

I. Introduction

Dans le cadre de notre projet de fin de formation en **Administration Système DevOps**, nous avons réalisé la mise en place d' une infrastructure complète de type **production**, déployée en **environnement hybride**. Ce projet vise à démontrer notre capacité à concevoir, automatiser, sécuriser et superviser une **plateforme applicative distribuée**, en s' appuyant sur des technologies et pratiques modernes d' ingénierie **DevOps**.

💡 **DevOps** est un ensemble de pratiques visant à unifier le développement logiciel (Dev) et l' administration des systèmes d' exploitation (Ops), en mettant l' accent sur l' automatisation, la collaboration, et la livraison continue.

L' objectif principal est de déployer plusieurs applications métiers (WordPress, Odoo, GLPI) sur un **cluster Kubernetes**, avec une chaîne **CI/CD** automatisée (Jenkins, SonarQube, Trivy, ArgoCD), un **monitoring temps réel** (Prometheus, Grafana), et une **infrastructure sécurisée** (Nginx Ingress Controller, Cert-Manager, Crowdsec).

L' ensemble de l' infrastructure est décrit sous forme de code (**IaC**, Infrastructure as Code) via **Terraform** et **Ansible**, garantissant la **réplicabilité**, la cohérence et la traçabilité des changements.

📄 **CI/CD** (Continuous Integration / Continuous Delivery) est une méthode DevOps permettant d' intégrer, tester, vérifier et déployer automatiquement les modifications de code.

II. Environnements Utilisés

Afin de simuler un environnement de production réaliste, nous avons adopté une architecture **hybride** combinant des ressources cloud (Microsoft Azure) et des machines locales via VirtualBox.

Rôle	Type	Technologie / OS
VM Master	Kubernetes Cloud	Azure Ubuntu Server 22.04
VM Worker	Kubernetes Cloud	Azure Ubuntu Server 22.04
VM Stockage (NFS)	Cloud	Azure Ubuntu Server 22.04
VM Administration	Locale	Debian 12

Composants principaux déployés :

- **Infrastructure** : Azure (IaaS), VirtualBox (virtualisation locale), Terraform
- **Automatisation** : Ansible pour la configuration déclarative
- **Orchestration** : Kubernetes, Helm
- **CI/CD** : Jenkins (choisi pour sa flexibilité d'intégration), SonarQube, Trivy, ArgoCD
- **Monitoring** : Prometheus, Grafana
- **Sécurité** : Nginx Ingress Controller, Cert-Manager, Crowdsec
- **Applications** : WordPress, Odoo, GLPI, PostgreSQL, MariaDB
- **Stockage** : NFS (Network File System) utilisé comme système de fichiers partagé entre les pods

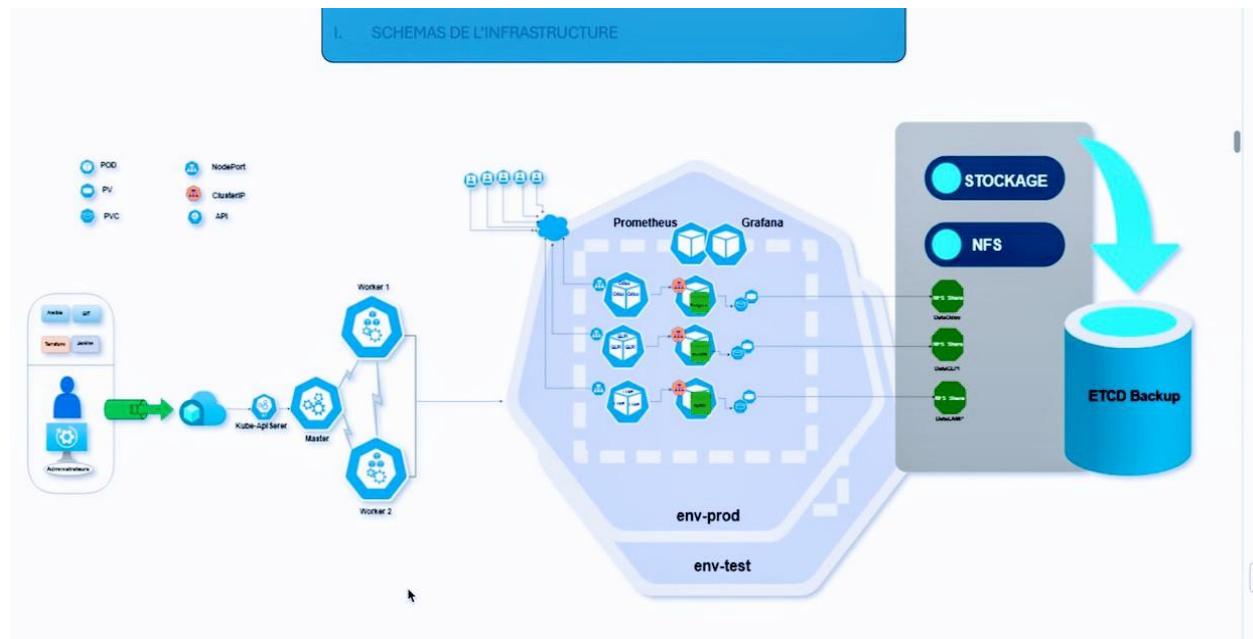
🔍 **IaC** (Infrastructure as Code) consiste à décrire l'architecture système dans un fichier texte (souvent en YAML ou HCL), permettant de la recréer automatiquement et de la versionner dans un dépôt Git.

III. Déploiement de l'Infrastructure

1. Schéma global de l'architecture

Un schéma d'architecture illustre les composants principaux du projet, notamment :

- Les VMs
- Le réseau et les flux de communication
- Le cluster Kubernetes
- Les ressources partagées comme le NFS
- Le pipeline CI/CD et les outils de monitoring



✂ **Note** : Ce type de schéma est crucial pour comprendre les relations entre les différentes briques de l'infrastructure.

2. Provisionnement des ressources cloud avec Terraform

L'ensemble de l'infrastructure cloud a été déployé sur **Microsoft Azure** via des **scripts Terraform**. Les modules ont été conçus pour être réutilisables et facilement modifiables.

Composants Terraform :

- **Réseau** : Création du Virtual Network (VNet), sous-réseaux (subnets) et groupes de sécurité (NSG)
- **Machines virtuelles** : Création des VM Kubernetes Master, Worker, et stockages NFS
- **Accès** : Configuration de l'accès SSH, règles de sécurité, et DNS interne

Exécution des commandes :

une partie du Script **terraform** pour la création des différentes ressources sur le **cloud azure**

```

provider "azurerm" {
  features {}
  subscription_id = "98e6d5a0-0135-4134-b9a2-ea682be29547"
}

# Réseau Virtuel (VNet)
resource "azurerm_virtual_network" "moyo_vnet" {
  name                = "moyo-vnet"
  location             = "North Central US" # Harmonisation de la région
  resource_group_name = "moyo"
  address_space       = ["10.0.0.0/16"]
  dns_servers         = ["168.63.129.16"]
}

# Sous-réseau
resource "azurerm_subnet" "default" {
  name                 = "default"
  resource_group_name = "moyo"
  virtual_network_name = azurerm_virtual_network.moyo_vnet.name # Utilisation de la VNet créée précédemment
  address_prefixes     = ["10.0.0.0/24"]
}

# Adresses IP publiques
resource "azurerm_public_ip" "master_ip" {
  name                = "master-vm-ip"
  location             = "North Central US" # Harmonisation de la région
  resource_group_name = "moyo"
  allocation_method    = "Static"
  domain_name_label   = "master-vm-unique"
}

resource "azurerm_public_ip" "worker1_ip" {
  name                = "worker1-vm-ip"
  location             = "North Central US" # Harmonisation de la région
  resource_group_name = "moyo"
  allocation_method    = "Static"
  domain_name_label   = "worker1-vm"
}

resource "azurerm_public_ip" "stockage_ip" {

```

Une partie de la commande **terraform plan**

```

+ fqdn                = (known after apply)
+ id                  = (known after apply)
+ idle_timeout_in_minutes = 4
+ ip_address          = (known after apply)
+ ip_version          = "IPv4"
+ location             = "westus"
+ name                 = "worker1-vm-ip"
+ resource_group_name = "moyo"
+ sku                  = "Standard"
+ sku_tier             = "Regional"
}

# azurerm_subnet.default will be created
+ resource "azurerm_subnet" "default" {
+   address_prefixes = [
+     "10.0.0.0/24",
+   ]
+   default_outbound_access_enabled = true
+   id                             = (known after apply)
+   name                           = "default"
+   private_endpoint_network_policies = "Disabled"
+   private_link_service_network_policies_enabled = true
+   resource_group_name            = "moyo"
+   virtual_network_name           = "moyo-vnet"
+ }

# azurerm_virtual_network.moyo_vnet will be created
+ resource "azurerm_virtual_network" "moyo_vnet" {
+   address_space = [
+     "10.0.0.0/16",
+   ]
+   dns_servers = [
+     "168.63.129.16",
+   ]
+   guid         = (known after apply)
+   id           = (known after apply)
+   location     = "westus"
+   name         = "moyo-vnet"
+   private_endpoint_vnet_policies = "Disabled"
+   resource_group_name            = "moyo"
+   subnet                        = (known after apply)
+ }

```










Plan: 11 to add, 0 to change, 3 to destroy.

Liste des ressources créer sur **azure** après application du script **terraform** via la commande **terraform apply**

```

vboxuser@vbox:~/projet_personnel$ az vm list -g moyo --out table
Name      ResourceGroup  Location      Zones
-----
master-vm  moyo           northcentralus
stockage-vm moyo           northcentralus
worker1-vm moyo           northcentralus
vboxuser@vbox:~/projet_personnel$ terraform state list
azurerm_linux_virtual_machine.master
azurerm_linux_virtual_machine.stockage
azurerm_linux_virtual_machine.worker1
azurerm_network_interface.master_nic
azurerm_network_interface.stockage_nic
azurerm_network_interface.worker1_nic
azurerm_public_ip.master_ip
azurerm_public_ip.stockage_ip
azurerm_public_ip.worker1_ip
azurerm_subnet.default
azurerm_virtual_network.moyo_vnet
vboxuser@vbox:~/projet_personnel$

```

<input type="checkbox"/>	 master-vm	...	Machine virtuelle	North Central US	moyo	Abonnement Azure 1	il y a 9 minutes
<input type="checkbox"/>	 stockage	...	Machine virtuelle	North Central US	moyo	Abonnement Azure 1	il y a 6 jours
<input type="checkbox"/>	 worker1-vm	...	Machine virtuelle	North Central US	moyo	Abonnement Azure 1	il y a 6 jours
<input type="checkbox"/>	 master-nic-nsg	...	Groupe de sécurité réseau	North Central US	moyo	Abonnement Azure 1	il y a 11 jours
<input type="checkbox"/>	 worker1-nic-nsg	...	Groupe de sécurité réseau	North Central US	moyo	Abonnement Azure 1	il y a 13 jours
<input type="checkbox"/>	 stockage-nsg	...	Groupe de sécurité réseau	North Central US	moyo	Abonnement Azure 1	il y a 20 jours
<input type="checkbox"/>	 moyo	...	Groupe de ressources		moyo	Abonnement Azure 1	il y a 20 jours
<input type="checkbox"/>	 moyo-vnet	...	Réseau virtuel	North Central US	moyo	Abonnement Azure 1	il y a 20 jours
<input type="checkbox"/>	 stockage-ip	...	Adresse IP publique	North Central US	moyo	Abonnement Azure 1	il y a 20 jours

🔍 **Terraform** est un outil open-source d'IaC développé par HashiCorp. Il permet de provisionner automatiquement des ressources cloud multi-fournisseurs.

3. Déploiement de la VM d'administration locale

La VM d'administration, hébergée sur **VirtualBox**, sert de centre de contrôle pour le déploiement et la supervision. Elle a été créée à l'aide d'un script Python s'appuyant sur la bibliothèque pyVBox.

Une partie du script python

```
)

# Configure la machine virtuelle
vm.memory_size = 2048 # Taille de la RAM (2 Go)
vm.cpu_count = 2 # Nombre de cœurs CPU
vm.vram_size = 16 # Taille de la vidéo mémoire

# Ajoutez un disque dur virtuel
vdi = vbox.create_hard_disk("VDI", "/path/to/vms/vm_admin.vdi")
vdi.create_base_storage(20 * 1024 * 1024 * 1024, [pyvbox.MediumFormat.VDI]) # 20 Go de stockage

# Ajoutez le disque dur à la machine virtuelle
vm.attach_device("SATA", 0, 0, vdi)

# Configure le réseau de la machine virtuelle
vm.set_network_adapter(0, type=pyvbox.NetworkAdapterType.NAT)

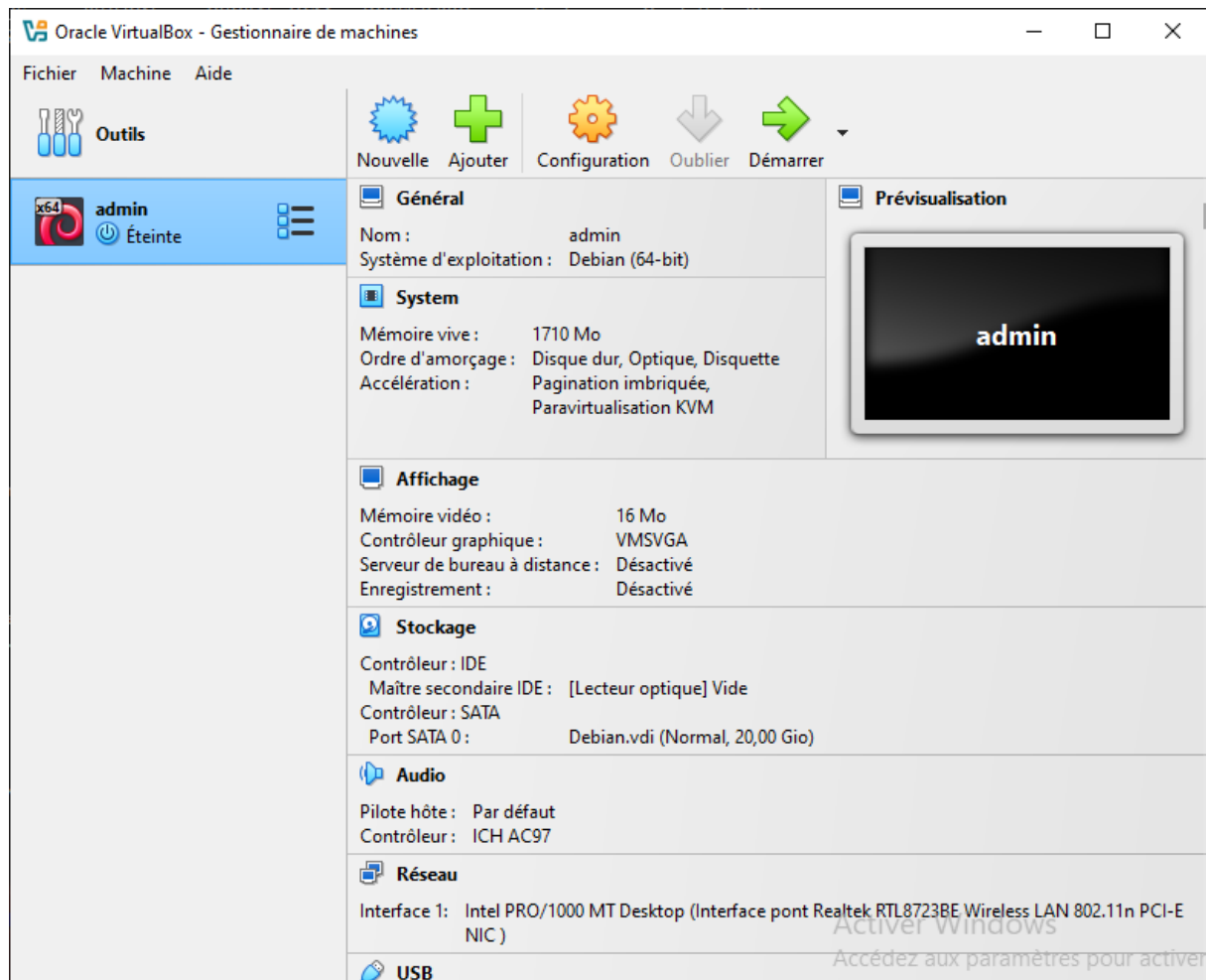
# Crée un contrôleur de disque
vm.add_storage_controller(name="SATA", controller_type=pyvbox.StorageControllerType.SATA)

# Crée la machine virtuelle
vbox.register_machine(vm)

# Démarre la machine virtuelle
vm.launch_vm_process(session=None, name="headless")

print(f"Machine virtuelle {vm_name} créée et démarrée avec succès.")

# Lancer la création de la machine virtuelle
```



Elle embarque :

- Terraform
- Ansible
- kubectl
- Jenkins
- Helm, etc.

💡 **Pourquoi en local ?** Le choix d'une VM locale permet de tester le provisionnement sans dépendre exclusivement du cloud, et de simuler un environnement "on-premise" (sur site en français, fait référence à une infrastructure qui est hébergée et gérée directement sur les locaux de l'entreprise, plutôt que dans le cloud), et aussi du à la limitation de notre abonnement cloud azure limiter à 5 cœur .

4. Configuration automatisée des VMs avec Ansible

L'outil **Ansible** a été utilisé pour configurer automatiquement les différentes VMs, selon leur rôle via un fichier inventory .

```
[master]
master-vm ansible_host=172.214.212.212 ansible_user=azureuser ansible_become=True ansible_ssh_common_args='-o StrictHostKeyChecking=no'

[worker]
worker1-vm ansible_host=172.183.176.149 ansible_user=azureuser ansible_become=True ansible_ssh_common_args='-o StrictHostKeyChecking=no'

[stockage]
stockage-vm ansible_host=192.168.15.86 ansible_user=azureuser ansible_become=True ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

```
vboxuser@vbox:~/projet_personnel$ ansible -i inventory.ini all -m ping --ask-pass --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:
worker1-vm | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
stockage-vm | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
master-vm | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
vboxuser@vbox:~/projet_personnel$
```

Des playbooks ont été conçus pour :

- Installer Kubernetes : kubeadm, kubelet, kubectl
- Initialiser le cluster avec kubeadm init et le joindre avec kubeadm join
- Installer et configurer le serveur NFS
- Déployer les outils système comme curl, htop, fail2ban

Structure des playbooks :

- kubernetes_master.yml


```

---
- name: Installer Minikube sur les VMs
  hosts: all
  become: yes
  tasks:
    - name: Mettre à jour la liste des paquets
      apt:
        update_cache: yes

    - name: Installer les dépendances nécessaires
      apt:
        name:
          - apt-transport-https
          - curl
          - virtualbox
          - docker.io
          - conntrack
        state: present

    - name: Télécharger Minikube
      get_url:
        url: https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
        dest: /usr/local/bin/minikube
        mode: '0755'

    - name: Vérifier si Minikube est installé
      command: minikube version
      register: minikube_version
      ignore_errors: yes

    - name: Afficher la version de Minikube
      debug:
        msg: "Minikube Version: {{ minikube_version.stdout }}"

```

```

SSH password:
BECOME password[defaults to SSH password]:

PLAY [Installer Minikube sur les VMs] *****
TASK [Gathering Facts] *****
k: [worker1-vm]
ok: [stockage-vm]
ok: [master-vm]

TASK [Mettre à jour la liste des paquets] *****
changed: [worker1-vm]
changed: [master-vm]
changed: [stockage-vm]

TASK [Installer les dépendances nécessaires] *****
changed: [master-vm]
changed: [stockage-vm]
changed: [worker1-vm]

TASK [Télécharger Minikube] *****
changed: [master-vm]
changed: [worker1-vm]
changed: [stockage-vm]

TASK [Vérifier si Minikube est installé] *****
changed: [master-vm]
changed: [stockage-vm]
changed: [worker1-vm]

TASK [Afficher la version de Minikube] *****
k: [master-vm] => {
  "msg": "Minikube Version: minikube version: v1.35.0\ncommit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty"
}
ok: [worker1-vm] => {
  "msg": "Minikube Version: minikube version: v1.35.0\ncommit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty"
}
ok: [stockage-vm] => {
  "msg": "Minikube Version: minikube version: v1.35.0\ncommit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty"
}

PLAY RECAP *****
master-vm      : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
stockage-vm    : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker1-vm     : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

- nfs_server.yml

```

vboxuser@vbox:~/projet_personnel$ vboxuser@vbox:~/projet_personnel$ ansible-playbook -i inventory.ini install_nfs.yml --ask-pass --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:

PLAY [Installer et configurer NFS sur la VM de stockage] *****

TASK [Gathering Facts] *****
ok: [stockage-vm]

TASK [Installer le serveur NFS] *****
changed: [stockage-vm]

TASK [Créer les répertoires partagés] *****
changed: [stockage-vm] => (item=postgres)
changed: [stockage-vm] => (item=wordpress)

TASK [Exporter les répertoires via NFS] *****
changed: [stockage-vm] => (item=postgres)
changed: [stockage-vm] => (item=wordpress)

TASK [Redémarrer le service NFS] *****
changed: [stockage-vm]

PLAY RECAP *****
stockage-vm      : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

🔍 **Ansible** est un outil d'automatisation de la configuration basé sur SSH et des fichiers YAML, réputé pour sa simplicité d'utilisation et son approche agentless (sans agent , aucun besoin d' installer un logiciel ou un processus spécial sur les machines cibles).

IV. Configuration des Applications

Les applications ont été déployées dans des **pods Kubernetes**, associées à des services réseau, du stockage persistant via NFS, et parfois une base de données externe. L' ensemble a été organisé dans un namespace pour isoler l' environnement de production.

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
glpi-7c94d8d5dd-4vzpx               1/1     Running   0           16h
odoo-844978b8c5-76w7k               1/1     Running   0           6s
postgres-8585d4bbbb-9sxc8           1/1     Running   0           16h
wordpress-6c9974f448-qgt97          1/1     Running   0           16h
azureuser@master-vm:~/projet_CI_CD$

```

A. Odoo

Pourquoi Odoo ?

Odoo est une suite ERP open-source complète qui permet de démontrer l'intégration de composants complexes (backend, frontend, base de données) dans un environnement Kubernetes.

Architecture déployée :

- 1 Pod odoo-app connecté à un service de type ClusterIP
- 1 Pod postgres-odoo avec stockage persistant via NFS

- Fichier PersistentVolume (PV) et PersistentVolumeClaim (PVC) pour PostgreSQL
- Fichier de déploiement Kubernetes pour Odoo avec variables d'environnement injectées

Outils utilisés :

- Base de données PostgreSQL (choisie pour sa robustesse et sa compatibilité native avec Odoo)
- NFS comme solution de stockage partagé pour la persistance des données

Fichiers Kubernetes :

- odoo-deployment.yml

```
-rw-rw-r-- 1 azureuser azureuser 191 Apr  8 09:11 gipi-service.yaml
-rw-rw-r-- 1 azureuser azureuser 360 Mar 31 20:14 nfs-pv.yaml
-rw-rw-r-- 1 azureuser azureuser 420 Apr  8 11:41 odoo-argocd.yaml
-rw-rw-r-- 1 azureuser azureuser 535 Apr  8 09:13 odoo-deployment.yaml
-rw-rw-r-- 1 azureuser azureuser 224 Apr  8 09:11 odoo-pv.yaml
-rw-rw-r-- 1 azureuser azureuser 175 Apr  8 09:12 odoo-pvc.yaml
-rw-rw-r-- 1 azureuser azureuser 195 Apr  8 09:13 odoo-service.yaml
-rw-rw-r-- 1 azureuser azureuser 424 Apr  8 11:42 postgres-argocd.yaml
```

- odoo-service.yml
- postgres-deployment.yml
- pv-nfs.yml, pvc-nfs.yml

The screenshot shows a web browser window with the address bar displaying 'localhost:8070/web/database/selector'. The page features the Odoo logo at the top. Below the logo, a message states 'Odoo is up and running! Fill out this form to create a new database. You will install your first app afterwards.' The form includes fields for 'Database Name', 'Email', and 'Password'. There are also dropdown menus for 'Language' (set to 'English') and 'Country'. A checkbox labeled 'Load demonstration data (Check this box to evaluate Odoo)' is present. At the bottom, there is a blue button labeled 'Create database' and a link 'or restore a database'.

Choix :

PostgreSQL a été retenue pour sa robustesse et son intégration native avec Odoo.

B. GLPI

Pourquoi GLPI ?

GLPI est un outil ITSM largement utilisé en entreprise. Il permet de mettre en œuvre un scénario réaliste de gestion de parc informatique et de tickets d'incident, tout en exploitant une base MySQL.

Architecture déployée :

- 1 Pod glpi-app (conteneur PHP + Apache)

```
azureuser 420 Apr 8 11:42 glpi-argocd.yaml
azureuser 540 Apr 8 09:10 glpi-deployment.yaml
azureuser 223 Apr 8 09:08 glpi-pv.yaml
azureuser 174 Apr 8 09:09 glpi-pvc.yaml
azureuser 191 Apr 8 09:11 glpi-service.yaml
azureuser 360 Mar 31 20:14 nfs-pv.yaml
```

- 1 Pod glpi-db basé sur MariaDB

- PV et PVC NFS pour les données de GLPI
- Configuration de secrets Kubernetes pour sécuriser les accès à la base



🔍 **MariaDB** est un fork de MySQL plus léger, souvent recommandé dans les environnements cloud.

C. WordPress

Pourquoi WordPress ?

WordPress reste un incontournable des CMS. Il constitue un cas d'usage idéal pour tester l'interconnexion entre front-end, base de données et stockage persistant dans Kubernetes.

Architecture déployée :

- 1 Pod wordpress (PHP + Apache)
- 1 Pod mariadb-wordpress
- 1 Service exposé en NodePort (ou via Ingress)
- Stockage NFS pour les contenus uploadés

Choix techniques :

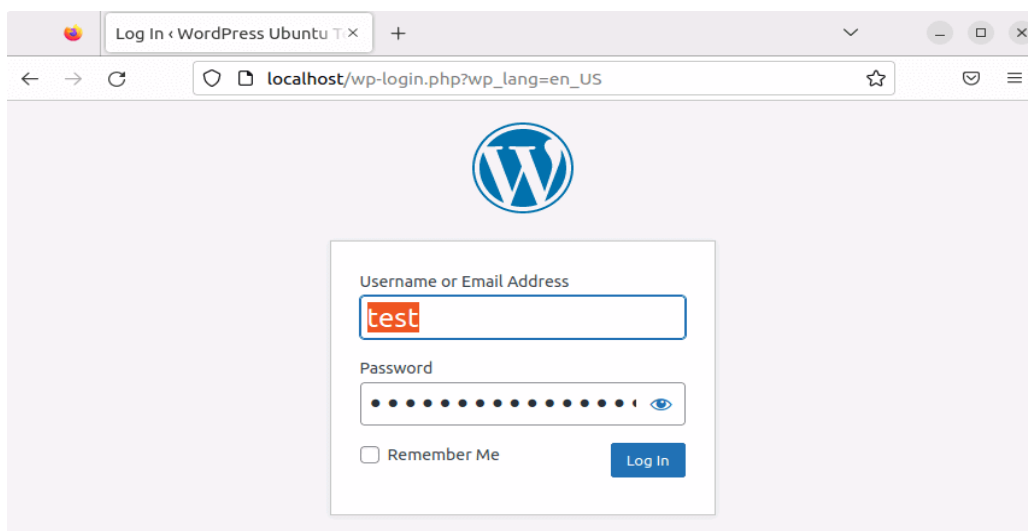
WordPress a été préféré à d' autres CMS comme Drupal pour sa simplicité de configuration et sa large documentation dans un contexte Kubernetes.

Fichiers Kubernetes :

- wordpress-deployment.yml

```
ser azureuser 425 Apr 8 11:41 wordpress-argocd.yml
ser azureuser 906 Apr 9 11:58 wordpress-deployment.yml
ser azureuser 382 Apr 9 09:36 wordpress-ingress.yml
ser azureuser 239 Apr 8 09:06 wordpress-pv.yml
ser azureuser 185 Apr 8 09:07 wordpress-pvc.yml
ser azureuser 206 Apr 8 09:08 wordpress-service.yml
~/projet_CI_CD$
```

- wordpress-service.yml
- wordpress-mariadb.yml
- wordpress-pvc-nfs.yml



🔍 **CMS** (Content Management System) : Plateforme de création et de gestion de contenus web.

D. Gestion des Namespaces

Toutes les applications ont été déployées dans un **namespace** dédié pour une meilleure **isolation logique** et une gestion plus fine des **ressources**.


Utiliser des namespaces dans Kubernetes présente plusieurs avantages concrets :

- ☒ **Isolation logique des applications** : chaque application ou environnement (dev, test, prod) fonctionne dans un espace isolé, évitant les conflits de configuration ou de noms entre les ressources (pods, services, secrets...).
- ☒ **Meilleure gestion des accès** : les RBAC (Role-Based Access Control) peuvent être définis par namespace, ce qui permet de restreindre les droits des utilisateurs ou des services à leur périmètre d' action.
- ☒ **Organisation claire du cluster** : en regroupant les ressources par namespace, on facilite la **lecture**, la **maintenance** et le **dépannage** des applications.
- ☒ **Quota de ressources** : il est possible de définir des **ResourceQuotas** ou des **LimitRanges** pour éviter qu'une application ne consomme trop de CPU ou de mémoire au détriment des autres.
- ☒ **Nettoyage plus simple** : supprimer un namespace entraîne la suppression de toutes les ressources associées (pods, services, configmaps, etc.), ce qui simplifie le nettoyage lors de tests ou de déploiements temporaires.

Ainsi, l' usage des namespaces contribue à la **sécurité**, à la **scalabilité** et à la **bonne gouvernance** du cluster Kubernetes.

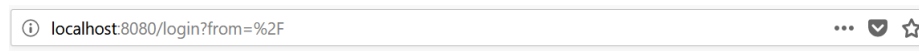
V. Chaîne CI/CD : Jenkins, SonarQube, Trivy et ArgoCD

La mise en place d' une *chaîne CI/CD* moderne permet d' automatiser l' intégration, la validation, la sécurité et le déploiement continu des applications. Notre pipeline s' appuie sur quatre composants majeurs, chacun jouant un rôle clé dans la chaîne DevOps.

 **CI/CD : Continuous Integration / Continuous Deployment**. Cela désigne un ensemble de pratiques d'automatisation pour intégrer et déployer plus rapidement les changements de code, tout en assurant qualité et stabilité.

A. Jenkins : le cœur de l'automatisation

Jenkins a été choisi pour sa modularité, sa grande compatibilité avec les outils de sécurité (Trivy, SonarQube), et sa capacité à orchestrer facilement des pipelines complexes via Jenkinsfile.



Welcome to Jenkins!

Sign in

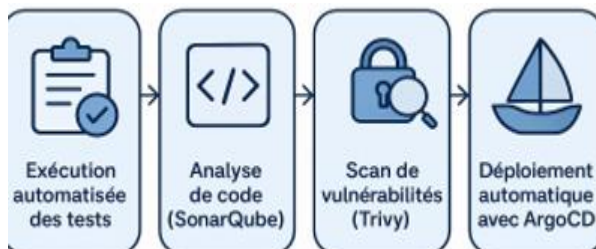
☐ Keep me signed in

Déploiement :

- Jenkins est installé sur la VM d'administration locale.
- Il fonctionne comme serveur autonome avec interface Web (port 8080).
- Les agents Jenkins tournent dans des pods Kubernetes, grâce au plugin Kubernetes Jenkins Agent.
- Les pipelines sont définis en tant que code via des Jenkinsfile.

Fonctionnalités intégrées :

- Exécution automatisée des tests



- Analyse de code (SonarQube)
- Scan de vulnérabilités (Trivy)
- Déploiement automatique avec ArgoCD

✳ **Important** : Jenkins assure l'orchestration de l'ensemble de la chaîne CI/CD. Il est le **point d'entrée unique** de chaque exécution.

B. SonarQube : Analyse de code

SonarQube a été retenu pour ses capacités avancées d'analyse statique de code, permettant de détecter les bugs, vulnérabilités, code smells et dettes techniques dès les premières phases de développement.

✓ Pourquoi SonarQube ?

- Multilangage : prise en charge de +25 langages (Java, JavaScript, Python, PHP, etc.).
- Intégration native avec Jenkins : facilite l'intégration continue (CI) avec génération automatique de rapports qualité.
- Tableaux de bord clairs : visualisation centralisée de la qualité du code avec historique et tendances.
- Support de règles personnalisées : possibilité de configurer ses propres règles selon les standards de l'équipe.

⚙ Déploiement dans Kubernetes :

- 1 Pod SonarQube a été déployé dans un namespace dédié (namespace: monitoring par exemple) pour une isolation logique.
- Le storage est assuré via un PersistentVolumeClaim (PVC), permettant de conserver les données entre les redémarrages.
- Le service exposé en interne permet aux autres composants (ex : Jenkins) d'y accéder facilement.
- L'image utilisée est celle de SonarQube officielle, avec des ressources CPU/mémoire limitées pour une meilleure gestion du cluster.

11d	service/prometheus-prometheus-node-exporter	ClusterIP	10.98.249.165	<none>	9100/TCP		
11d	service/sonarqube-service	ClusterIP	10.109.125.249	<none>	9000/TCP		
13s							
NAME			DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE
OR	AGE						
daemonset.apps/prometheus-prometheus-node-exporter			2	2	2	2	2
io/os=linux	11d						kubernetes.

Pipeline Jenkins associé :

- L' étape d' analyse SonarQube est appelée juste après le checkout du code.
- Résultats visibles dans SonarQube Web UI (problèmes de sécurité, duplications, bugs, couverture de test).

Définition : NFS (Network File System)




Protocole permettant de **partager un volume entre plusieurs machines** ou pods. Il est utilisé ici pour la persistance des données de SonarQube.

Extrait Jenkinsfile :

```
stage('Code Quality') {
  steps {
    script {
      sh 'sonar-scanner -Dsonar.projectKey=glpi -Dsonar.host.url=http://sonarqube.env-prod -Dsonar.login=$SONAR_TOKEN'
    }
  }
}
```

C. Trivy : Scan de vulnérabilités

Trivy a été retenu pour assurer l' analyse de sécurité des images Docker dans le pipeline CI/CD, grâce à ses nombreux avantages :

-  **Léger et rapide** : Trivy offre des scans efficaces sans ralentir la pipeline.
-  **Compatible Docker** : Il scanne directement les images Docker, sans besoin de registry intermédiaire.
-  **Facile à intégrer** : Une simple commande en ligne de commande permet de l' exécuter dans Jenkins.

- 📄 **Rapports exploitables** : Il génère des rapports au format .json, facilement intégrables ou visualisables par d'autres outils.
-

🔧 Utilisation dans la CI Jenkins :

- Trivy est **installé directement dans la VM Jenkins** (installation via apt ou binaire GitHub).
- Il est appelé **automatiquement dans le Jenkinsfile**, après l'étape docker build de chaque application.
- commande utilisée :

```
vm:~$ trivy image --format json -o trivy-report.json nom-de-l-image
```

- Le rapport .json est ensuite :
 - Archivé dans Jenkins (archiveArtifacts)
 - Utilisé pour des alertes ou des dashboards si nécessaire
-

🛡 Objectif

Grâce à cette étape d'analyse automatique, toute image contenant des vulnérabilités critiques est **identifiée avant le déploiement**, permettant d'assurer une meilleure **sécurité applicative dès la phase de développement**





Extrait Jenkinsfile :

```
stage('Vulnerability Scan') {  
  steps {  
    sh 'trivy image myapp:latest > trivy-report.json || true'  
  }  
}
```



□ **Note** : Le `|| true` permet de ne **pas faire échouer le pipeline** même si des vulnérabilités sont détectées, afin de conserver les rapports pour analyse.

D. ArgoCD : Déploiement GitOps


ArgoCD a été choisi pour orchestrer le déploiement applicatif dans Kubernetes selon les principes GitOps, où Git devient la source de vérité. Cette approche garantit :

-  Une traçabilité complète de chaque modification (via commit Git)
-  Une synchronisation automatique entre les manifests déclaratifs et l'état réel du cluster
-  Une réversibilité facile (rollback possible en revenant à un commit antérieur)
-  Une collaboration facilitée grâce à la gestion des versions et des PR




Déploiement Kubernetes

-  Namespace dédié argocd pour l'isoler des autres composants
-  Le **service ArgoCD Server** est de type LoadBalancer
- Une **IP publique** ou un **DNS cloud** permet d'y accéder depuis l'extérieur

Il est recommandé d'ajouter un **certificat TLS** sur le LoadBalancer (via cert-manager ou proxy HTTPS)

-  Accès sécurisé à l'interface Web ArgoCD : login admin, mot de passe initial stocké dans un Secret Kubernetes (argocd-initial-admin-secret)

□ Architecture GitOps avec ArgoCD

-  ArgoCD lit un dépôt Git centralisé contenant tous les fichiers manifest Kubernetes (YAML)
-  Chaque application est représentée comme une ressource Application dans ArgoCD, pointant vers le sous-dossier correspondant dans le dépôt
-  ArgoCD détecte automatiquement toute divergence entre l'état Git et l'état réel du cluster :

- En cas de différence, une alerte visuelle est affichée dans le dashboard
- Il est possible de déclencher une resynchronisation automatique ou manuelle

💡 Avantages clés

- CI/CD découplée : Jenkins gère la build et push de l' image Docker, ArgoCD gère la déclaration et déploiement
- Vision claire : Interface web conviviale avec statut des applis, santé, synchronisation, historique
- Auditabilité : Historique complet des synchronisations et des changements appliqués

The screenshot displays the ArgoCD web interface. The top section, titled 'Settings / Repositories', shows a table of configured repositories. Below this, the 'APPLICATION DETAILS NETWORK' section for the 'wordpress' application is visible. It includes a 'SYNC STATUS' section indicating the application is 'Synced to HEAD (9410ad3)' and a 'LAST SYNC' section showing the last sync was successful. The bottom part of the interface shows a network diagram with services like 'glpi', 'odoo', 'postgres-service', and 'wordpress' mapped to their respective pods in the cluster.

TYPE	NAME	PROJECT	REPOSITORY	CONNECTION STATUS
git	projetCI_CD	projet_CI_CD	https://github.com/moyoboza/projet_CI_CD.git	Successful

APP HEALTH: Progressing

SYNC STATUS: Synced to HEAD (9410ad3)

LAST SYNC: Sync OK to 9410ad3

APP CONDITIONS: 2 Warnings

⚠ **Important** : ArgoCD détecte automatiquement les écarts entre l' état Git et le cluster, ce qui permet un **rollback immédiat** en cas d'erreur.

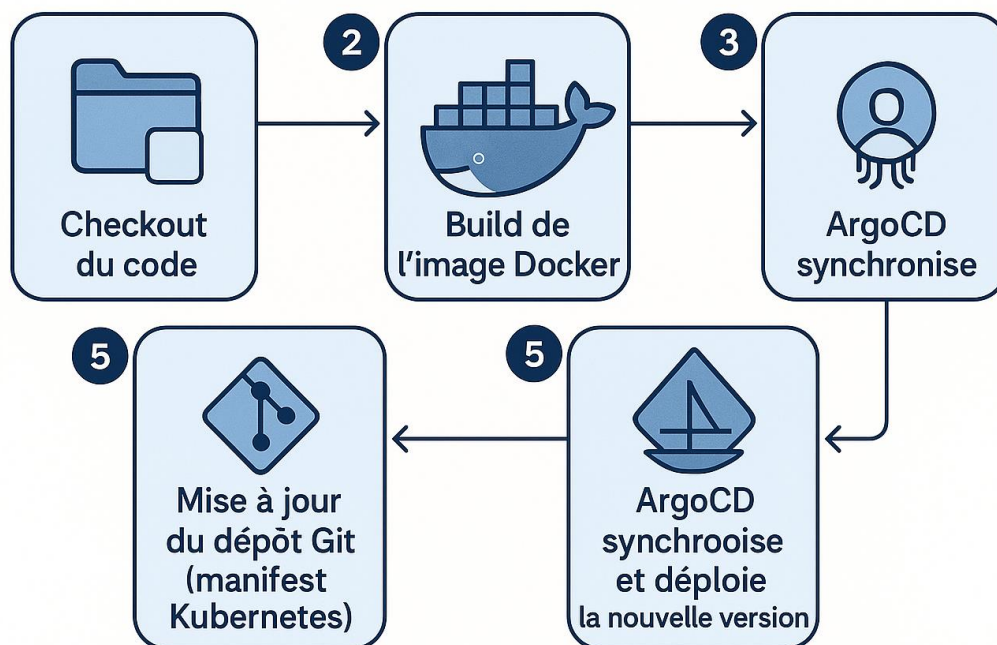
E. Déclenchement automatique des Pipelines

L' intégration continue est assurée via des **Webhooks GitHub**, qui déclenchent Jenkins à chaque git push.

Cycle CI/CD complet :

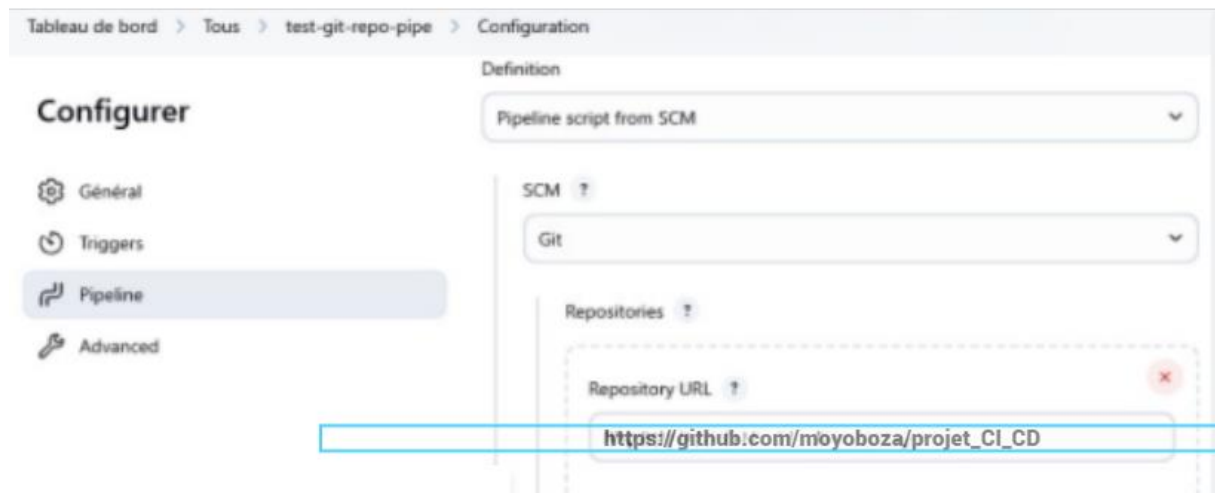
1. Checkout du code
2. Analyse SonarQube
3. Build de l' image Docker
4. Scan Trivy
5. Push de l' image vers un registre local
6. Mise à jour du dépôt Git (manifest Kubernetes)
7. ArgoCD synchronise et déploie la nouvelle version

Cycle CI/CD complet:

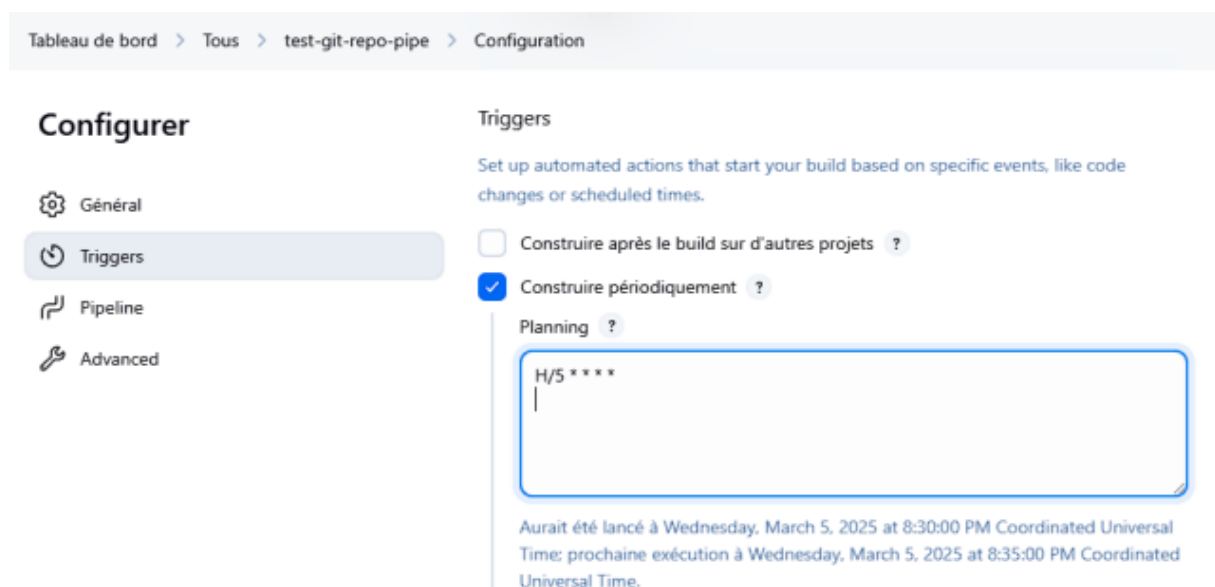


📌 **Note pédagogique :** Cette boucle CI/CD s'aligne parfaitement avec les principes DevOps d'automatisation, transparence et traçabilité.

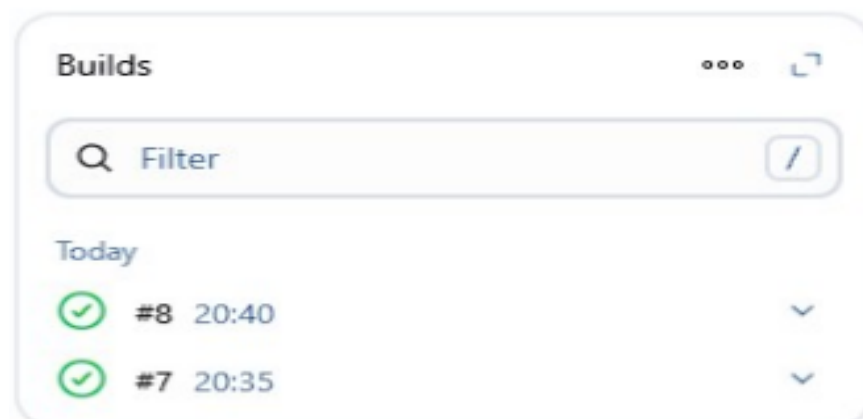
Configuration du pipeline sur jenkins



Configuration d'un triggers pour un build toutes les 5 min



Builds effectuer



VI. État des Applications Déployées

Cette section présente l'état final des applications déployées dans l'environnement Kubernetes ainsi que les vérifications techniques et fonctionnelles qui ont permis de valider leur bon fonctionnement.

A. Vérification des Composants Déployés

Chaque application (Odoo, GLPI, WordPress) a été déployée à l'aide de manifests Kubernetes versionnés dans un dépôt Git. Les déploiements ont été orchestrés par ArgoCD, garantissant l'alignement permanent entre la configuration déclarative et l'état réel du cluster.

- Tous les pods dans l'état **Running**
- Services exposés correctement (ClusterIP et NodePort)
- Ingress configurés avec nom DNS local
- Applications accessibles via navigateur

Application	Namespace	Pods	Accès Ingress	Base de données	Status
GLPI	env-prod	2 (Web + DB)	http://glpi.env-prod.local	MariaDB	OK
Odoo	env-prod	3 (Web+DB+Worker)	http://odoo.env-prod.local	PostgreSQL	OK
WordPress	env-prod	2 (WP + DB)	http://wp.env-prod.local	MySQL	OK

C. Persistance des Données

Toutes les bases de données sont associées à des volumes persistants provisionnés via NFS. Les volumes sont montés automatiquement via des PersistentVolumeClaims (PVC).

Installer et configurer le client NFS sur les nœuds Kubernetes

```
- name: Installer et configurer le client NFS sur les nœuds Kubernetes
hosts: master, worker
become: true
tasks:

  - name: Installer le client NFS
    apt:
      name: nfs-common
      state: present
      update_cache: yes

  - name: Créer les répertoires de montage
    file:
      path: "{{ item }}"
      state: directory
      mode: '0755'
    loop:
      - /mnt/postgres
      - /mnt/wordpress

  - name: Ajouter les entrées NFS à /etc/fstab
    lineinfile:
      path: /etc/fstab
      line: "{{ item }}"
      create: yes
    loop:
      - "192.168.15.86:/data/postgres /mnt/postgres nfs defaults 0 0"
      - "192.168.15.86:/data/wordpress /mnt/wordpress nfs defaults 0 0"

  - name: Monter les partages NFS
    command: mount -a

  - name: Vérifier les montages NFS
    command: df -h
    register: mount_output

  - name: Afficher la sortie des montages
    debug:
      msg: "{{ mount_output.stdout_lines }}"
```

Configurer Kubernetes pour utiliser NFS

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /data/postgres
    server: stockage-vm
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi

```

creation du volume persistant sur le worker et sur le master

```

azureuser@master-vm:~$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
master-vm     Ready    control-plane  23d    v1.28.15
worker1-vm    Ready    <none>         19d    v1.28.15

```

```

azureuser@worker1-vm:~$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
master-vm     Ready    control-plane  3d6h   v1.28.15
worker1-vm    Ready    <none>         3d3h   v1.28.15
azureuser@worker1-vm:~$ kubectl apply -f nfs-pv.yaml
persistentvolume/postgres-pv created
persistentvolumeclaim/postgres-pvc created
azureuser@worker1-vm:~$ kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM                STORAGECLASS  REASON  AGE
postgres-pv   5Gi       RWX           Retain          Bound   default/postgres-pvc  default      5m3s
azureuser@worker1-vm:~$ kubectl get pvc
NAME          STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
postgres-pvc  Bound   postgres-pv     5Gi       RWX           default      5m7s

```

NFS est bien monté sur Master et Worker

```

azureuser@master-vm:~$ sudo mount -t nfs 172.214.128.162:/data/postgres /mnt/postgres
azureuser@master-vm:~$ mount | grep nfs
172.214.128.162:/data/postgres on /mnt/postgres type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsiz=262144,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=10.0.0.6,local_lock=none,addr=172.214.128.162)
azureuser@master-vm:~$ ls -l /mnt/postgres
total 0
azureuser@master-vm:~$ df -h | grep /mnt/postgres
172.214.128.162:/data/postgres 29G 1.6G 28G 6% /mnt/postgres
azureuser@master-vm:~$ mount | grep nfs
172.214.128.162:/data/postgres on /mnt/postgres type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsiz=262144,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=10.0.0.6,local_lock=none,addr=172.214.128.162)
172.214.128.162:/data/wordpress on /mnt/wordpress type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsiz=262144,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=10.0.0.6,local_lock=none,addr=172.214.128.162)
azureuser@master-vm:~$ df -h | grep /mnt/postgres
172.214.128.162:/data/postgres 29G 1.6G 28G 6% /mnt/postgres
azureuser@master-vm:~$ ls -l /mnt/postgres
total 0

```

Vérifier que le PVC est bien lié au PV

```

azureuser@master-vm:~$ kubectl get pvc
NAME          STATUS    VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgres-pvc  Bound    postgres-pv     5Gi        RWX            default        49m
azureuser@master-vm:~$ kubectl describe pvc postgres-pvc
Name:          postgres-pvc
Namespace:     default
StorageClass:  default
Status:        Bound
Volume:        postgres-pv
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      5Gi
Access Modes:  RWX
VolumeMode:    Filesystem
Used By:       <none>
Events:        <none>

```

D. Résilience et Redémarrage

Des tests de résilience ont été menés pour simuler des pannes :

- Suppression manuelle de pods → redéploiement automatique (ReplicaSet)

```

azureuser@master-vm:~$ kubectl delete pods odoo-844978b8c5-45cm2
pod "odoo-844978b8c5-45cm2" deleted

```

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
glpi-7c94d8d5dd-xrcbl              1/1    Running   0           107s
odoo-844978b8c5-tjjxh              1/1    Running   0           73s
postgres-8585d4bbb-zjdtx           1/1    Running   0           25h
wordpress-6c9974f448-lvqmk         1/1    Running   0           3m10s

```

- Redémarrage du nœud Worker → pods reprogrammés avec succès

```

azureuser@worker1-vm:~$ sudo reboot
[sudo] password for azureuser:
Connection to 172.183.176.149 closed by remote host.
Connection to 172.183.176.149 closed.

```

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get nodes
NAME          STATUS    ROLES          AGE   VERSION
master-vm     Ready     control-plane   23d   v1.28.15
worker1-vm    NotReady  <none>          19d   v1.28.15
azureuser@master-vm:~/projet_CI_CD$

```

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
master-vm     Ready    control-plane  23d    v1.28.15
worker1-vm    Ready    <none>         19d    v1.28.15

```

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get pods -o wide --all-namespaces | grep worker1-vm
argocd          argocd-application-controller-0          1/1      Running    0          3h53m
92.168.94.252   worker1-vm <none> <none>
argocd          argocd-applicationset-controller-555cf564b4-q24cn 1/1      Running    0          3h58m
92.168.94.250   worker1-vm <none> <none>
argocd          argocd-dex-server-55874cb5fd-2rxwd        1/1      Running    0          3h58m
92.168.94.200   worker1-vm <none> <none>
argocd          argocd-notifications-controller-8f5c7f7ff-mctkt 1/1      Running    0          3h58m
92.168.94.230   worker1-vm <none> <none>
argocd          argocd-redis-85888cc66-pxhmm             1/1      Running    0          3h58m
92.168.94.244   worker1-vm <none> <none>
argocd          argocd-repo-server-6c4b9ffb7-qt4fj       1/1      Running    0          3h58m
92.168.94.231   worker1-vm <none> <none>
argocd          argocd-server-746f4c996d-j7r2c          1/1      Running    0          3h58m
92.168.94.227   worker1-vm <none> <none>
cert-manager    cert-manager-9f74c854d-vf5zq            1/1      Running    0          3h21m
92.168.94.245   worker1-vm <none> <none>
cert-manager    cert-manager-cainjector-665cd78979-xkkgh  1/1      Running    0          3h21m
92.168.94.253   worker1-vm <none> <none>
cert-manager    cert-manager-webhook-65767c6f65-sqkpr    1/1      Running    0          3h13m
92.168.94.236   worker1-vm <none> <none>
default         glpi-7c94d8d5dd-cngkl                   1/1      Running    0          3h58m
92.168.94.201   worker1-vm <none> <none>
default         odoo-844978b8c5-8cj8j                   1/1      Running    44 (5m32s ago) 4h1m
92.168.94.232   worker1-vm <none> <none>
default         postgres-8585d4bbbb-dp2ft               1/1      Running    0          3h58m
92.168.94.247   worker1-vm <none> <none>
default         wordpress-6c9974f448-pbgn8              1/1      Running    0          3h58m

```

Ces tests ont permis de confirmer la robustesse de l'orchestration Kubernetes, ainsi que l'autonomie des composants dans un environnement distribué.



Important : Résilience Kubernetes offre des mécanismes natifs de tolérance aux pannes via ses *controllers* (ReplicaSet, Deployment...).

VII. Infrastructure Sécurisée

La sécurisation de l'infrastructure est un volet central de ce projet. L'objectif est de contrôler les accès, sécuriser les flux réseau, garantir la confidentialité des échanges via TLS (*Transport Layer Security*) anciennement SSL (*Secure Sockets Layer*) qui sont des protocoles de **sécurité**, et mettre en place une protection active contre les menaces externes.

Pour cela, nous avons intégré les composants suivants :

- **Nginx Ingress Controller** : Stable, bien documenté, adapté aux environnements Kubernetes , il permet de gérer les routes HTTP/S dans Kubernetes
 - **Cert-Manager** : pour la génération et le renouvellement automatique des certificats TLS il est une Solution native Kubernetes, simple à automatiser, évite les erreurs humaines sur les certificats.
 - **Crowdsec** : Moderne, collaboratif, conçu pour le cloud natif (contrairement à fail2ban) , il est très utile pour la détection des comportements malveillants et la mitigation automatique
-

A. Sécurité de base : durcissement système

Avant toute chose, un certain nombre de bonnes pratiques de sécurité ont été appliquées à l' ensemble des VMs :

- Mise à jour régulière via Ansible (apt upgrade)
 - Désactivation de SSH root login
 - Configuration de fail2ban en complément local
 - Firewall ufw configuré sur les ports nécessaires (SSH, HTTP/S, Kubernetes, NFS)
 - Accès SSH restreint par clé publique
-

B. Nginx Ingress Controller

1. Installation et configuration

Le contrôleur Nginx a été installé dans le cluster via **Helm**

Une fois déployé, il permet de :

- Centraliser la gestion des routes HTTP/S
- Appliquer des règles de sécurité (rate limiting, whitelisting IP, etc.)
- Terminer les connexions TLS

Pourquoi Nginx ?

Nous avons choisi **Nginx** pour sa **stabilité**, sa **flexibilité**, et sa **documentation étendue**. Il s'intègre parfaitement avec Kubernetes contrairement à Apache ou Envoy, plus complexes à configurer dans ce contexte.

2. Exposition via Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: wordpress-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: wordpress.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

```
azureuser@master-vm:~$ kubectl get pods -n ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-6d7f74b8ff-5kzxq  1/1     Running   0           18m
```

C. Cert-Manager : gestion des certificats TLS

Cert-Manager a été intégré afin de sécuriser automatiquement tous les Ingress en HTTPS. Il gère :

- La génération de certificats auto-signés (staging)
- Le renouvellement automatique des certificats
- L'intégration avec Let's Encrypt via ACME HTTP challenge

```
azureuser@master-vm:~/projet_CI_CD$ kubectl apply -f clusterissuer-staging.yaml
clusterissuer.cert-manager.io/letsencrypt-staging created
azureuser@master-vm:~/projet_CI_CD$ kubectl get pods -n cert-manager
NAME                                READY   STATUS    RESTARTS   AGE
cert-manager-9f74c854d-vf5zq        1/1     Running   0           6m39s
cert-manager-cainjector-665cd78979-xkkgh  1/1     Running   0           6m39s
cert-manager-webhook-65767c6f65-d4c4w  1/1     Running   0           82s
```

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    # Adresse email pour les notifications de Let's Encrypt
    email: ton.email@example.com
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-staging
    solvers:
      - http01:
          ingress:
            class: nginx

```

```

azureuser@master-vm:~/projet_CI_CD$ kubectl describe clusterissuer letsencrypt-staging
Name:      letsencrypt-staging
Namespace:
Labels:     <none>
Annotations: <none>
API Version: cert-manager.io/v1
Kind:      ClusterIssuer
Metadata:
  Creation Timestamp:  2025-04-21T05:12:27Z
  Generation:         2
  Resource Version:    886523
  UID:                 342e0f9f-8323-40e3-9447-1018e910bb8d
Spec:
  Acme:
    Email:  jujumoyo5@gmail.com
    Private Key Secret Ref:
      Name:  letsencrypt-staging
    Server:  https://acme-v02.api.letsencrypt.org/directory
    Solvers:
      http01:
        Ingress:
          Class:  nginx
Status:
  Acme:
    Conditions:
      Last Transition Time:  2025-04-21T05:12:58Z

```

Pourquoi Cert-Manager ?

Nous avons retenu **Cert-Manager** car il est **nativement compatible avec Kubernetes** et prend en charge les mécanismes ACME, contrairement à des solutions comme Traefik ou acme.sh qui nécessitent plus de configuration manuelle.

D. Protection active avec Crowdsec

1. Présentation de Crowdsec

Crowdsec est une solution open-source de détection comportementale, inspirée de fail2ban mais pensée pour les environnements cloud et conteneurisés. Elle permet de détecter les tentatives de brute-force, de scan de port, ou de comportements anormaux, et d'y répondre automatiquement via des "Bouncers".


```

azureuser@master-vm:~/projet_CI_CD$ sudo systemctl status crowdsec
● crowdsec.service - Crowdsec agent
   Loaded: loaded (/lib/systemd/system/crowdsec.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2025-04-21 05:33:18 UTC; 20s ago
     Process: 2698 ExecStartPre=/usr/bin/crowdsec -c /etc/crowdsec/config.yaml -t -error (code=exited, status=0/SUCCESS)
    Main PID: 2759 (crowdsec)
      Tasks: 7 (limit: 4915)
     CGroup: /system.slice/crowdsec.service
            └─2759 /usr/bin/crowdsec -c /etc/crowdsec/config.yaml

Apr 21 05:33:14 master-vm systemd[1]: Starting Crowdsec agent...
Apr 21 05:33:18 master-vm systemd[1]: Started Crowdsec agent.

```

Pourquoi Crowdsec ?

Crowdsec a été préféré à fail2ban pour son architecture distribuée, sa base de données communautaire de menaces, et son intégration possible avec Nginx via Bouncer.

2. test du Bouncer Nginx

Le bouncer Crowdsec pour Nginx a été installé pour bloquer les IP malveillantes directement au niveau du reverse proxy.

```

azureuser@master-vm:~/projet_CI_CD$ for i in {1..50}; do curl http://localhost/wp-login.php; done
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.14.0 (Ubuntu)</center>
</body>
</html>
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.14.0 (Ubuntu)</center>
</body>
</html>
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.14.0 (Ubuntu)</center>
</body>
</html>
<html>
<head><title>404 Not Found</title></head>
<body bgcolor="white">
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.14.0 (Ubuntu)</center>
</body>
</html>

```

```

    if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {
      document.body.classList.add('dark');
      svg.forEach(element => {
        element.setAttribute('fill', 'white');
      });
      button.innerHTML = sunSvgString;
    } else {
      button.innerHTML = moonSvgString;
    }
    button.addEventListener('click', function() {
      document.body.classList.toggle('dark');
      svg.forEach(element => {
        element.getAttribute('fill') === 'black' ? element.setAttribute('fill', 'white') : element.setAttribute('
fill', 'black');
      });
      if (document.body.classList.contains('dark')) {
        button.innerHTML = sunSvgString;
      } else {
        button.innerHTML = moonSvgString;
      }
    });
  });
</script>
</body>
</html>
azureuser@master-vm:~/projet_CI_CD$ cat /etc/nginx/conf.d/crowdsec.conf
deny 127.0.0.1;

```

🔗 **Note pédagogique** : L'agent a été enrôlé dans la console Crowdsec Central pour bénéficier des scénarios collaboratifs

La combinaison de ces outils assure un **niveau de sécurité réseau et applicatif solide**, compatible avec les exigences d'un environnement de production.

VIII. Supervision et Observabilité

Assurer une visibilité en temps réel sur l'état de l'infrastructure et des applications est essentiel en environnement DevOps. Pour cela, nous avons mis en place une solution complète de monitoring basée sur **Prometheus** pour la collecte de métriques et **Grafana** pour leur visualisation.

A. Installation de Prometheus et Grafana via Helm

Les deux outils ont été installés à l'aide de la stack communautaire kube-prometheus-stack (Norme de facto pour la supervision Kubernetes, avec dashboards prêts à l'emploi), qui facilite une intégration native dans Kubernetes :

```

azureuser@master-vm:~/projet_CI_CD$ kubectl get all -n monitoring
NAME                                READY    STATUS    RESTARTS    AGE
pod/alertmanager-prometheus-kube-prometheus-alertmanager-0  2/2     Running   0            159m
pod/prometheus-grafana-6b97dd6875-7rm65                      3/3     Running   0            164m
pod/prometheus-kube-prometheus-operator-6b65c566bd-8rmq4     1/1     Running   0            164m
pod/prometheus-kube-state-metrics-5d84f96dc4-jjsmj           1/1     Running   3 (151m ago) 164m
pod/prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0            159m
pod/prometheus-prometheus-node-exporter-2b8vz                1/1     Running   2 (151m ago) 11d
pod/prometheus-prometheus-node-exporter-r6njd                1/1     Running   1 (160m ago) 11d

```

Ce chart installe automatiquement :

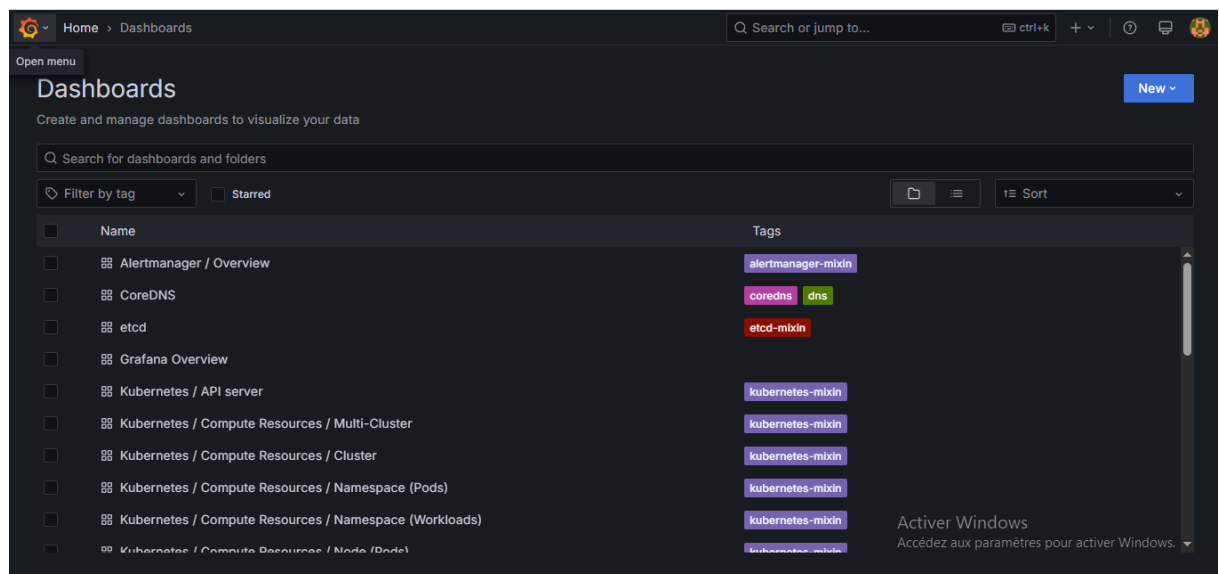
- Prometheus Server
- Node Exporter (métriques système des nœuds)
- Kube-State-Metrics (métriques liées aux objets Kubernetes)
- Grafana avec des dashboards préconfigurés

Pourquoi Prometheus & Grafana ?

Ces outils sont **standard dans l' écosystème cloud-native**, largement adoptés par la communauté Kubernetes. D' autres solutions comme Zabbix ou Datadog nécessitent davantage de configuration ou ne sont pas open source.

B. Configuration des Dashboards Grafana

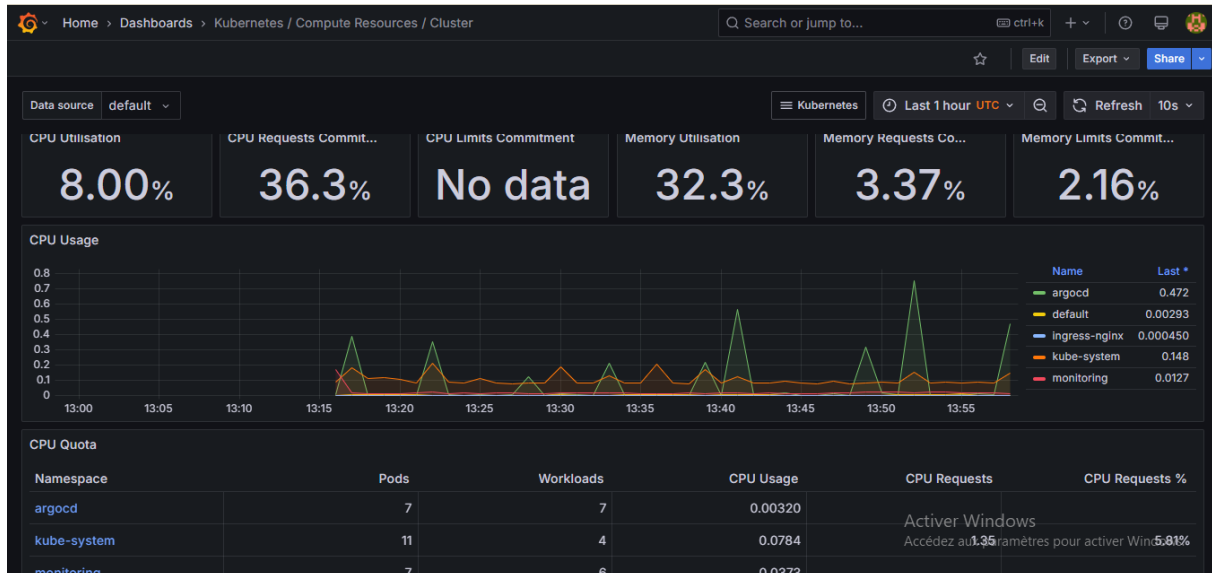
Une fois Grafana exposé via un Ingress sécurisé, plusieurs dashboards clés ont été configurés :



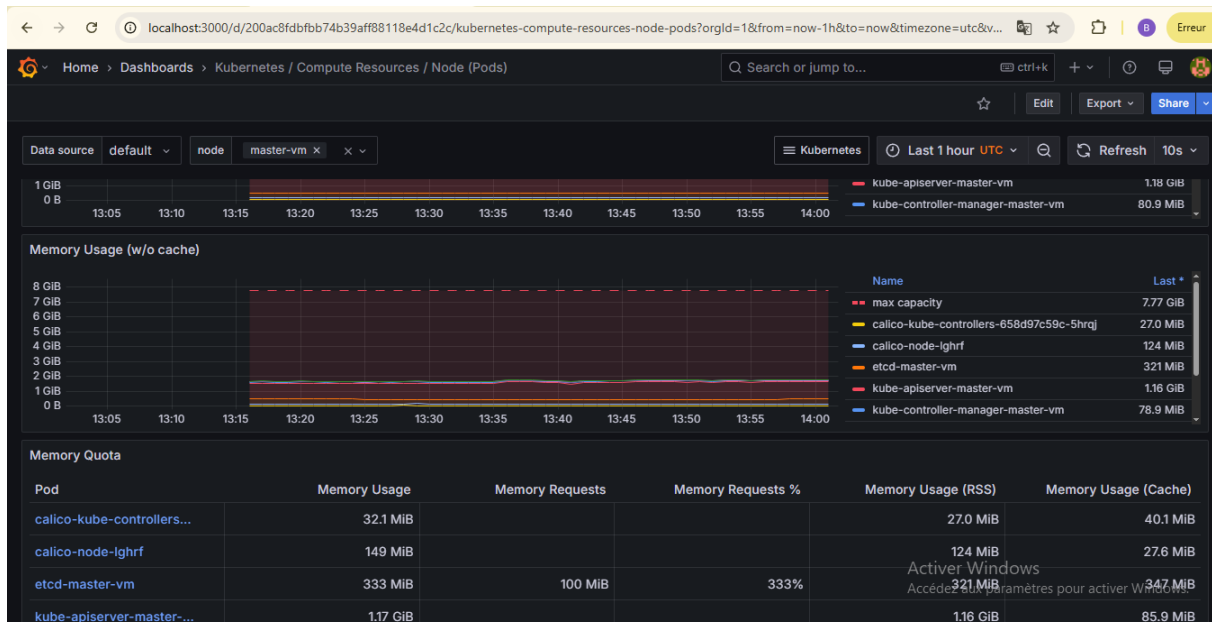
- Utilisation CPU/mémoire des nœuds Kubernetes
- État des pods et des namespaces

- Monitoring réseau (latence, trafic)
- Statistiques personnalisées pour les applications (Odo, GLPI...)

Dashboard général du cluster



Compute ressources / nodes (pods)



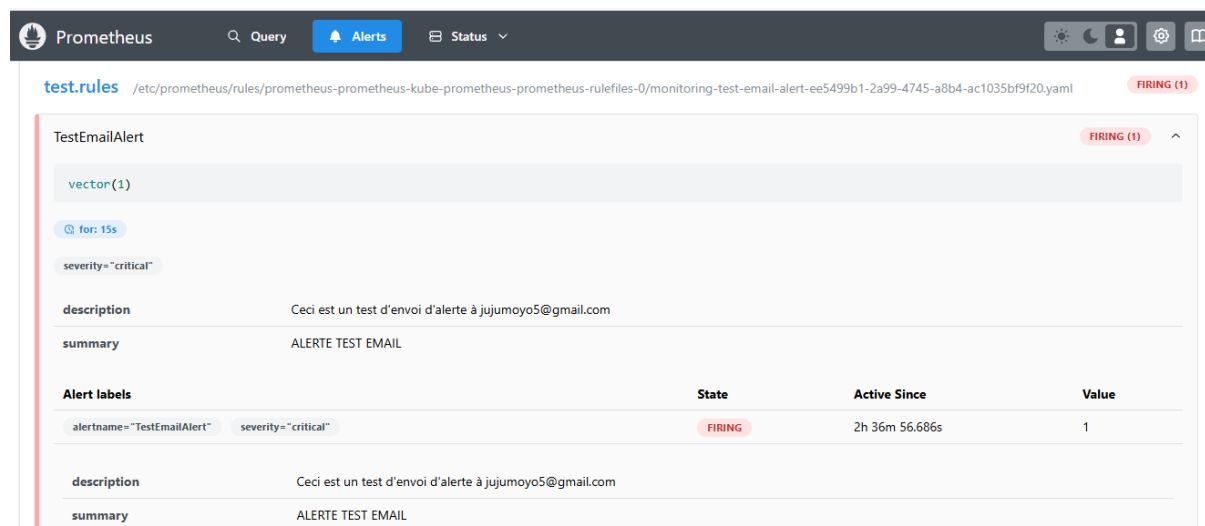
C. Alerting

Prometheus et Alertmanager sont configurés pour générer des alertes critiques :

- Nœud hors ligne
- Pod en crashloop
- Charge CPU > 90% pendant 5 minutes
- Service inatteignable (probes)

```
alertmanager:
  config:
    global:
      smtp_smarthost: 'smtp.gmail.com:587'
      smtp_from: 'jujumoyo5@gmail.com'
      smtp_auth_username: 'jujumoyo5@gmail.com'
      smtp_auth_password: 'APP_PASSWORD'
      smtp_require_tls: true
    route:
      receiver: email-alerts
    receivers:
      - name: email-alerts
        email_configs:
          - to: 'jujumoyo5@gmail.com'
            send_resolved: true
```

```
# test-email-alert.yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: test-email-alert
  namespace: monitoring
  labels:
    release: prometheus
spec:
  groups:
    - name: test.rules
      rules:
        - alert: TestEmailAlert
          expr: vector(1)
          for: 15s
          labels:
            severity: critical
          annotations:
            summary: "ALERTE TEST EMAIL"
            description: "Ceci est un test d'envoi d'alerte à jujumoyo5@gmail.com"
```



Ces alertes sont actuellement visibles dans l'interface web d'Alertmanager. Une future amélioration consisterait à les envoyer vers des canaux comme Discord, Slack ou email.

D. Métriques applicatives personnalisées

Pour les applications comme WordPress et Odoo, nous avons ajouté des endpoints /metrics exposés grâce à des exporters (notamment pour Nginx et PostgreSQL). Ces données sont exploitées dans des dashboards dédiés.

Conclusion de la section

Cette stack permet un suivi en **temps réel**, une **visualisation conviviale**, et une **réaction rapide aux anomalies**, ce qui est crucial pour la résilience et la fiabilité de l'infrastructure.

IX. Conclusion et Perspectives

Ce projet de fin de formation en Administration Système DevOps et Cybersécurité nous a permis de mettre en œuvre, de manière concrète et réaliste, l'ensemble des compétences techniques et méthodologiques acquises durant notre parcours.

Nous avons conçu, automatisé, sécurisé et supervisé une infrastructure de production complète, reposant sur des outils modernes tels que :

- Terraform & Ansible pour l' Infrastructure as Code (IaC)
- Kubernetes pour l' orchestration et la haute disponibilité
- Jenkins, SonarQube, Trivy, ArgoCD pour la chaîne CI/CD
- Nginx, Cert-Manager, Crowdsec pour la sécurisation des flux et la détection d' intrusions
- Prometheus & Grafana pour le monitoring temps réel

Les choix technologiques ont été guidés par des critères de **robustesse**, **compatibilité** avec Kubernetes, **communauté active**, et **intégration fluide** entre les outils.

Limites rencontrées

Malgré la réussite du projet, certaines limites ont été identifiées :

- La gestion des ressources Azure avec Terraform aurait pu être optimisée en utilisant un backend distant (remote state)
 - L'exposition des applications reste centralisée via un seul Ingress Controller, ce qui crée un point de défaillance unique
 - La gestion fine des rôles RBAC Kubernetes pourrait être améliorée
-

Perspectives d' évolution

Plusieurs axes peuvent être explorés pour enrichir ce projet :

- Mise en place d' un système de log centralisé (ELK ou Loki)
 - Déploiement multi-cluster pour la haute disponibilité géographique
 - Automatisation des tests fonctionnels dans la pipeline CI/CD
 - Renforcement de la sécurité via des politiques PodSecurityAdmission
-

Conclusion personnelle

Ce projet fut l' opportunité de consolider mes compétences techniques tout en apprenant à travailler selon une approche DevOps complète. Il m' a permis de comprendre les **enjeux réels de production**, d' intégrer les **bonnes**

pratiques de cybersécurité, et de concevoir une infrastructure **scalable**, **résiliente et observable**, prête pour le monde professionnel.

Bibliographie / Références Techniques

Voici les principales ressources techniques consultées et utilisées tout au long du projet :

Documentation Officielle

- Kubernetes Documentation : <https://kubernetes.io/docs/>
- Helm Charts Repository : <https://artifacthub.io/>
- Terraform Documentation :
<https://developer.hashicorp.com/terraform/docs>
- Ansible Documentation : <https://docs.ansible.com/>
- Jenkins Documentation : <https://www.jenkins.io/doc/>
- ArgoCD Docs : <https://argo-cd.readthedocs.io/>
- SonarQube Docs : <https://docs.sonarsource.com/>
- Trivy Security Scanner : <https://aquasecurity.github.io/trivy/>
- Cert-Manager Docs : <https://cert-manager.io/docs/>
- CrowdSec Docs : <https://docs.crowdsec.net/>
- Prometheus Docs : <https://prometheus.io/docs/>
- Grafana Docs : <https://grafana.com/docs/>

Référentiels GitHub & Helm Charts

- [bitnami/chart](https://github.com/bitnami/chart) - Charts Helm pour WordPress, PostgreSQL, etc.
- [cloudnativelabs/kube-router](https://github.com/cloudnativelabs/kube-router)
- [prometheus-community/helm-charts](https://github.com/prometheus-community/helm-charts)

- [crowdsecurity/helm-charts](#)

Articles et Blogs de Référence

- DevOps Stack Setup on Azure - Medium, ITNEXT, TowardsDev
- Kubernetes Best Practices - CNCF blog
- Infrastructure as Code Examples - Terraform Registry