Projet DevOps : Déploiement WordPress sur Kubernetes

Mise en place d'une infrastructure DevOps complète avec Kubernetes, CI/CD (Jenkins, GitHub, Docker Hub), stockage persistant (NFS) et supervision (Prometheus, Grafana).

Titre professionnel visé : Administrateur Systèmes DevOps (ASD)

Nom : VARELA DURAN David Hugo

Centre : M2i Formation Date : Juin 2025

Projet DevOps : Déploiement WordPress sur Kubernetes 1
Remerciements 2
Dédicace 3
Introduction 3
Objectifs pédagogiques 4
Architecture de l'infrastructure 5
Rôle des machines virtuelles 8
Schéma de l’infrastructure 11
Étapes techniques du déploiement 12
Création de l'infrastructure (Azure) 12
Installation du cluster Kubernetes (K3s) 15
Mise en place du stockage persistant (NFS) 21
Vérification finale de l’infrastructure 47
Technologies et outils utilisés 49
Supervision et CI/CD 52
Difficultés rencontrées et résolutions 53
Bilan personnel 56
Améliorations possibles 57
Références 58
Annexes 59

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 **Docker Hub** 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 **Docker Hub** pour la gestion des images.

Superviser l'infrastructure en temps réel : déployer **Prometheus**, **Node Exporter**, **Alertmanager** et **Grafana**, avec des Dashboard 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.

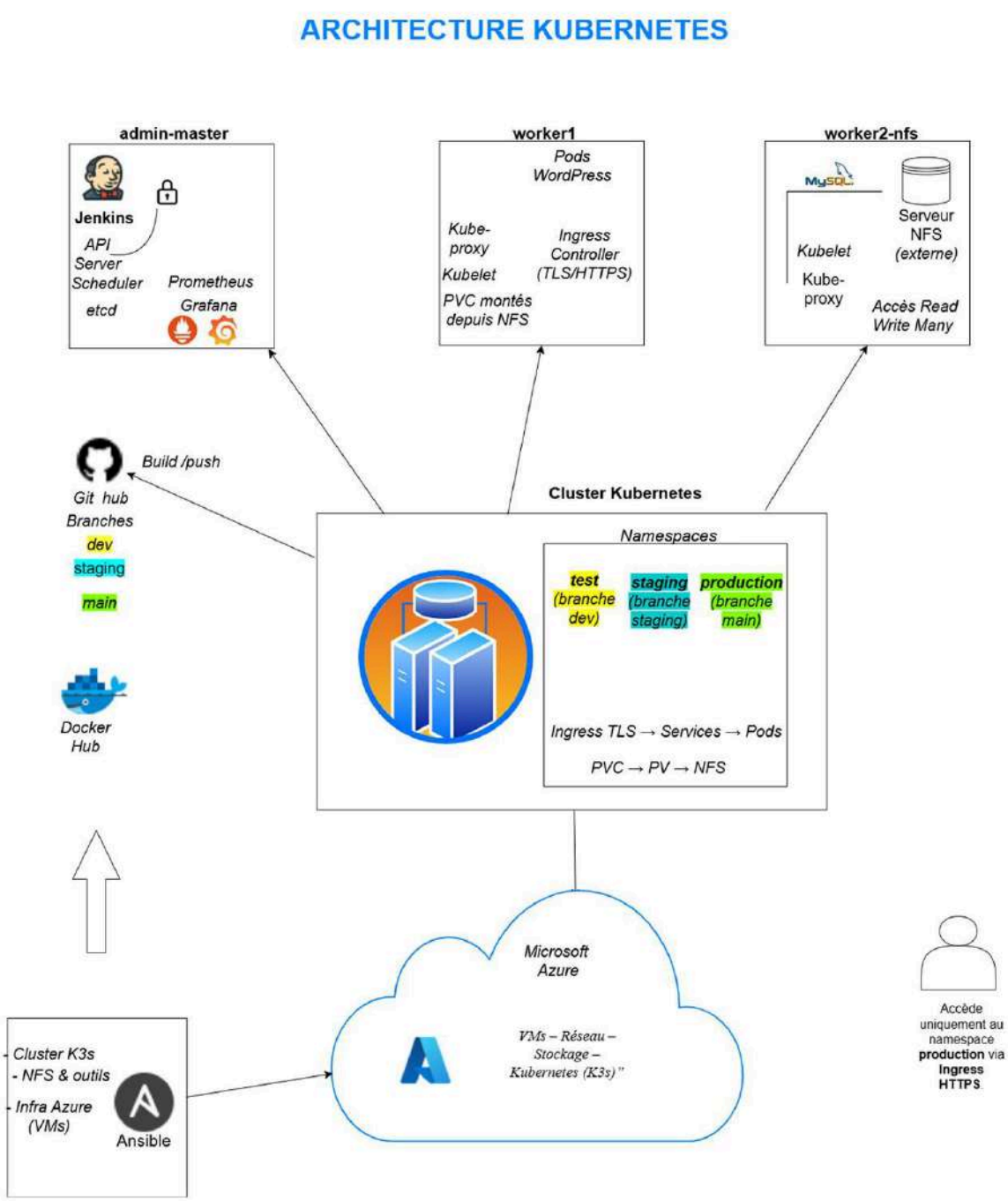
Voici la répartition :

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 illustre l'architecture de mon projet, déployée sur trois machines virtuelles Azure :

- **Admin-master** : nœud maître, hébergeant Ansible, Jenkins et les utilitaires Kubernetes (kubeadm, kubectl, Docker).
- **Worker1** : exécute WordPress avec un client NFS pour la persistance des données.
- **Worker2-nfs** : héberge MySQL et le serveur NFS pour stocker les données applicatives.
- **Supervision** : Prometheus et Grafana assurent le suivi en temps réel de la santé du cluster.

Cette architecture met en pratique les principes DevOps de **séparation des rôles, modularité, automatisation et supervision**.

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.

Rôle des machines virtuelles

Mon cluster Kubernetes repose sur trois machines virtuelles Azure, avec une répartition claire des rôles :

VM	Rôle Kubernetes	Rôle applicatif
admin-master	Master	Outils d'orchestration (Ansible, Jenkins) et supervision (Prometheus, Grafana)
Worker 1	Worker	WordPress + client NFS
Worker 2-nfs	Worker	MySQL + serveur NFS pour la persistance des données

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 utilisés dans le projet

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**.

Docker Hub, pour héberger les **images Docker personnalisées** générées par Jenkins (via Jenkins file).

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 **Dashboard 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**.

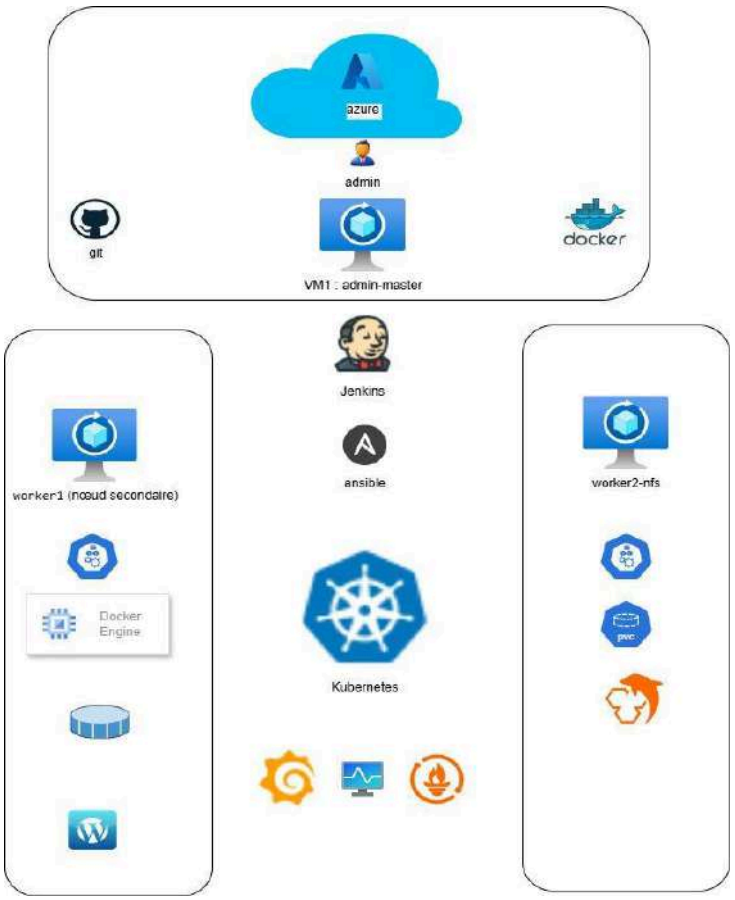


Illustration – Répartition des outils DevOps dans le cluster

Ce schéma représente la distribution des composants sur les trois machines virtuelles du cluster Kubernetes hébergé sur Microsoft Azure :

VM1 – admin-master

- Rôle : nœud maître
- Outils : Ansible (automatisation), Jenkins (CI/CD), Docker, accès GitHub

VM2 – worker1

- Rôle : nœud worker
- Outils : WordPress (frontend), Docker Engine, client NFS

VM3 – worker2-nfs

- Rôle : nœud worker + serveur de stockage
- Outils : MySQL (base de données), serveur NFS, PVC (stockage persistant)

Supervision

- Prometheus, Grafana et Node Exporter déployés dans le cluster pour l'observabilité et le monitoring

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 **Docker Hub**,

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 **Dashboard 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 CPU, 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>
```

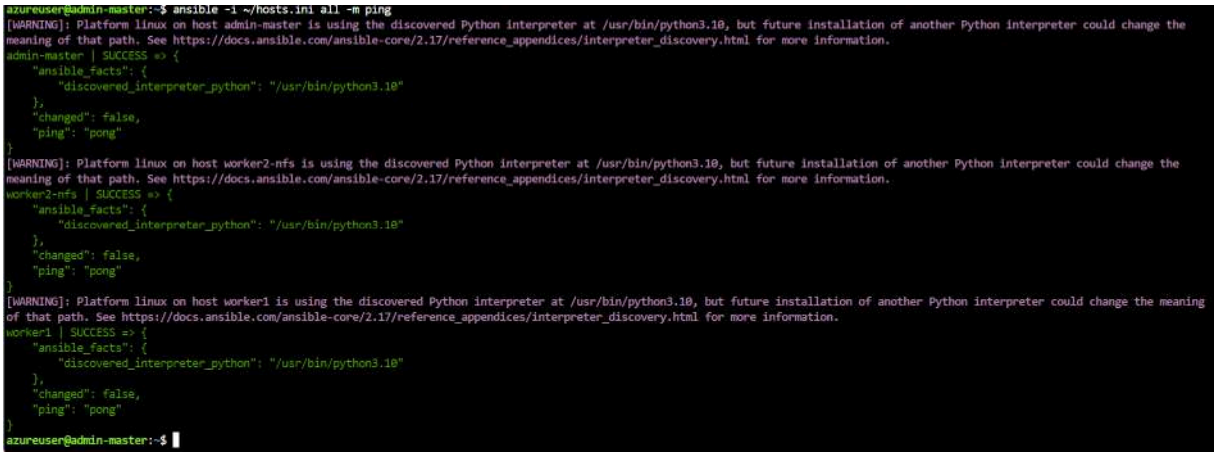
Une fois la connectivité SSH établie entre toutes les machines, j'ai validé les échanges avec Ansible en exécutant un simple ping sur l'inventaire :

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

Capture 1 – Création de la VM admin-master avec Azure CLI

Cette capture illustre la création de la machine virtuelle *admin-master* via l'outil **Azure CLI**.
La commande az vm create a été utilisée avec l'option --generate-ssh-keys pour générer automatiquement une paire de clés SSH et les injecter dans la VM, et --image Ubuntu2204 pour déployer une distribution compatible avec Docker et Kubernetes.
L'utilisation de la CLI permet de gagner en efficacité, de vérifier immédiatement les informations critiques (IP publique, nom de la VM, état) et de reproduire rapidement la création des autres nœuds du cluster.

Capture 2 – Vérification de la connectivité SSH avec Ansible

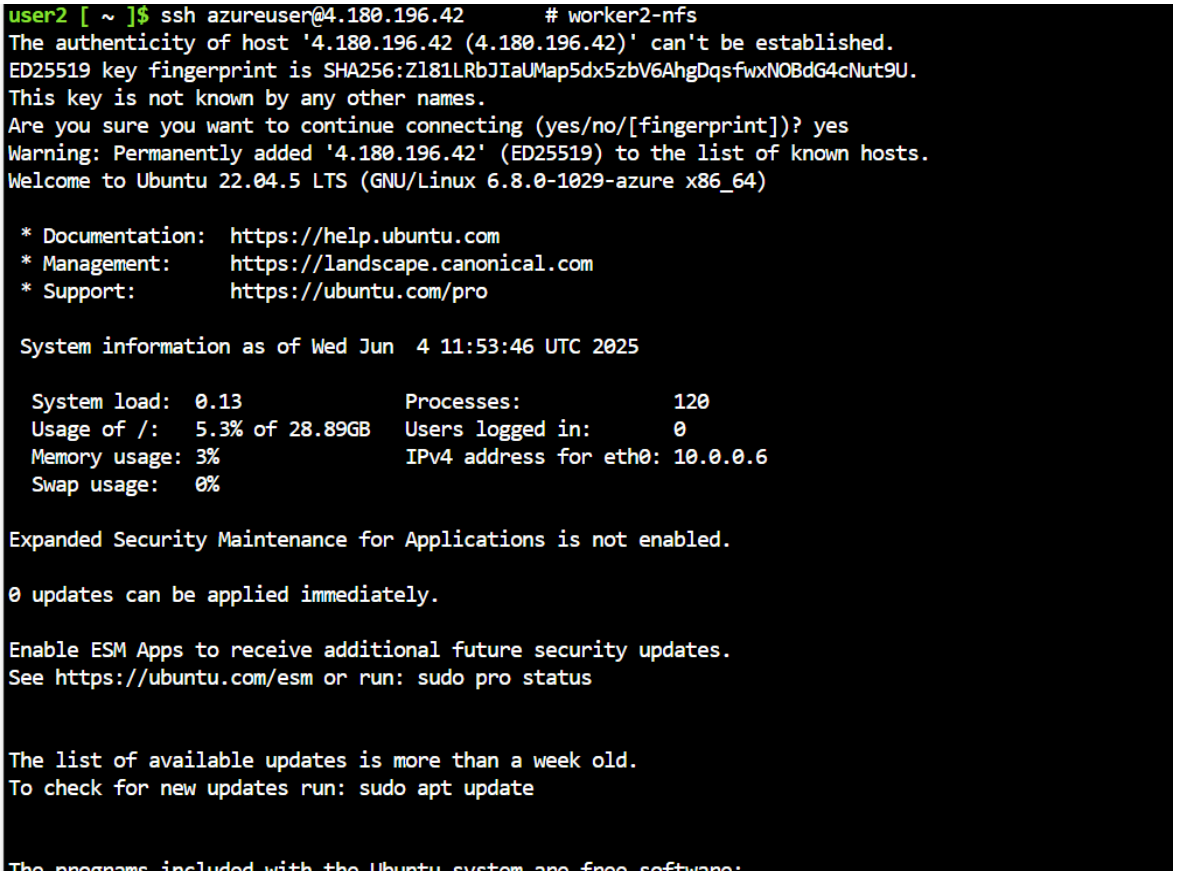


Cette capture montre le test de connectivité entre les trois machines virtuelles grâce à la commande :

ansible -i hosts.ini all -m ping

Le retour "Pong" sur chaque hôte confirme que **admin-master**, **worker1** et **worker2-nfs** sont accessibles et prêts à être administrés via Ansible.
Ce test constitue une étape de validation essentielle avant d'automatiser les déploiements.

Capture – Connexion SSH à worker2-nfs



Cette capture illustre une connexion **SSH réussie** à la machine virtuelle *worker2-nfs*.
Cela confirme que :

- La clé publique SSH a bien été injectée,
- Le port 22 est ouvert et accessible via les NSG,
- L'adresse IP publique de la VM est fonctionnelle.

Cette étape est indispensable pour préparer l'exécution des **playbooks Ansible** et garantir l'administration distante de la machine.

Installation du cluster Kubernetes (K3s)

Une fois l'infrastructure déployée et les connexions SSH validées, j'ai installé **K3s**, une distribution allégée et optimisée de Kubernetes.

Ce choix permet d'avoir un cluster **léger, rapide et fiable**, parfaitement adapté à un environnement **auto-hébergé avec ressources limitées**, comme c'est le cas ici sur Azure.

Installation du nœud master (*admin-master*)

Sur la VM *admin-master*, j'ai lancé la commande suivante :

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

Cette commande officielle :

- Télécharge et exécute le script sécurisé des mainteneurs de K3s,
- Installe le **serveur K3s**,
- Démarre automatiquement un **agent kubelet**,
- Génère le fichier kubeconfig :
- /etc/rancher/k3s/k3s.yaml

Vérification du cluster

Pour confirmer que le cluster avait bien démarré :

```
sudo kubectl get nodes
```

Résultat attendu :

- Le nœud admin-master apparaît en statut **Ready**,
- Ce qui valide l'initialisation correcte du cluster.

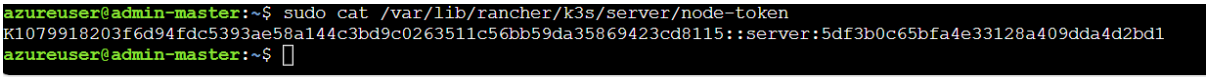
Récupération du token d'authentification

Afin de permettre aux nœuds agents (*worker1* et *worker2-nfs*) de rejoindre le cluster, j'ai récupéré le **token généré automatiquement par K3s** :

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

Ce **token**, combiné à l'**IP privée** du nœud *admin-master*, sera utilisé pour initialiser les nœuds secondaires et les intégrer au cluster.

Capture : récupération du token sur *admin-master*



La commande suivante m'a permis d'afficher le **token d'authentification généré automatiquement par K3s** lors de l'installation du nœud maître :

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

- Ce **token unique** est indispensable pour permettre aux nœuds *worker1* et *worker2-nfs* de rejoindre le cluster de manière sécurisée.
- Dans mon projet, je l'ai utilisé comme **variable d'environnement** dans mon **playbook Ansible**, ce qui m'a permis d'automatiser l'enrôlement des agents sans intervention manuelle.

Remarque : pour des raisons de sécurité, le contenu du token doit être flouté dans la version publique du rapport.

Installation sur les nœuds *worker1* et *worker2-nfs*

Sur chacun des deux nœuds secondaires, l'installation de l'agent K3s se fait grâce à la commande suivante :

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

Automatisation avec Ansible

J'ai intégré cette étape dans un playbook Ansible dédié, qui :

- Télécharge le script d'installation via curl,
- Injecte automatiquement la valeur du token et de l'IP du master,
- Exécute l'installation avec les privilèges administrateur (become: true).

Cette automatisation assure :

- La répétabilité de l'installation (utile pour un redéploiement),
- La sécurité (pas de copie manuelle du token),
- La cohérence (tous les nœuds agents rejoignent le cluster avec la même procédure).

Captures – Automatisation de l’installation du cluster K3s avec Ansible

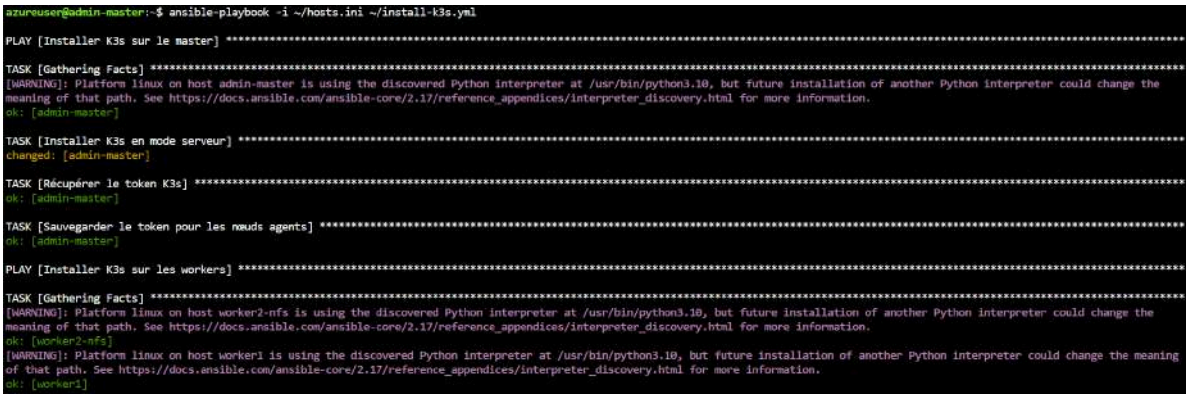
1️⃣ Installation du nœud master (*admin-master*)

```
azureuser@admin-master:~$ cat  install-k3s.yml
---
- name: Installer K3s sur le master
  hosts: k3s_master
  become: true
  tasks:
    - name: Installer K3s en mode serveur
      shell: curl -sfL https://get.k3s.io | sh -
      args:
        creates: /etc/rancher/k3s/k3s.yaml

    - name: Récupérer le token K3s
      shell: cat /var/lib/rancher/k3s/server/node-token
      register: k3s_token
      changed_when: false

    - name: Sauvegarder le token pour les nœuds agents
      set_fact:
        k3s_token: "{{ k3s_token.stdout }}"

- name: Installer K3s sur les workers
  hosts: k3s_nodes
  become: true
  tasks:
    - name: Installer K3s en mode agent
      shell: |
        curl -sfL https://get.k3s.io | K3S_URL=https://{{ hostvars['admin-master']['ansible_host'] }}:6443 K3S_TOKEN={{ hostvars['admin-master']['k3s_token'] }} sh -
      args:
        creates: /etc/rancher/k3s/k3s-agent.yaml
azureuser@admin-master:~$
```

La première capture montre l’exécution du rôle Ansible permettant d’installer **K3s en mode serveur** sur le nœud *admin-master*.

- Le serveur K3s est initialisé.
- Le fichier k3s.yaml (kubeconfig) est généré automatiquement.
- Le **token d’enrôlement** est produit par K3s.

2️⃣ Récupération et stockage du token



```
azureuser@admin-master:~$ ansible-playbook -i ~/hosts.ini ~/install-k3s.yml

PLAY [Installer K3s sur le master] *********************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: Platform linux on host admin-master is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the
meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [admin-master]

TASK [Installer K3s en mode serveur] *******************************************
changed: [admin-master]

TASK [Récupérer le token K3s] **************************************************
ok: [admin-master]

TASK [Sauvegarder le token pour les nœuds agents] ******************************
ok: [admin-master]

PLAY [Installer K3s sur les workers] *******************************************

TASK [Gathering Facts] *********************************************************
[WARNING]: Platform linux on host worker2-nfs is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the
meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [worker2-nfs]
[WARNING]: Platform linux on host worker1 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the meaning
of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [worker1]
```

La seconde capture illustre la **récupération automatique du token K3s** via Ansible :

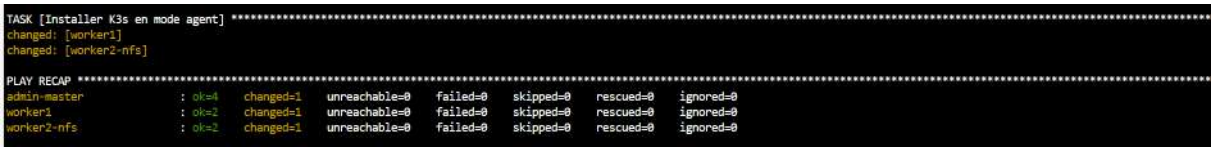
- name: Get node token

  command : cat /var/lib/rancher/k3s/server/node-token

  register: k3s_token

Ce token est stocké dans la variable k3s_token afin de pouvoir l’utiliser ensuite pour connecter les nœuds agents.

3️⃣ Ajout des nœuds agents (*worker1* et *worker2-nfs*)



La troisième capture présente la partie du playbook qui exécute l'installation de **K3s en mode agent** sur les deux nœuds secondaires.

- Les variables K3S_URL (adresse du master) et K3S_TOKEN (clé d'enrôlement) sont injectées automatiquement.
- Chaque worker rejoint le cluster sans intervention manuelle.

Ce playbook install-k3s.yml m'a permis de :

- **Gagner du temps** par rapport à une installation manuelle,
- **Standardiser** toutes les étapes d'installation,
- **Assurer la cohérence** et la **reproductibilité** du cluster,
- Respecter les **bonnes pratiques DevOps** (automatisation, IaC).

Vérification finale du cluster

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

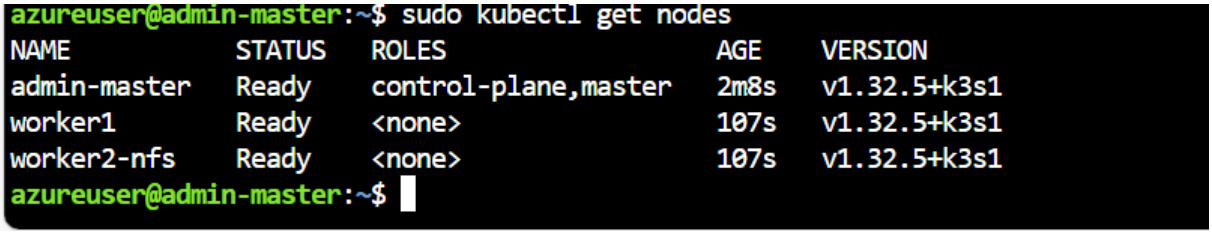
```
kubectl get nodes
```

Cette fois, les trois nœuds (admin-master, worker1, worker2-nfs) apparaissaient en statut Ready, avec les rôles correctement assignés :

- Control-plane, master pour le nœud admin-master
- worker pour les nœuds worker1 et worker2-nfs

Cette étape confirme que l'installation du cluster K3s est une réussite et que je peux passer à la suite du projet : la mise en place du stockage persistant.

Capture – Vérification de l'état du cluster



Cette capture montre le résultat de la commande kubectl get nodes exécutée depuis le nœud **admin-master**.
On y voit que :

- Les trois nœuds sont bien détectés,
- Chacun est en statut **Ready**,
- Le rôle est correctement attribué (control-plane pour admin-master, worker pour les autres).

Résumé technique des commandes utilisées

Étape	Commande utilisée
Installer K3s sur le master	`curl -sfL https://get.k3s.io | sh -`
Vérifier le 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 | K3S_URL=https://<IP_MASTER>:6443 K3S_TOKEN=<TOKEN> sh -`

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 : Configuration du service NFS sur worker2-nfs

```
azureuser@admin-master:~$ cat  install-nfs-server.yml
---
- name: Installer et configurer un serveur NFS sur worker2-nfs
  hosts: worker2-nfs
  become: true

  tasks:
    - name: Installer le serveur NFS
      apt:
        name: nfs-kernel-server
        state: present
        update_cache: true

    - name: Créer le répertoire de partage NFS
      file:
        path: /srv/nfs/kubedata
        state: directory
        owner: nobody
        group: nogroup
        mode: '0777'

    - name: Ajouter l'export NFS
      lineinfile:
        path: /etc/exports
        line: "/srv/nfs/kubedata *(rw,sync,no_subtree_check,no_root_squash)"
        state: present

    - name: Redémarrer le service NFS
```

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

- **rw** : autorise la lecture et l'écriture pour les clients,
- **sync** : garantit que les écritures sont immédiatement enregistrées 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 (pratique dans un cluster auto-géré).

Cette configuration rend possible le montage du dossier NFS par les nœuds **admin-master** et **worker1**, afin d'assurer la persistance des volumes pour **WordPress** et **MySQL**.
Elle est gérée automatiquement par **Ansible**, comme l'indique le bloc # BEGIN ANSIBLE MANAGED BLOCK.

Configuration des clients NFS (admin-master et worker1)

Sur les deux nœuds clients, j'ai effectué les étapes suivantes :

1. Installation du client NFS :
2.

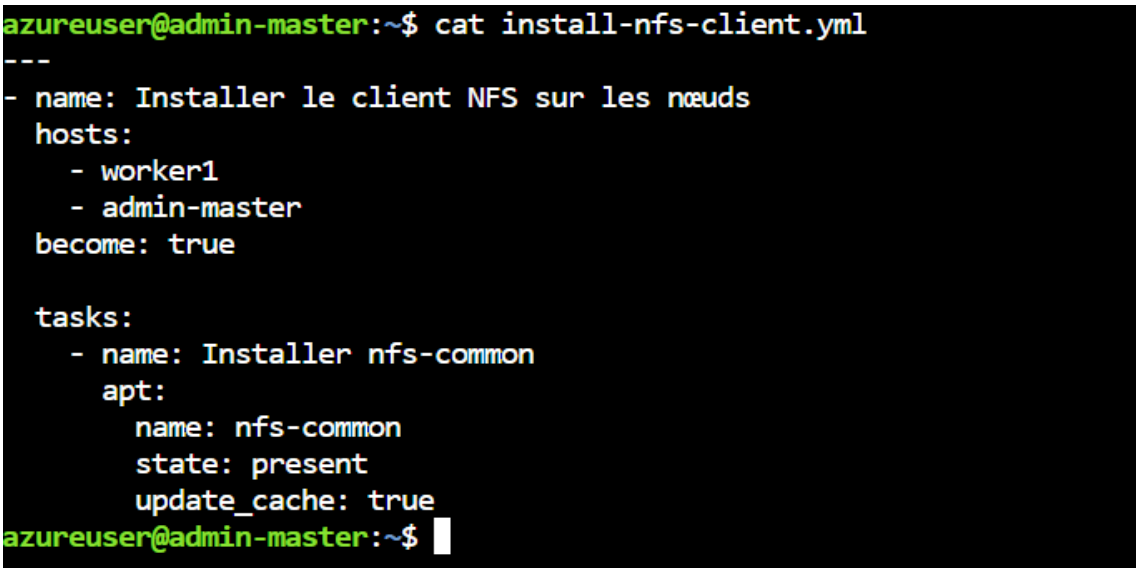
```
sudo apt install -y nfs-common
```

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

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

Ce test m'a permis de vérifier que la connexion au serveur NFS fonctionnait correctement avant de déléguer son utilisation à Kubernetes.

Capture : Validation du montage NFS sur les clients



Cette capture montre le contenu du dossier exporté via NFS sur le nœud worker2-nfs. On y retrouve :

- **Des fichiers de test** (test.txt, fichier-worker1.txt, nfs-test.txt) créés depuis différentes VMs du cluster pour valider le montage et la persistance.
- **Des dossiers applicatifs** (wordpress, wp-admin, wp-content, mysql) utilisés par les pods WordPress et MySQL pour stocker durablement leurs données.
- **Des fichiers propres à WordPress** (wp-config.php, wp-settings.php, etc.), preuve que le CMS est bien installé et que ses données persistent même après un redémarrage.

Cette validation confirme que le stockage partagé est pleinement fonctionnel, accessible par tous les nœuds, et qu'il assure la **résilience et la durabilité des données** dans l'infrastructure Kubernetes.

Intégration dans Kubernetes

Après avoir validé le bon fonctionnement de NFS, j'ai intégré la solution de stockage au cluster Kubernetes en créant des ressources YAML.

Ressources définies

- Un PersistentVolume (PV) pointant vers le chemin NFS partagé /srv/nfs/kubedata.
- Deux PersistentVolumeClaims (PVC) spécifiques :
    - MySQL : pour stocker durablement la base de données.
    - WordPress : pour stocker les fichiers dynamiques du site (médias, plugins, thèmes).

Exemple simplifié d'un PersistentVolume (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
```

Lien avec les déploiements

Les PersistentVolumeClaims (PVC) créés sont ensuite associés directement aux pods WordPress et MySQL via les champs :

- VolumeMounts → pour monter le volume dans le conteneur.
- Volumes → pour définir le lien avec le PVC.

Captures : Déclaration du stockage persistant dans Kubernetes

```
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 illustre le contenu du fichier **pv-pvc-nfs. yaml**, utilisé pour déclarer le stockage partagé et persistant dans Kubernetes.

Détails de la configuration

- **PersistentVolume (nfs-pv)**
    - Pointe vers le répertoire partagé **/srv/nfs/kubedata** situé sur le nœud **worker2-nfs (10.0.0.6)**.
    - Accès configuré en **ReadWriteMany (RWX)**, permettant à plusieurs pods, répartis sur différents nœuds, de lire et écrire simultanément dans le volume.
    - Garantit la compatibilité avec un **cluster distribué**.
- **PersistentVolumeClaim (wp-data)**
    - Créé dans le namespace **wordpress**.
    - Demande dynamiquement **1 Go de stockage** à partir du volume nfs-pv.
    - Sert d'interface pour les déploiements WordPress et MySQL afin d'accéder au volume partagé.

Résultat obtenu

Grâce à cette configuration :

- Les données de **WordPress** (uploads, médias, thèmes, plugins) et celles de **MySQL** (base de données) sont stockées de manière **centralisée et persistante**.
- Même après un **redémarrage du cluster** ou un **redeploiement des pods**, les données restent intactes.

- Cette solution assure la **robustesse** et la **continuité de service** de l'application e-commerce déployée sur Kubernetes.

Déploiement de WordPress et MySQL

L'objectif principal de cette étape est de déployer les deux composants essentiels de l'application e-commerce :

- La base de données MySQL,
- Le CMS WordPress,

Dans le cluster Kubernetes K3s, avec une configuration sécurisée, modulaire et persistante.

Création des Secrets et ConfigMap

- **Secret pour sécuriser les identifiants MySQL :**

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

Ce Secret contient le mot de passe administrateur MySQL, utilisé par le pod MySQL et par WordPress.

- ConfigMap pour stocker les paramètres applicatifs de WordPress (ex. site_url, site_title, etc.), afin d'éviter de coder en dur ces valeurs dans les manifests.

Déploiement de MySQL

- Fichier mysql-deployment. yaml :

Ce manifest définit :

- Un Deployment avec 1 réplique du pod MySQL,
- Un volume persistant attaché au PVC lié à NFS,
- Des variables d'environnement (MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, etc.),
- Un Service ClusterIP exposant MySQL uniquement au sein du cluster (sécurisé, non accessible publiquement).

Commande appliquée :

```
Kubectl apply -f mysql-deployment.yaml -n WordPress
```

Résultat attendu :

- Le pod MySQL est déployé et accessible uniquement par WordPress dans le namespace WordPress.
- Les données sont persistantes grâce au PVC connecté au serveur NFS.
- La sécurité est renforcée par l'usage de Secrets et ConfigMap, en ligne avec les bonnes pratiques DevOps**.**

Déploiement de WordPress

Le déploiement du CMS **WordPress** a été réalisé à l'aide du manifest wordpress-deployment.yaml.
Ce fichier contient :

Un initContainer chargé d'attendre la disponibilité de MySQL (exécution d'une boucle sleep + test TCP).

Le conteneur principal WordPress avec :

- Un **volume monté** sur /var/www/html,
- Un **PVC** connecté au serveur NFS pour garantir la persistance des fichiers WordPress (thèmes, plugins, uploads).

Un Service de type NodePort exposant WordPress sur le port **30080**, accessible via l'IP publique ou privée du nœud worker1.

Commande appliquée :

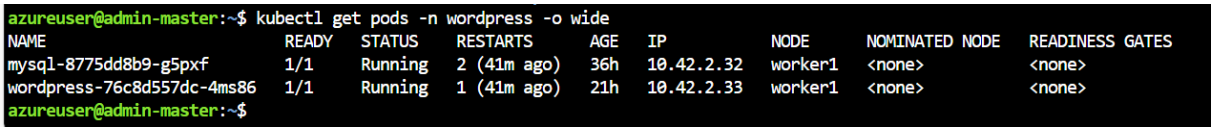
```
kubectl apply -f wordpress-deployment.yaml -n WordPress
```

Vérifications après déploiement

- **État des pods** :

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

Capture : État des pods WordPress et MySQL



Cette capture montre la liste des pods déployés dans le namespace WordPress.

- Les pods **wordpress-xxxxx** (CMS frontend) et **mysql-xxxxx** (base de données) apparaissent en statut **Running**,
- Aucun redémarrage anormal n'est constaté,
- Les conteneurs s'exécutent correctement sur leurs nœuds respectifs (worker1 pour WordPress, worker2-nfs pour MySQL),
- La communication entre WordPress et MySQL est assurée (attente via initContainer si nécessaire),
- Les volumes **NFS** sont bien montés et accessibles.

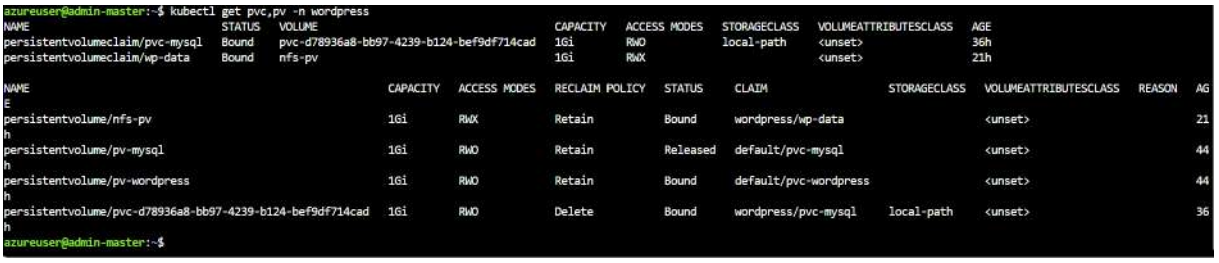
Vérification du stockage persistant

Après le déploiement de WordPress et MySQL, j'ai vérifié que les volumes étaient correctement créés et liés dans Kubernetes.

Commande utilisée :

```
kubectl get pvc,pv -n wordpress
```

Capture : État des Persistent Volumes (PV) et PersistentVolumeClaims (PVC)



Cette capture montre la vérification du stockage persistant dans Kubernetes.

- Les **PersistentVolumeClaims (PVC)** ont bien le statut **Bound**, ce qui signifie qu'ils sont correctement liés à des **Persistent Volumes (PV)** disponibles.
- Les PVC portent des noms explicites comme **wp-data**, correspondant aux besoins de **WordPress** et de **MySQL**.
- Le mode d'accès est **ReadWriteMany (RWX)**, ce qui permet à plusieurs pods (sur plusieurs nœuds) d'utiliser le volume simultanément, grâce au serveur **NFS**.

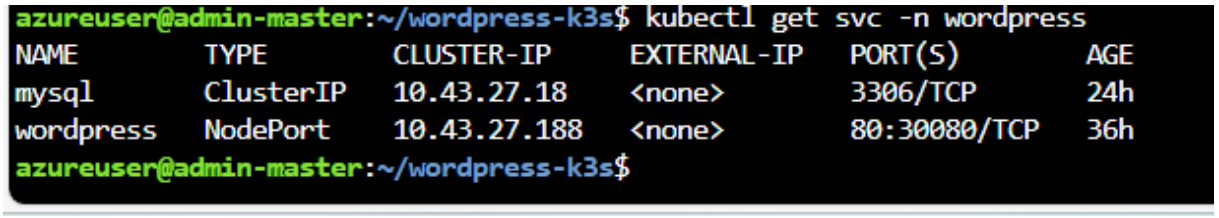
Vérification des services Kubernetes

Après le déploiement de WordPress et MySQL, j'ai vérifié la configuration des services exposés dans le namespace WordPress.

Commande utilisée :

```
kubectl get svc -n WordPress
```

Capture : État des services WordPress et MySQL



Cette capture montre les services créés par Kubernetes pour mon application :

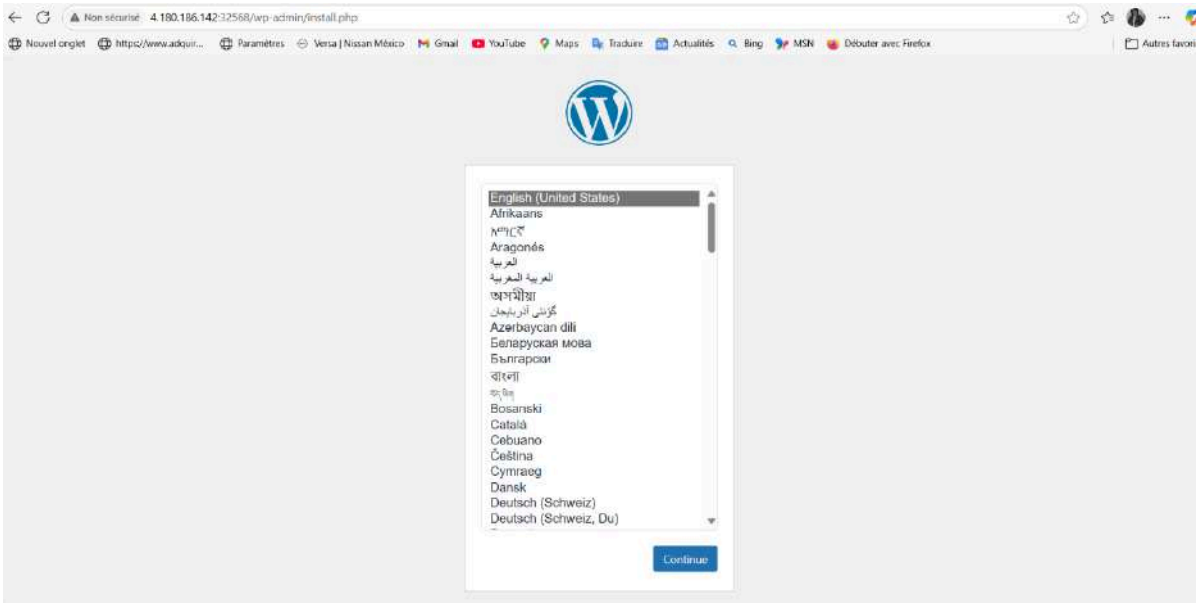
- **mysql** : service de type **Cluster IP**, interne au cluster. Il n'est pas accessible depuis l'extérieur, ce qui garantit la sécurité (uniquement les pods internes peuvent communiquer avec la base).
- **WordPress** : service de type **Node Port**, qui expose le port **80** du pod vers le port **30080** sur le nœud.

Accès frontend depuis le navigateur

Une fois le déploiement terminé, j'ai accédé au site WordPress depuis un navigateur web via l'URL exposée par le service Node Port :

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

Capture : Écran d'installation de WordPress



Cette capture montre l'écran d'accueil de l'installateur WordPress, accessible via l'adresse : http://4.180.186.142:32568/wp-admin/install.php

Cet écran n'apparaît que si :

- Le pod **WordPress** est bien déployé et exposé via le **Node Port**
- La base **MySQL** est fonctionnelle et joignable depuis WordPress
- Les fichiers WordPress sont stockés sur le volume NFS et persistent correctement

Validation obtenue

- Le CMS WordPress est accessible publiquement
- La communication WordPress ↔ MySQL est opérationnelle
- Le stockage persistant via NFS est bien fonctionnel
- Preuve technique et visuelle que l'architecture DevOps (K3s + NFS + CI/CD) est aboutie et opérationnelle

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 **Docker Hub** pour le stockage des images.

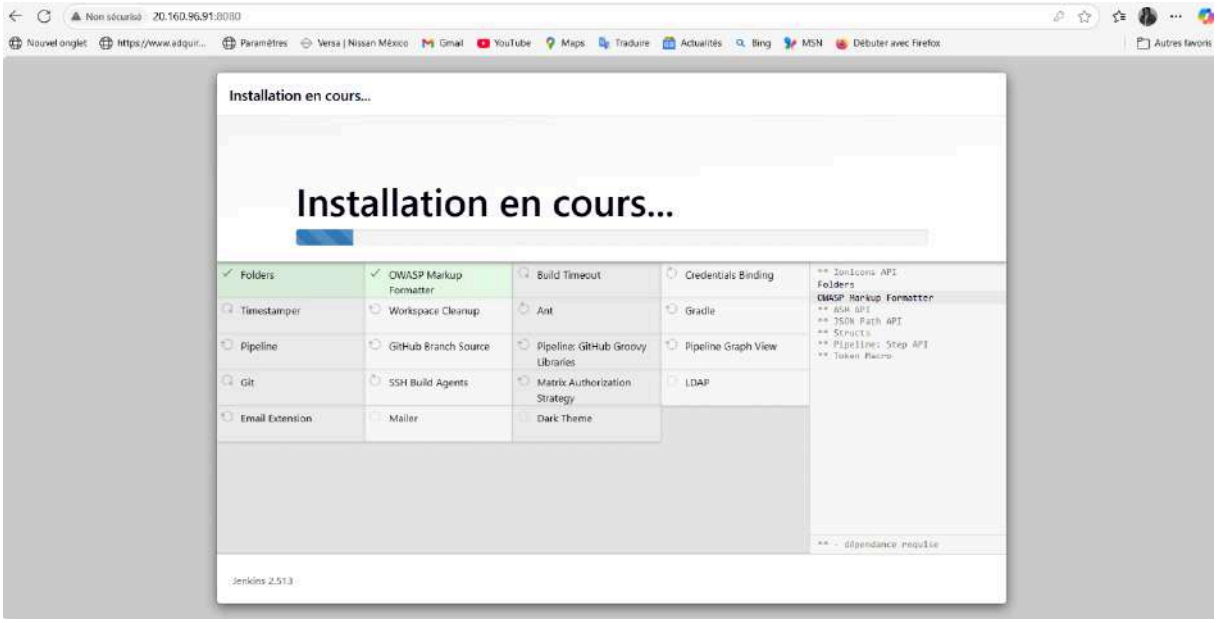
Objectif :
Automatiser toutes les étapes — construction de l'image Docker, tests, push vers Docker Hub, et déploiement automatique sur le cluster Kubernetes (K3s) à chaque mise à jour du code.

Installation et sécurisation de Jenkins (admin-master)

L'installation a été réalisée **via Ansible** sur la VM admin-master, selon les étapes suivantes :

```
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 : Installation de Jenkins et configuration initiale



Cette capture montre l'interface web de Jenkins lors de la configuration initiale, accessible via l'URL : http://20.160.96.91:8080

- Jenkins est installé sur la VM **admin-master** et tourne sur le port **8080**.
- L'utilisateur est en phase de première configuration, avec l'installation des plugins essentiels : **Git**, **Pipeline**, **GitHub Intégration**, **Email**, etc.
- Cette étape valide le bon déploiement de Jenkins dans l'infrastructure.

Prochaines étapes prévues :

- Mise en place d'un reverse proxy Nginx + HTTPS (Let's Encrypt)
- Ajout des credentials GitHub et Docker Hub
- Création d'un pipeline CI/CD basé sur un **Jenkins file**

Cette étape marque le début concret de l'automatisation, en posant les bases de la chaîne **CI/CD** du projet DevOps.

Connexion à GitHub et Docker Hub

J’ai créé un dépôt GitHub contenant :

Le code WordPress personnalisé (avec un thème préinstallé)

Un Docker file pour construire l’image

Un Jenkins file pour décrire le pipeline CI/CD

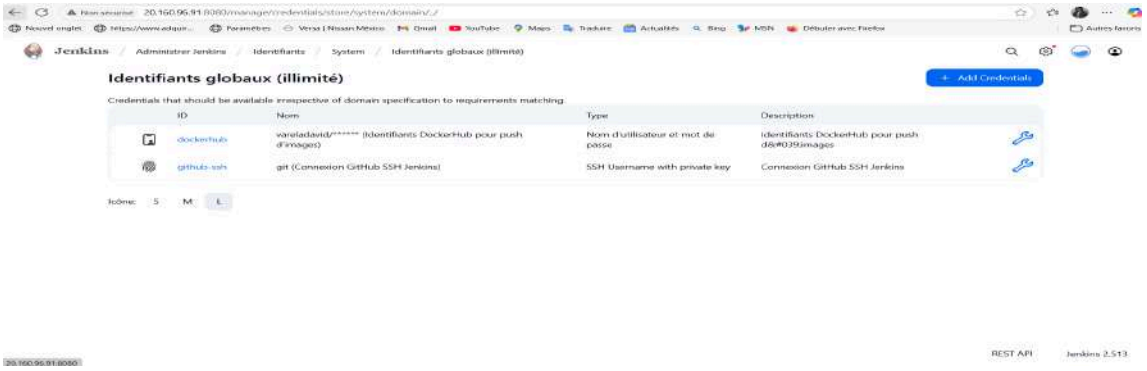
Les manifestes Kubernetes (wordpress-deployment.yaml, etc.)

Puis, dans Jenkins :

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

J’ai enregistré mes identifiants Docker Hub (login + token/mot de passe sécurisé)

Capture : Configuration des credentials Jenkins



Cette capture montre l’onglet **"Identifiants globaux"** (Credentials > global) dans Jenkins, où deux éléments essentiels ont été configurés :

- **docker hub**
  - Type : *Nom d’utilisateur + mot de passe*
  - Rôle : permet à Jenkins de **pusher les images Docker** vers Docker Hub
  - Référencé dans le Jenkins file via son ID sécurisé
- **github-ssh**
  - Type : *SSH Username with private key*
  - Rôle : permet à Jenkins de **cloner le dépôt GitHub en SSH**, déclencher des builds et interagir avec les branches du projet
  - Stocké dans le credential store de Jenkins

Ces credentials sont **sécurisés, réutilisables dans les pipelines** et indispensables pour une **chaîne CI/CD professionnelle**.
Ils garantissent une communication fiable et sécurisée avec GitHub et Docker Hub, étapes critiques avant la mise en place de l’automatisation complète.

Fichier Jenkins file : pipeline as code

Le Jenkins file, versionné sur GitHub avec le reste du projet, définit le pipeline CI/CD complet.
Il est écrit en syntaxe déclarative et décrit les étapes automatisées :

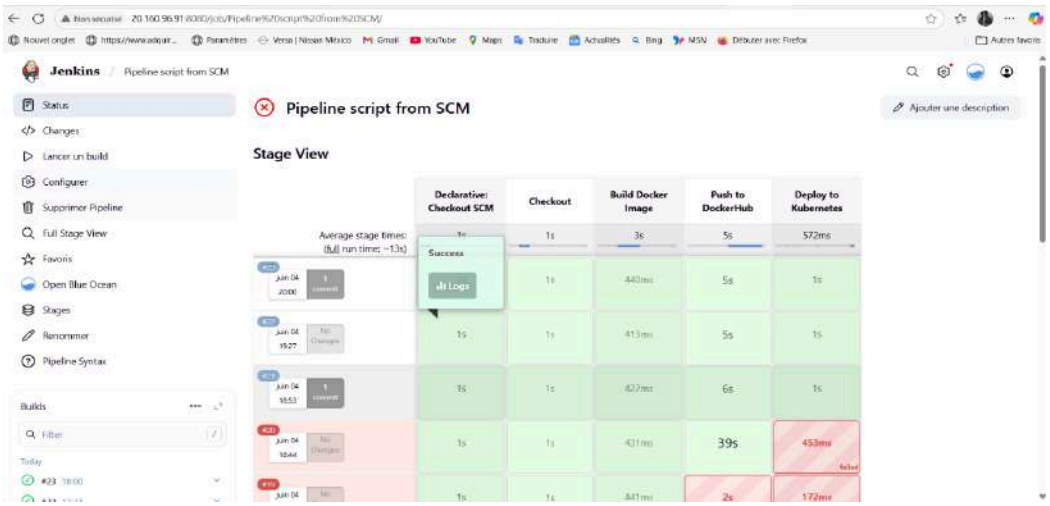
```
pipeline {
    agent any
    environment {
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-creds-id')
    }
    stages {
        stage('Build Docker image') {
            steps {
                sh 'docker build -t vareladavidhugo/wordpress-custom:latest
.'
            }
        }
        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'
            }
        }
    }
}
```

Explications :

- Stage 1 – Build Docker image
  Construit l'image personnalisée wordpress-custom:latest à partir du Docker file.
- Stage 2 – Push to Docker Hub
  Authentifie Jenkins sur Docker Hub (via dockerhub-creds-id) et pousse l'image construite.
- Stage 3 – Deploy to Kubernetes
  Déploie automatiquement l'application dans le cluster K3s en appliquant les manifests présents dans le dossier k8s/app/.

Exécution du pipeline CI/CD dans Jenkins

Capture : Vue par étapes (Stage View)



Cette capture montre la **vue Stage View** d'un pipeline Jenkins multibranches, défini dans un Jenkins file et relié directement à un dépôt GitHub.

On y distingue les étapes principales :

- **SCM Checkout** : récupération du code source depuis GitHub
- **Checkout** : préparation de l'espace de travail (Workspace Jenkins)
- **Build Docker Image** : construction de l'image Docker WordPress personnalisée
- **Push to Docker Hub** : envoi de l'image vers le registre DockerHub
- **Deploy to Kubernetes** : déploiement de l'application dans le cluster K3s via kubectl apply

Les builds #21, #22, #23 sont indiqués comme **Success** ✅, ce qui confirme que toutes les étapes de la chaîne CI/CD s'exécutent sans erreur.

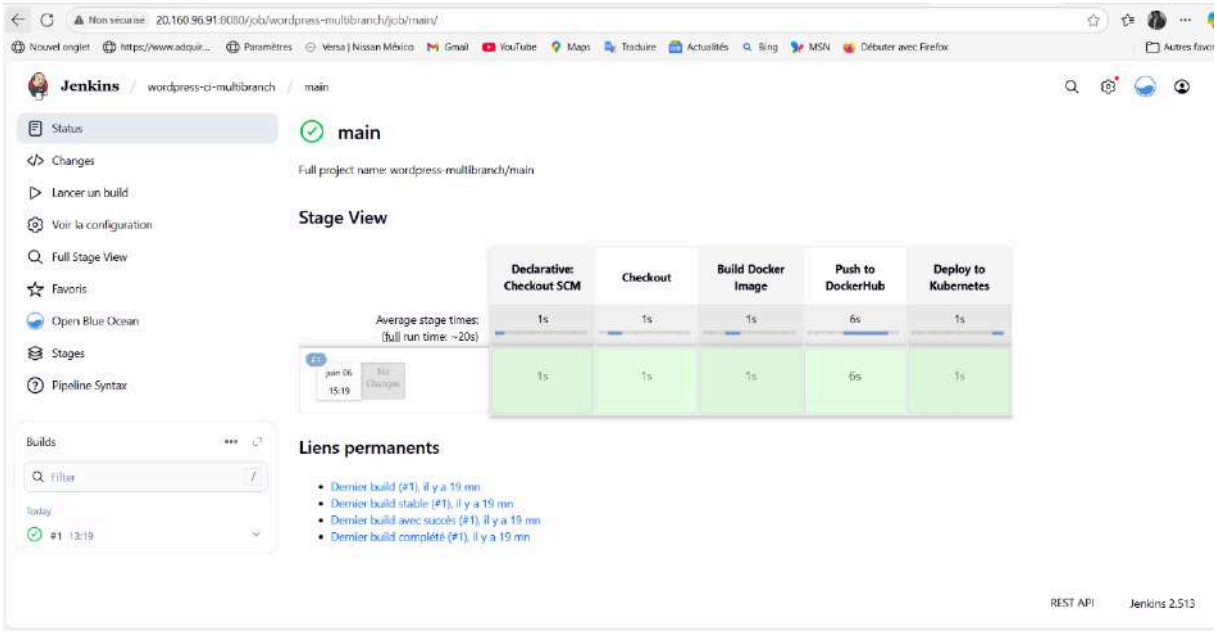
Pipeline multibranches GitHub

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

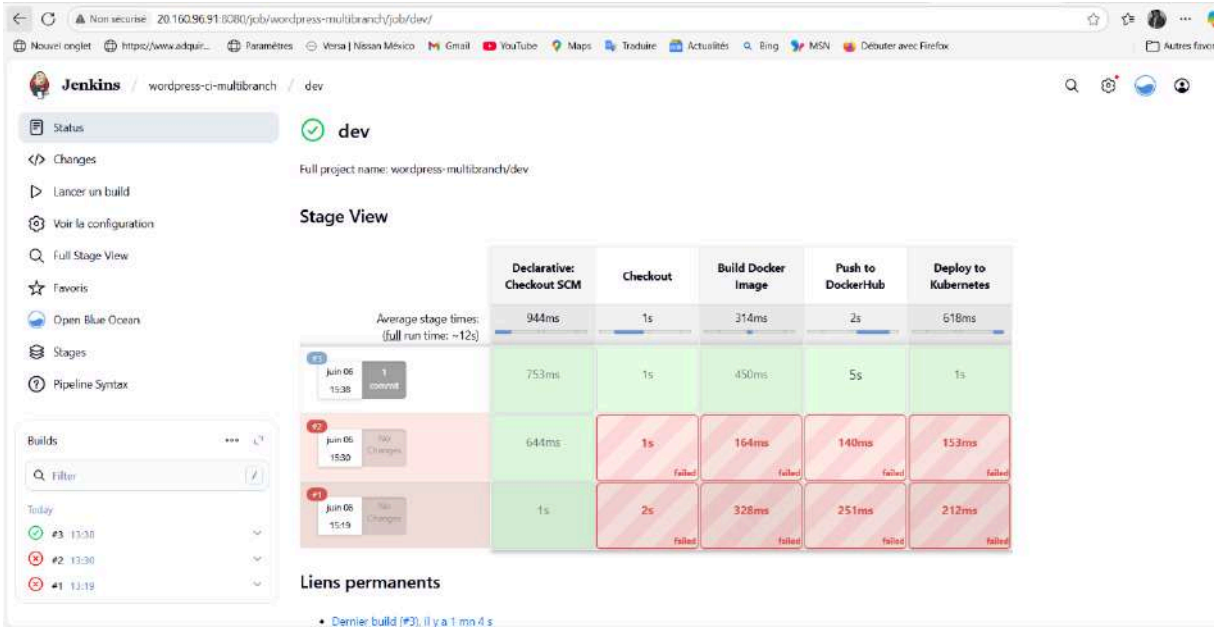
Séparation claire des environnements selon la branche

Capture 1 : Vue par étapes (Stage View)



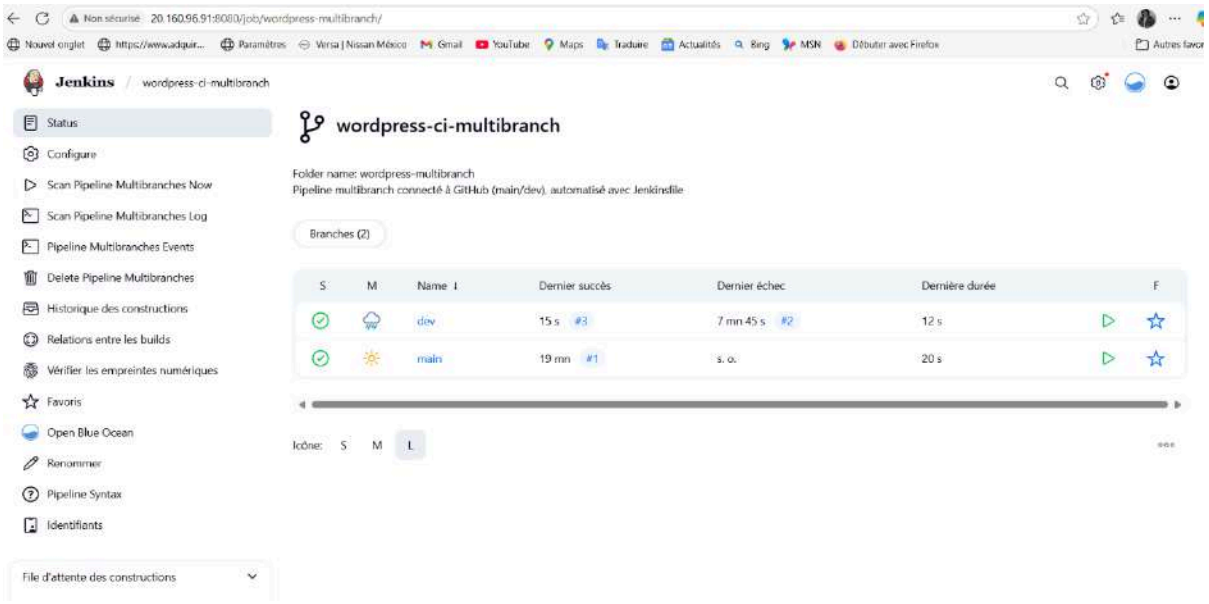
Cette capture illustre la **vue par étapes (Stage View)** du pipeline Jenkins.
On distingue les différentes étapes (Checkout, Build, Push, Deploy), toutes exécutées avec succès

Capture 2 : Historique des builds



Cette capture présente l'historique des builds (#21, #22, #23), tous marqués **Success**.
Cela prouve la **stabilité de la chaîne CI/CD**, capable de gérer plusieurs commits consécutifs sans échec.

Capture 3 : Job Multibranch



Cette capture montre l'interface d'un **job multibranch Jenkins** connecté à GitHub.
On observe :

- La détection automatique des branches **main** (production) et **dev** (tests)
- L'exécution individuelle d'un pipeline par branche
- Un suivi clair et séparé des environnements de déploiement

.

Grâce à cette configuration multibranches, le projet respecte les bonnes pratiques

Git + CI/CD :

main → déploiement en production

dev → déploiement en environnement de test

Automatisation déclenchée **à chaque git push**

Vérifications et tests finaux

Étapes de validation effectuées

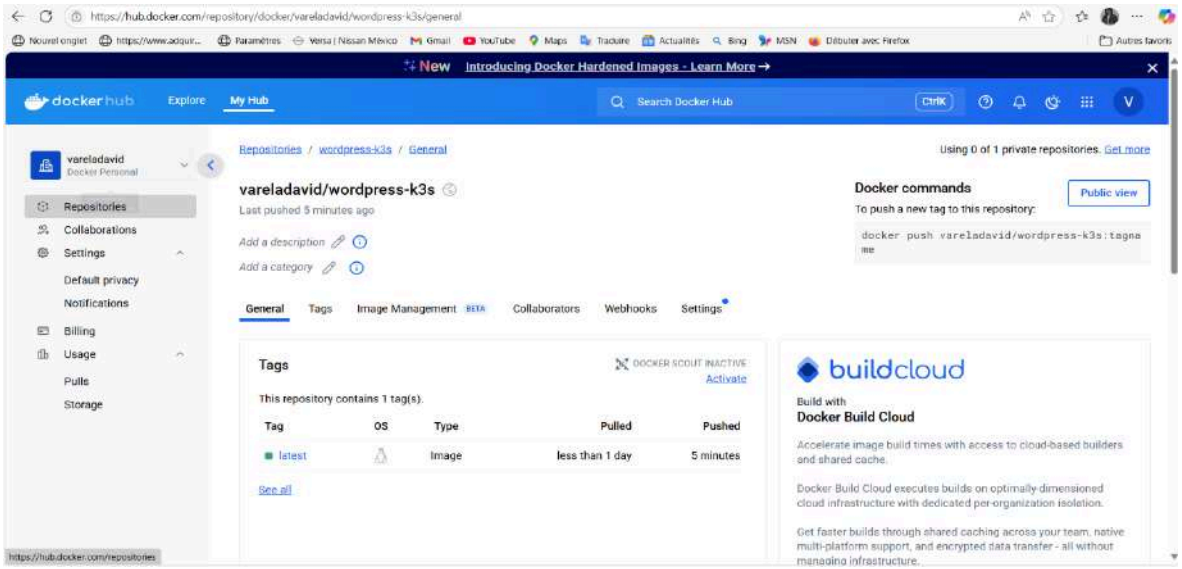
- Lancement manuel de jobs pour valider chaque étape du pipeline
- Vérification que l'image Docker est bien poussée sur Docker Hub
- Contrôle que les déploiements Kubernetes sont automatiquement mis à jour après chaque build

Commandes utilisées :

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

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

Capture : Dépôt Docker Hub



Cette capture illustre le dépôt **vareladavid/wordpress-k3s** sur Docker Hub.
On y voit l'image Docker générée et mise à jour automatiquement par **Jenkins** après chaque push sur la branche dev

Résultat obtenu

Chaque commit déclenche automatiquement un pipeline complet CI/CD
Les étapes build → push → déploiement Kubernetes sont entièrement automatisées
La cohérence est assurée entre les branches dev et prod
Le processus est reproductible, sécurisé, et maintenable

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 :

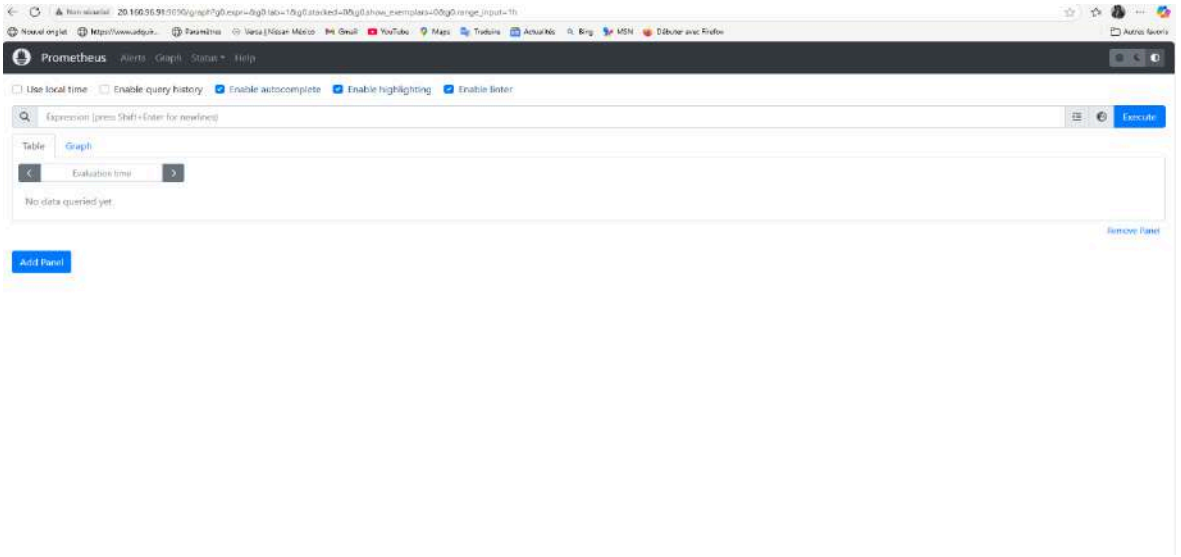
- **Prometheus** (collecte et stockage des métriques)
- **Node Exporter** (export des métriques système depuis les nœuds)
- **Grafana** (visualisation et tableaux de bord)

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
```

Supervision avec Prometheus et Grafana (suite)



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 :

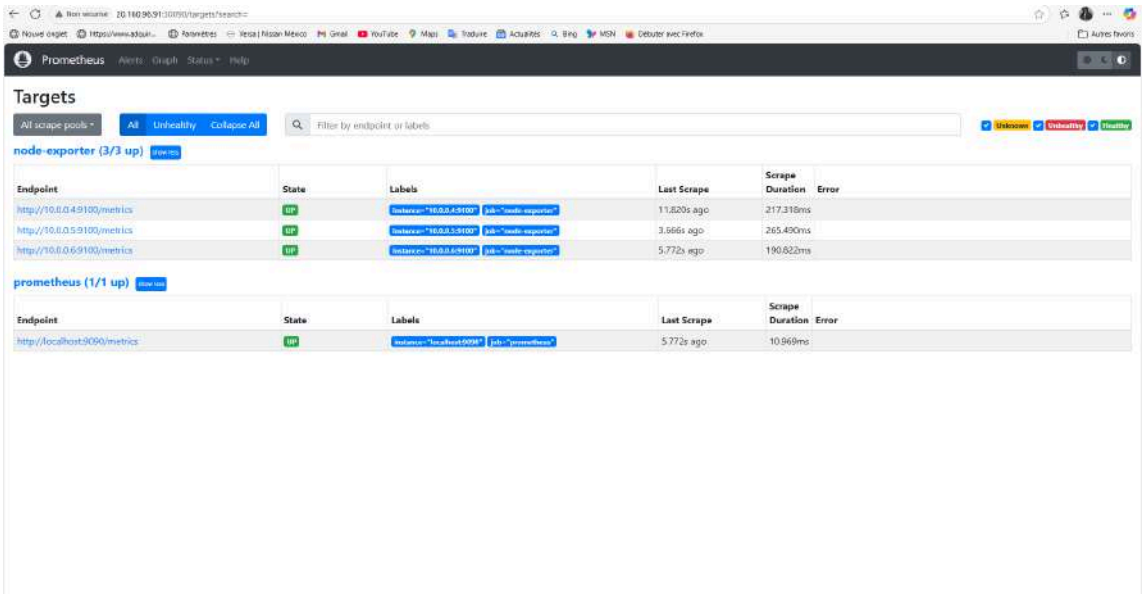
- Le déploiement de Prometheus est fonctionnel sur la VM admin-master
- Le port **9090** est bien ouvert dans les règles de sécurité réseau (NSG) de la VM Azure
- L'utilisateur peut accéder à Prometheus pour :
    - Consulter les métriques
    - Exécuter des requêtes PromQL
    - Surveiller les **Targets** (machines du cluster K3s)
    -

Installation de Node Exporter

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

```
# Télécharger 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 Node Exporter en arrière-plan

sudo nohup node_exporter &
```

Supervision Prometheus – Scraping des nœuds



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

Extrait du fichier prometheus.yml modifié pour ajouter manuellement les cibles :

scrape_configs:

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

Cette configuration permet à Prometheus de récupérer en continu les métriques système de chaque VM (CPU, RAM, disque, réseau), garantissant une supervision fiable et centralisée.

Intégration avec Grafana

Installation de 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 et activer le service

sudo systemctl start grafana-server

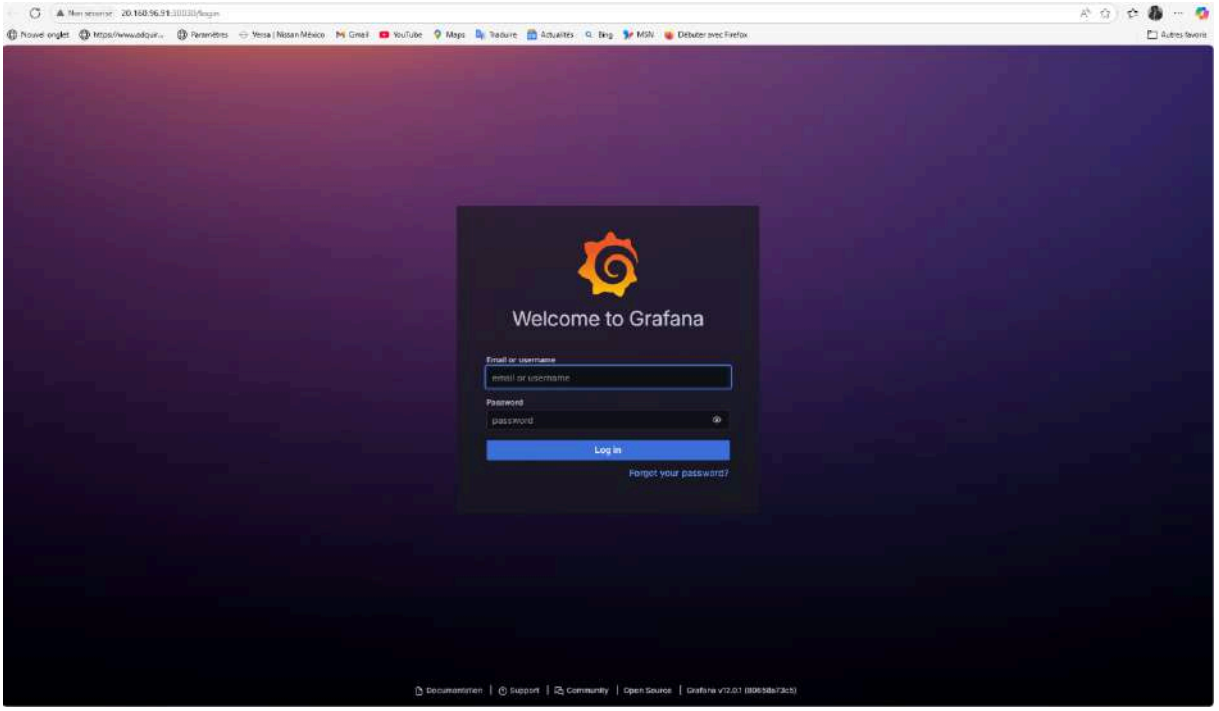
sudo systemctl enable grafana-server

Grafana est ensuite accessible à l'adresse :

```
http://20.160.96.91:3000
```

Identifiants par défaut : admin / admin

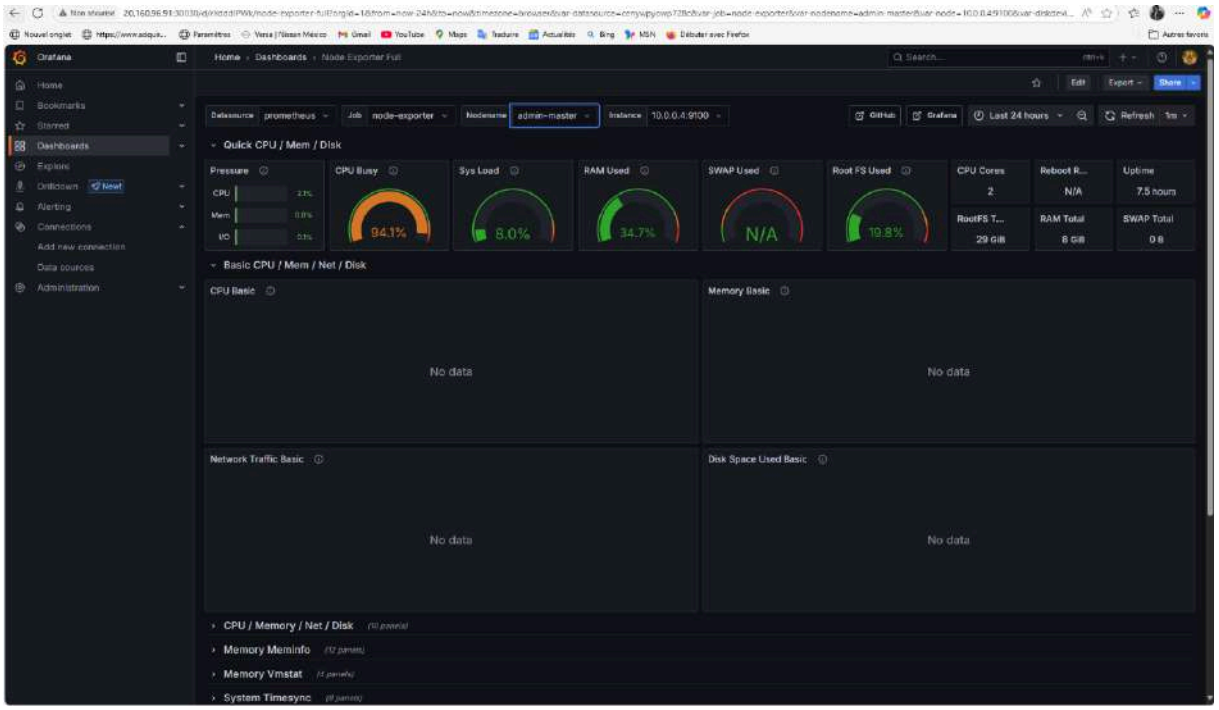
Capture – Interface de connexion Grafana (port 3000)



Cette capture montre l'accès à l'interface sécurisée de **Grafana**, accessible via le port **3000**.

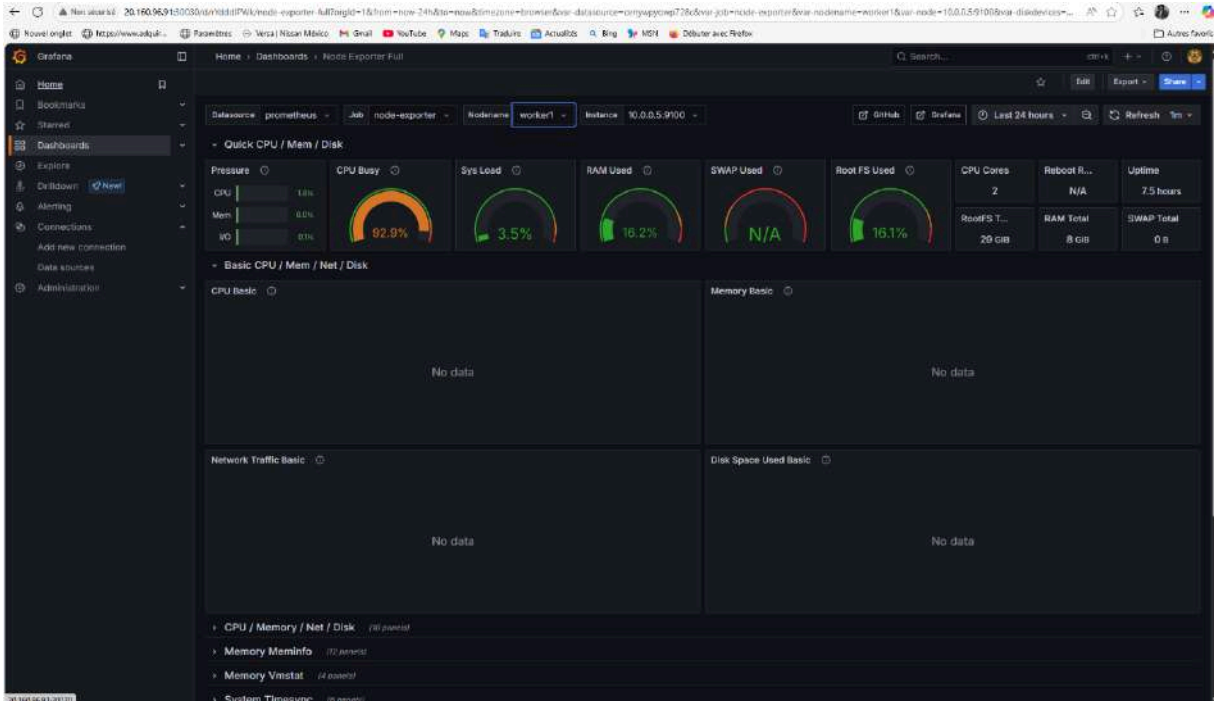
- L'administrateur doit s'authentifier avec son identifiant et mot de passe pour accéder aux Dashboard.
- Une fois connecté, il est possible d'ajouter **Prometheus comme source de données** et de créer des **Dashboard personnalisés**.
- Grafana permet une **visualisation graphique et interactive** des métriques collectées par Prometheus (CPU, mémoire, disque, services Kubernetes).

Capture – Supervision Grafana : ressources système du nœud admin-master



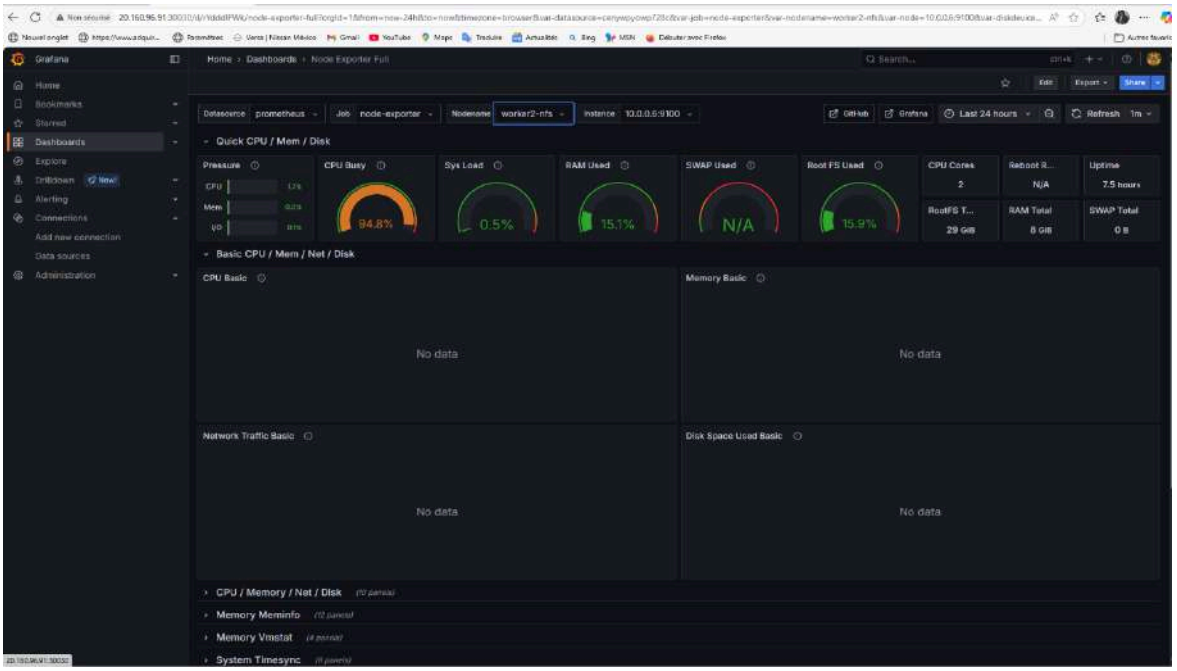
Ce panneau montre l'état des ressources système sur le nœud admin-master. On observe ici un taux d'utilisation CPU élevé, une RAM utilisée à 34,7 % et un espace disque racine utilisé à 19,8 %.

Capture – Supervision Grafana : ressources système du nœud worker1



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

Capture – Supervision Grafana : ressources système du nœud worker2-nfs



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 Jenkins file est bien pris en compte :

Build de l'image Docker

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

Push vers **Docker Hub**

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 Dashboard 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 validé la stabilité et la cohérence globale de l'architecture déployée.

Le système est désormais :

- Stable : le cluster Kubernetes fonctionne correctement et est dimensionné pour supporter l'application.
- Sécurisé : l'usage des secrets, du chiffrement HTTPS et des outils de supervision garantit la protection et la fiabilité des services.
- Automatisé : la chaîne CI/CD assure une intégration et un déploiement continus, réduisant les erreurs humaines.
- Résilient : le stockage NFS permet la persistance des données, même en cas de redémarrage ou d'incident.

Cette étape clôture la partie technique du projet en démontrant que toutes les fonctionnalités attendues sont en place. L'infrastructure répond non seulement aux objectifs pédagogiques de la formation, mais est également prête à être utilisée dans un contexte professionnel.

.

Technologies et outils utilisés

1. Infrastructure & Systèmes

- **Microsoft Azure** : hébergement des machines virtuelles (Vms), configuration du réseau, sécurisation via NSG.
- **Ubuntu Server 22.04** : système Linux stable, adapté aux environnements DevOps.

2. Conteneurisation & Orchestration

- **Docker** : construction d'images personnalisées (WordPress avec thème) et exécution de conteneurs.
- **Kubernetes (K3s)** : orchestration légère et adaptée à un cluster auto-hébergé, gestion des pods, services et volumes.

3. Infrastructure as Code (IaC)

- **Ansible** : automatisation de l'installation (K3s, NFS, Jenkins…), playbooks modulaires et réutilisables.

4. CI/CD – Intégration et Déploiement Continu

- **Jenkins** : pipeline CI/CD complet (build, push Docker, déploiement Kubernetes).
- **GitHub** : hébergement du code source, Docker file, Jenkinsfile et manifests YAML.
- **Docker Hub** : stockage des images Docker générées automatiquement par Jenkins.

5. Supervision & Monitoring

- **Prometheus** : collecte des métriques des nœuds et pods.
- **Node Exporter** : exposition des métriques système (CPU, RAM, disque) à Prometheus.
- **Grafana** : visualisation des données sous forme de Dashboard (ex. *Node Exporter Full*).

6. Stockage & Persistance

- **NFS (Network File System)** : stockage partagé pour WordPress et MySQL, garantissant la persistance des données.

7. Outils complémentaires

- **kubectl** : CLI pour interagir avec Kubernetes.
- **openssl / certbot** : génération et gestion des certificats TLS via Let's Encrypt (HTTPS).
- **Visual Studio Code** : éditeur principal pour YAML, Ansible et Jenkins file.

8. Choix technologiques

Ces outils ont été choisis pour leur :

- **Fiabilité** (solutions reconnues en production)
- **Compatibilité cloud** (Azure, Kubernetes, CI/CD)
- **Simplicité d'intégration** (outils interopérables entre eux)
- **Pertinence DevOps** (chaîne complète d'automatisation + supervision).

Ils ont permis de construire une infrastructure **automatisée, supervisée et prête pour un usage professionnel**, alignée avec les standards modernes du DevOps.

Résultat final obtenu

Le projet aboutit à un environnement de production **complet, automatisé et fonctionnel**, construit selon les standards professionnels du DevOps.

Réalisations concrètes

Cluster Kubernetes K3s opérationnel, composé de 3 nœuds (1 master + 2 workers), déployé automatiquement avec **Ansible**.

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

Chaîne CI/CD automatisée avec **Jenkins + GitHub + Docker Hub**, permettant le build, le push et le déploiement automatique à chaque commit.

Supervision en temps réel via **Prometheus + Node Exporter + Grafana**, offrant visibilité et traçabilité sur les nœuds et pods.

Accès sécurisé aux applications grâce à **Nginx reverse proxy** et certificats **HTTPS Let's Encrypt**.

Valeur ajoutée du projet

L'infrastructure répond pleinement aux objectifs pédagogiques du **Titre Professionnel ASDEV**.

Elle applique des **bonnes pratiques professionnelles** : cloud, automatisation, sécurité, résilience et monitoring.

Elle est **évolutive**, prête à accueillir :

- de nouveaux services,
- un découpage multi-environnements (test/production),
- ou une migration vers un cluster managé (**AKS / GKE**) pour la montée en charge.

Ce projet m'a permis de démontrer concrètement ma capacité à **concevoir, déployer et superviser une infrastructure DevOps complète**, compétence directement transférable en entreprise.

Supervision et CI/CD

La supervision et l'automatisation des déploiements sont les deux **piliers majeurs** de l'infrastructure 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 **Node Exporter**.
Ces données sont visualisées dans **Grafana** à travers des tableaux de bord dynamiques que j'ai personnalisés.

Indicateurs clés surveillés :

- Utilisation CPU et RAM de chaque nœud
- Espace disque consommé
- État des pods et services Kubernetes

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

Intégration et déploiement continus avec Jenkins

L'automatisation des déploiements a été assurée via **Jenkins**, avec un pipeline "as code" défini dans un **Jenkins file**.

À chaque **push GitHub**, Jenkins déclenche automatiquement :

1. Le **build** de l'image Docker personnalisée de WordPress
2. Le **tag** et le **push** vers Docker Hub (:test, :stable)
3. Le **déploiement** dans Kubernetes avec kubectl apply

Branches gérées :

- main → production
- dev → tests

Cette chaîne CI/CD m'a permis de :

- **Automatiser** tout le processus de mise à jour
- **Réduire les erreurs humaines**
- **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 DevOps.

Réalisations concrètes :

- Cluster **Kubernetes K3s** (3 nœuds) déployé via Ansible
- Application **WordPress e-commerce + MySQL** avec persistance NFS
- Chaîne **CI/CD complète** (Jenkins + GitHub + Docker Hub)
- **Supervision centralisée** (Prometheus + Grafana)
- **Sécurité renforcée** avec HTTPS (Nginx reverse proxy + Let's Encrypt)

Cette architecture répond aux exigences techniques du **Titre Professionnel ASDEV**, tout en restant **évolutive** : intégration de nouveaux services, environnements multi-stages, ou migration vers un cluster managé (**AKS, GKE**).

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** : clés SSH mal distribuées + port 22 bloqué dans le 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 ~/.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 dans la variable K3S_URL.
- **Solution** :
    - Vérification des IPs avec ping et IP a
    - Correction des rôles Ansible
    - Redéploiement propre avec un playbook automatisé

Problème 3 : Volumes NFS non montés

- **Symptôme** : les pods WordPress et MySQL restaient en état **Pending**.
- **Analyse** : les clients NFS ne pouvaient pas monter les volumes → paquet nfs-common manquant.
- **Solution** :
    - Installation de nfs-common via Ansible
    - Vérification 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** : absence de webhooks + credentials mal configurés.
- **Solution** :
    - Ajout des credentials sécurisés (GitHub PAT + Docker Hub)
    - Activation des **webhooks** dans GitHub
    - Test avec un 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 des variables GIT_SYNC_USERNAME et GIT_SYNC_PASSWORD.

- **Solution** :
    - Génération d'un nouveau **Personal Access Token (PAT)** GitHub
    - Ajout des variables d'environnement dans le Déploiement
    - Redéploiement du pod + vérification du bon clonage

Ce que ces problèmes m'ont appris

Ces difficultés ont été pour moi :

- Un exercice concret de **troubleshooting en conditions réelles**
- Une opportunité de renforcer mes **connaissances techniques** (SSH, Kubernetes, NFS, CI/CD)
- Une incitation à adopter une **méthodologie rigoureuse et documentée**

J'ai appris à :

- **Rechercher activement les causes racines**
- **Tester mes solutions de manière itérative**
- **M'auto-former en autonomie** grâce à la documentation officielle et aux forums spécialisés

Ces compétences sont directement transférables dans un contexte **professionnel d'administrateur Systèmes & DevOps**, où les imprévus techniques sont fréquents et exigent une résolution rapide et fiable.

Bilan personnel

Une expérience immersive et professionnalisante

Ce projet a représenté bien plus qu'un simple exercice technique.
Il a été une **expérience immersive**, me confrontant à des problématiques concrètes similaires à 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 pu consolider mes compétences et en développer de nouvelles :

- **Kubernetes** : déployer et administrer un cluster auto-hébergé, stable et sécurisé.
- **Ansible** : automatiser une infrastructure complète avec des rôles modulaires et réutilisables.
- **CI/CD** : mettre en place une chaîne fiable avec Jenkins, GitHub et Docker Hub.
- **Supervision** : collecter et visualiser les métriques avec Prometheus, Grafana et Node Exporter.
- **Troubleshooting** : diagnostiquer, corriger et documenter des erreurs complexes en conditions proches de la production.

Au-delà des aspects techniques, j'ai renforcé :

- ma **capacité d'apprentissage autonome**,
- ma **rigueur professionnelle**,
- et ma **méthodologie de travail**, indispensables dans un contexte DevOps.

Un tournant dans mon parcours

Ce projet constitue une étape déterminante dans ma **reconversion vers l'ingénierie Systèmes & DevOps**.
Il m'a permis de :

- prendre confiance en mes capacités techniques,
- valider mes acquis dans un environnement concret,
- et me préparer à **intégrer une équipe en entreprise**.

Je ressors de cette expérience avec :

- un **socle solide de compétences**,
- une **vision claire des bonnes pratiques DevOps**,
- et une **motivation renforcée** pour continuer à progresser, apprendre et contribuer à des projets d'infrastructure modernes, automatisés et fiables

Améliorations possibles

Même si le projet est abouti, stable et fonctionnel, plusieurs pistes d'évolution permettraient d'atteindre un **niveau de maturité supérieur** en matière de sécurité, de résilience et de scalabilité, en cohérence avec les standards DevOps modernes.

Sécurité

- Mise en place de **politiques RBAC personnalisées** pour restreindre finement les permissions des composants Kubernetes.
- Intégration d'un **gestionnaire de secrets** (HashiCorp Vault, Sealed Secrets, External Secrets Operator) afin de protéger les données sensibles (tokens, mots de passe, clés API).

Fiabilité et tests

- Ajout de **probes liveness/readiness** dans les manifestes Kubernetes, pour renforcer la supervision native et assurer un redémarrage automatique en cas d'anomalie.
- Intégration de **tests de performance et de disponibilité** dans la pipeline CI/CD (ex. avec *k6* ou *curl*) pour valider la montée en charge et la stabilité du service.

Packaging et automatisation

- Conversion des manifests YAML en **charts Helm**, afin d'industrialiser la gestion des déploiements multi-environnements (dev, staging, prod).
- Création de **Makefiles** pour centraliser les tâches récurrentes (build, push, apply, test) et simplifier le quotidien d'exploitation.

GitOps et déploiement continu

- Adoption d'une approche **GitOps** avec des outils comme ArgoCD ou Flux, permettant un déploiement **déclaratif, traçable et synchronisé** directement depuis GitHub.

Supervision avancée

- Intégration de **Prometheus Alertmanager** pour générer des alertes automatiques (CPU élevé, pod non disponible, disque saturé…).
- Mise en place d'une **centralisation des logs** avec Loki, afin d'accélérer le diagnostic et le débogage des applications.

Conclusion partielle :
Ces évolutions correspondent à une montée en maturité progressive (niveau **3 à 4 DevOps**).
Elles renforceraient la **sécurité**, la **fiabilité**, et l'**industrialisation** de l'environnement, tout en rapprochant ce projet d'une exploitation réelle en entreprise ou d'une production cloud managée.

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 Jenkins file, 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

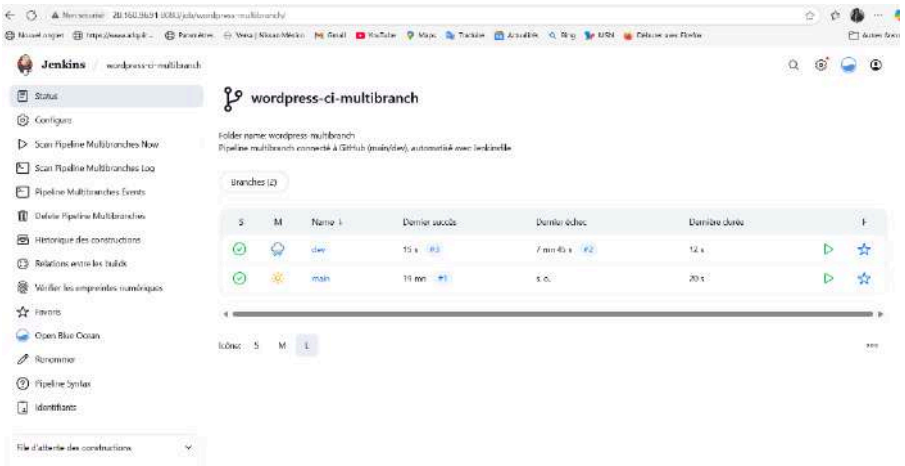
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client
provisioner: nfs.csi.k8s.io
parameters:
  server: 10.0.0.5        # IP du serveur NFS (ex: worker2-nfs)
```

```
  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.

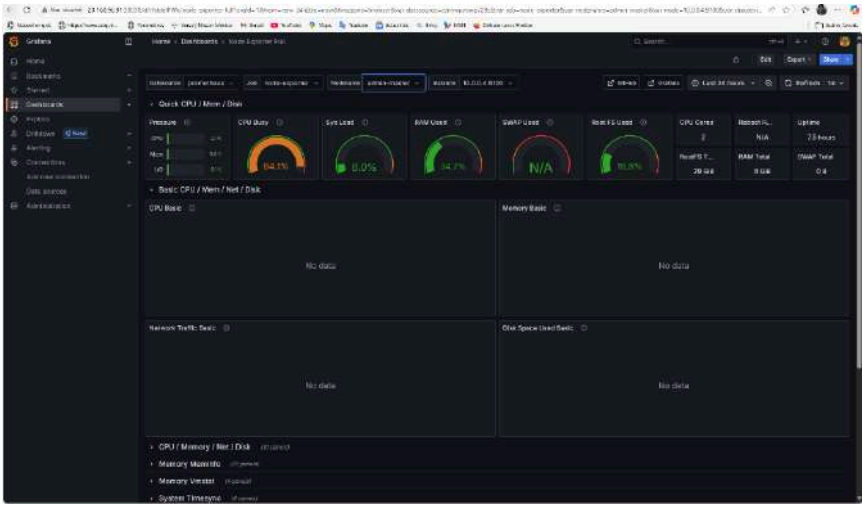
Interface Jenkins – Pipeline CI/CD



Vue de Jenkins exécutant un pipeline multibranche. Le déclenchement s’effectue automatiquement après un *git push*, et les étapes *Build* → *Push* → *Deploy* apparaissent en succès

2. Dashboard Grafana

Dashboard Grafana – Supervision en temps réel



Affichage d'un tableau de bord Grafana basé sur *Node Exporter* et métriques Kubernetes. Les indicateurs clés (CPU, RAM, filesystem, uptime, statut des pods WordPress/MySQL) permettent de suivre en direct l'état et la performance de l'infrastructure.

3. Sorties de commandes Kubernetes

Kubernetes – Pods WordPress et MySQL en exécution

```
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:~$
```

Résultat de la commande kubectl get pods -n wordpress montrant que les pods **WordPress** et **MySQL** sont en état *Running*. Cette sortie confirme que les déploiements critiques de l'application sont opérationnels et stables dans le cluster.

Kubernetes – Volumes persistants WordPress et MySQL

```
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$
```

Résultat des commandes kubectl get pv et kubectl get pvc -n WordPress.
Les PersistentVolume (PV) et PersistentVolumeClaim (PVC) associés à **WordPress** et **MySQL** apparaissent en statut **Bound**, garantissant un **stockage persistant fonctionnel** via NFS.

Kubernetes – Services exposés (WordPress & MySQL)

```
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$
```

Résultat de la commande kubectl get svc -n default.
Les services **WordPress** et **MySQL** sont visibles avec leurs **ClusterIP** et leur type (**LoadBalancer** ou **Cluster IP**).
Cette sortie confirme la **connectivité interne et externe** des applications déployées dans le cluster.

4. Commande df -h sur les VMS avec NFS

Montage NFS sur les nœuds du cluster

```
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:~$
```

Résultat de la commande df -h | grep nfs exécutée sur les VMS (worker1 ou worker2-nfs).
Les volumes partagés depuis le serveur **NFS** sont correctement montés.
La présence de **/srv/nfs** confirme que le **stockage persistant** est bien opérationnel dans le cluster Kubernetes.

5. Statut des services critiques

Statut des services critiques (NFS, Jenkins, Prometheus)

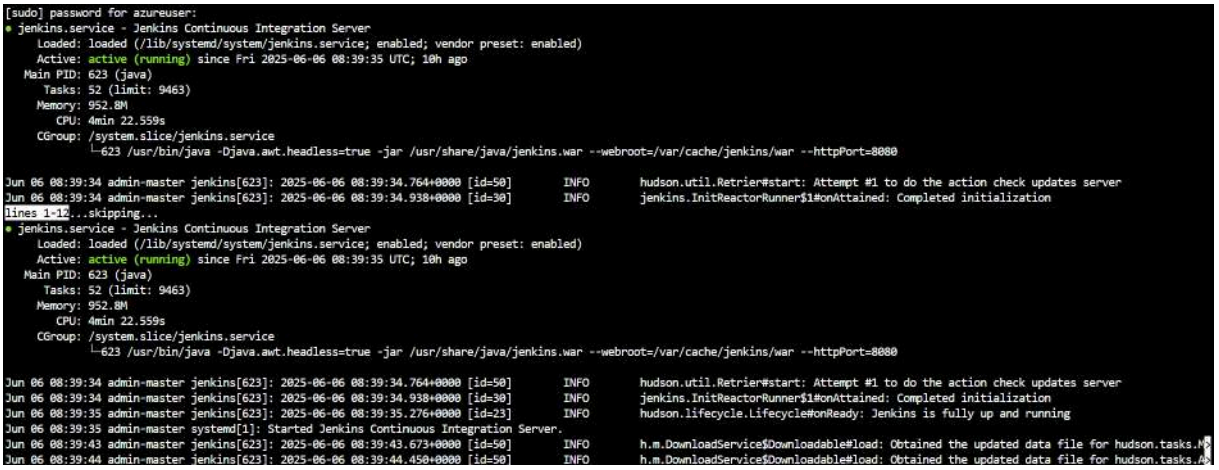
```
azureuser@worker2-nfs:~$ systemctl status nfs-kernel-server
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
    Drop-In: /run/systemd/generator/nfs-server.service.d
             └─order-with-mounts.conf
     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.
```

Sorties de la commande systemctl status sur les VMS du cluster :

- Sur **worker2-nfs** : le service **nfs-kernel-server** est actif et assure le partage des volumes.
- Sur **admin-master** : les services **Jenkins** et **Prometheus** sont en cours d'exécution.

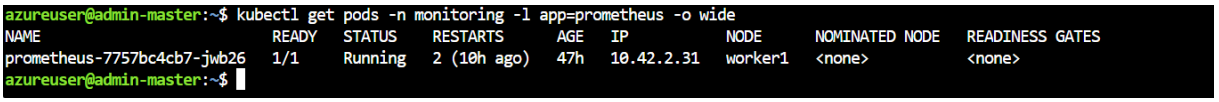
Ces vérifications confirment que les services essentiels à l'infrastructure (stockage, CI/CD, supervision) sont opérationnels et stables.

Statut du service Jenkins sur admin-master



Sortie de la commande systemctl status jenkins sur le nœud *admin-master*.
Le service Jenkins est actif (running) et pleinement opérationnel, avec confirmation des logs indiquant que Jenkins est « fully up and running ».

Statut du service Prometheus sur admin-master

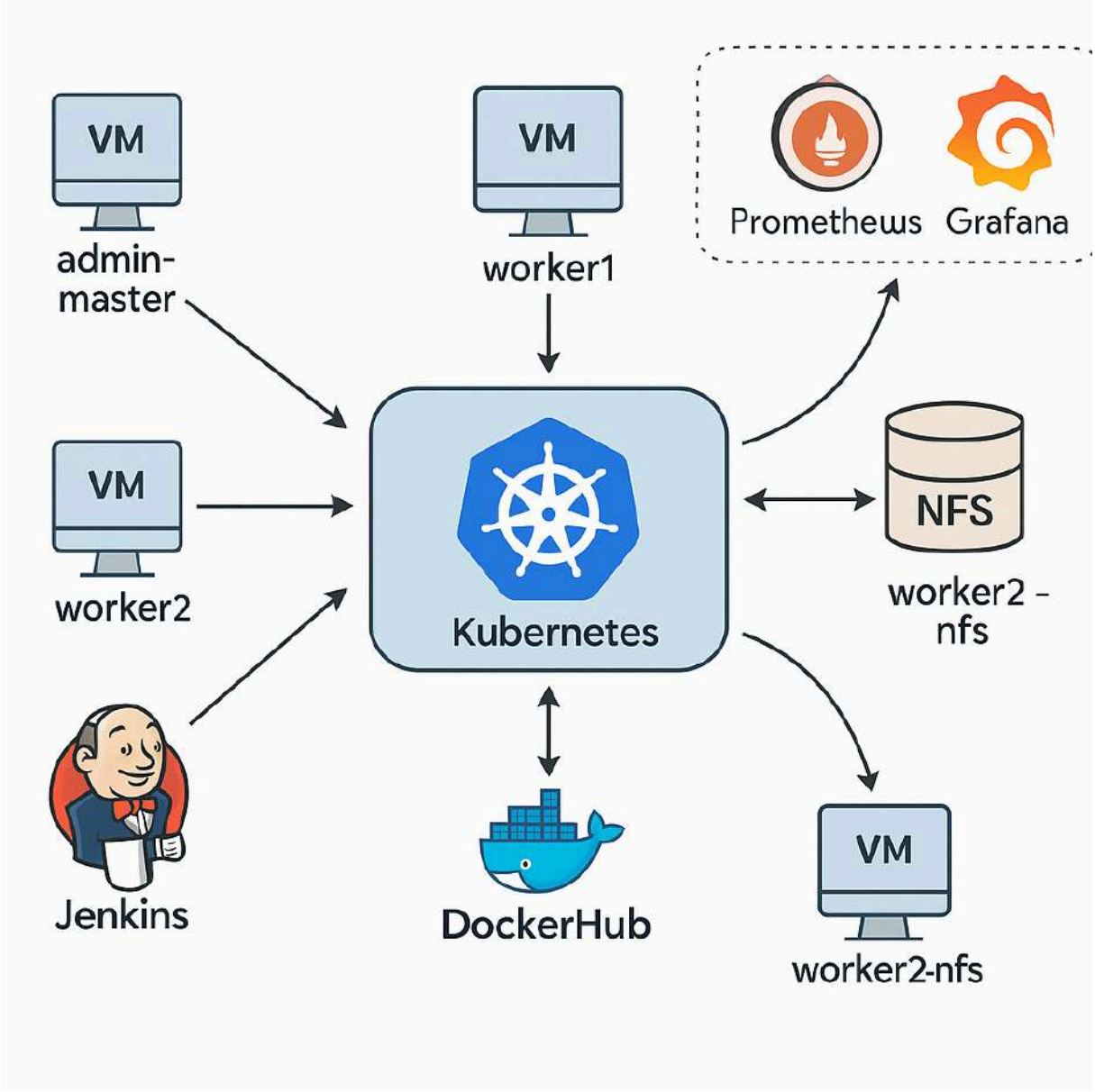


Sortie de la commande systemctl status prometheus sur le nœud *admin-master*.
Le service **Prometheus est actif (running)**, ce qui valide la supervision des métriques du cluster. Combiné avec Jenkins et NFS, cela démontre que les services critiques de l'infrastructure fonctionnent correctement.

Comme ça, tu auras le **bloc complet "Services critiques"** avec Jenkins + Prometheus + NFS validés.

D. Schéma de l’infrastructure

Schéma global de l’infrastructure DevOps



Ce diagramme illustre l’architecture complète du projet :

- **Cluster K3s** réparti sur trois VMS (admin-master, worker1, worker2-nfs)
- **Chaîne CI/CD** : GitHub → Jenkins → Docker Hub → Kubernetes
- **Stockage persistant** assuré par un serveur **NFS**
- **Supervision** avec Prometheus et Grafana

E. Liens utiles personnels

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

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

Docker Hub (images utilisées dans le projet) : hub.docker.com/u/vareladavidhugo

Portfolio professionnel (DevOps) : www.varelasolutions.fr

Ces annexes complètent le rapport principal en apportant des preuves tangibles (captures, scripts, schémas) et des ressources externes (profils, dépôts).
Elles démontrent à la fois :

la réussite technique du projet,

sa reproductibilité,

et ma capacité à documenter et valoriser un travail DevOps de bout en bout.

Conclusion générale

Ce projet a démontré ma capacité à concevoir, déployer et automatiser une infrastructure complète et réaliste, en appliquant les bonnes pratiques DevOps.

Mise en place d'un cluster Kubernetes auto-hébergé et sécurisé,

Chaîne CI/CD complète avec Jenkins, GitHub et DockerHub,

Supervision avancée avec Prometheus et Grafana,

Persistance des données via NFS.

Au-delà de l'aspect technique, ce projet représente une étape clé de ma reconversion vers les métiers d'ingénierie système et DevOps.
Il m'a apporté :

la confiance,

les compétences solides,

et une vision claire pour contribuer efficacement à des environnements professionnels modernes, sécurisés et évolutifs.