## UNIVERSITÉ JOSEPH KI-ZERBO

**(UJKZ)**

## \*\*\*\*\*\*

Année 2024-2025

**TP : Déploiement d’une Infrastructure Multi-Machines avec Vagrant et Bash «**[https://github.com/Mensan2024/V\_projet](https://github.com/Mensan2024/V_projet )**»**

Membre du groupe :

1. ZOUNDI JEAN PHILIPPE

2.OUEDRAOGO R. Daouda

2. BOGNINI Benjamine

4. CODJIA M. Moréno

Sous la supervision de :

**Dr Flavien**

**INSTITUT BURKINABÈ DES ARTS ET MÉTIERS (IBAM)**



### **PLAN**

### Introduction

### Contexte du projet

### Objectifs généraux et pédagogiques

### Technologies utilisées (Vagrant, VirtualBox, Bash, Apache, MySQL, Nginx, Prometheus, Grafana)

### Description et architecture du projet

### Présentation de l’architecture globale

### Architecture réseau et schéma de topologie

### Rôles des machines

### Organisation et structure des répertoires du projet

### Mise en place de l’environnement

### Installation des prérequis : Vagrant, VirtualBox, Git

### Création du fichier **Vagrantfile** et explication de sa structure

### Écriture et organisation des scripts de provisioning Bash pour chaque rôle

### Déploiement de l’infrastructure

### Lancement de l’infrastructure avec **vagrant up**

### Vérification de l’état des machines (**vagrant status**)

### Vérification de la connectivité réseau (**ping**, **ssh**, **host**)

### Configuration des services

### Serveur web (web1 et web2)

### Load Balancer

### Installation et configuration de Nginx

### Mise en œuvre de l’équilibrage de charge (round-robin)

### Infrastructure MySQL

### Installation de MySQL sur db-master et db-slave

### Mise en place de la réplication maître-esclave (configuration, utilisateurs, synchronisation)

### Supervision et Monitoring

### Installation de Prometheus + **node\_exporter**

### Installation de Grafana

### Tests et validation avec le client

### Test sur l’interface de connexion du Client

### Test de répartition des charges

### Test de connexion sur les serveurs web1 et web2

### Test de connexion aux bases de données

### Test du fonctionnement du serveur Monitoring

### Test du remplissage du formulaire sur les serveurs

### Test de résilience lors des arrêts des serveurs

### Test de vérification de la réplication entre le db-master et le db-slave

### Test de consommation des nœuds en pourcentage

### Test de consommation de la mémoire

### Conclusion

### Bilan du projet

### Compétences techniques et transversales acquises

### Limites éventuelles et perspectives d’amélioration

### Lien vers le dépôt Git du projet (code source, README, scripts, etc.)

1. Bibliographie

Table des matières

[**1.** **INTRODUCTION** 5](#_Toc199593659)

[1.1. Contexte du projet 5](#_Toc199593660)

[1.2 Objectifs généraux et pédagogiques 5](#_Toc199593661)

[1.3. Technologies utilisées 5](#_Toc199593662)

[**2.** **Description de l’architecture** 6](#_Toc199593663)

[2.1 Présentation de l’architecture globale 6](#_Toc199593664)

[2.2 Architecture réseau 6](#_Toc199593665)

[2.3 Rôles des machines 7](#_Toc199593666)

[2.4 Organisation et structure des répertoires du projet 7](#_Toc199593667)

[**3.** **Mise en place de l’environnement** 8](#_Toc199593670)

[**3.1.** Installation des prérequis : Vagrant, VirtualBox, Git 8](#_Toc199593671)

[**3.1.1.** Vagrant 8](#_Toc199593672)

[3.2 Création du fichier Vagrantfile et explication de sa structure 10](#_Toc199593674)

[3.3 Écriture et organisation des scripts de provisioning Bash pour chaque rôle 11](#_Toc199593675)

[**4.** **Déploiement de l’infrastructure** 12](#_Toc199593676)

[4.1 Lancement de l’infrastructure avec vagrant up 12](#_Toc199593677)

[4.2 Vérification de l’état des machines (vagrant status) 13](#_Toc199593678)

[4.3 Vérification de la connectivité réseau (ping, ssh, host) 13](#_Toc199593679)

[5. Configuration des services 14](#_Toc199593681)

[5.1. **Serveurs Web (web1, web2)** 14](#_Toc199593682)

[**5.1.1 Installation d’Apache** 14](#_Toc199593683)

[**5.2.1 Déploiement d’une page d’une page de test personnalisée** 14](#_Toc199593684)

[5.2 Load Balancer 14](#_Toc199593685)

[5.3](#_Toc199593688) [Déploiement de la base de données db-master et db-slave dans le Vagrantfile 16](#_Toc199593689)

[5.4 Supervision et Monitoring 24](#_Toc199593699)

[**5.5** **Tests et validation avec le client** 30](#_Toc199593706)

[**6.** **Conclusion** 38](#_Toc199593719)

[**7.** **Bibliographie** 40](#_Toc199593723)

# 

# TABLE DES FIGURES

[Figure 1 : l’architecture globale du projet 6](#_Toc199583728)

[Figure 2 : Tableau montrant les correspondances des adresses IP par machine 7](#_Toc199583729)

[Figure 3 : Visualisation des dossiers créer sur à travers notre PowerShell 8](#_Toc199583730)

[Figure 4 : Version de Vagrant utilisé 9](#_Toc199583731)

[Figure 5 : Version du VirtualBox utilisé 10](#_Toc199583732)

[Figure 6: version de git utilisée 10](#_Toc199583733)

[Figure 7 : répertoire scripts contenant nos dossiers 12](#_Toc199583734)

[Figure 8 : Les scripts sont remplis avec nos contenues 12](#_Toc199583735)

[Figure 9 : Tous les machine sont démarrées sans problème 13](#_Toc199583736)

[Figure 10 : Ping avec chaque IP pour vérifier les connexions 13](#_Toc199583737)

[Figure 11 : affichage des contenus de nos pages web 14](#_Toc199583738)

[*Figure 12 : Lb en fonctionnement* 14](#_Toc199583739)

[Figure 13 : Interface graphique du serveur Web 2 15](#_Toc199583740)

[Figure 14 : Interface graphique du serveur Web 1 15](#_Toc199583741)

[Figure 15 : projetdB 17](#_Toc199583742)

[Figure 16 : Affichage de la table actuelle 17](#_Toc199583743)

[Figure 17 : Insertion d’un nouvel étudiant 18](#_Toc199583744)

[Figure 18: capture mysql installé 18](#_Toc199583745)

[Figure 19 : Ecoute sur toutes les interfaces réseaux 19](#_Toc199583746)

[Figure 20 : Capture fichier de config mysql sur db-slave 19](#_Toc199583747)

[Figure 21 : Vérification de la réplication 20](#_Toc199583748)

[Figure 22 : La réplication fonctionne 21](#_Toc199583749)

[Figure 23 : fichier sql crée 22](#_Toc199583750)

[Figure 24 : Insertion d’un nouvel étudiant 23](#_Toc199583751)

[Figure 25 : test de réplication 23](#_Toc199583752)

[Figure 26 : fichier de configuration prometheus.yml 24](#_Toc199583753)

[Figure 27 : interface de prometheus 26](#_Toc199583754)

[Figure 28 : Interface gafrana 27](#_Toc199583755)

[Figure 29 : création de source de données 28](#_Toc199583756)

[Figure 30 : la répartition de charge entre web1 et web2 30](#_Toc199583757)

[Figure 31 : Contenu du server web1 31](#_Toc199583758)

[Figure 32 : Contenu du server web2 31](#_Toc199583759)

[Figure 33 : Test de connexion sur db-master et db-slave 31](#_Toc199583760)

[Figure 34 : Test sur le serveur 32](#_Toc199583761)

[Figure 35 : code html du site 32](#_Toc199583762)

[Figure 36 : Test de saisie 32](#_Toc199583763)

[Figure 37 : Réplication effective 35](#_Toc199583764)

[*Figure 38 : Réponse du serveur web1*](#_Toc199583765)

[Figure 39 : consommation en pourcentage des nœuds du load balancer 36](#_Toc199583766)

[Figure 40 : consommation du CPU 37](#_Toc199583767)

[*Figure 41 : Consommation de la mémoire pour le db master* 37](#_Toc199583768)

# **INTRODUCTION**

## Contexte du projet

La transformation numérique a profondément modifié les pratiques informatiques contemporaines. Face à la montée en puissance des services numériques, les entreprises et les professionnels de l’informatique doivent relever des défis croissants en matière de flexibilité, de performance, de sécurité et de résilience. Dans ce contexte, la virtualisation s’impose comme une solution stratégique : elle permet d’optimiser l’usage des ressources matérielles, de réduire les coûts d’infrastructure et de déployer rapidement des environnements complexes.

C’est dans cette logique que s’inscrit notre projet. Réalisé dans le cadre du cours de Virtualisation et Cloud Computing, il consiste à concevoir et à mettre en œuvre une infrastructure virtualisée multi-machines, simulant un environnement de production réaliste, modulaire et évolutif.

## 1.2 Objectifs généraux et pédagogiques

Ce travail pratique poursuit plusieurs objectifs pédagogiques clairs :

* Comprendre les principes de la virtualisation et de l’orchestration d’environnements complexes à l’aide d’outils open-source ;
* Maîtriser l’écriture de scripts d’automatisation en Bash, afin de provisionner rapidement et efficacement les machines ;
* Mettre en œuvre des services critiques dans un environnement virtualisé : équilibrage de charge, base de données distribuée, supervision et monitoring ;
* Tester la résilience et la tolérance aux pannes dans un environnement isolé mais réaliste, afin de reproduire les contraintes d’un système en production.

## 1.3. Technologies utilisées

Notre projet repose exclusivement sur des solutions open-source, garantissant une approche accessible, flexible et pédagogique.

Afin de structurer notre démarche, ce rapport est organisé comme suit : nous commencerons par une description détaillée de l’architecture de l’infrastructure. Ensuite, nous expliquerons la mise en place de l’environnement, le déploiement automatique des services, puis la configuration de chaque composant fonctionnel. Enfin, nous terminerons par une série de tests de validation sur le client avant de conclure sur les compétences acquises et les perspectives d’amélioration.

# **Description de l’architecture**

## 2.1 Présentation de l’architecture globale

L’objectif principal de notre projet est de simuler un environnement de production complet, cohérent et interconnecté, capable de démontrer les principes de base de la virtualisation, de l’automatisation et de la résilience des services. Pour cela, nous avons conçu une architecture multi-machines qui reflète les standards actuels des infrastructures déployées en entreprise.

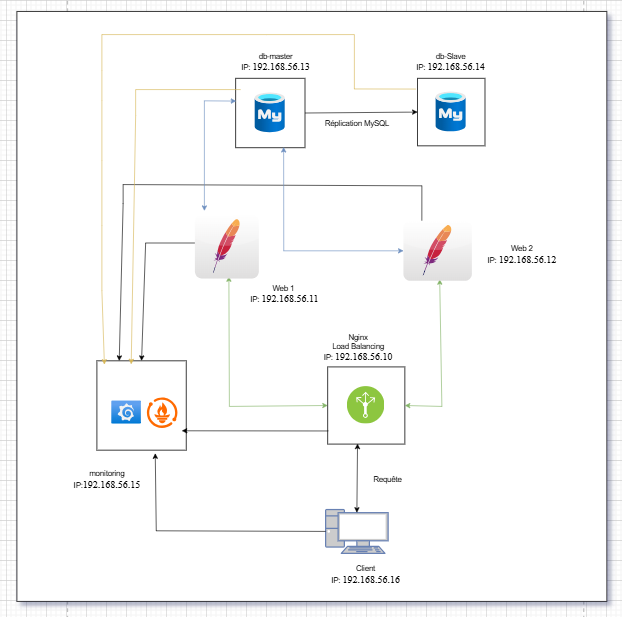


Figure 1 : l’architecture globale du projet

## 2.2 Architecture réseau

L’ensemble des machines virtuelles sont reliées via un réseau privé en mode host-only configuré sur le sous-réseau **192.168.56.0/24**. Chaque nœud dispose d’une adresse IP statique, ce qui facilite les connexions inter-machines et l’automatisation des configurations.

Voici un aperçu de la table de correspondance des adresses IP par machine :

|  |  |  |
| --- | --- | --- |
| **Nom de la machine** | **Rôle** | **Adresse IP** |
| **lb** | Load Balancer (Nginx) | **192.168.56.10** |
| **web1** | Serveur Web (Apache) | **192.168.56.11** |
| **web2** | Serveur Web (Apache) | **192.168.56.12** |
| **db-master** | Base de données MySQL Maître | **192.168.56.13** |
| **db-slave** | Base de données MySQL Esclave | **192.168.56.14** |
| **monitoring** | Supervision (Prometheus + Grafana) | **192.168.56.15** |
| **client** | Poste de test et validation | **192.168.56.16** |

Figure 2 : Tableau montrant les correspondances des adresses IP par machine

## 2.3 Rôles des machines

* **lb** (Load Balancer) : Ce nœud utilise Nginx comme répartiteur de charge HTTP. Il distribue intelligemment les requêtes entrantes vers **web1** et **web2**, garantissant un partage équitable de la charge et une haute disponibilité.
* **web1** et **web2** (Serveurs Web) : Ils hébergent l’application web via le serveur Apache2. Ces serveurs sont identiques et leur contenu est synchronisé via les scripts de provisioning, assurant la redondance.
* **db-master** (Serveur MySQL maître) : Il centralise les écritures de la base de données. Sa configuration permet la réplication des données vers l’esclave.
* **db-slave** (Serveur MySQL esclave) : Ce serveur reçoit les mises à jour de la base depuis le maître. Il peut être utilisé pour des lectures, des sauvegardes ou comme bascule en cas de panne du maître.
* **Monitoring** (Serveur de supervision) : Il héberge Prometheus pour la collecte des métriques système, et Grafana pour leur visualisation graphique à travers des tableaux de bord dynamiques.
* **Client** (Machine de test) : Ce poste permet de simuler l’expérience utilisateur finale, de tester les services (ping, requêtes web, accès aux bases de données), et d’observer le comportement du système en conditions réelles.

## 2.4 Organisation et structure des répertoires du projet

### Pour assurer une bonne organisation de notre infrastructure virtuelle, nous avons adopté une structure de répertoires claire, logique et modulaire. Chaque élément du projet est rangé dans un emplacement précis en fonction de son rôle :

* **.Vagrant :** est essentiel pour que Vagrant puisse suivre et gérer l'état des environnements virtuels créés dans un projet donné. Sans ce fichier, Vagrant ne saurait pas quelle machine est associée au projet, ni son état.
* **configs/** : contient les fichiers de configuration personnalisés pour les services comme Prometheus ou Grafana.
* **docs/** : documentation du projet, avec les captures d’écran des étapes importantes.
* **logs/** : fichiers journaux pour suivre les éventuelles erreurs lors du provisioning.
* **scripts/** : contient tous les scripts de provisioning Bash, un par rôle (web, db, lb...).
* **sql/** : fichiers SQL utilisés pour initialiser la base de données ou tester la réplication
* **gitignore :** Il permet d'exclure des fichiers ou dossiers spécifiques des suivis de Git (évitant qu'ils ne soient commités/pushés).
* **Vagrantfile** : fichier de configuration principal qui décrit l’ensemble de l’infrastructure.

### Étape 1 :

Dans PowerShell, on n’a tapé ceci : « cd $HOME\Documents » pour aller dans notre dossier "Documents"

Étape 2 : Créer notre dossier de projet et sa structure

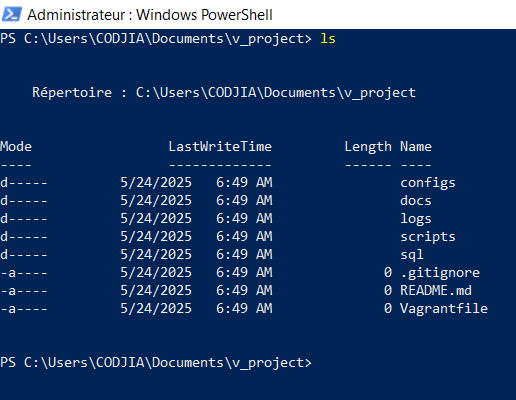


Figure 3 : Visualisation des dossiers créer sur à travers notre PowerShell

# **Mise en place de l’environnement**

# Installation des prérequis : Vagrant, VirtualBox, Git

# Vagrant

Vagrant est un outil d’orchestration qui permet de définir et déployer facilement des environnements virtualisés reproductibles via un fichier de configuration (**Vagrantfile**).

Étapes d’installation :

1. Se rendre sur le site officiel : <https://www.vagrantup.com/downloads>
2. Télécharger la version correspondant à votre système.
3. Installer Vagrant comme un logiciel classique (Next > Next > Finish).
4. Vérifier l’installation via un terminal ou PowerShell (en mode administrateur) à travers cette commande : vagrant –version.

Pour ce projet, nous avons installé vagrant 2.4.5

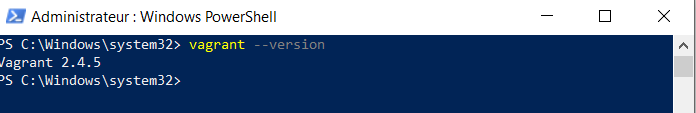


Figure 4 : Version de Vagrant utilisé

\****VirtualBox***

VirtualBox est un hyperviseur permettant d’exécuter des machines virtuelles sur un système hôte. Il est nécessaire pour que Vagrant puisse créer et gérer des VM localement.

Étapes d’installation :

1. Aller sur le site officiel : <https://www.virtualbox.org/>
2. Télécharger la version adaptée à votre système d’exploitation (Windows, macOS, Linux).
3. Suivre l’assistant d’installation en acceptant les options par défaut.
4. Vérifier que l’installation est réussie en lançant VirtualBox.

Pour notre projet, nous avons utilisé VirtualBox version 6.1.50

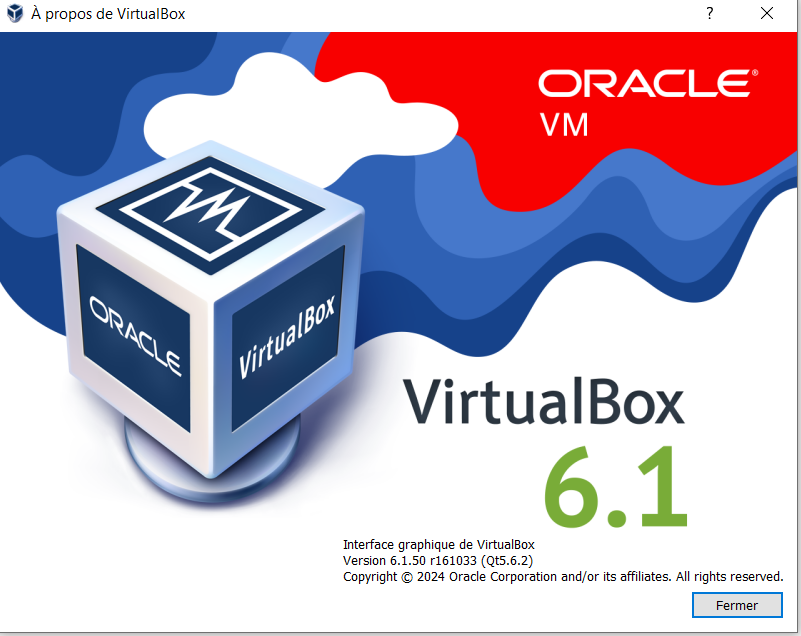


Figure 5 : Version du VirtualBox utilisé

**\*Git**

Système de contrôle de version, utilisé pour versionner notre code et collaborer efficacement.

### Sous Windows :

1. Rendez-vous sur le site officiel :  
   <https://git-scm.com/download/win>  
   Le téléchargement démarre automatiquement.
2. Une fois le fichier **.exe** téléchargé, lancez-le.
3. Lors de l’installation, vous pouvez laisser la configuration par défaut (Next → Next...).  
   Vous pouvez aussi cocher l’option “Git Bash Here” pour faciliter l’usage via clic droit.
4. Une fois installé, ouvrez Git Bash (une interface de terminal) pour l'utiliser.
5. Pour vérifier l’installation :



Pour ce projet, nous avons utilisé git version 2.49.0

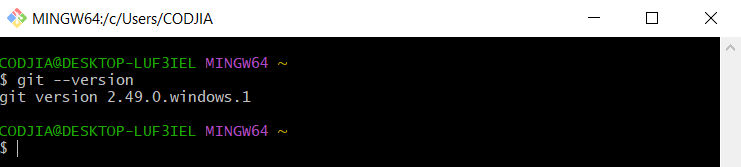


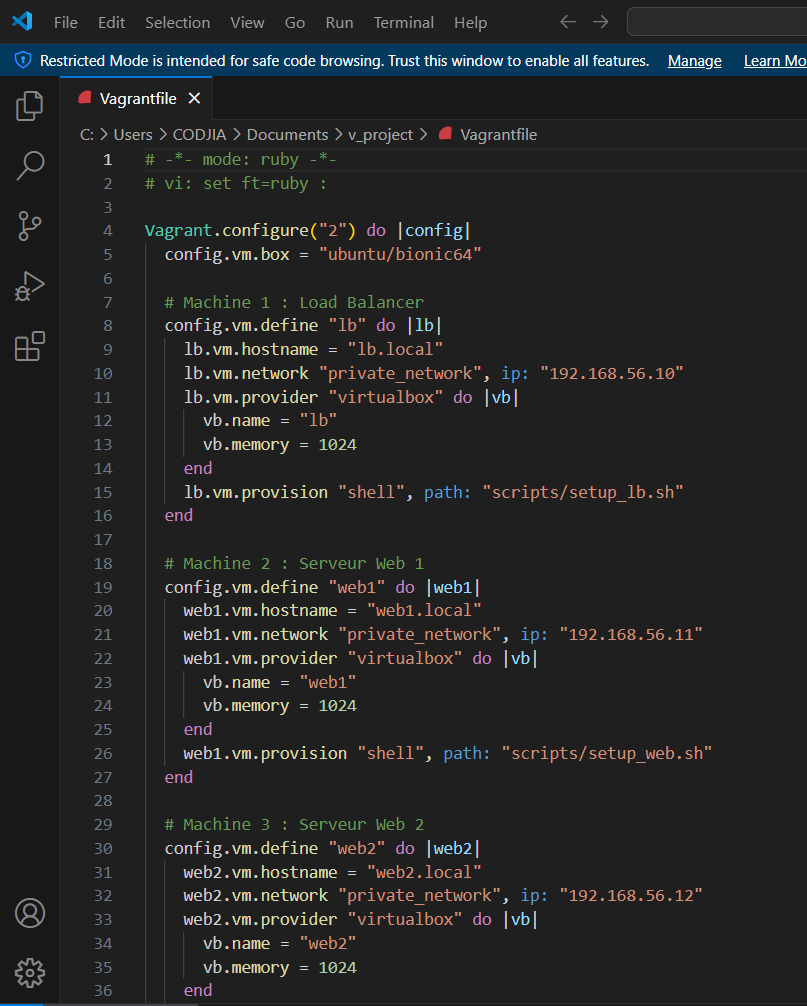
Figure 6: version de git utilisée

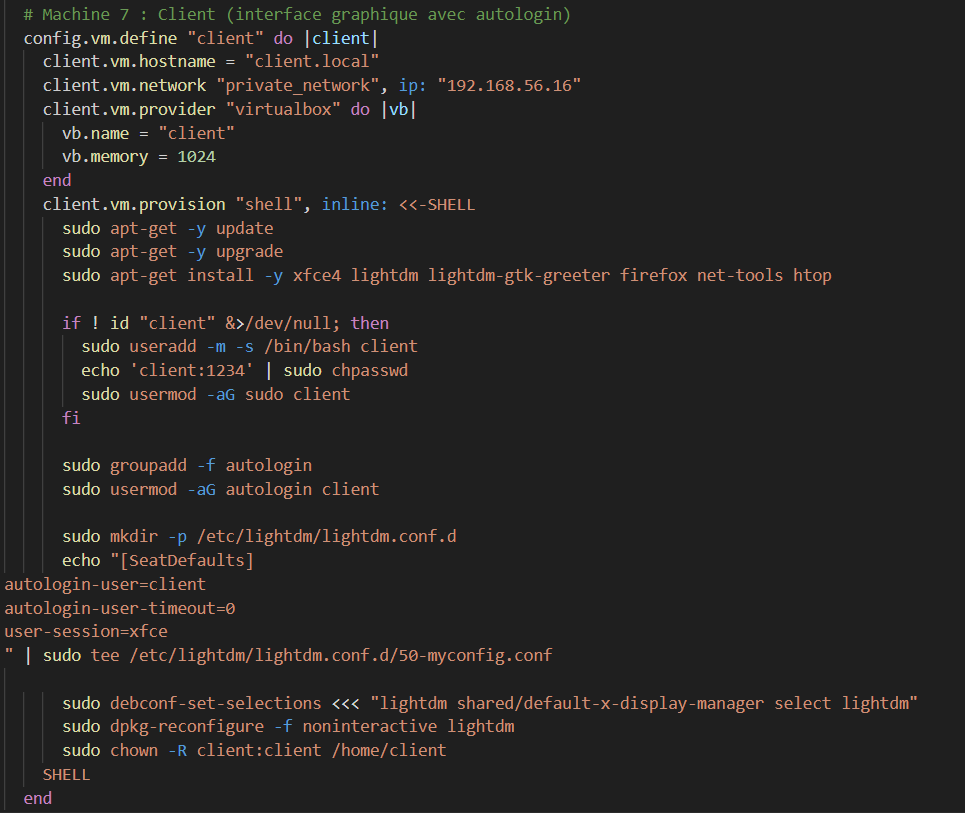
## 3.2 Création du fichier ****Vagrantfile**** et explication de sa structure

Le fichier Vagrantfile est un fichier de configuration écrit en **Ruby**, utilisé par **Vagrant** pour décrire l’environnement de machines virtuelles à créer. Il est le cœur de notre infrastructure car il décrit la configuration des 7 machines virtuelles, leurs adresses IP, leurs rôles respectifs et les scripts de provisioning à exécuter.

Voir l’aperçu de notre fichier Vagrantfile avec quelques descriptions :







## 3.3 Écriture et organisation des scripts de provisioning Bash pour chaque rôle

Notre dossier scripts doit contenir les fichiers suivants :

* scripts/
* ├── setup\_db\_slave.sh
* ├── setup\_db\_master.sh
* ├── setup\_web.sh
* ├── setup\_monitoring.sh
* ├── setup\_lb.sh

Pour le faire, nous allons d’abord créer tous les fichiers .sh par cette commande dans notre PowerShell :

foreach ($file in "setup\_lb.sh","setup\_web.sh","setup\_db\_master.sh","setup\_db\_slave.sh","setup\_monitoring.sh") { New-Item -ItemType File -Name $file -Force}

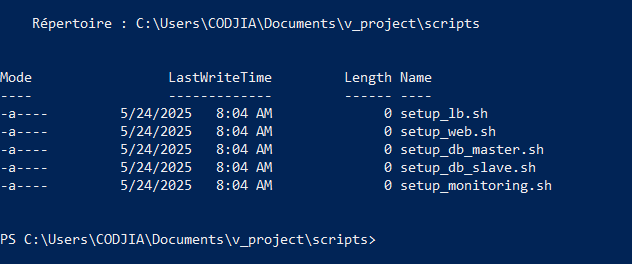


Figure 7 : répertoire scripts contenant nos dossiers

Maintenant, nous allons injecter directement le contenu de chaque script dans les fichiers .sh via PowerShell.

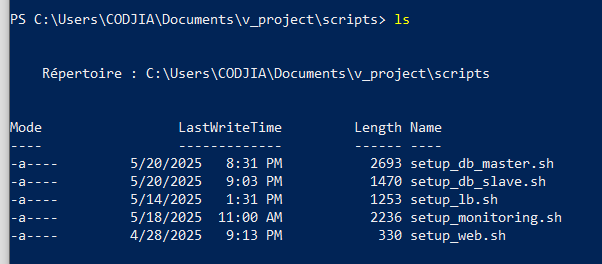


Figure  : Les scripts sont remplis avec nos contenues

# **Déploiement de l’infrastructure**

## Lancement de l’infrastructure avec **vagrant up**

Nous avons ouvert notre PowerShell et nous sommes rendus dans notre dossier **« PS C:\Users\CODJIA\Documents\v\_project> » puis exécuter la commande « vagrant up » cela va Crée et démarrer** chaque machine virtuelle définie dans notre Vagrantfile et **Configura automatiquement** chaque VM avec son script associé (installation, configuration réseau, services, etc.).

## 4.2 Vérification de l’état des machines (**vagrant status**)

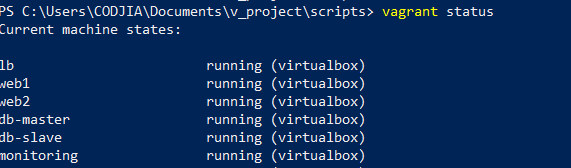


Figure 9 : Tous les machine sont démarrées sans problème

## 4.3 Vérification de la connectivité réseau (**ping**, **ssh**, **host**)

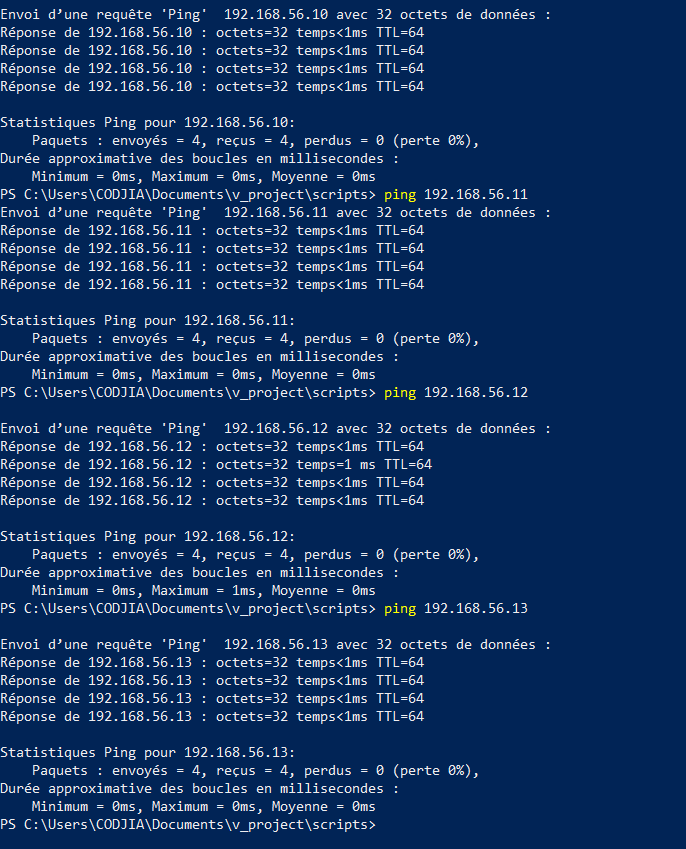


Figure 10 : Ping avec chaque IP pour vérifier les connexions

## 

## ****Les vérifications de la connexion par ssh****

vagrant@web1:~$

vagrant@web2:~$

vagrant@lb:~$

vagrant@db-master:~$

vagrant@db-slave:~$

# Configuration des services

# **Serveurs Web (web1, web2)**

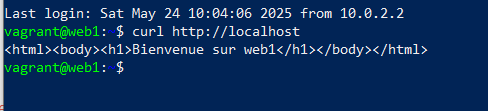
Les serveurs web constituent la couche applicative de l’infrastructure. Ils sont directement sollicités par le Load Balancer **lb** et doivent être fonctionnels en permanence. Grâce à la duplication de service sur **web1** et **web2**, le système garantit une meilleure résilience et une répartition de la charge.

### **5.1.1 Installation d’Apache**

Chaque machine **web1** et **web2** exécute le script de provisionnement suivant (**setup\_web.sh**) déjà crée.

### **5.2.1 Déploiement d’une page d’une page de test personnalisée**

On va ensuite ajouter une page HTML personnalisée. Ce qui revient à écrire dans le fichier setup\_web.sh :



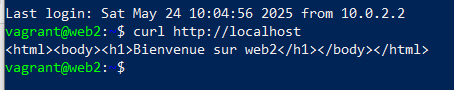


Figure 11 : affichage des contenus de nos pages web

## Load Balancer

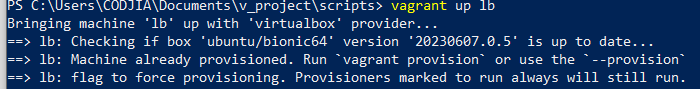
Le serveur **lb** (Load Balancer) utilise Nginx, un serveur HTTP performant capable de faire office de répartiteur de charge (reverse proxy).

### **5.2.1 Installation et configuration de Nginx**

Le script de provisionnement **setup\_lb.sh** déjà créer permet d’installer et configurer automatiquement Nginx.

Ensuite lançons la machine lb avec **vagrant up lb pour vérification.**

Une fois la machine **lb** lancée avec **vagrant up lb on n’a :**

*Figure 12 : Lb en fonctionnement*

**5.2.2 Mise en œuvre et vérification de l’équilibrage de charge**

Vérification du bon fonctionnement de notre lb en se connectant à travaers son IP : 192.168.56.10.

Automatiquement on n’est connecté sur le web2 :

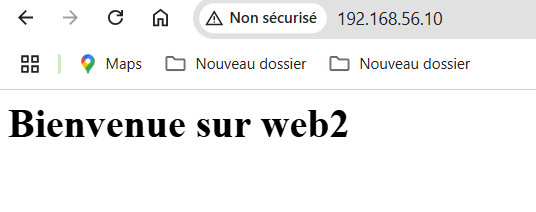


Figure 13 : Interface graphique du serveur Web 2

En actualisant la page web on passe au web1 :



Figure 14 : Interface graphique du serveur Web 1

## Test de résilience

Nous avons testé la résilience du système en éteignant web1 et en vérifiant que l’application reste disponible via le load balancer car lorsque le serveur web 1 est éteint le serveur 2 continu de travailler avec le load balancer.

## Infrastructure MySQL

### Déploiement de la base de données db-master et db-slave dans le Vagrantfile



### **5.3.1 Installation de mysql sur db-master et db-slave**

Pendant le déploiement des serveurs db-master et db-slave mysql serveur est installé avec les variables ci-dessous qui se trouve dans le fichier « mysql\_dump\_transfer.sh » du dossier dossier sql



Le mot de passe du super administrateur mysql avec tous les privilèges

Un utilisateur de réplication et son mot de passe associé qui sera utilisé par le slave pour se connecter au master et recevoir les mises à jour.

L’adresse **0.0.0.0** signifie que MySQL acceptera les connexions depuis toutes les interfaces réseau, y compris les autres machines. Utile pour permettre au slave d’accéder au master à distance.

Le nom de la base de données qui sera créée : projetdb

## Création de la base de données projetdb sur db-master

Apres l’installation de mysql avec tout le paramétrage nécessaire nous avons procédé à la création de la base de données projetdb sur dB-master avec différentes tables.

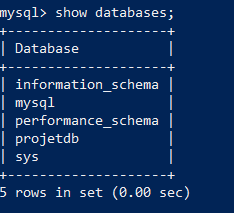


Figure 15 : projetdB

Nous avons ensuite copié le contenu de notre fichier « script\_sql\_projet.sql » présent dans notre dossier sql.

Ce script crée une base de données relationnelle simple et cohérente :

* 1 Professeur enseigne plusieurs cours.
* 1 Étudiant peut suivre plusieurs cours.
* **ETUDIANT\_COURS** fait le lien entre eux avec des infos utiles (paiement, note…).

### Étape 1 : dans le répertoire où se trouve notre fichier nous avons exécuté notre commande :

**mysql -u root -p projetdb < /vagrant/scripts/script\_sql\_projet.sql**

On n’est invité à entrer le mot de passe : @dmin25

Cela va créer toutes les tables et insérer toutes les données en une seule fois.

Maintenant, il faut se connecter à mysql du db-master pour vérifier si les données ont été enregistrer.

1. **Test pour s’assurer que l’insertion d’étudiants dans la table étudiant fonctionne**

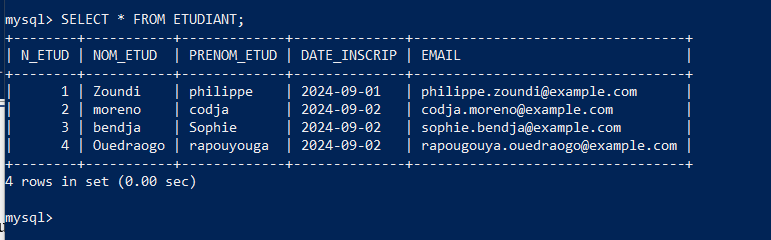


Figure 16 : Affichage de la table actuelle

1. **Insertion d’un nouvel étudiant**

Pour ajouter cet étudiant à ta table ETUDIANT, on va exécuter cette requête SQL dans MySQL :

INSERT INTO ETUDIANT (N\_ETUD, NOM\_ETUD, PRENOM\_ETUD, DATE\_INSCRIP, EMAIL)

VALUES (5, 'KONE', 'JACOB', '2025-05-20', 'jacob.kone@example.com');

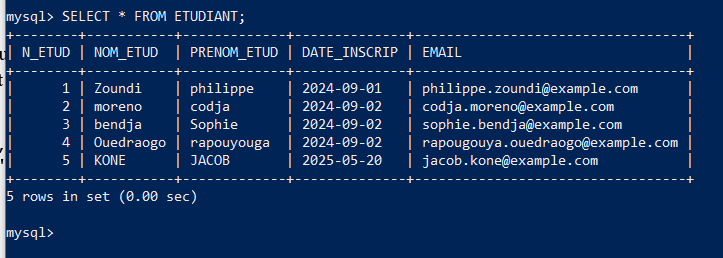


Figure 17 : Insertion d’un nouvel étudiant

### **5.3.2 Mise en place de la réplication maître-esclave (Configuration, utilisateurs, synchronisation)**

Le chemin vers le fichier de configuration de mysql.

/etc/mysql/mysql.conf.d/mysqld.cnf

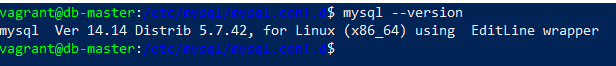


Figure 18: capture mysql installé

Configurer le fichier par cette commande

vagrant@db-master:~$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

* + - 1. Pour la réplication MySQL, voici les prochaines lignes à ajouter dans ce même fichier

**[mysqld]**

**server-id = 1**

**log\_bin = /var/log/mysql/mysql-bin.log**

**binlog\_do\_db = nom\_de\_ta\_base**

**bind-address = 0.0.0.0**

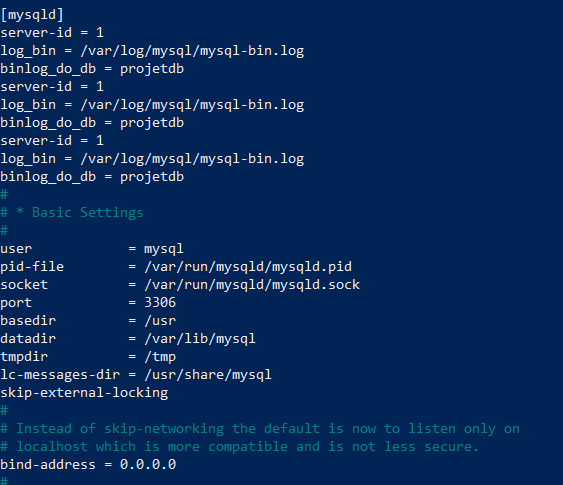


Figure 19 : Ecoute sur toutes les interfaces réseaux

Cette ligne permet à MySQL d'écouter sur toutes les interfaces réseau, ce qui est nécessaire pour que les autres machines (comme le **db-slave**) puissent se connecter pour la réplication.

**On fera les mêmes configurations sur le db-slave**

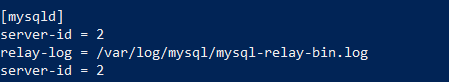


Figure 20 : Capture fichier de config mysql sur db-slave

NB :

Le server-id de **db-master**=1

Le server-id de **db-slave**=2

Ce qui montre le maitre et l’esclave

* + - 1. On doit redémarrer les deux serveurs pour valider la réplication et vérifier si la réplication marche :

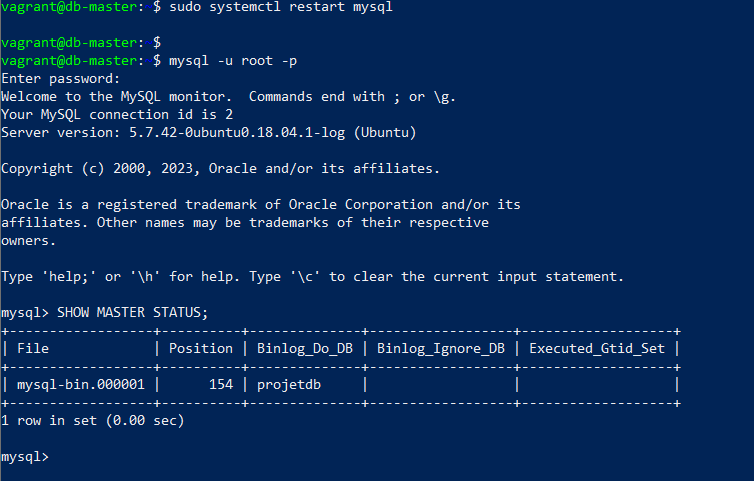


Figure 21 : Vérification de la réplication

Le **SHOW MASTER STATUS;** affiche désormais :

* File : **mysql-bin.000001** → le fichier de log binaire actif
* Position : **154** → la position actuelle dans le fichier binaire
* Binlog\_Do\_DB : **projetdb** → seule cette base est répliquée

J’ai aussi changé le mot de passe de la réplication sur ma base de données.

CHANGE MASTER TO

MASTER\_HOST='192.168.56.13',

MASTER\_USER='user\_replica',

MASTER\_PASSWORD='replica\_pass',

MASTER\_LOG\_FILE='mysql-bin.000001',

MASTER\_LOG\_POS=154;

La réplication fonctionne correctement maintenant. Voici les signes qui confirment cela :

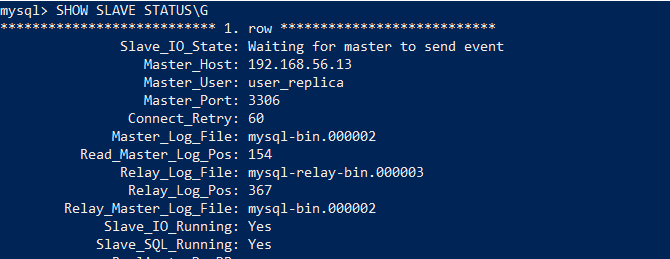


Figure 22 : La réplication fonctionne

Voici les signes qui confirment cela :

### **Éléments clés à vérifier :**

* **Slave\_IO\_Running: Yes** → La connexion au master est active.
* **Slave\_SQL\_Running: Yes** → Le slave applique bien les événements.
* **Slave\_IO\_State: Waiting for master to send event** → Tout est synchronisé, il attend simplement de nouveaux événements.
* **Seconds\_Behind\_Master: 0** → Le slave est à jour avec le master.

Sur le **master**, on va se connecter à MySQL :

mysql -u root -p

Et on insère des données dans la base projetdb, par exemple :

USE projetdb;

CREATE TABLE test\_replication (id INT PRIMARY KEY, nom VARCHAR(50));

INSERT INTO test\_replication VALUES (1, 'Test depuis master');

* + - 1. **Sauvegarde de la configuration de la base de données du db-master et transfert vers le db-salve**
* Exécuter la commande sur le db-master

FLUSH TABLES WITH READ LOCK;

Pour verrouiller temporairement les tables en lecture seule et empêcher de nouvellesécriturespendant qu’on fait une copie (dump) de la base de données.

### **Exécution du script mysql\_dump\_transfer.sh**

### Étape 1 : Lançons la sauvegarde (mysqldump) pendant que le verrou est actif

Dans notre machine **db-master**, sur un autre terminal (ou session SSH), exécutons le script (ou la commande) qui va faire le dump :

**mysqldump -u root -p**"@dmin25" **--databases projetdb --master-data=2 > projetdb\_dump.sql**

* Cette commande crée un fichier SQL avec les données de notre base verrouillée.
* L'option **--master-data=2** ajoutes dans le dump les informations de position de la réplication.

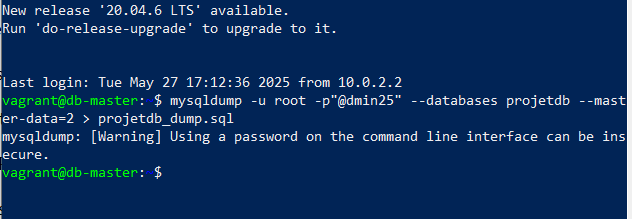
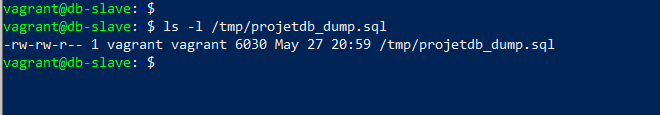


Figure 23 : fichier sql crée

### Étape 2 : Transférer le fichier dump vers le serveur esclave (db-slave)

Toujours dans un terminal (sur le db-master), lançons :

**scp projetdb\_dump.sql** [user@192.168.56.14:/tmp/](mailto:user@192.168.56.14:/tmp/)



Cela va créer un dump SQL propre et à jour qui sera transférer vers l’esclave, et va inclure automatiquement les bonnes infos de réplication.

Dans le répertoire du projet il s’agit de projetdb\_dump.sql

## Test de réplication sur le db-master et le db-slave

Nous allons vérifier que l’insertion de l’étudiant SAWADOGO Arouna dans db-master a été répliquée sur db-slave en tapant cette commande dans db-master pour insérer Arouna :

INSERT INTO ETUDIANT (N\_ETUD, NOM\_ETUD, PRENOM\_ETUD, DATE\_INSCRIP, EMAIL)

VALUES (6, 'SAWADOGO', 'Arouna', CURDATE(), 'arouna.sawadogo@example.com');

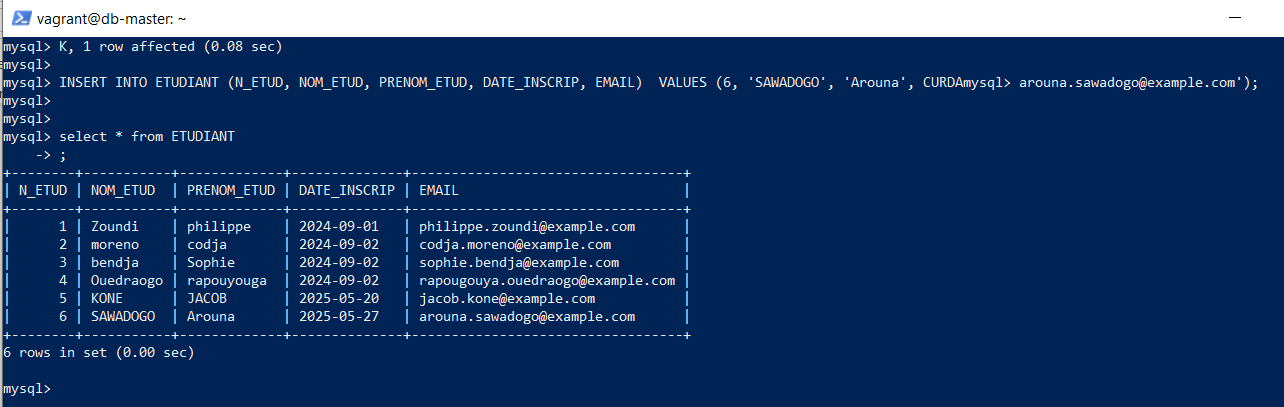


Figure 24 : Insertion d’un nouvel étudiant

**Vérifions notre réplication sur db-slave maintenant**

On va se connecter à db-slave et taper cette commande

SELECT\*FROM **ETUDIANT** WHERE **NOM\_ETUD**='SAWADOGO'AND **PRENOM\_ETUD**='Arouna'**;**

La réplication fonctionne bien car on :

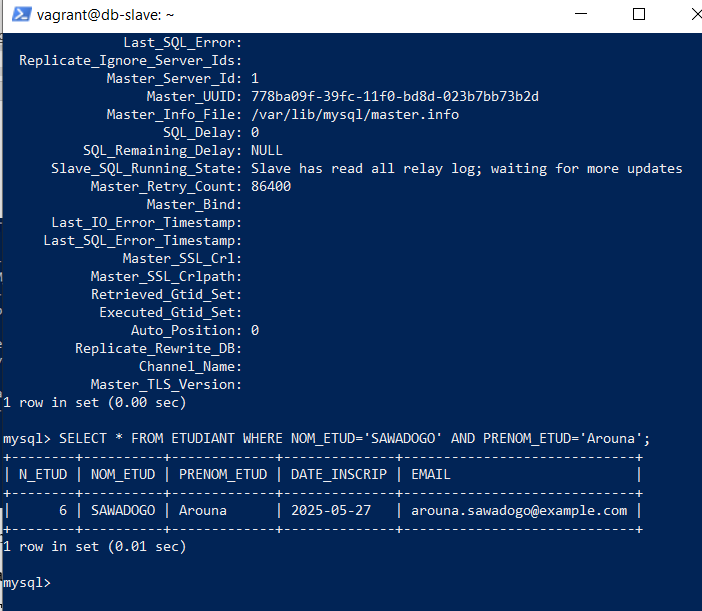


Figure 25 : test de réplication

## Supervision et Monitoring

## Installation de Prometheus et Node Exporter

L’outil de monitoring Prometheus, couplé à Node Exporter, a été déployé sur la machine dédiée au monitoring (IP : **192.168.56.15**).

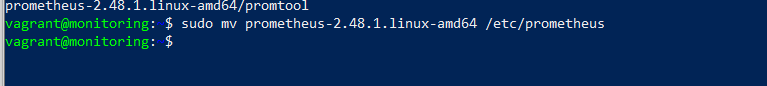
Voici les étapes principales de notre installation :

#### **Télécharger et installer Prometheus à travers cette commande :**

**wget https://github.com/prometheus/prometheus/releases/download/v2.48.1/prometheus-2.48.1.linux-amd64.tar.gz**

**tar -xvzf prometheus-2.48.1.linux-amd64.tar.gz**

**sudo** mv **prometheus-2.48.1.linux-amd64 /etc/prometheus**



#### **5.4.1.2 Créer un fichier de configuration prometheus.yml et l’éditer avec nano :**

**Sudo nano /etc/prometheus/prometheus.yml**

Et y insérer quelque chose comme :

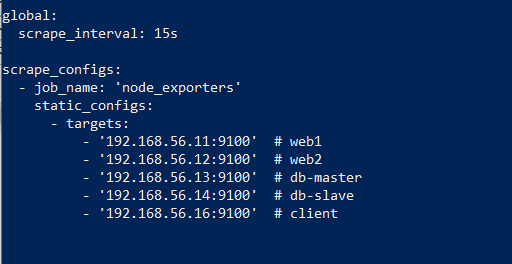


Figure 26 : fichier de configuration **prometheus.yml**

#### **5.4.1.3 Lancer Prometheus**

Installons d’abord *Node Exporter* sur tous les machine avec :

Pour cela, nous avons ajouter un **script de provisioning** pour que Node Exporter soit installé dès le lancement des machines.

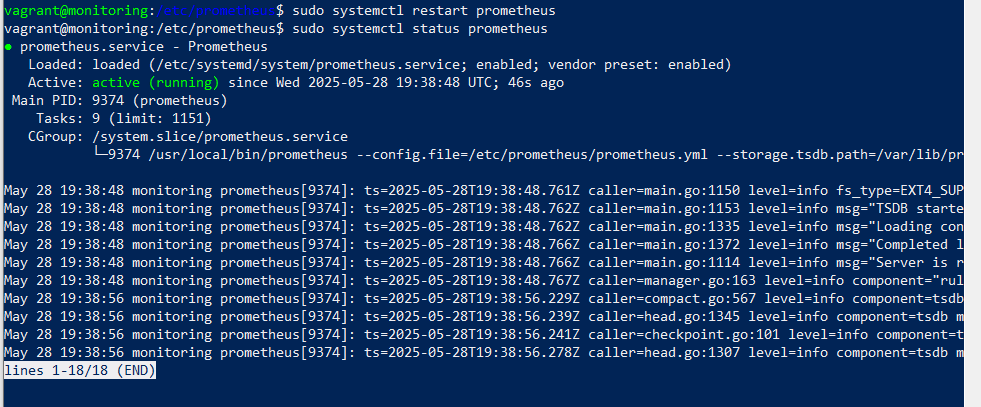
Nous pouvons vérifier la présence du Node Exporter sur chaque machine par cette commande :

ls /usr/local/bin/node\_exporter



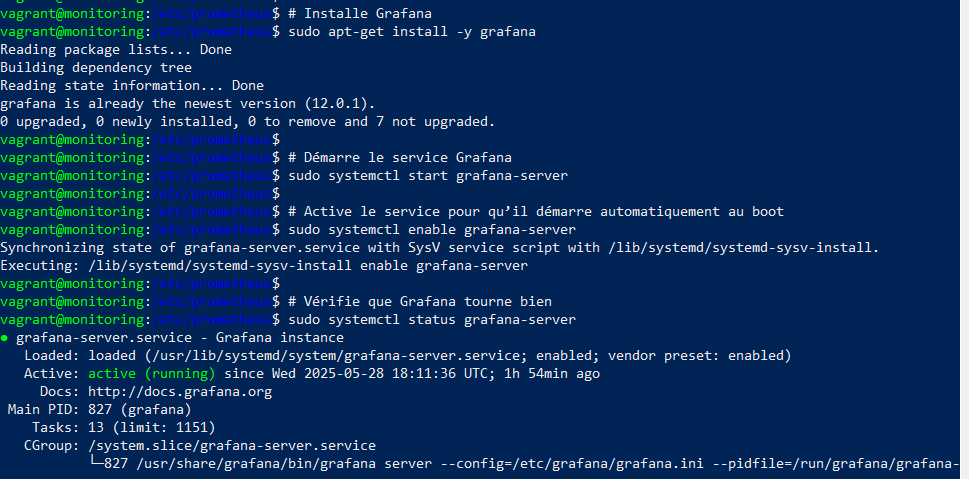
Lançons Prometheus

/usr/local/bin/prometheus --config.file=/etc/prometheus/prometheus.yml



#### 2. **Installation de Grafana**

Sur le monitoring, nous avons tapé « sudo apt-get install -y grafana »



### Pour accéder aux interfaces web :

1. Depuis notre client (l’ordinateur hôte, pas la VM), on ouvre un navigateur.
2. On tape dans la barre d’adresse :

<http://192.168.56.15:9090/targets> Pour Prometheus :

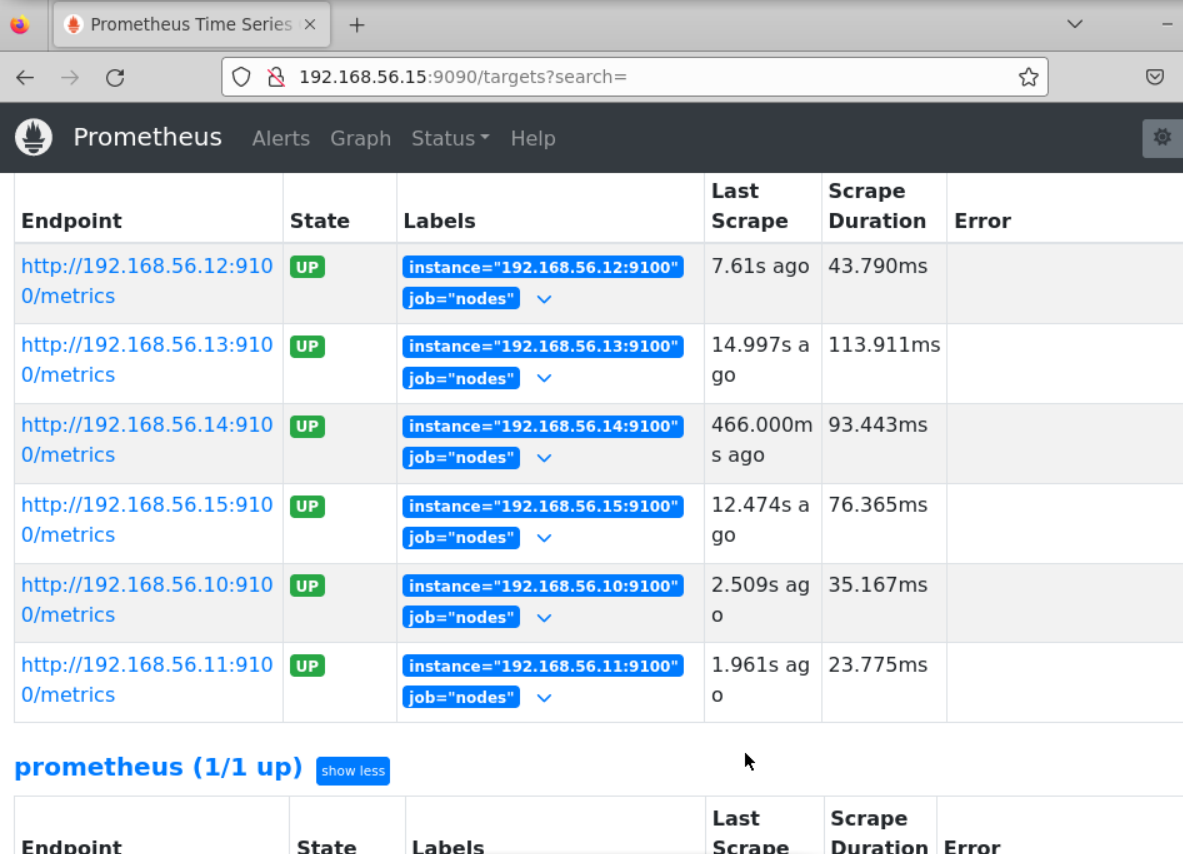


Figure 27 : interface de prometheus

### *Requête sur Prometheus pour avoir des informations sur le metrique consommation de CPU*

***100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle”} [5m])) by (host)***

* Pour Grafana (une fois installé et démarré) :

**http:**//192.168.56.15:3000

### **5.4.1.4 Configuration initiale de Grafana**

* Accède à l’interface web Grafana :  
  **http://<IP\_de\_ton\_serveur>:3000**
* Connecte-toi avec le login par défaut :
  + user: admin
  + password: admin

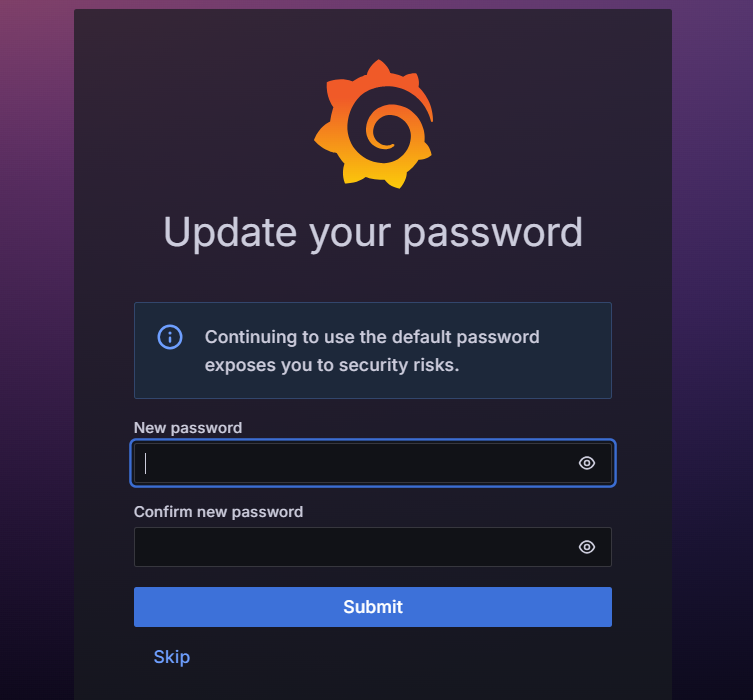


Figure 28 : Interface gafrana

**5.4.1.5 Création d’un dashboard de visualisation des métriques**

Une fois Grafana installé et lancé, nous avons procédé à la création d’un dashboard personnalisé pour surveiller les performances de nos machines virtuelles.

#### Étapes réalisées :

1. Ajout de la source de données Prometheus :
   * Depuis l’interface Grafana, aller dans "Configuration > Data Sources".
   * Ajouter une nouvelle source de données de type Prometheus.
   * Renseigner l’URL de Prometheus : <http://localhost:9090>.

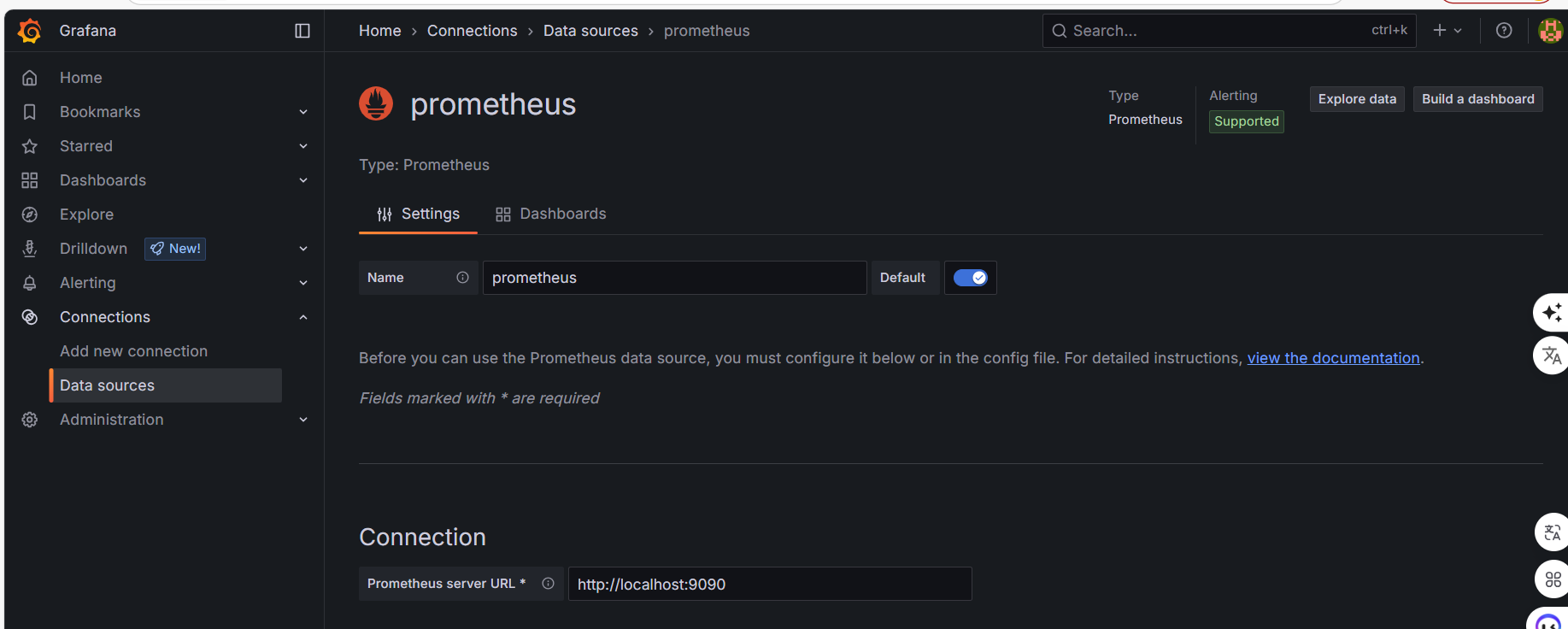


Figure 29 : création de source de données

### **5.4.1.6 Création d’un nouveau dashboard :**

### Créer un dashboard personnalisé

Dans Grafana → menu de gauche → "+" → Dashboard

Clique sur Add new panel

Dans le champ Query, entre une métrique comme :

* + **node\_cpu\_seconds\_total**
  + **node\_memory\_MemAvailable\_bytes**
  + **node\_network\_receive\_bytes\_total**

Choix du type de visualisation : graph, gauge, bar gauge, etc.

Clique sur Apply

Répète pour d’autres panels (CPU, RAM, etc.)

Clique sur Save dashboard

* + Cliquer sur "Create > Dashboard".
  + Ajouter un nouveau panneau (panel).

Ajout de panels avec des métriques pertinentes :

* + CPU usage :  
    Requête PromQL :

rate**(node\_cpu\_seconds\_total{mode="user"}[**1**m])**

* + Utilisation de la RAM :

**(node\_memory\_MemTotal\_bytes - node\_memory\_MemAvailable\_bytes) / node\_memory\_MemTotal\_bytes**

* + Espace disque disponible :

node\_filesystem\_avail\_bytes **/ node\_filesystem\_size\_bytes**

Personnalisation :

* + Choix de types de visualisation (graph, gauge, bar gauge, stat, etc.).
  + Attribution de noms, unités (%, MB, etc.), couleurs et seuils pour chaque panel.

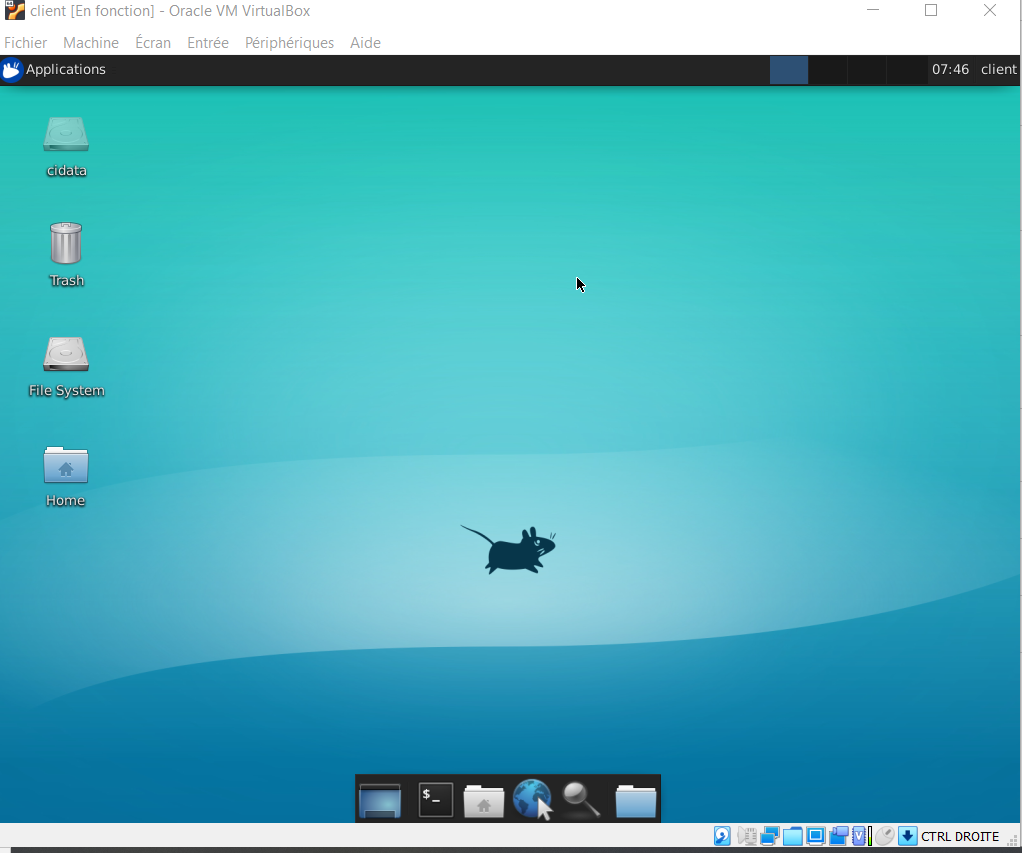
Sauvegarde du dashboard :

* + Nom du dashboard : **Monitoring Infrastructure**.
  + Partage possible par lien ou export JSON.

Grâce à cette mise en place, nous disposons désormais d’un système de supervision visuel et réactif, permettant de détecter en temps réel les variations de performances ou les incidents potentiels au sein de notre infrastructure.

# **Tests et validation avec le client**

## 5.5.1 Test sur Interface de connexion du client



## Test de répartition de charge

Curl http://192.168.56.10 pour se connecter sur le lb et vérifier le fonctionnement de la répartition des charges entre web1 et web2



Figure 30 : la répartition de charge entre web1 et web2

## Test sur les serveurs web

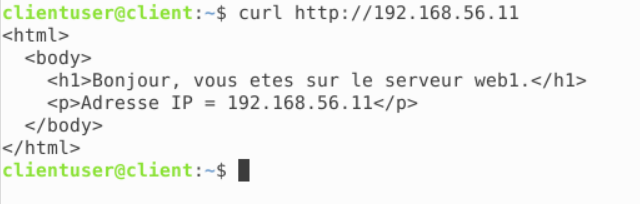


Figure 31 : Contenu du server web1

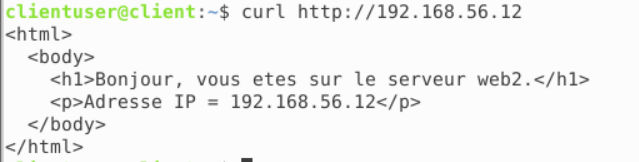


Figure 32 : Contenu du server web2

## 5.5.4 Test de connexion aux bases de données

Test de connexion sur db-master et db-slave

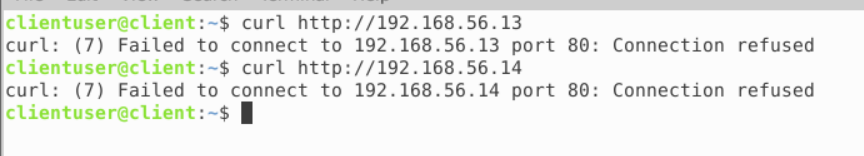


Figure 33 : Test de connexion sur db-master et db-slave

Les deux serveurs de base de données n’écoutent pas sur le port 80 donc la commande

Curl <http://192.168.56.13/14> est refusée

Nous allons effectuer le test sur le port d’écoute des deux bases de données à savoir le port 3306 avec la commande :

Nc –zv 192.168.56.13 3306

Nc –zv 192.168.56.14 3306

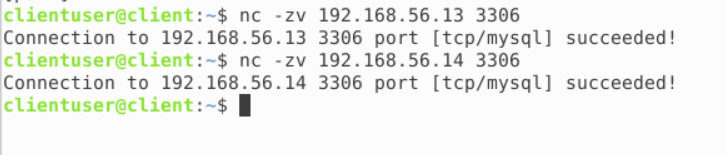


Figure 34 : Test sur le serveur

## 5.5.5 Test sur le serveur de monitoring

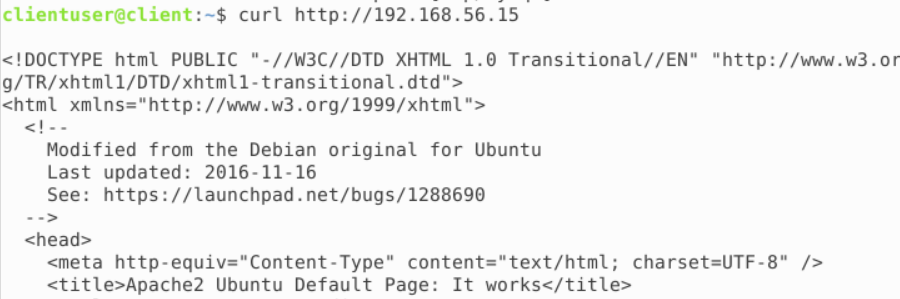


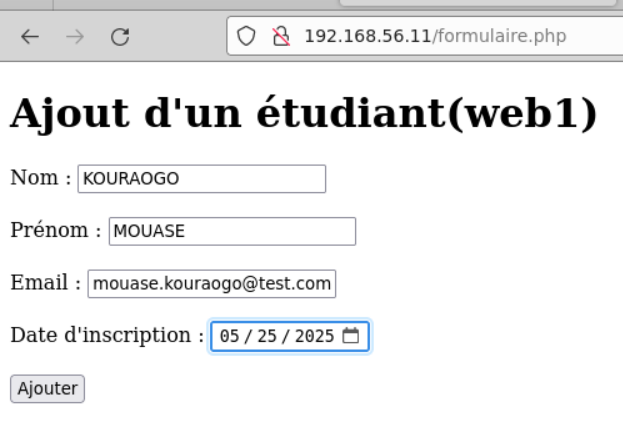
Figure 35 : code html du site

**Nous constatons que les tests de connectivité que les tests de connectivité sont concluants.**

## Test de remplissage du formulaire sur les serveurs web

Nous allons nous connecter à partir du client sur le serveur web1 et remplir un formulaire

En saisissant l’adresse <http://192.168.56.11/formulaire.php>



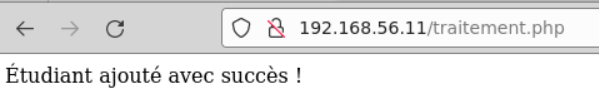
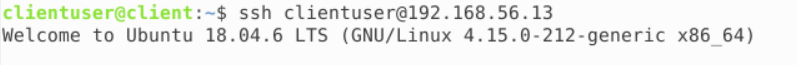
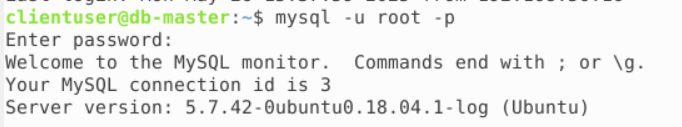


Figure 36 : Test de saisie

## Test de résilience (arrêt d’un web ou du maître, observation du comportement)





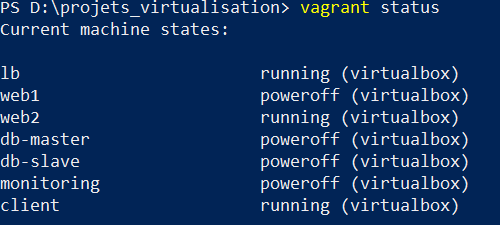
Nous avons éteint le serveur web2 pour effectuer le test

## 

## 



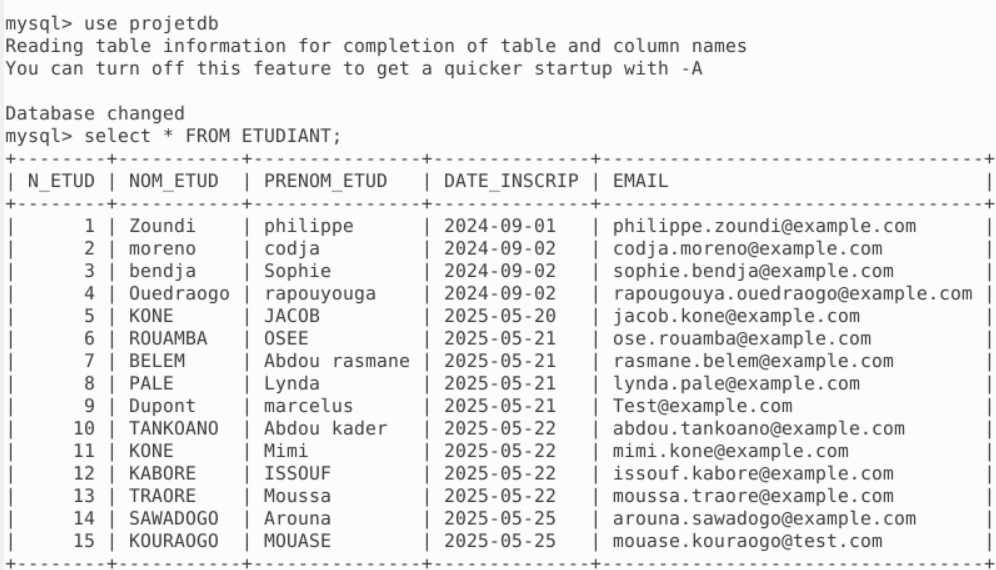
Nous allons éteindre maintenant le serveur web1





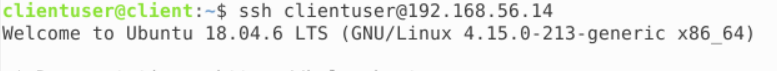
**En conclusion si l’un des serveurs web tombe en passe, l’autre reste toujours fonctionnel**

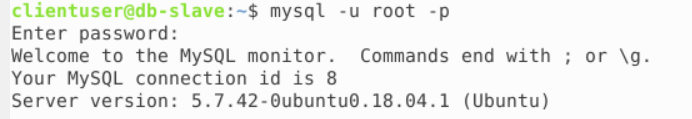
## *5.5.8 Test de* Vérification *et de réplication du* db-master et réplication sur db-slave



On observe que kouraogo mouase a bien été ajouté à db-master

Maintenant observons sur le db-slave





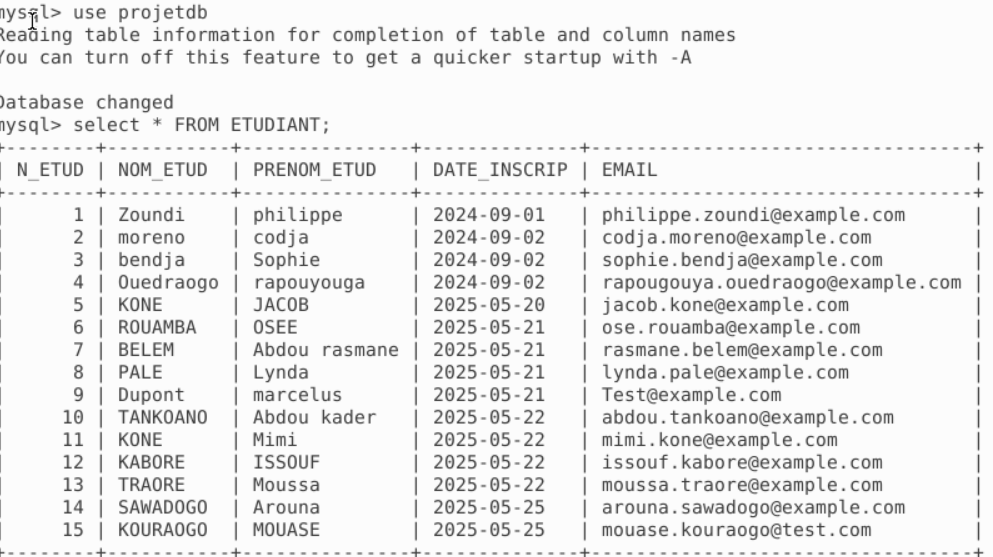


Figure 37 : Réplication effective

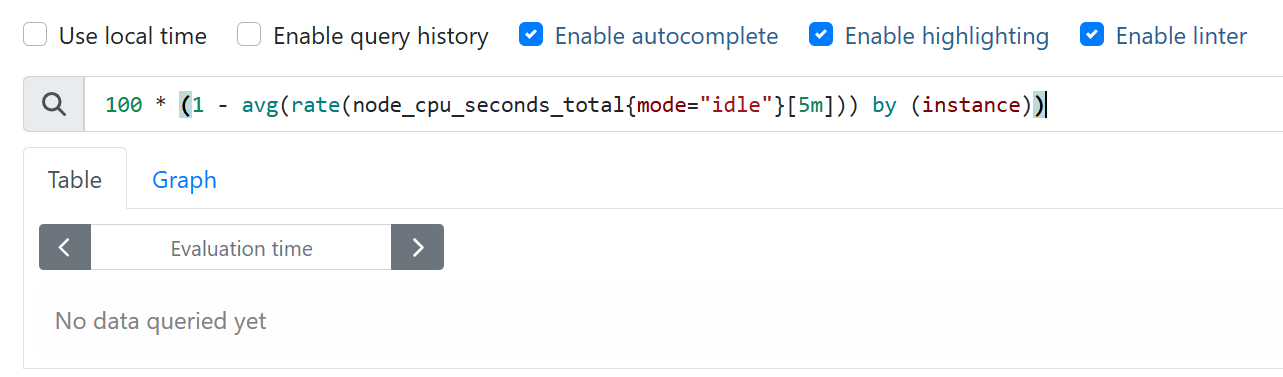
Nous constatons que la réplication est effective sur db-slave pour kouraogo Mouase

## Test de consommation des nœuds en pourcentage

Dans la fenêtre graph on a exécuté la commande up qui afficher **1** pour chaque cible en ligne.



Exécution de la commande : *100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance))* pour obtenir la consommation en pourcentage des différents nœuds toutes les 5m.



Dans cet exemple il s’agit du load balancer

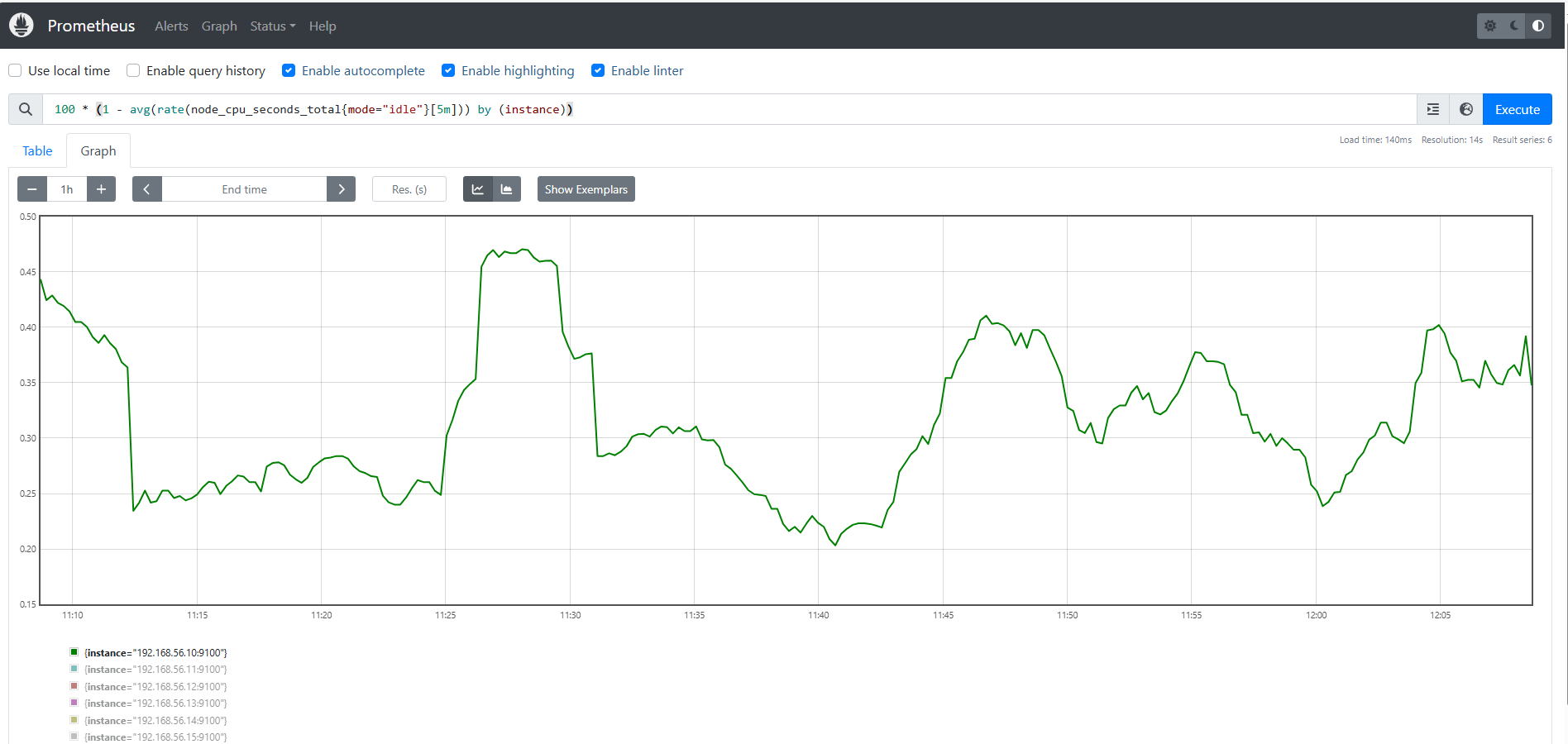
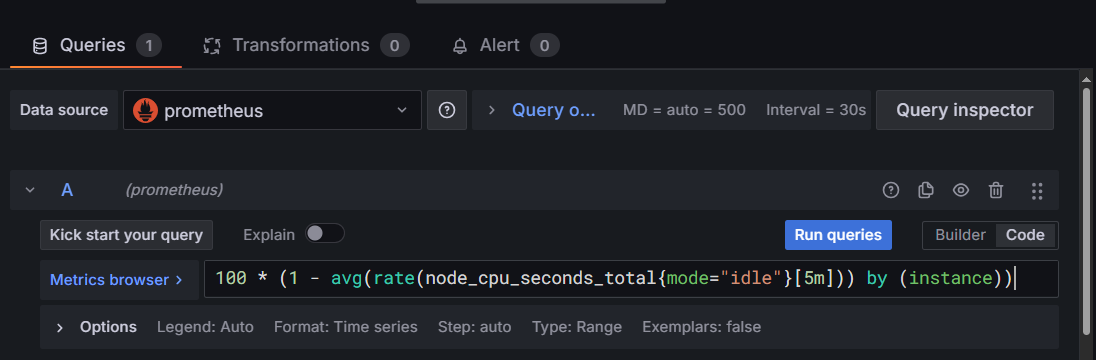


Figure 38 : consommation en pourcentage des nœuds du load balancer

Excécution du commande:

100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance))



Elle donne le pourcentage moyen d’utilisation du CPU sur chaque machine surveillée, calculé sur les 5 dernières minutes.

Résultat de la commande de vérification de consommation après l’exécution

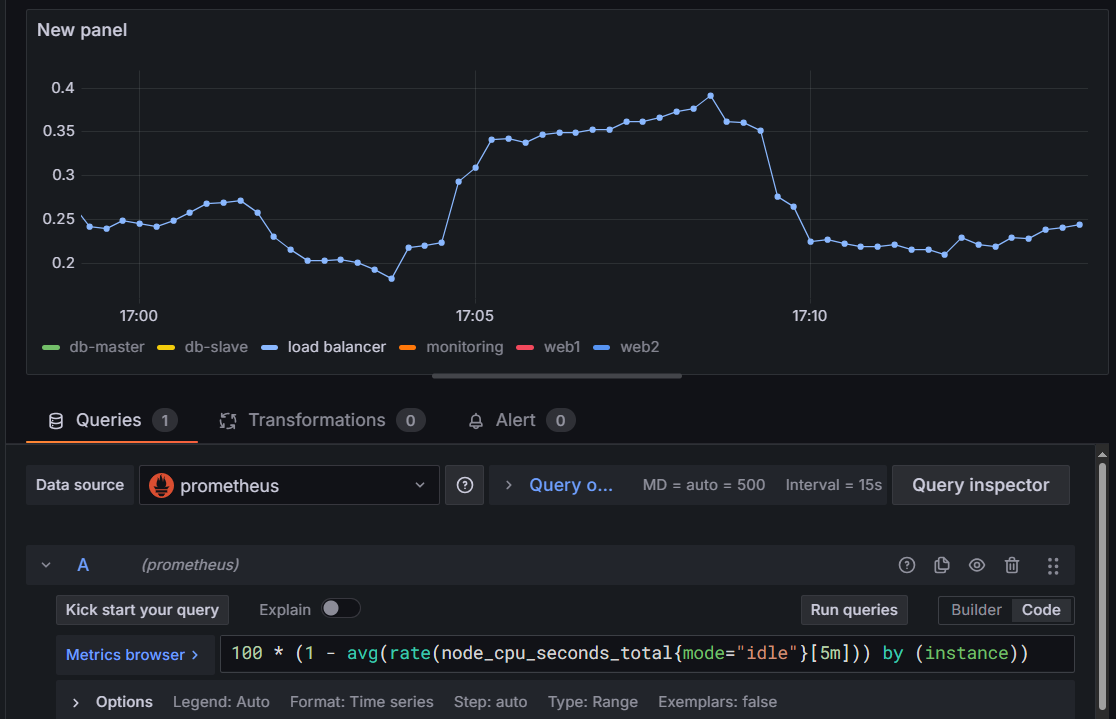
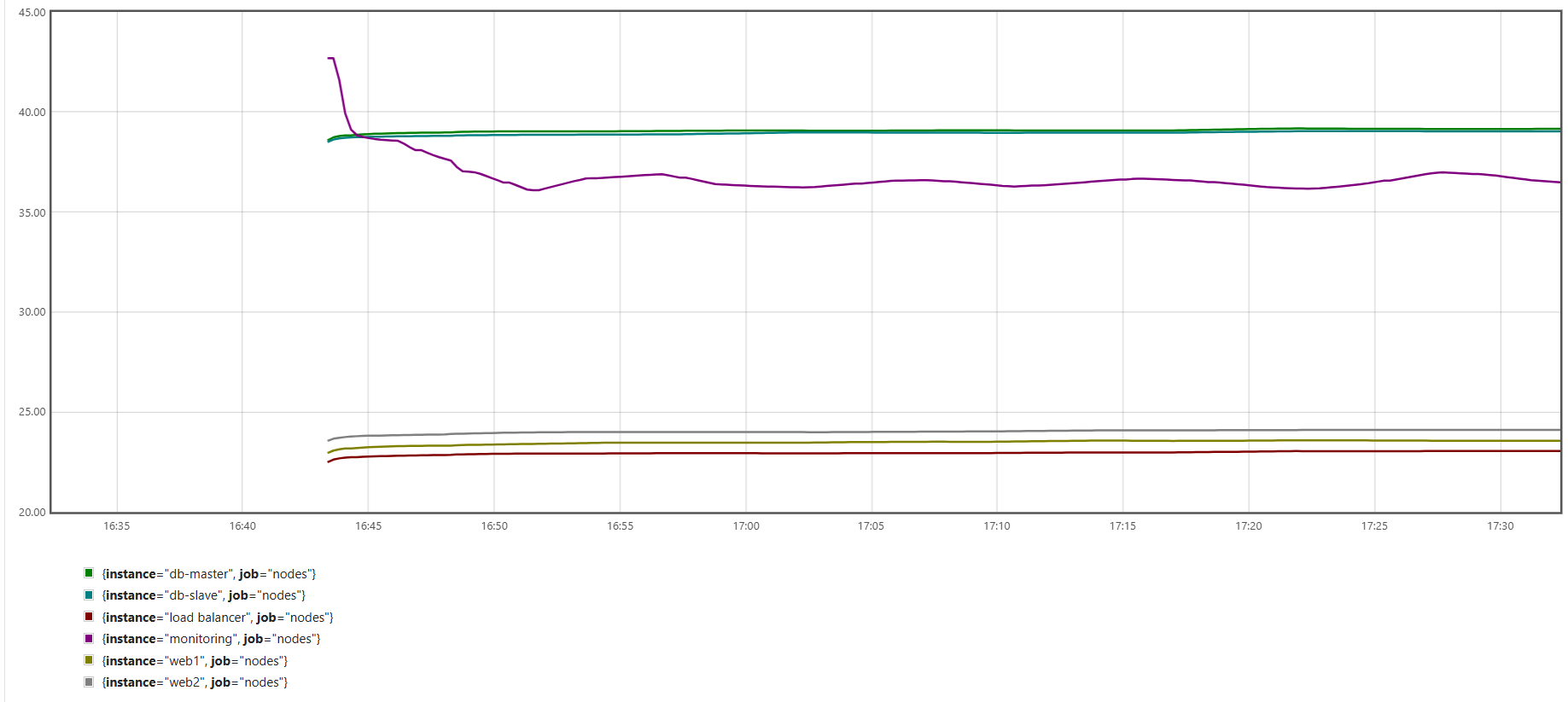
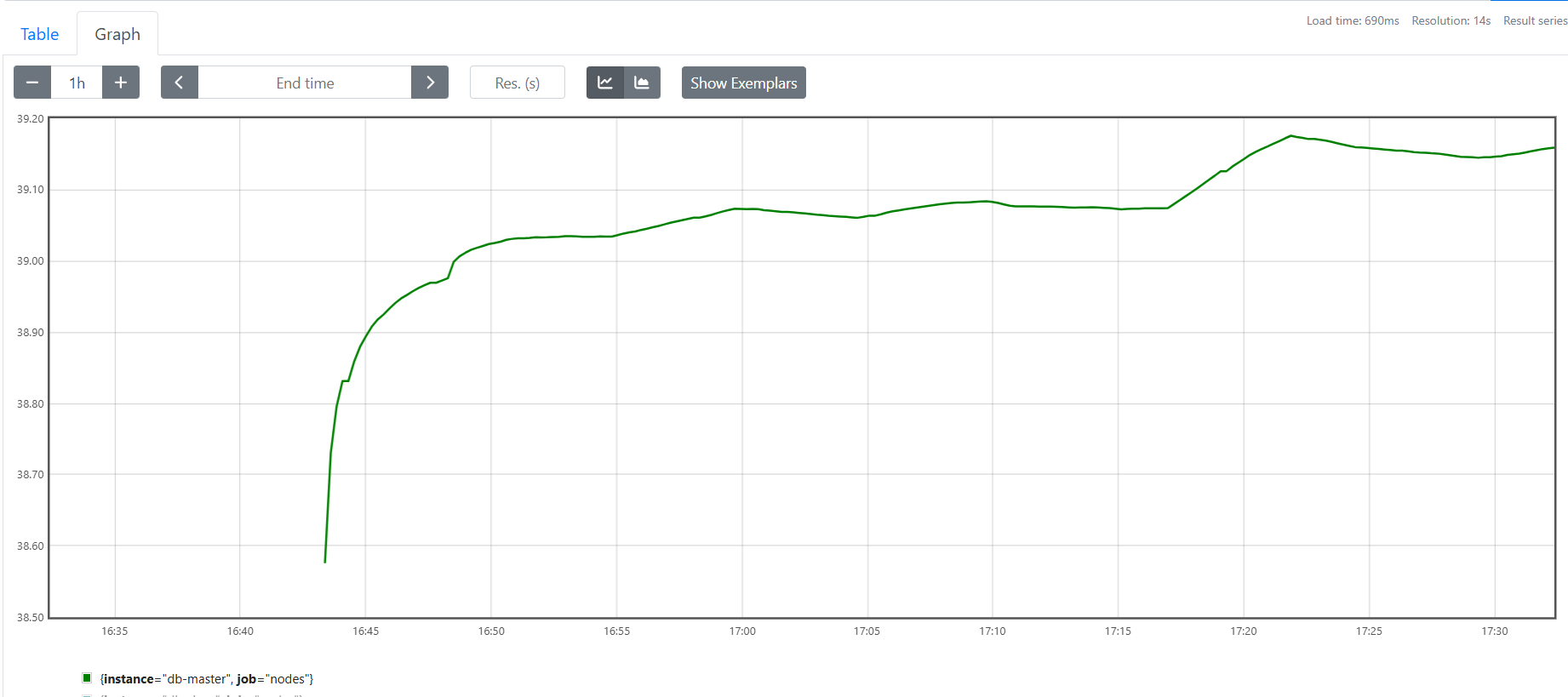


Figure 39 : consommation du CPU

## 5.5.10 Test de la consommation de la mémoire





*Figure 40 : Consommation de la mémoire pour le db master*

# **Conclusion**

## 6.1 Bilan du projet

La réalisation de ce projet d'infrastructure virtuelle multi-machines a permis de mettre en œuvre de manière concrète les concepts clés abordés en cours de virtualisation et cloud computing. Grâce à l'utilisation combinée de Vagrant pour l’orchestration et de Bash pour l’automatisation, nous avons réussi à construire une infrastructure complète, modulaire et fonctionnelle intégrant des composants critiques tels que la supervision, le load balancing et la réplication de base de données.

Cette approche a favorisé une compréhension approfondie des mécanismes d’interconnexion entre les services, de la gestion des dépendances, ainsi que des enjeux liés à la résilience, la redondance et la performance des systèmes virtualisés.

## 6.2 Compétences techniques et transversales acquises

Ce projet nous a permis de développer et renforcer plusieurs compétences :

* Techniques :
  + Déploiement automatisé d’infrastructures virtualisées
  + Configuration d’un système de répartition de charge avec Nginx
  + Mise en place d’un système de monitoring avec Prometheus et Grafana
  + Configuration d’une réplication MySQL maître-esclave
  + Résolution de problèmes liés à la connectivité, aux dépendances logicielles et à l’intégrité des services
* Transversales :
  + Travail collaboratif et gestion de version via Git
  + Capacité d’analyse et d’adaptation face aux imprévus techniques
  + Documentation technique rigoureuse et structuration claire du code
  + Gestion de projet en mode agile : planification, test, validation
  1. **Limites éventuelles et perspectives d’amélioration**

Malgré les résultats satisfaisants, certaines limites peuvent être identifiées :

* L’infrastructure ne dispose pas encore d’un mécanisme de scalabilité automatique, ce qui pourrait être envisagé via un outil comme Ansible, Docker Swarm ou Kubernetes.
* Les tests de résilience ont été réalisés manuellement ; une intégration continue avec des scripts de test automatisés renforcerait la robustesse.
* Le système de supervision pourrait être enrichi avec des alertes proactives, une journalisation plus fine, ou l’ajout de Loki pour la gestion des logs.

À terme, le projet pourrait évoluer vers un déploiement en environnement cloud public (AWS, Azure, GCP), afin de tester sa portabilité et sa flexibilité à plus grande échelle.

### **6.4 Lien vers le dépôt Git**

Le code source du projet, accompagné des scripts Bash, des fichiers de configuration Vagrant et des captures d’écran de validation, est disponible à l’adresse suivante :

[Lien GitHub du projet à insérer ici]

# **Bibliographie**

1. Rapport de projet de Virtualisation des étudiants deMaster 1 ISIE 2025
2. Le site officiel de Vagrant : documentation technique et tutoriels de référence.
3. Le site officiel de [MySQL](https://dev.mysql.com/doc/) : documentation sur la réplication, la gestion des utilisateurs et la sécurité.
4. Le site officiel de Grafana : déploiement, dashboards, intégration avec Prometheus.
5. Le site officiel de Prometheus : collecte de métriques, alerte et configuration.
6. [LinuxCommand.org](http://linuxcommand.org/) : Documentation sur la syntaxe Bash et les scripts shell.
7. WILLIAM, T. (2020). Scripts Bash pour administrateurs Linux : du débutant à l'expert. Libre Linux Press.
8. Formation OpenClassrooms (2023). Déployer son infrastructure avec Vagrant. OpenClassrooms.com
9. Réseau des grandes écoles d’ingénieurs (2023). La virtualisation au service de l’efficacité énergétique des data centers. Étude de cas interne.
10. CAIRE, C. (2022). Surveillance et performances des infrastructures IT. Guide technique chez Eyrolles.
11. Documentation Apache HTTP Server : <https://httpd.apache.org/docs/>
12. Documentation Nginx officielle : https://nginx.org/en/docs/
13. TUTOSI, L. (2022). L’art de l’automatisation système sous Linux. En ligne, revue LinuxTech.
14. Cours de Cloud Computing et Virtualisation (2024). Département Informatique, Université Virtuelle de Côte d’Ivoire.