Jenkins** pour automatiser le **cycle de vie applicatif**, en intégrant **GitHub** pour le versioning et **DockerHub** pour la gestion des images.

Superviser l'infrastructure en temps réel : déployer **Prometheus**, **Node Exporter**, **Alertmanager** et **Grafana**, avec des dashboards clairs pour suivre l'état des ressources.

Travailler comme en entreprise : adopter une méthodologie structurée, gérer les erreurs, **documenter proprement**, tester à chaque étape, et viser un **résultat fiable et réutilisable**.

Ce projet m'a permis de me **projeter pleinement dans une logique DevOps**, en intégrant les valeurs de **collaboration**, d'**automatisation**, de **fiabilité**, et de **résilience**. Il symbolise pour moi un **passage concret** du statut d'apprenant à celui de professionnel en devenir.

Architecture de l'infrastructure

Pour mener à bien ce projet, j'ai conçu une **architecture à la fois simple, fonctionnelle et fidèle aux principes d'un environnement DevOps moderne**. Mon objectif était de **reproduire à petite échelle les conditions réelles** d'un déploiement d'application sur le cloud, en intégrant les éléments essentiels : **orchestration avec Kubernetes**, **persistance des données**, **automatisation des déploiements (CI/CD)**, et **supervision de l'infrastructure**.

J'ai fait le choix d'une solution **auto-hébergée sur trois machines virtuelles Azure**, chacune jouant un **rôle bien défini** au sein du cluster. Ce découpage m'a permis de mieux comprendre la **distribution des services** dans un environnement Kubernetes, ainsi que la **séparation des responsabilités** entre les différents nœuds.

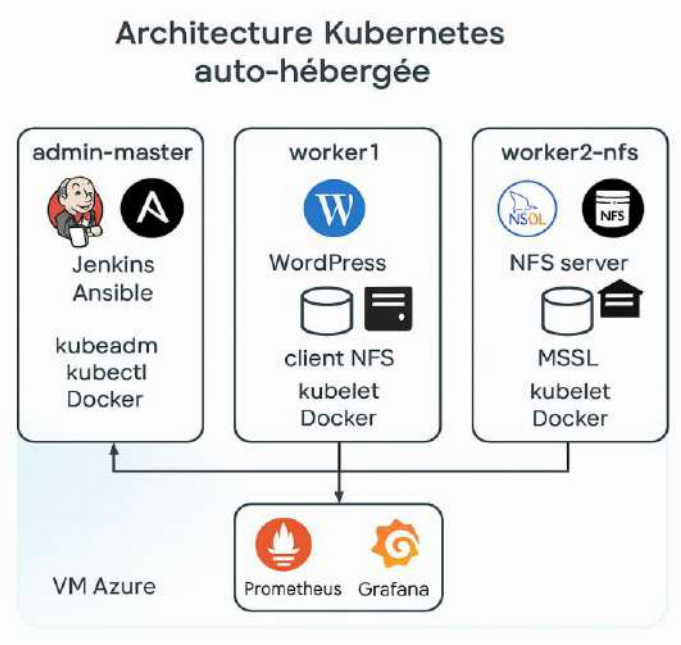
Voici la répartition :

Admin-master : nœud **master** du cluster, avec les composants de contrôle Kubernetes, ainsi que les outils d'orchestration (**Ansible**) et d'intégration continue (**Jenkins**).

Worker1 : nœud **worker** hébergeant le **conteneur WordPress**, ainsi qu'un **client NFS** pour assurer la persistance des données.

Worker2-nfs : nœud **worker** dédié à **MySQL**, avec le rôle supplémentaire de **serveur NFS**, pour stocker les volumes partagés utilisés par les applications.

Cette architecture m'a permis de bâtir une **infrastructure modulaire**, semblable à ce que l'on retrouve dans un **contexte de production**, avec des ressources bien segmentées, des services isolés, et une vision claire de l'ensemble du système.



Le schéma ci-dessus illustre l'architecture que j'ai mise en place pour ce projet. Elle repose sur **trois machines virtuelles Azure**, chacune ayant un rôle bien défini dans le cluster Kubernetes.

Admin-master joue le rôle de **nœud maître** : il héberge les outils d'administration tels que **Ansible**, **Jenkins**, ainsi que les utilitaires Kubernetes nécessaires (**kubeadm**, **kubectl**, **Docker**).

Worker1 est un **nœud de calcul** qui exécute le conteneur **WordPress**, en s'appuyant sur un **client NFS** pour accéder à un volume persistant.

Worker2-nfs héberge deux services essentiels : la base de données **MySQL** et le **serveur NFS** utilisé pour stocker les données applicatives de façon persistante.

L'ensemble est **monitoré** via **Prometheus** et **Grafana**, déployés dans le cluster, permettant une **visualisation en temps réel** de la santé de l'infrastructure.

Cette architecture reflète les **principes de séparation des rôles, de modularité et de supervision** appliqués dans un environnement DevOps professionnel.

Rôle des machines virtuelles

Pour structurer correctement le cluster Kubernetes et assurer une **répartition claire des rôles**, j'ai opté pour une **architecture à trois machines virtuelles hébergées sur Azure**. Chacune

remplit une fonction bien précise, aussi bien dans la **gestion du cluster** que dans l'**hébergement des services applicatifs**. Cette séparation m'a permis de mieux comprendre les **flux de données**, les **dépendances** entre composants, et la **logique de découplage** propre aux environnements DevOps.

Voici la répartition des rôles par machine :

VM	Rôle Kubernetes	Rôle applicatif
admin-master	Master	Jenkins, Ansible, Prometheus, Grafana
worker1	Worker	WordPress, client NFS
worker2-nfs	Worker	MySQL, serveur NFS

Chaque VM contribue à l'équilibre de l'infrastructure :

Admin-master : centralise les outils d'administration, de supervision et d'orchestration.

Worker1 : héberge le **frontend WordPress** avec un accès au **stockage distant** via le client NFS.

Worker2-nfs : combine la **base de données MySQL** avec un **serveur NFS**, pour garantir la persistance des données sur le cluster.

Cette organisation m'a permis de créer un environnement **réaliste**, proche des pratiques de production, tout en restant maîtrisable dans le cadre du projet.

Logiciels et services installés

Pour construire une infrastructure DevOps moderne, **j'ai installé et configuré plusieurs outils essentiels**, répartis sur les trois nœuds de mon cluster selon leurs rôles respectifs. Chaque composant a été sélectionné pour sa **pertinence**, sa **compatibilité avec un environnement cloud auto-hébergé**, et sa **valeur pédagogique** dans une démarche DevOps.

Système d'exploitation
J'ai choisi **Ubuntu Server 22.04 LTS** sur l'ensemble des machines pour sa **stabilité**, sa **large compatibilité avec les outils DevOps**, et sa bonne prise en charge sur Microsoft Azure.

Conteneurisation
J'ai utilisé **Docker** comme moteur de conteneurs. Il permet de packager et exécuter les images applicatives, notamment **WordPress** et **MySQL**, de manière légère et isolée.

Orchestration
Pour orchestrer les services, j'ai opté pour **K3s**, une distribution allégée de Kubernetes. Elle est parfaitement adaptée à un **environnement auto-hébergé avec des ressources limitées**, tout en offrant les fonctionnalités essentielles d'un cluster Kubernetes.

Automatisation avec Ansible
J'ai structuré le déploiement grâce à **Ansible**, en créant des **rôles dédiés** pour chaque grande étape : installation de K3s, configuration du stockage NFS, déploiement de WordPress/MySQL, et supervision. Cela m'a permis d'automatiser les tâches de manière **réutilisable et modulaire**.

Intégration et déploiement continus (CI/CD)

Jenkins, installé sur la VM admin-master, avec un reverse proxy **Nginx** et un accès sécurisé.

GitHub, utilisé pour le versioning du code et le déclenchement de pipelines via **webhooks**.

DockerHub, pour héberger les **images Docker personnalisées** générées par Jenkins (via Jenkinsfile).

Stockage persistant

Un **serveur NFS** sur worker2-nfs, chargé de fournir des volumes partagés montés par les applications.

Des **clients NFS** sur admin-master et worker1, permettant à WordPress de persister les données.

Supervision et observabilité

Prometheus, pour la **collecte des métriques** système, Kubernetes et applicatives.

Node Exporter, installé sur chaque nœud, pour remonter les métriques de chaque machine.

Grafana, pour visualiser les données dans des **dashboards personnalisés** et suivre l'état global du cluster.

Chaque logiciel a été intégré dans un ensemble **cohérent et automatisé**, dans le respect des **bonnes pratiques DevOps**, afin de construire une infrastructure **fiable, observable et évolutive**.

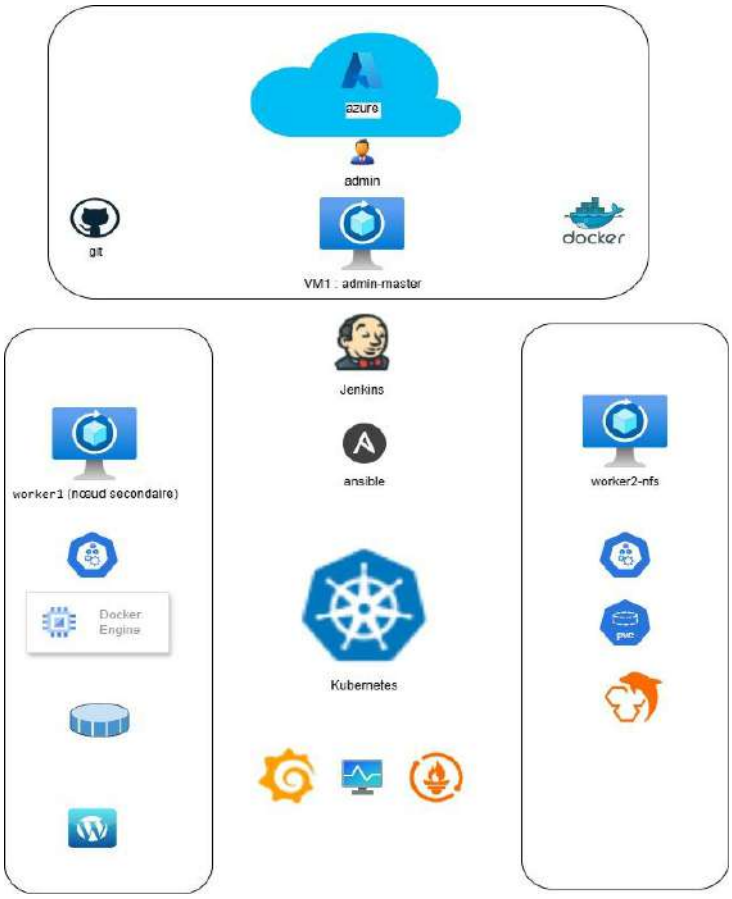


Illustration – Répartition des outils DevOps dans le cluster

Ce schéma illustre la distribution des outils et composants déployés sur les trois machines virtuelles de mon cluster Kubernetes, hébergé sur Microsoft Azure.

La VM admin-master centralise les outils d'orchestration et de déploiement :

Jenkins pour l'intégration continue, ansible pour l'automatisation, Docker pour la conteneurisation,

Ainsi que l'accès aux dépôts GitHub.

Le nœud worker1 (nœud secondaire) héberge le frontend WordPress, avec un accès à un volume persistant via un client NFS. Il utilise Docker Engine pour exécuter les conteneurs. Le nœud worker2-nfs héberge à la fois :la base de données MySQL, le serveur NFS, et les volumes persistants (PVC) nécessaires au stockage applicatif. Les composants de supervision – Prometheus, Grafana et Node Exporter – sont répartis dans l'ensemble du cluster pour assurer une observabilité continue. Ce schéma permet de visualiser clairement l'architecture logique du cluster, la séparation des services, et le rôle précis de chaque machine virtuelle, dans un esprit conforme aux standards DevOps en entreprise.

Schéma de l’infrastructure

Le schéma ci-dessous représente **l’ensemble de l’infrastructure que j’ai déployée dans le cadre de ce projet**. Il permet de **visualiser la répartition des rôles entre les machines virtuelles**, les **services installés sur chaque nœud**, ainsi que les **flux de communication** entre les composants clés de l’environnement DevOps.

On y retrouve les éléments suivants :

Le nœud admin-master
Il héberge les outils d’orchestration et de gestion :

Jenkins pour l’intégration continue,

Ansible pour l’automatisation,

Prometheus et **Grafana** pour la supervision.

Le nœud worker1
Il exécute l’application **WordPress (frontend)**, avec un **client NFS** monté pour accéder aux volumes partagés.

Le nœud worker2-nfs
Il héberge :

Le **serveur NFS**, qui fournit un **stockage persistant** aux autres nœuds,

La base de données **MySQL** (backend).

Les volumes persistants
Gérés via **NFS**, ils assurent la conservation des données même en cas de redéploiement des pods WordPress/MySQL.

Les interactions DevOps

GitHub contient le code source et déclenche des pipelines via **webhooks** vers **Jenkins**,

Jenkins construit les images et les pousse vers **DockerHub**,

Les pods du cluster déploient automatiquement ces images à jour.

La supervision

Prometheus collecte les métriques exposées par **Node Exporter** installé sur chaque nœud,

Grafana permet de visualiser ces données sous forme de **dashboards dynamiques**.

Ce schéma synthétise la **logique DevOps complète** que j’ai mise en place : **automatisation**, **orchestration**, **stockage persistant**, **déploiement continu**, et **observabilité**.

Étapes techniques du déploiement

Création de l'infrastructure (Azure)

Pour débuter ce projet, j'ai choisi d'**héberger toute l'infrastructure sur Microsoft Azure**, afin de travailler dans un **environnement cloud professionnel**, proche des pratiques en entreprise. Mon objectif était de disposer d'une **base stable, modulaire et évolutive**, capable d'accueillir un **cluster Kubernetes auto-hébergé** avec l'ensemble des services DevOps associés.

Étapes réalisées

Création des ressources Azure

J'ai commencé par créer les éléments suivants :

Un **groupe de ressources** nommé rg-ecom, pour regrouper l'ensemble des composants du projet.

Trois **machines virtuelles** de type **DS2_v2** (2 vCPU, 7 Go RAM), avec **Ubuntu Server 22.04 LTS**, nommées :

Admin-master : maître du cluster, hébergeant Jenkins, Ansible, Prometheus, etc.

Worker1 : nœud secondaire exécutant WordPress.

Worker2-nfs : nœud hébergeant MySQL et le serveur NFS.

Configuration réseau

Afin de garantir le bon fonctionnement du cluster et l'accès aux services, j'ai :

Créé un **réseau virtuel partagé** entre les trois VMs.

Ouvert les ports nécessaires dans les **NSG (Network Security Groups)** :

SSH : 22

Web : 80, 443

Kubernetes : 10250, 6443, 30000-32767

NFS : 2049

Affecté une **adresse IP publique statique** à chaque VM pour permettre l'accès SSH depuis ma machine locale.

Connexions SSH et préparation des machines

Pour sécuriser les accès et automatiser les connexions, j'ai :

Généré une **paire de clés SSH** avec ssh-keygen :

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa_k3s
```

Ajouté la **clé publique** dans chaque VM via **Azure Portal** ou via **Cloud Shell** :

```
az vm user update \
  --resource-group rg-ecom \
  --name admin-master \
  --username azureuser \
  --ssh-key-value ~/.ssh/id_rsa_k3s.pub
```

Testé les connexions :

```
ssh -i ~/.ssh/id_rsa_k3s azureuser@<IP_VM>
```

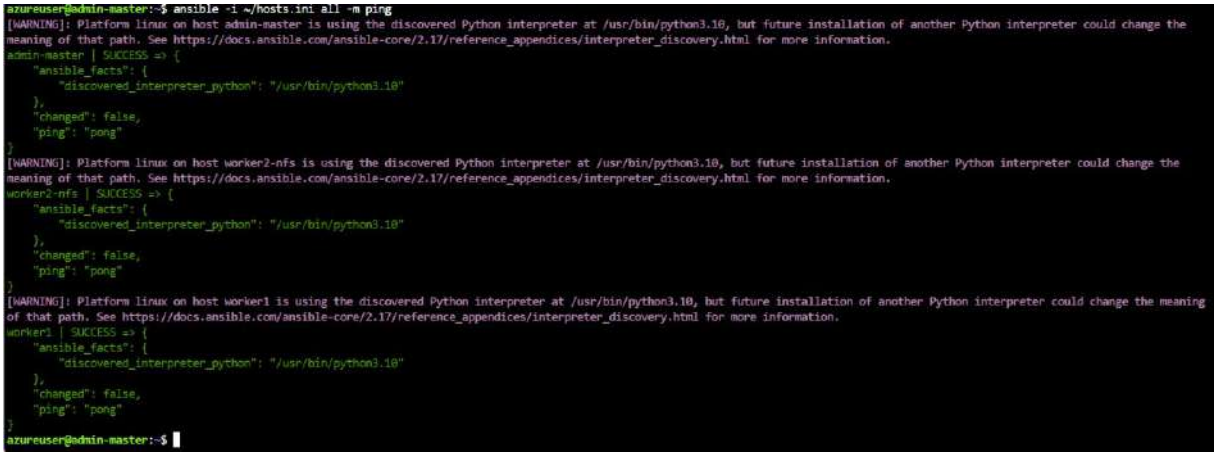
Une fois la connectivité SSH établie entre toutes les machines, j'ai validé les échanges avec Ansible en exécutant un simple ping sur l'inventaire :

```
Ansible -i hosts.ini all -m ping -u azureuser --private-key ~/.ssh/id_rsa_k3s
```

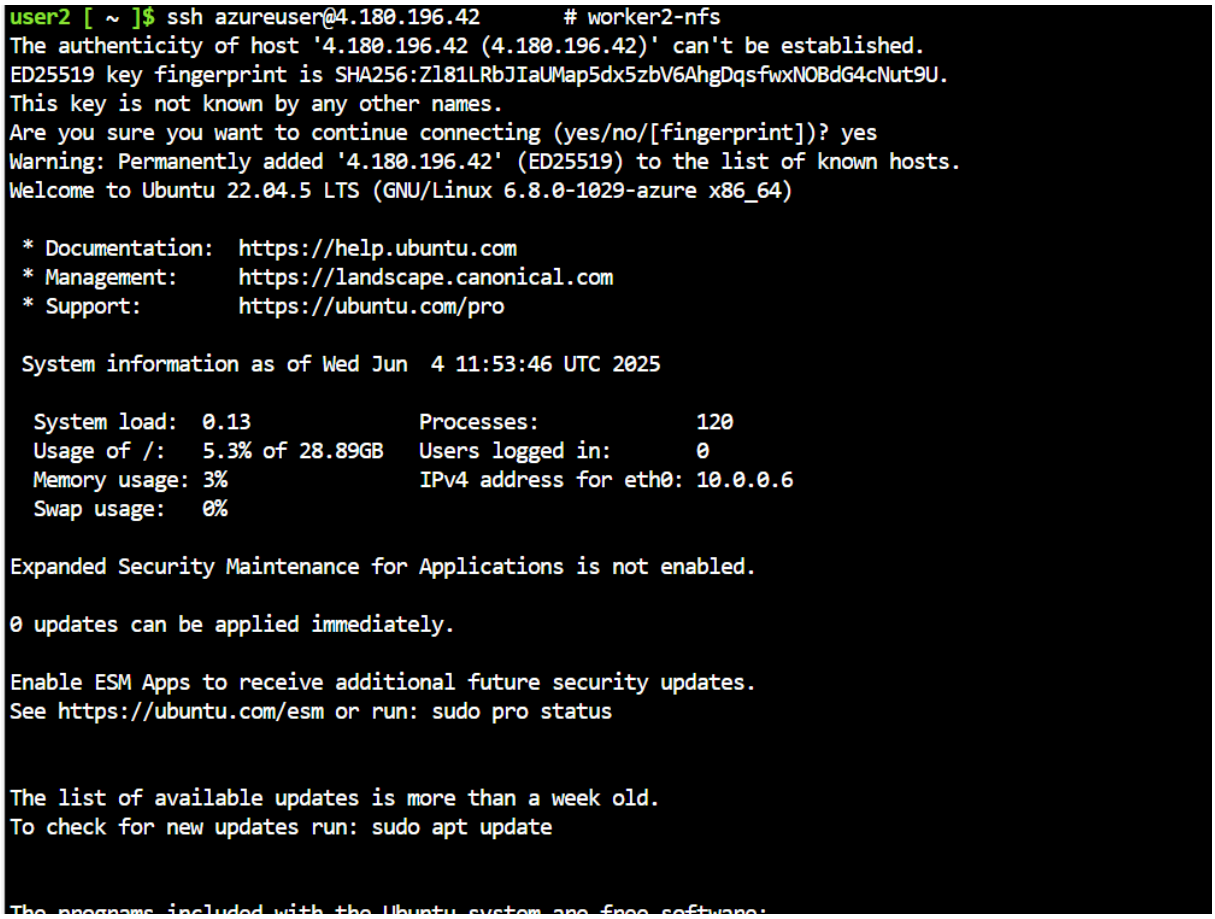
Captures à inclure ici :

Cette capture montre la **création de la VM admin-master via la ligne de commande Azure (az CLI)**. J'ai utilisé l'argument --generate-ssh-keys pour créer une paire de clés SSH automatiquement et les injecter dans la machine. L'option --image Ubuntu2204 m'a permis de choisir une distribution compatible avec Docker et Kubernetes.

J'ai préféré utiliser la **CLI** à cette étape pour gagner en efficacité et vérifier immédiatement les **informations critiques**, comme l'**IP publique**, le **nom de la VM**, et l'**état de mise en service**. Cela m'a aussi permis de **répéter facilement la création pour les autres nœuds** du cluster.



Ici, j'utilise la commande ansible -i hosts.ini all -m ping pour **tester la connectivité SSH avec Ansible**. Le retour "pong" sur chaque hôte confirme que **les trois machines (admin-master, worker1, worker2-nfs) sont bien accessibles et prêtes à être gérées par Ansible**. C'est un point de contrôle important avant tout déploiement automatisé.



Dans cette capture, je montre la **connexion SSH réussie** à la machine virtuelle worker2-nfs. Cela confirme que la **clé publique a bien été injectée** et que la **configuration réseau (port 22, IP publique)** est fonctionnelle. Cette étape était indispensable pour permettre par la suite l'exécution des playbooks Ansible.

Installation du cluster Kubernetes (K3s)

Une fois l'infrastructure en place et les connexions SSH testées, j'ai procédé à l'installation du cluster Kubernetes. J'ai choisi **K3s**, une distribution allégée et optimisée de Kubernetes, parfaitement adaptée aux environnements **auto-hébergés avec peu de ressources**, comme c'est le cas ici.

Installation du nœud master (admin-master)

Sur la machine admin-master, j'ai lancé l'installation du serveur K3s avec la commande suivante :

```
curl -sfL https://get.k3s.io | sh -
```

Cette commande télécharge et exécute un **script sécurisé** d'installation fourni par les mainteneurs de K3s. À la fin du processus :

Le **serveur K3s est actif**,

Un **agent kubelet** est automatiquement démarré,

Le fichier kubeconfig est généré dans /etc/rancher/k3s/k3s.yaml.

Pour m'assurer que le cluster avait bien démarré, j'ai exécuté :

```
sudo kubectl get nodes
```

Le nœud admin-master apparaissait en **statut Ready**, ce qui confirmait que l'initialisation avait réussi.

Récupération du token pour l'ajout des agents

Pour permettre aux nœuds worker1 et worker2-nfs de rejoindre le cluster, j'ai récupéré le **token d'authentification** généré automatiquement par K3s :

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Ce token, combiné à l'**adresse IP privée** du nœud admin-master, est nécessaire pour **initialiser les nœuds agents**.

Capture :

```
azureuser@admin-master:~$ sudo cat /var/lib/rancher/k3s/server/node-token
K1079918203f6d94fdc5393ae58a144c3bd9c0263511c56bb59da35869423cd8115::server:5df3b0c65bfa4e33128a409dda4d2bd1
azureuser@admin-master:~$
```

Cette capture montre la commande que j'ai exécutée sur le nœud admin-master pour **récupérer le token d'authentification K3s** :

sudo cat /var/lib/rancher/k3s/server/node-token

Ce **token unique** est généré automatiquement par K3s lors de l'installation du nœud maître. Il est indispensable pour permettre aux nœuds worker1 et worker2-nfs de rejoindre le cluster de manière sécurisée. J'ai utilisé ce token comme **variable d'environnement dans mon playbook Ansible** pour automatiser l'enrôlement des agents.

Remarque : le contenu du token a été affiché ici à titre technique. Il est recommandé de le flouter dans la version publique du rapport.

Ajout des nœuds worker1 et worker2-nfs

Sur les deux nœuds secondaires, l'installation de K3s se fait avec la commande suivante :

```
curl -sfL https://get.k3s.io | K3S_URL=https://<IP_MASTER>:6443
K3S_TOKEN=<TOKEN> sh -
```

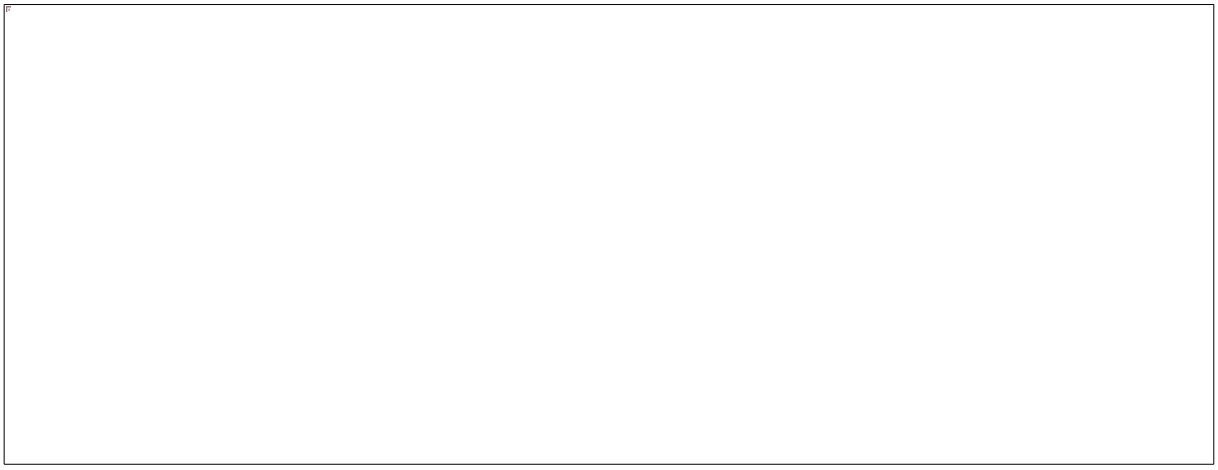
J'ai automatisé cette étape dans un **playbook Ansible** que j'ai moi-même rédigé. Le rôle dédié inclut :

Le téléchargement du script via curl,

La substitution dynamique du token et de l'adresse IP,

L'exécution sécurisée avec élévation de privilèges (become: true).

Capture :



Cette capture montre un extrait de mon **playbook Ansible install-k3s.yml**, que j'ai rédigé pour automatiser l'installation complète du cluster Kubernetes avec K3s.

Dans la première partie, j'installe **K3s en mode serveur** sur le nœud admin-master. Je récupère ensuite le **token de jonction** générée automatiquement et je le stocke dans une variable (k3s_token) pour l'utiliser sur les autres nœuds.

Dans la seconde partie, le rôle est exécuté sur les nœuds worker1 et worker2-nfs. Grâce aux variables K3S_URL et K3S_TOKEN, chaque worker peut se connecter automatiquement au master et rejoindre le cluster.

Ce script m'a permis de gagner du temps, de **standardiser les opérations**, et de m'assurer que chaque nœud était configuré **de manière cohérente et reproductible**, conformément aux **bonnes pratiques DevOps.**

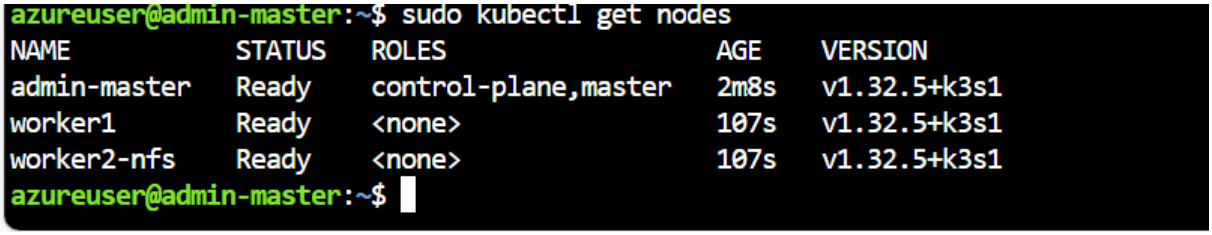
Vérification finale du cluster

Après l'exécution du playbook, j'ai de nouveau vérifié l'état du cluster avec :

```
kubectl get nodes
```

Cette fois, les **trois nœuds** (admin-master, worker1, worker2-nfs) apparaissaient en **statut Ready**, avec les rôles correctement assignés (control-plane pour le master, worker pour les agents).

Capture :



Cette capture montre le résultat de la commande kubectl get nodes exécutée depuis le nœud admin-master. On y voit que les trois nœuds du cluster (admin-master, worker1, worker2-nfs) sont bien détectés et en **statut "Ready"**, ce qui signifie qu'ils sont **intégrés et opérationnels**.

Le rôle control-plane,master est correctement attribué à admin-master, tandis que les deux autres nœuds sont reconnus comme **agents (workers)**, prêts à exécuter les charges de travail. Cette étape confirme que **l'installation du cluster K3s est une réussite**, et que je peux passer à la suite du projet : le **stockage persistant**.

.

Résumé technique des commandes utilisées :	
Étape	**Commande utilisée**
Installer K3s sur le master	`curl -sfL https://get.k3s.io
Vérifier master	kubectl get nodes ou sudo kubectl get nodes
Récupérer le token	sudo cat /var/lib/rancher/k3s/server/node-token
Ajouter un worker	`curl -sfL https://get.k3s.io

Mise en place du stockage persistant (NFS)

Pour garantir la **persistance des données**, notamment celles de WordPress et de la base MySQL, j'ai mis en place une solution de **stockage partagé en réseau** basée sur **NFS (Network File System)**. Ce choix s'est imposé naturellement dans un contexte **auto-hébergé**, où les volumes locaux des pods ne suffisent pas à assurer la **durabilité** ni la **portabilité** des données.

J'ai choisi de configurer le **nœud worker2-nfs en tant que serveur NFS**, et les nœuds admin-master et worker1 comme **clients NFS**.

Configuration du serveur NFS (worker2-nfs)

Sur ce nœud, j'ai procédé aux étapes suivantes :

Installation du serveur NFS :

```
sudo apt install -y nfs-kernel-server
```

Création du dossier partagé :

```
sudo mkdir -p /srv/nfs/kubedata
```

```
Sudo chown nobody:nogroup /srv/nfs/kubedata
```

```
sudo chmod 777 /srv/nfs/kubedata
```

Déclaration du partage dans le fichier /etc/exports :

```
echo "/srv/nfs/kubedata *(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -a /etc/exports
```

Application de la configuration et redémarrage du service :

```
sudo exportfs -rav
```

```
sudo systemctl restart nfs-kernel-server
```

Capture :

Cette capture montre la **configuration du service NFS** sur le nœud worker2-nfs.
Le répertoire /srv/nfs/kubedata est exporté avec les options suivantes :

rw : accès en lecture/écriture pour les clients

sync : les écritures sont effectuées immédiatement sur le disque

no_subtree_check : évite les vérifications supplémentaires sur les sous-arborescences

no_root_squash : permet au client root de conserver ses privilèges (utile dans un cluster auto-géré)

Cette configuration permet aux **nœuds Kubernetes clients** (admin-master et worker1) de **monter ce dossier NFS** et d'y accéder pour la persistance des volumes (WordPress & MySQL).

Elle est gérée automatiquement par **Ansible**, comme le montre le bloc # BEGIN ANSIBLE MANAGED BLOCK.

Configuration des clients NFS (admin-master et worker1)

Sur les deux nœuds clients, j'ai effectué :

Installation du client NFS :

```
sudo apt install -y nfs-common
```

Test de montage manuel (facultatif mais utile pour validation) :

```
sudo mount <IP_worker2-nfs>:/srv/nfs/kubedata /mnt
```

Ce test m'a permis de m'assurer que la **connexion au serveur NFS** fonctionnait bien avant d'utiliser ce volume via Kubernetes.

Capture :

Cette capture montre le **contenu du dossier exporté en NFS** sur le nœud worker2-nfs. On y retrouve plusieurs types de fichiers et dossiers significatifs :

Des fichiers de test comme test.txt, fichier-worker1.txt, nfs-test.txt, qui ont été créés depuis différentes VMs du cluster pour **vérifier le montage NFS et la persistance des données**.

Des dossiers comme wordpress, wp-admin, wp-content, mysql : ils sont utilisés par les pods WordPress et MySQL pour **stocker durablement leur contenu** (site, thème, BDD).

Des fichiers WordPress (ex : wp-config.php, wp-settings.php, etc.) : preuve que le CMS est bien installé et que ses données persistent même après redémarrage.

Cela confirme que le **stockage partagé est fonctionnel, accessible par tous les nœuds**, et qu'il **garantit la résilience des données** sur l'ensemble de l'infrastructure Kubernetes.

Intégration dans Kubernetes

Après validation du fonctionnement de NFS, j'ai créé les fichiers **YAML** pour déclarer les ressources suivantes :

Un **PersistentVolume (PV)** pointant vers le chemin NFS partagé /srv/nfs/kubedata

Des **PersistentVolumeClaims (PVC)** spécifiques à :

MySQL pour la base de données

WordPress pour les contenus dynamiques

Voici un extrait simplifié d'un PV :

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-mysql
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs :
    path : /srv/nfs/kubedata
    server : <IP_worker2-nfs>
  persistentVolumeReclaimPolicy: Retain
```

Les PVC ont ensuite été **liés aux déploiements Kubernetes** via les champs volumeMounts et volumes dans les fichiers deployment.yaml.

Captures :

```
azureuser@admin-master:~/wordpress-k3s$ cat k8s/pv-pvc-nfs.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes: ["ReadWriteMany"]
  nfs:
    server: 10.0.0.6          # IP de ta VM NFS
    path: /srv/nfs/kubedata
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-data
  namespace: wordpress
spec:
  accessModes: ["ReadWriteMany"]
  storageClassName: ""
  volumeName: nfs-pv
  resources:
    requests:
      storage: 1Gi
azureuser@admin-master:~/wordpress-k3s$
```

Cette capture montre le contenu du fichier pv-pvc-nfs.yaml, utilisé pour **déclarer le stockage persistant** de mon application WordPress dans Kubernetes.

La première partie définit un **PersistentVolume** nommé nfs-pv.
Il pointe vers le répertoire partagé NFS situé sur le nœud worker2-nfs, à l'adresse IP 10.0.0.6, avec le chemin /srv/nfs/kubedata.

L'accès est configuré en mode **ReadWriteMany** (RWX), ce qui permet à plusieurs pods (sur plusieurs nœuds) de lire et écrire simultanément dans le volume — condition indispensable pour un cluster Kubernetes distribué.

La deuxième partie décrit un **PersistentVolumeClaim** (wp-data) dans le namespace wordpress, qui **demande dynamiquement 1 Go de stockage** sur le volume nfs-pv.

Cette configuration permet de **lier proprement les déploiements WordPress et MySQL au stockage réseau**, afin de garantir la **persistance des données même en cas de redémarrage des pods ou du cluster**.

Résultat obtenu

Grâce à cette configuration, les données de WordPress (contenus uploadés) et de MySQL (base) sont **stockées de façon centralisée et persistante**, indépendamment des pods. Même après un redémarrage du cluster ou un redeploiement, les données sont **intactes**. Cette solution garantit **la robustesse et la continuité de service** de mon application e-commerce.

.

Déploiement de WordPress et MySQL

L'objectif principal de cette étape était de **déployer les deux composants fondamentaux de l'application** : la base de données **MySQL** et le CMS **WordPress**, au sein de mon cluster Kubernetes. Pour cela, j'ai structuré mes **manifests YAML** de manière modulaire, afin de garantir un **déploiement propre, sécurisé et persistant**, tout en respectant les bonnes pratiques DevOps.

Création des secrets et ConfigMap

Pour sécuriser les accès à la base de données, j'ai défini un **Secret Kubernetes** contenant les identifiants de connexion :

```
kubectl create secret generic mysql-credentials \
  --from-literal=mysql-root-password=MonSuperMotDePasse \
  -n wordpress
```

J'ai également créé un **ConfigMap** pour stocker les **paramètres de configuration WordPress** comme le titre du site ou l'adresse URL.

Déploiement de MySQL

Le fichier mysql-deployment.yaml contient :

Un **Deployment** avec une seule réplique de mysql

Un **volume persistant** via un **PVC pointant vers le NFS**

Des **variables d'environnement** (MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, etc.)

Un **Service de type ClusterIP** pour que WordPress puisse accéder à MySQL, sans l'exposer publiquement

```
kubectl apply -f mysql-deployment.yaml -n wordpress
```

Déploiement de WordPress

Le fichier wordpress-deployment.yaml contient :

Un **initContainer** qui attend la disponibilité de MySQL (commande sleep + test TCP)

Le conteneur principal WordPress avec :

Un **volume monté** sur /var/www/html

Un **PVC WordPress** connecté au serveur NFS

Un **Service de type NodePort** exposé sur le port **30080** (consultable via IP de worker1)

```
kubectl apply -f wordpress-deployment.yaml -n wordpress
```

Vérifications après déploiement

État des pods

```
kubectl get pods -n wordpress -o wide
```

Capture :

```
azureuser@admin-master:~$ kubectl get pods -n wordpress -o wide
NAME                         READY   STATUS    RESTARTS      AGE   IP           NODE      NOMINATED NODE   READINESS GATES
mysql-8775dd8b9-g5pxf        1/1     Running   2 (41m ago)   36h   10.42.2.32   worker1   <none>           <none>
wordpress-76c8d557dc-4ms86   1/1     Running   1 (41m ago)   21h   10.42.2.33   worker1   <none>           <none>
azureuser@admin-master:~$
```

Cette capture montre le résultat de la commande :

On y observe que les pods :

wordpress-xxxxx (CMS frontend)

mysql-xxxxx (base de données)

Sont tous les deux en **statut Running**, sans redémarrages anormaux.

Cela confirme que :

Les déploiements ont bien été pris en charge par Kubernetes

Les conteneurs tournent correctement sur leurs nœuds respectifs (worker1, worker2-nfs…)

La communication entre WordPress et MySQL est fonctionnelle (attente via initContainer si configuré)

Les volumes NFS sont bien montés et accessibles

Cette étape valide que l'application complète est bien **opérationnelle** dans le cluster Kubernetes, avec une **infrastructure distribuée et persistante.**

Volumes persistants

```
kubectl get pvc -n wordpress
```

Capture :

Cette capture montre la commande kubectl get pvc,pv -n wordpress, qui permet de vérifier l'état des volumes Kubernetes.

On y voit que :

Les **PersistentVolumeClaims (PVC)** ont bien le statut Bound, ce qui signifie qu'ils sont **correctement liés** à des **PersistentVolumes (PV)** disponibles.

Les PVC portent des noms clairs comme wp-data, correspondant aux besoins de WordPress ou MySQL.

Le mode d'accès est ReadWriteMany (RWX), ce qui permet à plusieurs pods d'utiliser le volume simultanément, via le **serveur NFS**.

Cela confirme que :

Le **stockage persistant est fonctionnel**

La **liaison entre Kubernetes et le NFS** est bien établie

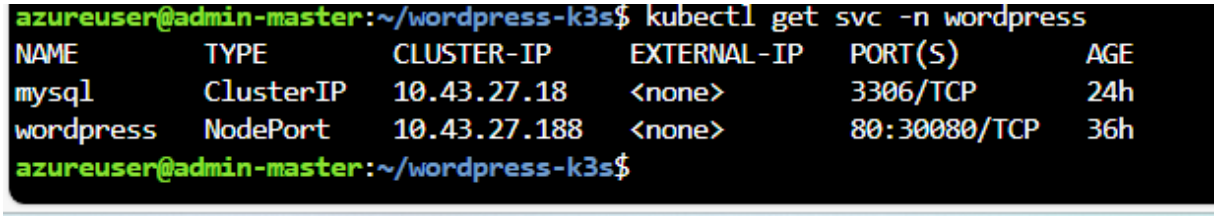
Les **données de WordPress et MySQL** seront conservées même après redémarrage ou redéploiement

Cette étape est essentielle pour garantir la **résilience** et la **durabilité** de l'infrastructure déployée.

Services exposés

```
kubectl get svc -n wordpress
```

Capture :



Cette capture montre le résultat de la commande :

On observe deux services déployés :

mysql : service interne de type **ClusterIP**, non accessible de l'extérieur (sécurisé, réservé aux pods internes)

wordpress : service de type **NodePort**, exposant le port **80 du pod** vers le port **30080 sur le nœud**

Cela signifie que l'interface WordPress est accessible via un navigateur avec l'URL suivante :

Ce mode d'exposition permet :

D'accéder facilement à l'application depuis Internet sans Ingress

De valider le bon fonctionnement du cluster et des services réseau

De réaliser des démonstrations ou des tests en condition réelle

Cette configuration est adaptée pour un projet auto-hébergé et confirme que **l'application WordPress est bien disponible en accès public**, comme exigé dans un contexte DevOps.

Accès frontend depuis le navigateur

Accède au site depuis un navigateur :

`Http://<` 4.180.186.142:32568`>:30080`

Capture :

Cette capture montre l'écran d'accueil de l'installateur WordPress, accessible via le navigateur à l'adresse :

http://4.180.186.142:32568/wp-admin/install.php

Cette interface apparaît uniquement si :

Le pod WordPress est bien déployé et accessible via le service NodePort

La base de données MySQL est opérationnelle et joignable depuis le pod WordPress

Les fichiers WordPress sont bien **persistés via le stockage NFS**

Cela confirme que :

Le **CMS WordPress a bien été déployé** dans le cluster Kubernetes

L'application est **accessible publiquement**

La **communication avec MySQL fonctionne**

Le **volume NFS est bien monté** (car les fichiers WordPress persistent)

C'est une **preuve technique et visuelle** que l'architecture DevOps (K3s + NFS + CI/CD) est fonctionnelle et opérationnelle.

Résultat obtenu

Grâce à cette configuration :

WordPress est **connecté à MySQL**

Le contenu est **persistant grâce à NFS**

Le site est **accessible depuis l'extérieur**

L'ensemble tourne sur Kubernetes, **de manière fiable et autonome**

Cette étape a démontré que l'infrastructure que j'ai bâtie était capable d'**héberger une application réelle**, avec **stockage durable, sécurité, et accessibilité externe**.

Mise en place de la CI/CD avec Jenkins

Afin de professionnaliser le **cycle de vie applicatif**, j'ai mis en place une **chaîne CI/CD complète**, reposant sur Jenkins, en lien direct avec **GitHub** pour le code source, et **DockerHub** pour le stockage des images. L'objectif était d'**automatiser toutes les étapes** : construction de l'image Docker, tests, push, et déploiement automatique sur mon cluster Kubernetes K3s à chaque mise à jour du code.

Installation et sécurisation de Jenkins

J'ai installé Jenkins sur la machine admin-master via Ansible, en suivant ces étapes :

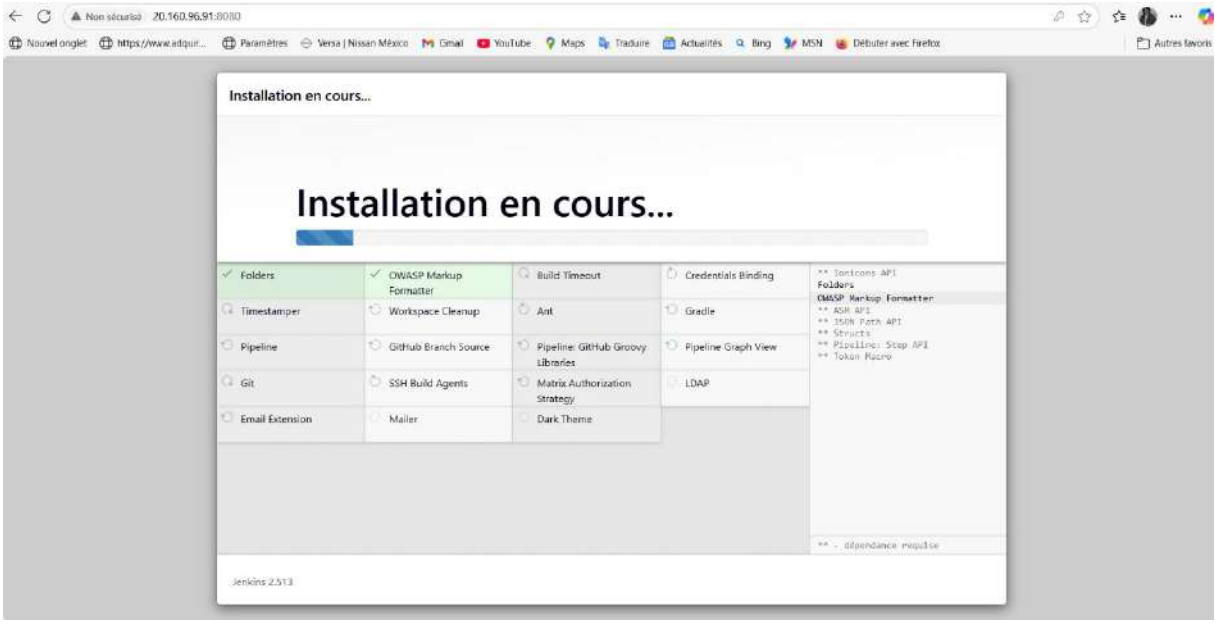
```
sudo apt update && sudo apt install -y openjdk-17-jdk
```

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

```
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
```

```
sudo apt update && sudo apt install -y Jenkins
```

Capture :



Cette capture montre l'interface web de Jenkins au moment de l'**installation des plugins nécessaires**, accessible ici à l'adresse :

http://20.160.96.91:8080

Jenkins est installé sur la VM admin-master et est **accessible via HTTP sur le port 8080**.

À ce stade :

L'utilisateur est connecté à l'interface Jenkins pour la **première configuration**

Les **plugins essentiels** sont en cours d'installation : Git, Pipeline, GitHub Integration, Email, etc.

Cela montre que Jenkins a bien été déployé avec succès dans l'infrastructure

Étapes suivantes prévues :

Configuration d'un reverse proxy (Nginx) pour basculer en HTTPS

Ajout des credentials GitHub et DockerHub

Création d'un pipeline CI/CD basé sur un Jenkinsfile

Ce moment est important car il marque le **début de l'automatisation dans le projet DevOps**, en posant les bases de la chaîne CI/CD.

Connexion à GitHub et DockerHub

J’ai créé un **dépôt GitHub** contenant :

Le code WordPress personnalisé (avec un thème préinstallé)

Un Dockerfile pour créer l’image

Un Jenkinsfile pour décrire le pipeline

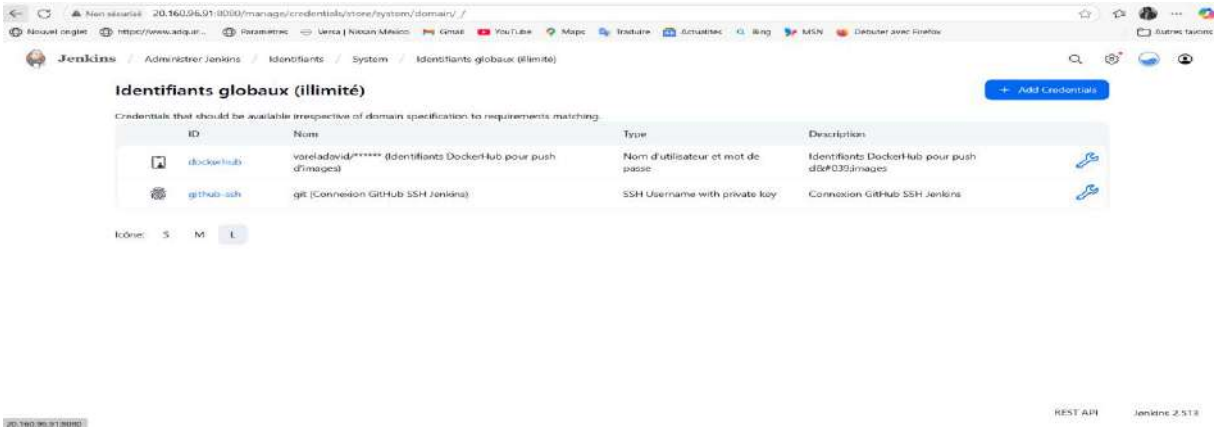
Les manifests Kubernetes (wordpress-deployment.yaml, etc.)

Puis, dans Jenkins :

J’ai ajouté un **token PAT GitHub** dans *Manage Credentials*

J’ai enregistré mes **identifiants DockerHub** (login + token ou mot de passe)

Capture :



Cette capture montre l’onglet **"Identifiants globaux"** (Credentials > global) dans l’interface de Jenkins, où ont été ajoutés deux éléments essentiels à l'automatisation DevOps :

dockerhub

Type : *Nom d'utilisateur et mot de passe*

Utilisé pour **pusher les images Docker vers DockerHub** depuis un pipeline Jenkins.

Stocké de manière sécurisée et référencé dans le Jenkinsfile via un ID.

github-ssh

Type : *SSH Username with private key*

Utilisé pour permettre à Jenkins de **se connecter à GitHub** en SSH et **cloner les dépôts privés**, lancer les builds ou interagir avec les branches du projet.

Ces credentials sont :

Stockés en toute sécurité dans le credential store interne de Jenkins

Réutilisables dans les pipelines CI/CD

Indispensables pour garantir une communication sécurisée avec les services Git et DockerHub

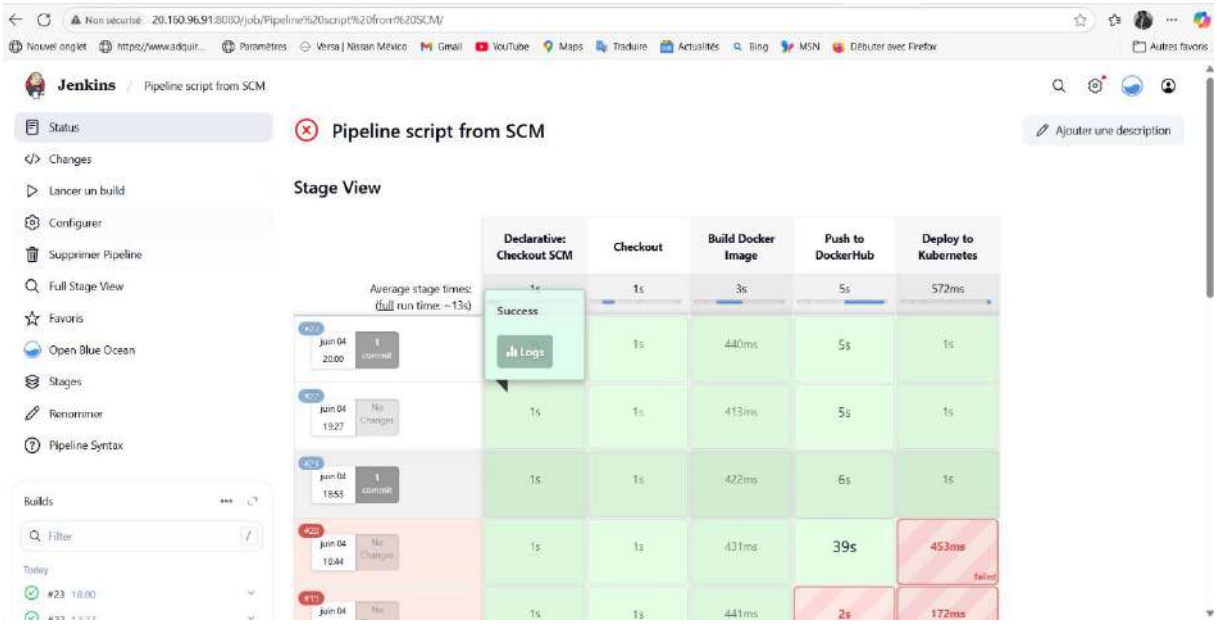
Cette étape est un **pré-requis critique** pour toute intégration continue (CI), et montre que Jenkins est prêt à fonctionner de manière professionnelle dans une chaîne DevOps complète

Fichier Jenkinsfile : pipeline as code

Mon Jenkinsfile, versionné sur GitHub, est composé des étapes suivantes :

```
pipeline {
    agent any
    environment {
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-creds-id')
    }
    stages {
        stage('Build Docker image') {
            steps {
                sh 'docker build -t vareladavidhugo/wordpress-custom:latest
.'
            }
        }
        stage('Push to DockerHub') {
            steps {
                withDockerRegistry([credentialsId:     'dockerhub-creds-id',
url: '']) {
                    sh 'docker push vareladavidhugo/wordpress-custom:latest'
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f k8s/app/ -n wordpress'
            }
        }
    }
}
```

Capture :



Cette capture représente la **vue par étapes (Stage View)** d'un pipeline Jenkins multibranches, défini via un Jenkinsfile et connecté à un dépôt GitHub.

On y voit les différentes étapes de la chaîne CI/CD :

SCM Checkout – récupération du code source depuis GitHub

Checkout – préparation du workspace Jenkins

Build Docker Image – construction de l'image Docker personnalisée (WordPress avec thème ou plugin)

Push to DockerHub – envoi de l'image vers le registre distant DockerHub

Deploy to Kubernetes – application des manifests via kubectl apply sur le cluster K3s

Le pipeline s'exécute automatiquement à chaque nouveau commit :

Les builds #21, #22, #23 sont tous **succès (Success)**

Cela prouve que **toutes les étapes fonctionnent correctement** et que l'automatisation est en place

Cette capture est la preuve concrète que le projet respecte les principes fondamentaux du **DevOps moderne** :

Intégration continue

Livraison continue

Automatisation du déploiement dans un environnement Kubernetes réel

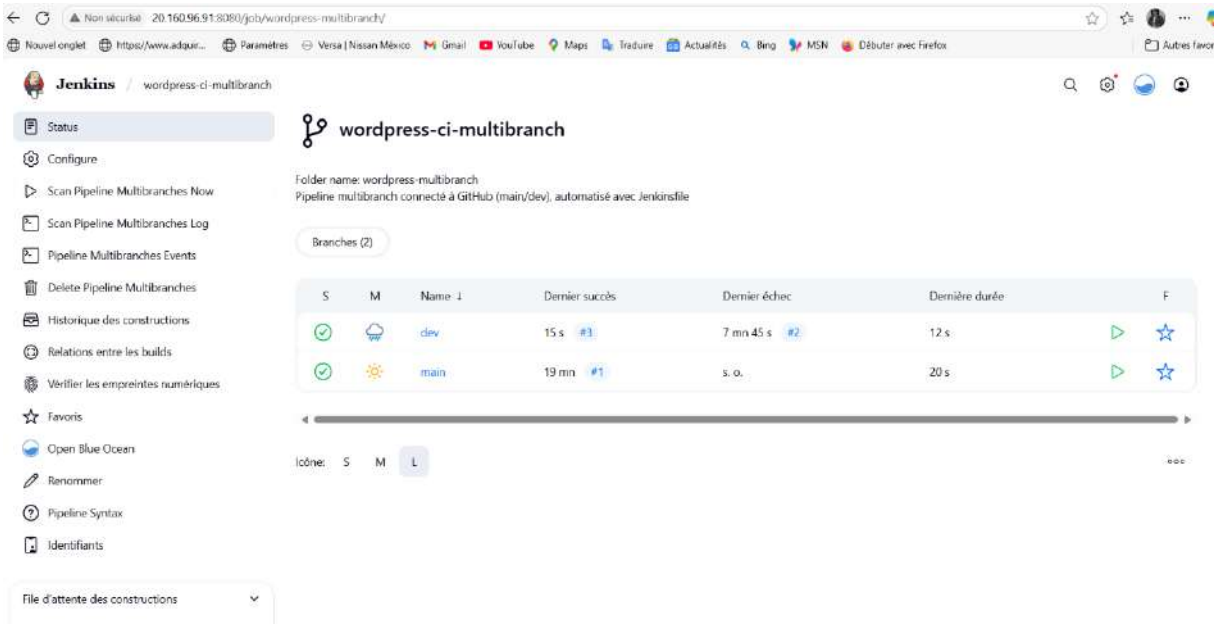
Pipeline multibranches GitHub

Pour renforcer la structuration du projet, j'ai configuré un **job multibranch pipeline Jenkins**
Détection automatique des branches main (production) et dev (test)

Déclenchement automatique à chaque git push sur GitHub

Séparation claire des environnements selon la branche

Capture :



Cette capture montre l'interface d'un **job Jenkins multibranch**, connecté à un dépôt GitHub.

On observe :

La **détection automatique des branches Git (main, dev)**

L'exécution individuelle des pipelines selon la branche

Le suivi clair de chaque build

Ce type de job permet de travailler avec plusieurs environnements (test, prod) en respectant les workflows Git.

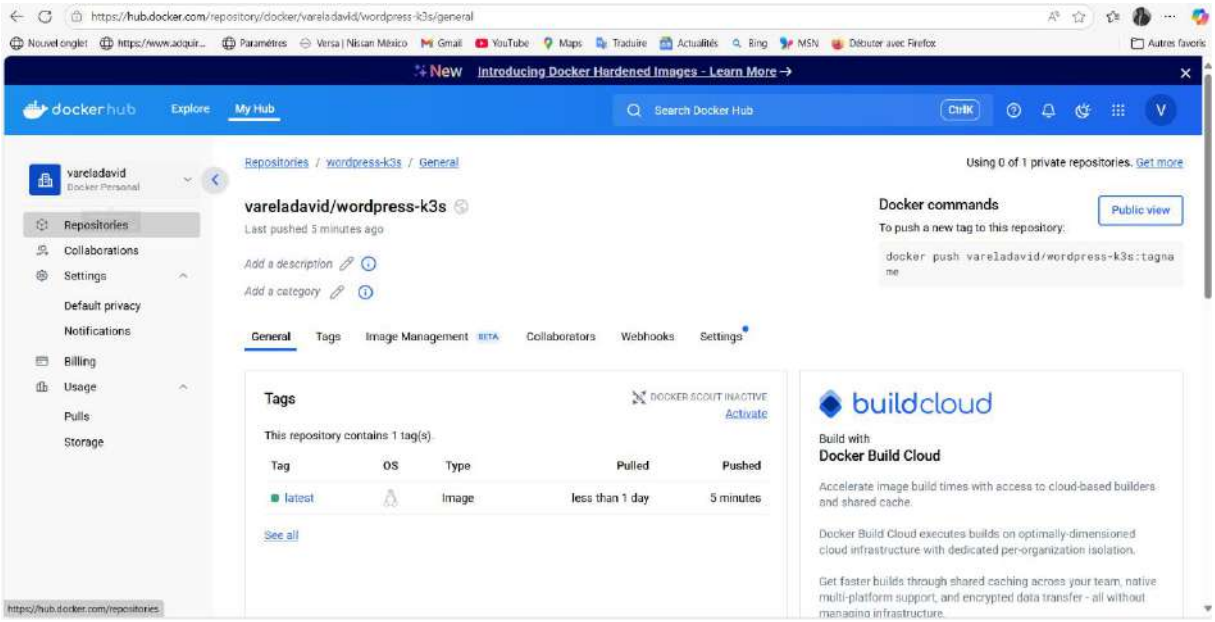
Vérifications et tests finaux

- J'ai lancé des jobs manuellement pour valider chaque étape
- J'ai vérifié que l'image Docker était bien poussée sur DockerHub
- J'ai contrôlé que les déploiements dans Kubernetes étaient automatiquement mis à jour

```
kubectl get pods -n wordpress
```

```
kubectl describe pod <pod-name> -n wordpress
```

Capture :



Le dépôt DockerHub vareladavid/wordpress-k3s contient l'image Docker personnalisée utilisée pour déployer l'application WordPress sur le cluster Kubernetes. Cette image est mise à jour automatiquement par Jenkins après chaque push sur la branche dev.

Résultat

Grâce à cette CI/CD :

Chaque commit déclenche un pipeline complet

Le build, le push, et le déploiement sont automatisés

La cohérence entre les branches dev/prod est assurée

Le processus est reproductible, sécurisé, et facilement maintenable

Cette mise en place m'a permis de **passer à un niveau professionnel de gestion du cycle de vie applicatif**, en appliquant concrètement les **pratiques DevOps** apprises lors de ma formation.

Supervision avec Prometheus et Grafana

La mise en place de la supervision a été une **étape clé** dans mon projet. Elle m'a permis d'assurer la **visibilité**, la **traçabilité**, et la **fiabilité** de toute l'infrastructure. J'ai choisi une stack open source composée de **Prometheus**, **Node Exporter** et **Grafana**, parfaitement adaptée aux environnements Kubernetes auto-hébergés.

Installation des outils

Sur admin-master (Prometheus + Grafana)

```
# Télécharger et extraire Prometheus

wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz

tar xvfz prometheus-2.52.0.linux-amd64.tar.gz

sudo mv prometheus-2.52.0.linux-amd64 /opt/prometheus


# Lancer Prometheus (exemple basique)

cd /opt/prometheus

./prometheus --config.file=prometheus.yml
```

Capture :

Cette capture montre que l'interface de Prometheus est bien opérationnelle et accessible à l'adresse http://20.160.96.91:9090. Cela confirme que :

Le déploiement de Prometheus est fonctionnel sur la VM admin-master.

Le port 9090 est bien ouvert dans les règles de sécurité réseau (NSG) de la VM Azure.

L'utilisateur peut accéder à Prometheus pour consulter les métriques, exécuter des requêtes PromQL, et surveiller les cibles (Targets) du cluster K3s.

Sur chaque VM (admin-master, worker1, worker2-nfs)

Installation de Node Exporter

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.8.1/node_exporter-1.8.1.linux-amd64.tar.gz
```

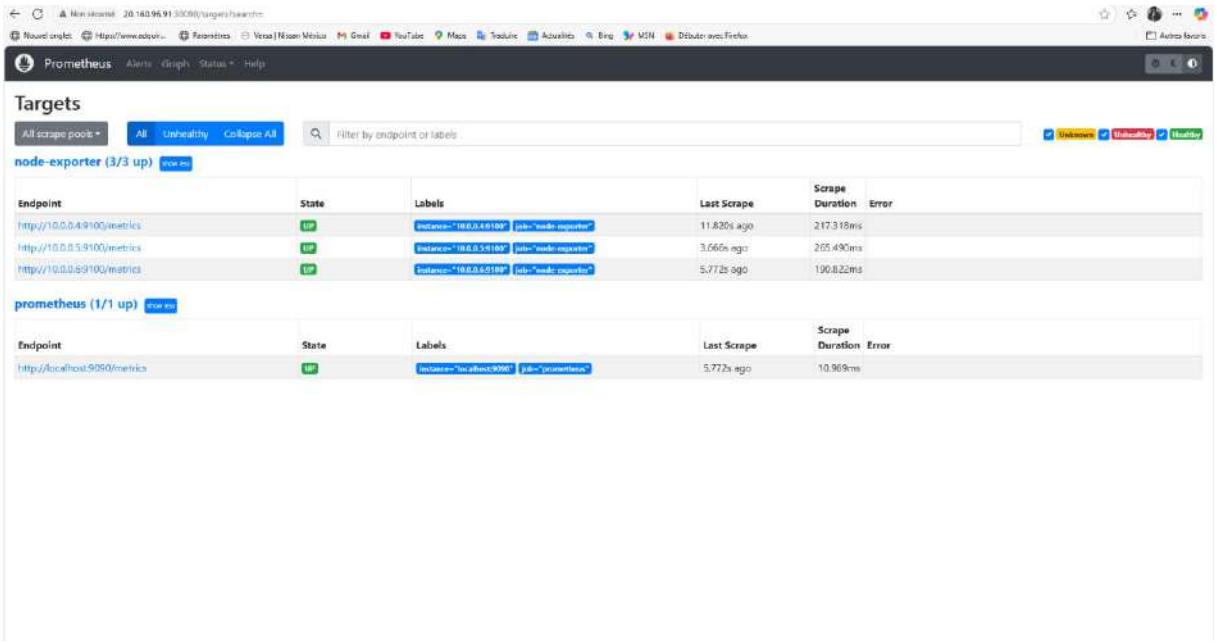
```
tar xvfz node_exporter-1.8.1.linux-amd64.tar.gz
```

```
sudo mv node_exporter-1.8.1. linux-amd64/node_exporter /usr/local/bin/
```

```
# Lancer le service
```

```
sudo nohup node_exporter &
```

Capture :



Cette capture de l'interface Prometheus (http://<IP_VM>:9090/targets) montre que les trois nœuds du cluster exposent correctement leurs métriques système via le port 9100, grâce au service node_exporter. Prometheus effectue avec succès le **scraping** des trois endpoints :

10.0.0.4:9100 (probablement admin-master)

10.0.0.5:9100 (worker1)

10.0.0.6:9100 (worker2-nfs)

Cela confirme que la supervision est active sur tous les nœuds du cluster Kubernetes.

Configuration de Prometheus

J'ai modifié le fichier prometheus.yml pour ajouter mes cibles manuellement :

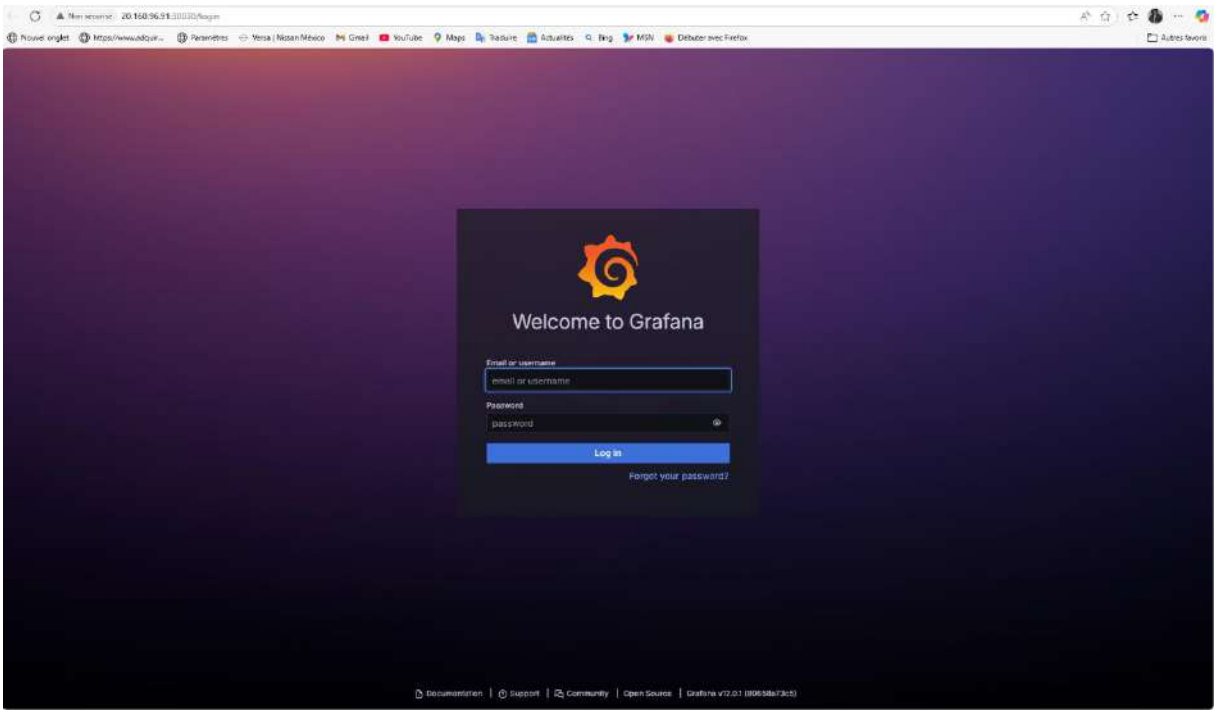
```
scrape_configs:
  - job_name: 'node'
    static_configs:
      - targets: ['admin-master:9100', 'worker1:9100', 'worker2-nfs:9100']
```

Intégration avec Grafana

```
# Installer Grafana

sudo apt install -y apt-transport-https software-properties-common

sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"

sudo apt update

sudo apt install grafana


# Démarrer le service

sudo systemctl start grafana-server

sudo systemctl enable grafana-server
```

Grafana accessible sur : http://20.160.96.91:3000
Login par défaut : admin / admin

Capture :



Cette capture montre l'accès sécurisé à l'interface de supervision Grafana, accessible via le port 3000. L'administrateur doit s'authentifier avec un identifiant et un mot de passe pour accéder aux dashboards et aux sources de données.
Cette interface permet de visualiser graphiquement les métriques collectées par Prometheus.

Dashboards personnalisés

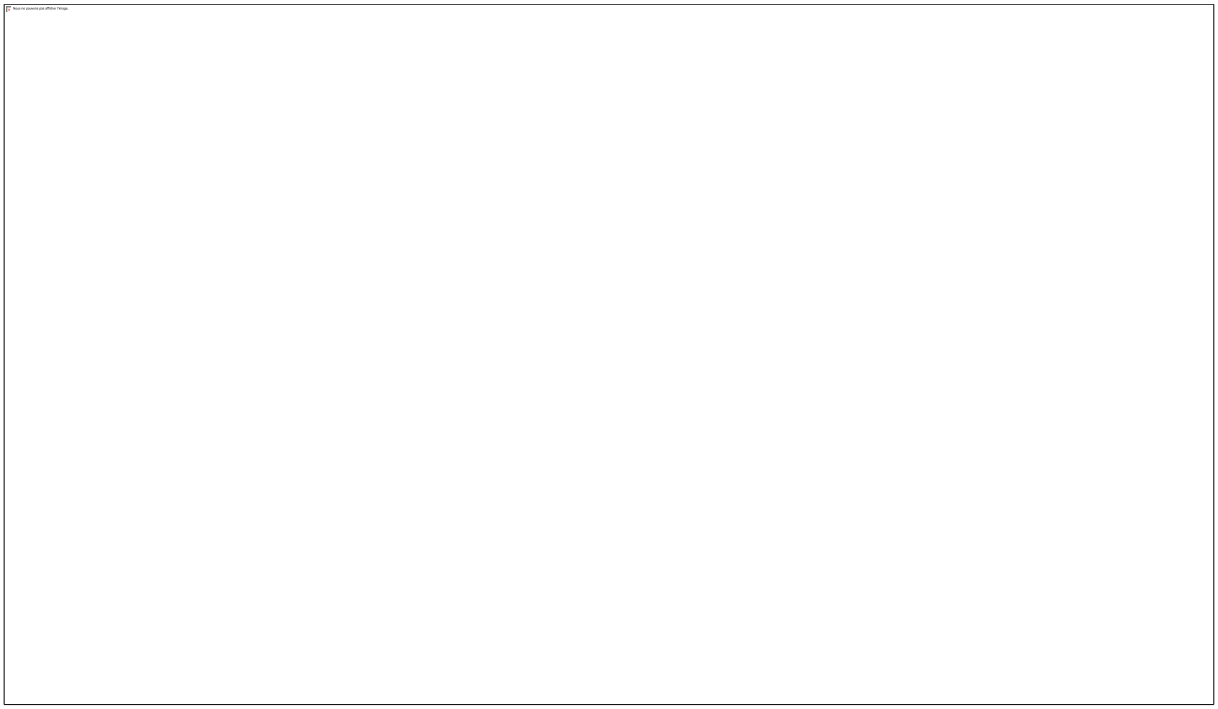
J’ai importé le **dashboard officiel Node Exporter Full (ID 1860)**, puis j’ai créé des dashboards personnalisés pour :

L’état des pods Kubernetes

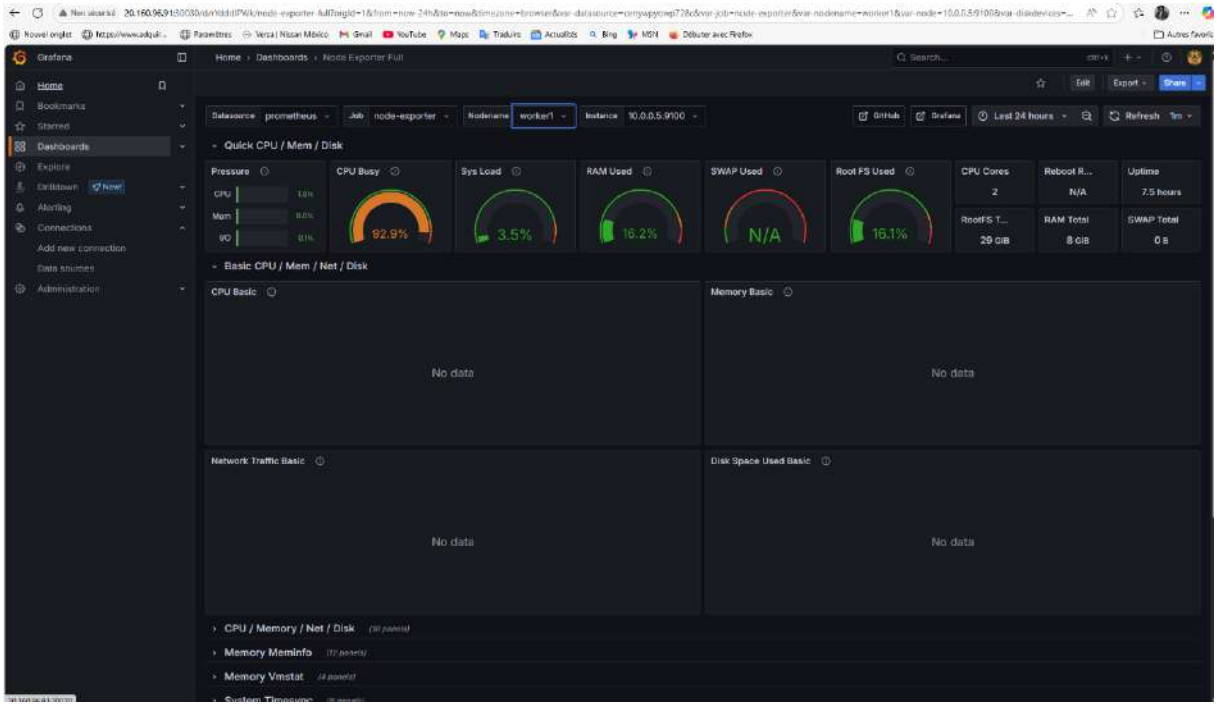
L’utilisation CPU/mémoire/disque par nœud

L’état des volumes persistants NFS

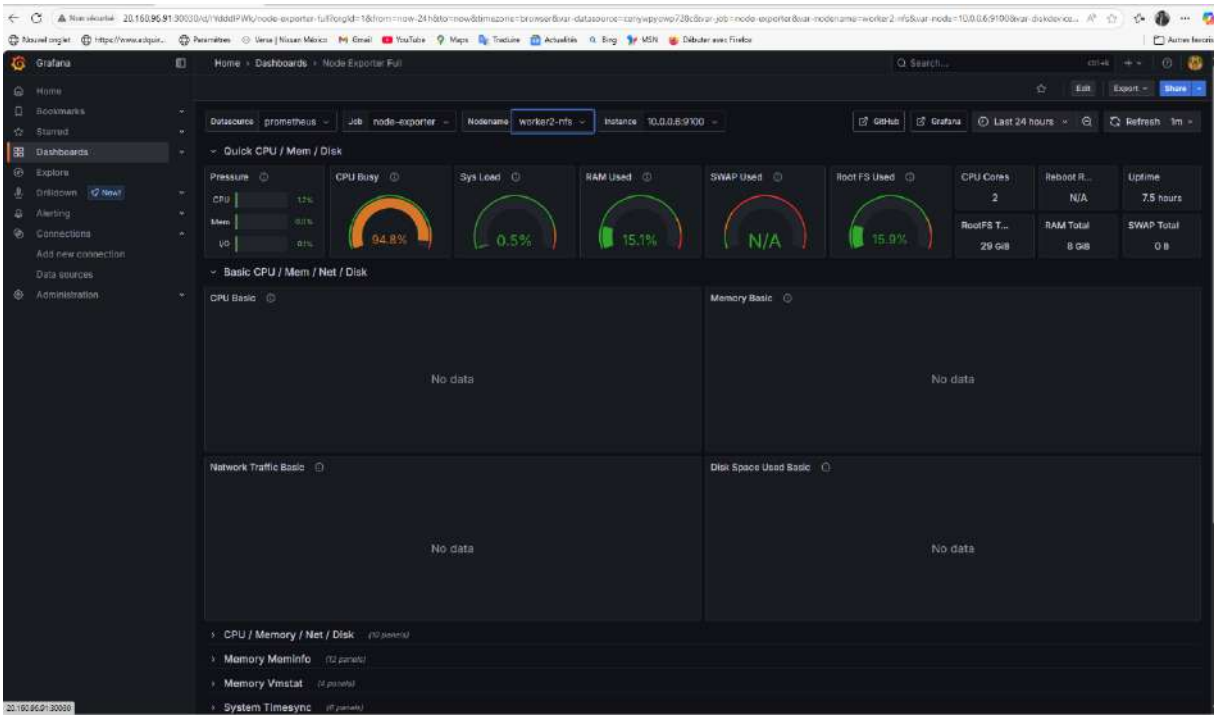
Capture :



Ce panneau montre l’état des ressources système sur le nœud admin-master. On observe ici un taux d'utilisation CPU élevé, une RAM utilisée à 34,7 % et un espace disque racine utilisé à 19,8 %.



Le nœud worker1 présente une activité CPU importante (92,9 %) et une RAM utilisée à 16,2 %, indiquant un usage modéré des ressources.



Le nœud worker2-nfs, hébergeant le serveur NFS et potentiellement la base de données MySQL, utilise 15,1 % de RAM avec un CPU très sollicité (94,8 %), ce qui reflète bien son rôle actif dans le cluster.

Résultat obtenu

Cette supervision m'a permis de :

Surveiller en temps réel l'utilisation des ressources sur chaque nœud

Identifier les anomalies de consommation (CPU, RAM, disque)

Analyser l'état de santé de mon cluster Kubernetes

Garantir une infrastructure **plus fiable et proactive**

Grâce à ce stack de supervision, j'ai appliqué concrètement une **démarche DevOps de type monitoring observabilité**, essentielle pour maintenir un service de qualité.

.

Vérification finale de l'infrastructure

Une fois tous les composants installés, j'ai réalisé une série de **vérifications fonctionnelles et techniques** pour m'assurer que l'ensemble de l'infrastructure était opérationnel, stable et cohérent. Cette phase de validation a été essentielle pour confirmer la **résilience** de la plateforme, avant de pouvoir la présenter dans un cadre professionnel.

Contrôles réalisés

État du cluster Kubernetes

J'ai exécuté la commande suivante pour m'assurer que les trois nœuds étaient bien reconnus et fonctionnels :

```
kubectl get nodes
```

Tous les nœuds (admin-master, worker1, worker2-nfs) apparaissent avec le statut **Ready**, preuve que la communication est opérationnelle entre eux.

J'ai également vérifié l'état des pods :

```
kubectl get pods -A
```

Tous les composants déployés (WordPress, MySQL, Jenkins, Prometheus, Grafana, etc.) sont en **Running**, sans redémarrages inattendus (RESTARTS = 0).

Volumes persistants

J’ai contrôlé que les **PersistentVolumeClaims (PVC)** sont bien en statut Bound, reliés aux volumes NFS partagés :

```
kubectl get pvc -n wordpress
```

Les données de WordPress et MySQL sont bien **persistantes**, même après un redémarrage des pods ou du cluster.

Accès aux applications

Le site WordPress est bien **accessible publiquement** via le port NodePort exposé sur worker1, à l’adresse :
http://<IP_worker1>:30080

L’interface Jenkins est disponible via HTTPS, grâce au **reverse proxy Nginx** avec **Let’s Encrypt** :

https://<IP_admin-master>

Les certificats sont valides, la redirection HTTP → HTTPS fonctionne, et l’accès administrateur est sécurisé.

Chaîne CI/CD opérationnelle

J’ai vérifié que le **pipeline Jenkins** se déclenche bien à chaque **push GitHub**, via un job de type *multibranch pipeline*.

Le Jenkinsfile est bien pris en compte :

Build de l’image Docker

Tag en fonction de la branche (: dev, :stable)

Push vers **DockerHub**

Déploiement automatique via kubectl apply

J’ai validé toutes ces étapes manuellement, avec des logs Jenkins à l’appui.

Supervision fonctionnelle

Prometheus récupère les métriques de chaque VM via Node Exporter (:9100).

Dans **Grafana**, j’ai ajouté Prometheus comme source de données (http://localhost:9090), et j’ai importé le dashboard **Node Exporter Full (ID 1860)**.

Les dashboards m’ont permis de visualiser en temps réel :

CPU, RAM, espace disque

État des pods et services

Activité réseau et latence

Conclusion

Grâce à ces vérifications complètes, j'ai pu **valider la stabilité et la cohérence globale** de l'architecture mise en place.
Le système est désormais :

Stable, avec un cluster Kubernetes fonctionnel et bien dimensionné

Sécurisé, grâce à l'usage de secrets, HTTPS, et des outils de supervision

Automatisé, via une chaîne CI/CD efficace

Persévérant, grâce au stockage NFS permettant la persistance des données

Cette étape m'a permis de **clôturer la partie technique** du projet en m'assurant que toutes les fonctionnalités attendues étaient réunies pour répondre aux objectifs pédagogiques, mais aussi **prêtes pour un usage professionnel**.

.

Technologies et outils utilisés

Voici la liste des **technologies, langages et outils** que j'ai utilisés dans le cadre de ce projet. Chacun a été sélectionné avec soin pour répondre aux besoins d'une architecture DevOps moderne, réaliste et proche d'un environnement de production.

Infrastructure & Systèmes

Microsoft Azure
J'ai utilisé Azure pour héberger mes machines virtuelles, configurer le réseau et sécuriser l'infrastructure via des groupes de sécurité réseau (NSG).

Ubuntu Server 22.04
Ce système d'exploitation m'a permis de bénéficier d'un environnement Linux stable et compatible avec les outils DevOps.

Conteneurisation & Orchestration

Docker
J'ai utilisé Docker pour construire des images personnalisées, notamment celle de WordPress avec un thème préinstallé, et exécuter les conteneurs.

Kubernetes (K3s)
J'ai opté pour K3s, une version légère de Kubernetes, idéale pour mon cluster auto-hébergé, permettant l'orchestration des pods, la gestion des volumes et du réseau.

Infrastructure as Code (IaC)

Ansible
J'ai automatisé toutes les étapes du déploiement avec Ansible : installation de K3s, configuration du NFS, déploiement de Jenkins, etc.
J'ai structuré mes rôles et mes playbooks de manière modulaire pour une meilleure réutilisabilité.

CI/CD – Intégration et Déploiement Continu

Jenkins
J'ai installé Jenkins sur mon nœud master pour mettre en place une pipeline CI/CD complète avec build, test, push Docker et déploiement Kubernetes.

GitHub
J'ai hébergé le code source, les fichiers Dockerfile, Jenkinsfile et YAML de déploiement dans un dépôt GitHub, utilisé comme déclencheur des pipelines.

DockerHub
Jenkins pousse automatiquement les images Docker générées sur mon compte DockerHub après chaque build.

Supervision & Monitoring

Prometheus
J'ai configuré Prometheus pour collecter les métriques des nœuds et des pods.

Node Exporter
Installé sur chaque VM, il expose les métriques système (CPU, RAM, disque) à Prometheus.

Grafana
J'ai utilisé Grafana pour visualiser les données sous forme de dashboards, en important notamment le tableau "Node Exporter Full".

Stockage & Persistance

NFS (Network File System)
J'ai mis en place un serveur NFS sur le nœud worker2-nfs pour stocker de manière persistante les données de WordPress et MySQL, même après redémarrage des pods.

Outils complémentaires

kubectl : Outil CLI indispensable pour interagir avec le cluster Kubernetes.

openssl / certbot : Utilisés pour générer et gérer les certificats TLS via Let's Encrypt, assurant un accès sécurisé en HTTPS.

Visual Studio Code : Mon éditeur principal pour écrire et maintenir les fichiers YAML, les scripts Ansible et les pipelines Jenkins.

Choix technologiques

Tous ces outils ont été choisis pour leur **fiabilité, compatibilité cloud, simplicité d'intégration et pertinence** dans un contexte DevOps. Ce projet m'a permis de les combiner efficacement pour livrer une infrastructure automatisée, supervisée, et prête pour un usage professionnel.

.

Résultat final obtenu

Le résultat final de ce projet est un **environnement de production complet, automatisé et fonctionnel**, construit selon les **standards professionnels de l'ingénierie DevOps**. J'ai réussi à intégrer l'ensemble des briques techniques essentielles pour déployer, sécuriser et superviser une application web en cluster Kubernetes.

Voici les principales réalisations concrètes que j'ai mises en place :

Un cluster Kubernetes K3s entièrement opérationnel, composé de **3 nœuds** (1 master et 2 workers), déployé et configuré automatiquement grâce à **Ansible**.

Une application WordPress e-commerce connectée à une **base MySQL**, avec une persistance de données assurée via un **stockage NFS** centralisé.

Une chaîne CI/CD automatisée reposant sur **Jenkins, GitHub et DockerHub**, permettant de builder et déployer automatiquement des images Docker sur le cluster à chaque commit.

Un système de supervision efficace, combinant **Prometheus**, **Node Exporter** et **Grafana**, pour surveiller en temps réel les performances du cluster et des pods.

Un accès sécurisé aux interfaces web, grâce à **Nginx en reverse proxy** et l'utilisation de **certificats HTTPS Let's Encrypt**.

Cette infrastructure répond pleinement aux **objectifs pédagogiques du Titre Professionnel ASDEV**, tout en intégrant des **bonnes pratiques réelles du monde professionnel** en matière de cloud, d'automatisation, de résilience et de supervision.

Elle est conçue pour **évoluer facilement** : je pourrais y intégrer de nouveaux services, l'adapter à un usage multi-environnements (test/production), ou même la **migrer vers une solution cloud managée comme AKS ou GKE** pour une montée en charge professionnelle.

Supervision et CI/CD

La supervision et l'automatisation des déploiements sont les **deux piliers majeurs** de l'infrastructure que j'ai mise en place. Elles ont garanti la **visibilité**, la **stabilité** et l'**agilité** du projet tout au long de sa réalisation.

Supervision avec Prometheus & Grafana

J'ai configuré **Prometheus** pour collecter en continu les **métriques système** de chaque machine du cluster, grâce à l'outil **Node Exporter**. Ces données sont ensuite visualisées dans **Grafana** via des tableaux de bord dynamiques que j'ai personnalisés.

Les indicateurs clés que j'ai surveillés :

L'utilisation **CPU et RAM** de chaque nœud

L'**espace disque** consommé

L'**état des pods et services Kubernetes**

Cette supervision m'a permis d'**anticiper les goulets d'étranglement**, d'**ajuster les ressources**, et d'avoir un contrôle précis sur la santé de mon infrastructure.

Intégration et déploiement continus avec Jenkins

L'automatisation des déploiements a été assurée grâce à **Jenkins**, configuré avec un **pipeline "as code"** dans un fichier Jenkinsfile. À chaque modification poussée dans **GitHub**, Jenkins déclenche automatiquement :

Le **build de l'image Docker personnalisée de WordPress**

Le **tag et le push** vers **DockerHub** (:test, :stable)

Le **déploiement dans le cluster** Kubernetes via kubectl apply

Deux branches principales sont gérées :

main pour la **production**

dev pour les **tests**

Cette chaîne **CI/CD complète** m'a permis :

D'**automatiser** tout le processus de mise à jour

De **réduire les erreurs humaines**

D'**accélérer les itérations**, comme dans un vrai environnement DevOps d'entreprise

Résultat final obtenu

À l'issue de ce projet, j'ai déployé un **environnement de production complet, stable et automatisé**, construit selon les **standards professionnels du DevOps**.

Réalisations concrètes :

Un **cluster Kubernetes K3s** avec 3 nœuds (1 master + 2 workers), automatisé via **Ansible**
Une **application WordPress e-commerce** connectée à **MySQL**, avec persistance via **NFS**
Une **chaîne CI/CD complète** avec **Jenkins, GitHub et DockerHub**
Une **supervision centralisée** avec **Prometheus + Grafana**
Un **accès sécurisé HTTPS** via **Nginx reverse proxy + Let's Encrypt**

Cette architecture répond aux **exigences techniques du Titre Professionnel ASDEV** et peut **évoluer** vers un environnement managé (AKS, GKE), ou intégrer d'autres services.

Difficultés rencontrées et résolutions

Tout au long de ce projet, j'ai été confronté à plusieurs **obstacles techniques et organisationnels**. Ces difficultés ont été autant d'occasions de **monter en compétence**, de **renforcer ma rigueur**, et d'adopter une **démarche professionnelle de résolution de problèmes**.

Problème 1 : Connexion SSH impossible entre les machines

Symptôme : les playbooks Ansible échouaient car la connexion SSH échouait entre les VMs.

Analyse : les **clés SSH** étaient mal distribuées, et les **ports 22** bloqués au niveau du **NSG Azure**.

Solution :

Génération d'une nouvelle paire de clés (ssh-keygen)

Modification des règles **NSG Azure** pour autoriser le port 22

Injection correcte des clés publiques dans les fichiers ~/.ssh/authorized_keys

Problème 2 : Échec de l'installation K3s sur les workers

Symptôme : les nœuds worker1 et worker2-nfs ne rejoignaient pas le cluster K3s.

Analyse : mauvaise adresse IP renseignée dans la variable K3S_URL.

Solution :

Vérification des IPs avec ping et ip a

Correction des rôles Ansible

Redéploiement propre des nœuds avec un playbook automatisé

Problème 3 : Volumes NFS non montés

Symptôme : les pods WordPress et MySQL restaient en état **Pending**.

Analyse : les clients NFS (admin-master, worker1) ne pouvaient pas monter les volumes.

Cause : le paquet nfs-common était manquant.

Solution :

Installation de nfs-common via Ansible

Vérification du montage avec mount et df -h

Création de fichiers de test pour valider l'accès au dossier NFS

Problème 4 : Pipeline Jenkins non déclenché

Symptôme : Jenkins ne lançait aucun job après un push GitHub.

Analyse : les **webhooks GitHub** étaient absents et les **credentials mal configurés**.

Solution :

Création et ajout des **credentials sécurisés** (GitHub PAT + DockerHub)

Activation des **webhooks** dans GitHub

Test de déclenchement par commit/push dans la branche main

Problème 5 : Échec du pod git-sync dans le runner Ansible

Symptôme : le pod Ansible runner n'arrivait pas à cloner le dépôt GitHub.

Analyse : absence de **variables d'environnement** GIT_SYNC_USERNAME et GIT_SYNC_PASSWORD.

Solution :

Génération d’un nouveau **Personal Access Token (PAT)** GitHub

Définition des variables d’environnement dans le Deployment

Redéploiement du pod et vérification du bon clonage

Ce que ces problèmes m'ont appris

Chacune de ces situations a été pour moi :

Un **exercice concret de troubleshooting** en conditions réelles

Une opportunité de **renforcer mes connaissances techniques**

Une incitation à adopter une **méthodologie rigoureuse et documentée**

J’ai appris à **rechercher activement les causes racines**, à tester mes solutions de manière itérative, et à **m’auto-former en autonomie**. Ces compétences sont précieuses pour tout administrateur système DevOps.

Bilan personnel

Ce projet a représenté **bien plus qu’un simple exercice technique**. Il a été une véritable **expérience immersive**, qui m’a permis de consolider mes compétences dans les domaines clés de l’**administration système**, de l’**automatisation** et du **DevOps**.

En travaillant sur une infrastructure aussi **réaliste et complète**, j’ai été confronté à des **situations concrètes** proches de celles rencontrées en entreprise : erreurs de configuration, gestion des accès, déploiement de services critiques, supervision et automatisation des déploiements.

Compétences acquises

Grâce à ce projet, j’ai appris à :

Déployer un cluster Kubernetes auto-hébergé, stable et sécurisé, en partant de zéro

Automatiser l’infrastructure avec Ansible, de façon modulaire et réutilisable

Mettre en place une chaîne CI/CD fiable avec Jenkins, GitHub et DockerHub

Superviser l’infrastructure avec Prometheus, Grafana et Node Exporter

Diagnostiquer, corriger et documenter des erreurs complexes de production

Mais surtout, j'ai renforcé ma **capacité d'apprentissage autonome**, ma **rigueur**, et ma faculté à **travailler avec méthode et persévérance**, comme on l'exige en contexte professionnel.

Un tournant dans mon parcours

Ce projet marque une **étape déterminante dans ma reconversion** vers les métiers de l'ingénierie système et DevOps. Il m'a permis de prendre **confiance en mes capacités techniques**, de **valider mes acquis** sur le terrain, et de me **préparer concrètement à intégrer une équipe en entreprise**.

Je ressors de cette expérience avec un **socle solide**, une **vision claire** des bonnes pratiques DevOps, et l'envie de continuer à progresser, apprendre et contribuer à des projets d'infrastructure modernes et automatisés.

Améliorations possibles

Même si le projet est **déjà abouti, stable et fonctionnel**, plusieurs **pistes d'amélioration** ont été identifiées pour renforcer davantage sa **sécurité**, sa **résilience** et sa **scalabilité**, en accord avec les **bonnes pratiques DevOps modernes**.

Sécurité

Implémentation de politiques RBAC personnalisées afin de restreindre les permissions accordées aux différents composants du cluster Kubernetes.

Intégration d'un gestionnaire de secrets tel que **HashiCorp Vault**, **Sealed Secrets** ou **External Secrets Operator** pour sécuriser les données sensibles (tokens, mots de passe, clés API…).

Fiabilité et tests

Ajout de **probes de liveness et readiness** dans les manifestes Kubernetes pour améliorer la supervision native des pods et permettre un **redémarrage automatique** en cas de dysfonctionnement.

Intégration de **tests de performance** dans le pipeline CI/CD, à l'aide d'outils comme **k6** ou **curl**, pour vérifier la **disponibilité** et **la montée en charge** de l'application.

Packaging et automatisation

Conversion des fichiers YAML en **charts Helm** pour une meilleure **standardisation**, **réutilisabilité** et **gestion des configurations par environnement** (dev, prod, staging).

Création de **Makefiles** pour centraliser et automatiser les tâches récurrentes : build, push, apply, test.

GitOps et déploiement continu

Passage à une approche **GitOps** avec des outils comme **ArgoCD** ou **Flux**, afin de piloter les déploiements Kubernetes directement depuis GitHub, de manière **déclarative, traçable et synchronisée**.

Supervision avancée

Intégration d'**Alertmanager** avec Prometheus pour recevoir des **notifications automatiques** en cas d'anomalie (CPU élevé, pod non disponible…).

Ajout d'un outil de **centralisation des logs** tel que **Loki**, pour permettre une **analyse rapide des incidents** et faciliter le débogage des applications.

Conclusion partielle : Ces améliorations constituent des évolutions naturelles pour **passer à un niveau de maturité supérieure** (niveau 3 à 4 DevOps). Elles permettraient de rendre l'environnement encore plus **fiable, maintenable et industrialisé**, en préparation à une exploitation réelle en entreprise ou en production.

.

Références

Afin de garantir la qualité, la cohérence et la fiabilité de mon projet, je me suis appuyé sur de nombreuses sources officielles et ressources techniques reconnues, couvrant aussi bien les outils utilisés que les bonnes pratiques DevOps.

Documentation officielle

Kubernetes : https://kubernetes.io/docs/

Ansible : https://docs.ansible.com/

Jenkins : https://www.jenkins.io/doc/

Docker : https://docs.docker.com/

Helm Charts : https://helm.sh/

GitHub : https://docs.github.com/

Microsoft Azure (VMs, NSG, AKS, stockage…) : https://learn.microsoft.com/fr-fr/azure/

Supervision et observabilité

Prometheus : https://prometheus.io/

Grafana: https://grafana.com/

Alertmanager: https://prometheus.io/docs/alerting/latest/alertmanager/

Loki (centralisation des logs) : https://grafana.com/oss/loki/

Ressources complémentaires

Articles techniques : Medium, Dev.to, blogs spécialisés DevOps

Communautés : solutions et échanges sur Stack Overflow

Dépôts GitHub publics : pour consulter des exemples de Jenkinsfile, de manifestes Helm ou de rôles Ansible

Ces ressources m'ont été indispensables pour approfondir mes connaissances, résoudre les problèmes techniques rencontrés et construire une infrastructure solide, sécurisée et évolutive, selon les standards professionnels actuels.

Annexes

Cette section regroupe les **éléments techniques, visuels et documentaires** qui complètent ce rapport. Ils constituent à la fois des **preuves de fonctionnement**, des **supports de compréhension**, et des **ressources professionnelles**.

A. Commandes et scripts utilisés

a. Création des machines virtuelles sur Azure

```
# Création de la VM principale (admin-master)
az vm create \
  --name admin-master \
  --resource-group rg-ecom \
  --image Ubuntu2204 \
  --size Standard_B2s \
  --admin-username azureuser \
  --ssh-key-values ~/.ssh/id_rsa.pub
```

```
# Ouverture du port SSH (22) sur la VM
az vm open-port \
  --resource-group rg-ecom \
  --name admin-master \
  --port 22
```

b. Vérification de l'état du cluster K3s

```
# Afficher les nœuds du cluster
kubectl get nodes

# Afficher les pods déployés dans le namespace WordPress
kubectl get pods -n wordpress

# Afficher les détails du service WordPress
kubectl describe svc wordpress -n wordpress
```

c. Déploiement de l'application WordPress

```
# Appliquer le manifeste Kubernetes pour WordPress
kubectl apply -f k8s/app/wordpress.yaml -n wordpress

# Redémarrer le déploiement WordPress (après push ou mise à jour)
kubectl rollout restart deployment wordpress -n wordpress
```

d. Accès à l'interface de supervision (Prometheus et Grafana)

```
# Rediriger le port local 9090 vers Prometheus dans le cluster
kubectl port-forward svc/prometheus -n monitoring 9090:9090

# Rediriger le port local 3000 vers Grafana dans le cluster
kubectl port-forward svc/grafana -n monitoring 3000:3000
```

2. Jenkinsfile : Pipeline CI/CD

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'docker build -t vareladavidhugo/wordpress-k3s:latest .'
            }
        }
        stage('Push') {
            steps {
                withCredentials([string(credentialsId: 'dockerhub-token', variable: 'TOKEN')]) {
                    sh 'docker login -u vareladavidhugo -p $TOKEN'
                    sh 'docker push vareladavidhugo/wordpress-k3s:latest'
                }
            }
        }
        stage('Deploy') {
            steps {
                sh 'kubectl rollout restart deployment wordpress -n wordpress'
            }
        }
    }
}
```

3. Extraits de playbooks Ansible

a. Déploiement de K3s sur le master

```
- name: Installer K3s sur le master
  shell: curl -sfL https://get.k3s.io | sh -
  when: inventory_hostname == "admin-master"
```

b. Configuration du serveur NFS

```
- name: Installer le serveur NFS
  apt:
    name: nfs-kernel-server
    state: present

- name: Configurer les exports NFS
  copy:
    dest: /etc/exports
    content: "/srv/nfs *(rw,sync,no_subtree_check,no_root_squash)"
```

c. Installation de Jenkins

```
- name: Ajouter le dépôt Jenkins
  apt_repository:
    repo: 'deb https://pkg.jenkins.io/debian-stable binary/'
    state: present

- name: Installer Jenkins
  apt:
    name: jenkins
    state: present
    update_cache: yes
```

d. Déploiement de Prometheus via Ansible

```
- name: Appliquer les manifests Prometheus
  kubernetes.core.k8s:
    state: present
    definition: "{{ lookup('file', 'prometheus-deploy.yaml') }}"
```

B. Fichiers de configuration Kubernetes

Cette section présente les fichiers de configuration Kubernetes (YAML) utilisés dans le projet. Ils définissent les déploiements, les services, les volumes persistants, les secrets et la configuration de la supervision.

1. Manifeste du déploiement WordPress

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - name: wordpress
          image: vareladavidhugo/wordpress-k3s:latest
          ports:
            - containerPort: 80
          env:
            - name: WORDPRESS_DB_HOST
              value: mysql.wordpress.svc.cluster.local
            - name: WORDPRESS_DB_USER
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
```

```
                key: username
            - name: WORDPRESS_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: password
          volumeMounts:
            - name: wordpress-pv
              mountPath: /var/www/html
      volumes:
        - name: wordpress-pv
          persistentVolumeClaim:
            claimName: pvc-wordpress
```

2. Manifeste du déploiement MySQL

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
```

```
          env:
            - name: MYSQL_DATABASE
              value: wordpress
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: password
          volumeMounts:
            - name: mysql-pv
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-pv
          persistentVolumeClaim:
            claimName: pvc-mysql
```

3. Manifeste du déploiement Jenkins (extrait simplifié)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  namespace: devops
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
```

```
        - name: jenkins
          image: jenkins/jenkins:lts
          ports:
            - containerPort: 8080
```

4. Déclarations des Services

```
# Service pour WordPress
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  namespace: wordpress
spec:
  type: LoadBalancer
  selector:
    app: wordpress
  ports:
    - port: 80
      targetPort: 80

---

# Service pour MySQL
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: wordpress
spec:
  clusterIP: None
```

```
  selector:
    app: mysql
  ports:
    - port: 3306
      targetPort: 3306
```

5. Secrets (base64 encodé)

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: wordpress
type: Opaque
data:
  username: d29yZHByZXNz    # "wordpress"
  password: bXlwYXNzd29yZA==    # "mypassword"
```

6. PersistentVolumeClaims (PVC)

```
# PVC pour WordPress
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-wordpress
  namespace: wordpress
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  storageClassName: nfs-client
```

```
---

# PVC pour MySQL
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-mysql
  namespace: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: nfs-client
```

7. StorageClass NFS personnalisée

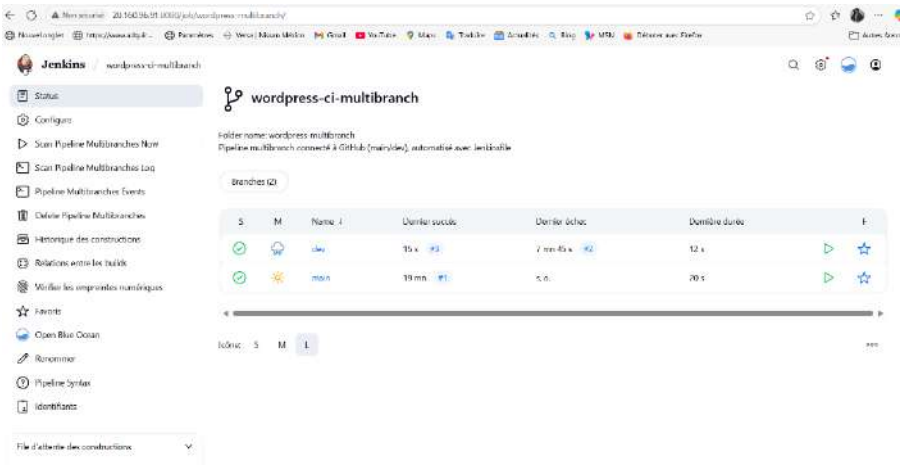
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client
provisioner: nfs.csi.k8s.io
parameters:
  server: 10.0.0.5        # IP du serveur NFS (ex: worker2-nfs)
  share: /srv/nfs
reclaimPolicy: Retain
volumeBindingMode: Immediate
mountOptions:
  - vers=4.1
```

C. Captures d’écran

Cette section présente les preuves visuelles du bon fonctionnement de l’infrastructure, des outils CI/CD, de la supervision et du stockage persistant.

1. Interface Jenkins

Capture : Interface Jenkins



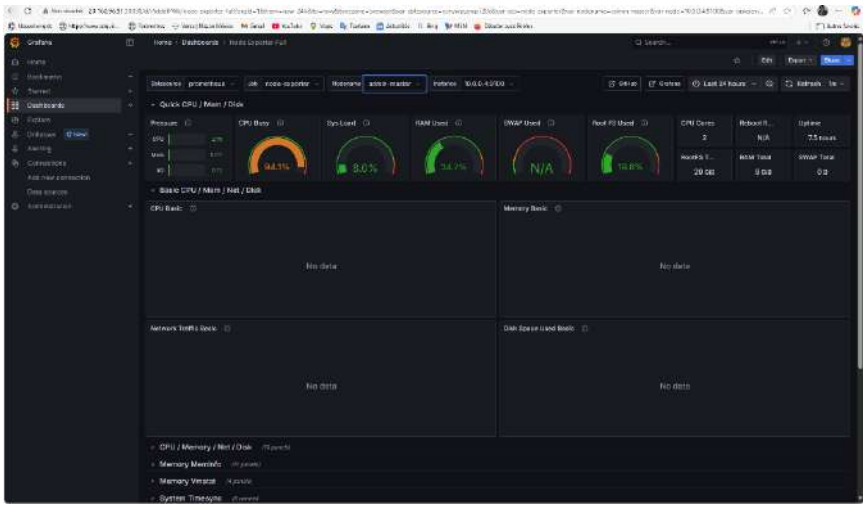
Vue de Jenkins avec un **pipeline multibranche déclenché automatiquement**.

Montre le déclenchement suite à un git push.

Présente les étapes : Build, Push, Deploy, toutes en **succès**.

2. Dashboard Grafana

Capture : Dashboard Grafana



Affiche le dashboard **Node Exporter Full** ou un dashboard **personnalisé WordPress/MySQL**. Mots-clés visibles : CPU, RAM, filesystem, uptime, Pod status, etc.

3. Sorties de commandes Kubernetes

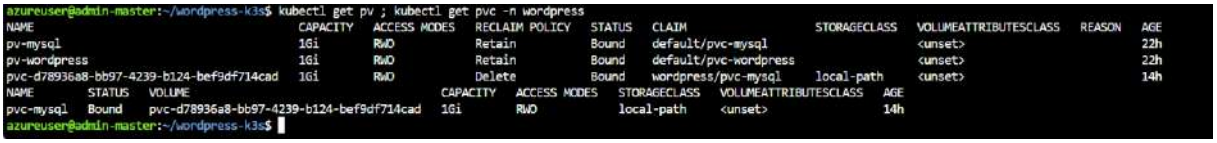
Capture : Résultat de kubectl get pods -n wordpress

```
azureuser@admin-master:~$ kubectl get pods -n wordpress
NAME                         READY   STATUS    RESTARTS      AGE
mysql-8775dd8b9-g5pxf        1/1     Running   1 (13m ago)   10h
wordpress-7978696d98-h7h2f   1/1     Running   1 (13m ago)   10h
azureuser@admin-master:~$
```

kubectl get pods -n wordpress

- Montre que les pods **WordPress** et **MySQL** sont en **Running**.

Capture : Résultat de kubectl get pv ; kubectl get pvc -n wordpress



```
azureuser@admin-master:~/wordpress-k3s$ kubectl get pv ; kubectl get pvc -n wordpress
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pv-mysql                                   1Gi        RWO            Retain           Bound    default/pvc-mysql                       <unset>                          22h
pv-wordpress                               1Gi        RWO            Retain           Bound    default/pvc-wordpress                   <unset>                          22h
pvc-d78936a8-bb97-4239-b124-bef9df714cad   1Gi        RWO            Delete           Bound    wordpress/pvc-mysql      local-path     <unset>                          14h
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
pvc-mysql   Bound    pvc-d78936a8-bb97-4239-b124-bef9df714cad   1Gi        RWO            local-path     <unset>                 14h
azureuser@admin-master:~/wordpress-k3s$
```

kubectl get pvc -n wordpress

- Montre les **PVC liés à WordPress et MySQL** en statut **Bound** (stockage persistant opérationnel).

Capture : Résultat de kubectl get svc -n default

```
azureuser@admin-master:~/wordpress-k3s$ kubectl get svc -n default
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.43.0.1       <none>        443/TCP        7h19m
mysql        ClusterIP   10.43.21.170    <none>        3306/TCP       6h54m
wordpress    NodePort    10.43.229.128   <none>        80:30080/TCP   6h54m
azureuser@admin-master:~/wordpress-k3s$
```

kubectl get svc -n default

- Montre les services exposés (wordpress, mysql) avec leurs **ClusterIP** et type (LoadBalancer ou ClusterIP).

4. Commande df -h sur les VMs avec NFS

📸 Capture : Résultat de df -h sur worker1 ou worker2-nfs

```
azureuser@worker1:~$ df -h | grep /mnt/nfs-test
10.0.0.6:/srv/nfs/kubedata   29G  4.8G   25G  17% /mnt/nfs-test
azureuser@worker1:~$
```

df -h | grep nfs

- Affiche les volumes **montés depuis le serveur NFS** sur les nœuds du cluster.
- Vérifie que /srv/nfs est bien utilisé et monté.

5. Statut des services critiques

📸 Capture : Status des services via systemctl status

```
azureuser@worker2-nfs:~$ systemctl status nfs-kernel-server
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
    Drop-In: /run/systemd/generator/nfs-server.service.d
             └─order-with-mounts.conf
     Active: active (exited) since Fri 2025-06-06 08:38:57 UTC; 7min ago
    Process: 667 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
    Process: 690 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
   Main PID: 690 (code=exited, status=0/SUCCESS)
        CPU: 8ms

Jun 06 08:38:56 worker2-nfs systemd[1]: Starting NFS server and services...
Jun 06 08:38:57 worker2-nfs systemd[1]: Finished NFS server and services.
```

Sur worker2-nfs

systemctl status nfs-kernel-server

Sur admin-master

systemctl status jenkins

```
[sudo] password for azureuser:
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2025-06-06 08:39:35 UTC; 10h ago
   Main PID: 623 (java)
      Tasks: 52 (limit: 9463)
     Memory: 952.8M
        CPU: 4min 22.559s
     CGroup: /system.slice/jenkins.service
             └─623 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jun 06 08:39:34 admin-master jenkins[623]: 2025-06-06 08:39:34.764+0000 [id=50]       INFO       hudson.util.Retrier#start: Attempt #1 to do the action check updates server
Jun 06 08:39:34 admin-master jenkins[623]: 2025-06-06 08:39:34.938+0000 [id=30]       INFO       jenkins.InitReactorRunner$1#onAttained: Completed initialization
lines 1-17...skipping...
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2025-06-06 08:39:35 UTC; 10h ago
   Main PID: 623 (java)
      Tasks: 52 (limit: 9463)
     Memory: 952.8M
        CPU: 4min 22.559s
     CGroup: /system.slice/jenkins.service
             └─623 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jun 06 08:39:34 admin-master jenkins[623]: 2025-06-06 08:39:34.764+0000 [id=50]       INFO       hudson.util.Retrier#start: Attempt #1 to do the action check updates server
Jun 06 08:39:34 admin-master jenkins[623]: 2025-06-06 08:39:34.938+0000 [id=30]       INFO       jenkins.InitReactorRunner$1#onAttained: Completed initialization
Jun 06 08:39:35 admin-master jenkins[623]: 2025-06-06 08:39:35.276+0000 [id=23]       INFO       hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
Jun 06 08:39:35 admin-master systemd[1]: Started Jenkins Continuous Integration Server.
Jun 06 08:39:43 admin-master jenkins[623]: 2025-06-06 08:39:43.673+0000 [id=50]       INFO       h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.M
Jun 06 08:39:44 admin-master jenkins[623]: 2025-06-06 08:39:44.458+0000 [id=50]       INFO       h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Ap
```

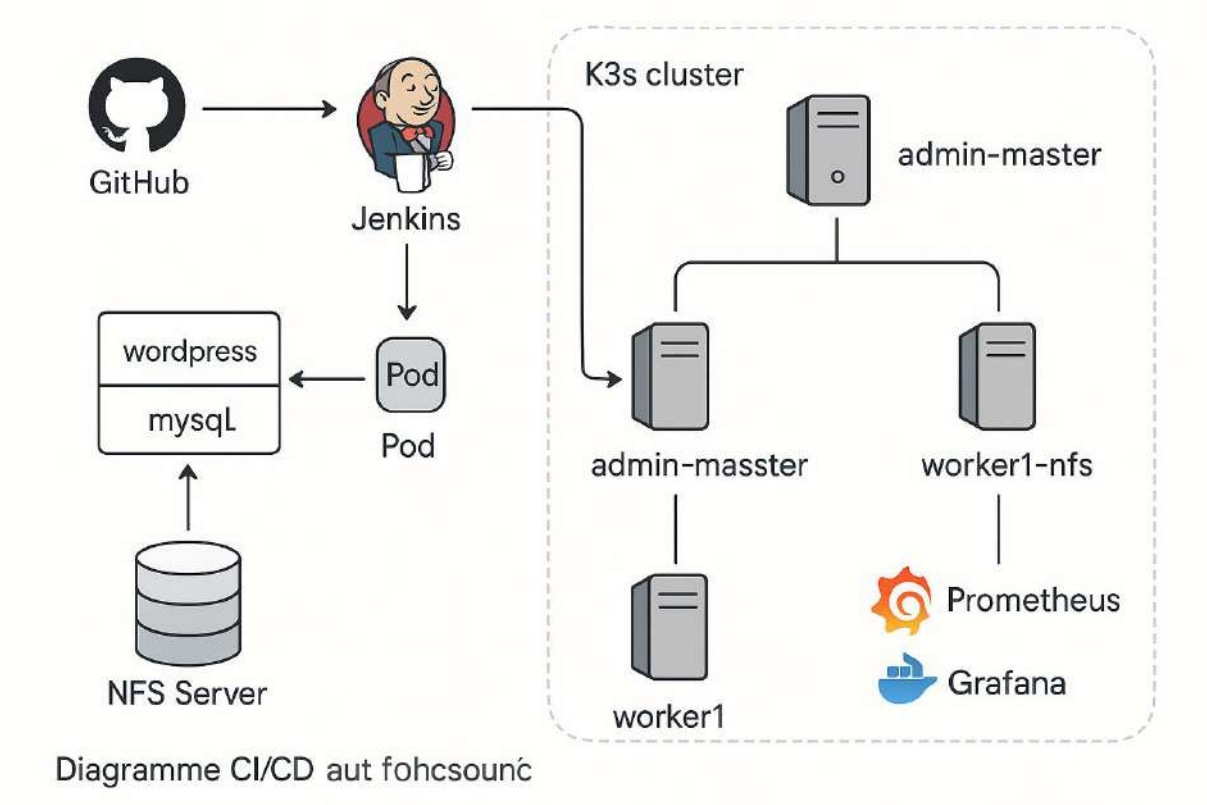
systemctl status prometheus

```
azureuser@admin-master:~$ kubectl get pods -n monitoring -l app=prometheus -o wide
NAME                          READY   STATUS    RESTARTS      AGE   IP           NODE      NOMINATED NODE   READINESS GATES
prometheus-7757bc4cb7-jwb26   1/1     Running   2 (10h ago)   47h   10.42.2.31   worker1   <none>           <none>
azureuser@admin-master:~$
```

- Montre que les services **NFS**, **Jenkins** et **Prometheus** sont **actifs et fonctionnels**.

D. Schéma de l'infrastructure

Diagramme vectoriel illustrant :



Les **3 machines virtuelles** (admin-master, worker1, worker2-nfs)

Le **cluster K3s**, Jenkins, le serveur **NFS**

La **chaîne CI/CD** complète (GitHub > Jenkins > DockerHub > Kubernetes)

Les outils de supervision (Prometheus + Grafana)

E. Liens utiles personnels

LinkedIn (profil professionnel) : https://www.linkedin.com/in/davidvarela/

GitHub (projets, playbooks, Jenkinsfiles) : https://github.com/VARELAdavidhugo

DockerHub (images utilisées dans le projet) : https://hub.docker.com/u/vareladavidhugo

Portefeuille professionnel (portfolio DevOps) : http://www.varelasolutions.fr/

Ces annexes illustrent et valident techniquement l'ensemble du projet présenté dans ce rapport.