Pierre-Alexandre STERBIK

PROJET :

API pour le cabinet LA CLEF IMMO

Sommaire

I. Cahier des charges 4

Suivi des modifications 4
Approbation 4
1. Contexte et définition du problème 5
2. Objectif(s) 5
3. Périmètre 6
4. Description fonctionnelle 6
5. Enveloppe budgétaire 9
6. Délais 9

II. Automatiser la création de serveurs à l'aide de scripts 10

7. Le serveur est fonctionnel 10
8. La configuration est conforme au cahier des charges 12
9. Les documentations sont mises à jour 12

III. Automatiser le déploiement d'une infrastructure 13

10. Les serveurs sont fonctionnels 13
11. L'architecture est conforme au cahier des charges 17
12. Mettre l'infrastructure en production dans le cloud 17

IV. Sécuriser l'infrastructure 18

13. Les mises à jour sont testées avant le déploiement : 18
14. L'application est sécurisée 19

V. Mettre l'infrastructure en production dans le cloud 21

15. Le serveur est fonctionnel 21
16. La configuration est conforme au cahier des charges 22
17. La documentation est mise à jour 23

VI. Exemple concret 24

18. Introduction 24
19. L'Api et sa database : 24
20. Le pipeline 25

VII. (Extra) : La supervision 29

21. Définir et mettre en place les statistiques de service 29

VIII. Exemple de recherche effectuée 32

22. Choisir la bonne technologie 32

IX. Conclusion 34

- *ANNEXE 1 : EXEMPLE DE DEPLOIEMENT RAPIDE AVEC VAGRANT 35*
- *ANNEXE 2 : CREATION DE SERVEURS LINUX 39*

PAGE INTENTIONALLY LEFT BLANK

I. Cahier des charges

Date de création du document	02-04-2024	**Auteur(s)**	P.Alexandre
Chef de projet	P.Alexandre	**Sponsor**	Cabinet laClef Immobilier
Membres de l'équipe projet	P.Alexandre Joël	Autres intervenants	Thibault Lejeune Alexandre Pesenti

Suivi des modifications

Version	Date	Auteur(s)	Modification(s) apportée(s)
V0.1	01/07/2024	P.Alexandre	Initialisation du projet
V0.2	29/08/2024	P.Alexandre	Projet arrivé à terminaison
V0.2.1	24/09/2024	P.Alexandre	Mise en place de l'infrastructure dans le cloud

Approbation

Version	Date	Nom	Rôle	Signature	Observation(s)
V0.1	04/07/2024	Thibault Lejeune	Directeur		Structure mise en place. Rien à déclarer, en attente
V0.2.1	31/08/2024	Thibault Lejeune	Directeur		Api testée et en production. Satisfait

1. Contexte et définition du problème

Les cabinets immobiliers payent cher le transfert de leurs annonces en passant par des passerelles qui les distribuent ainsi aux autres plateformes. S'ils avaient la possibilité d'avoir leur propre API avec leurs informations toujours enregistrées et disponibles à la diffusion, alors ils pourraient économiser sur ce qu'ils payent au niveau des passerelles au moins sur certains biais de diffusions. Ils auraient également un meilleur contrôle sur le transfert de leurs données et sur la garantie de leur sécurité.

2. Objectif(s)

- **Interopérabilité** :
L' API permet à différents systèmes, applications ou services d'interagir entre eux, même s'ils sont construits avec des technologies ou des langages différents. Elles facilitent la communication entre des plateformes distinctes.
- **Réutilisabilité** :
L' API permet de réutiliser des fonctionnalités existantes sans avoir à réinventer la roue. Par exemple, une application peut utiliser une API de géolocalisation ou de paiement sans avoir à développer ces services en interne.
- **Séparation des responsabilités** :
L' API permet de séparer le front-end (interface utilisateur) et le back-end (traitement des données). Cela permet à différentes équipes de se concentrer sur des parties spécifiques du projet sans interférence.
- **Facilité d'intégration** :
Grâce aux API, il est possible d'intégrer des services tiers (comme les services de paiement, de messagerie, ou encore des réseaux sociaux) rapidement et facilement dans une application.
- **Scalabilité** :
L' API permet aux applications d'évoluer plus facilement en fonction des besoins. Par exemple, un service basé sur une API peut répondre à une plus grande demande en augmentant simplement la capacité de son infrastructure.
- **Sécurité** :
L' API offre des moyens de contrôle d'accès et de sécurité, comme l'authentification via des clés API, des tokens, ou des protocoles comme OAuth, ce qui permet de protéger les données et les services.
- **Automatisation** :
L' API permet l'automatisation de tâches répétitives ou complexes en permettant à des programmes d'interagir directement entre eux sans intervention humaine.
- **Économie de temps et de coûts** :
En réutilisant des API existantes, les entreprises économisent du temps et des ressources en développement, ce qui réduit les coûts.

Coût de la mise en place estimée :

- Conception, Orchestration, Documentation
- Maquettage et mise en place du MVP
- Sécurisation, scalabilité
- Code de l'application
- UATs
- Budget total estimé :
    - TJM développeur : ~30€/J
    - Nombre de développeurs recommandés : 2
    - Total estimé : 1 mois
    - Budget estimé : 1800 euros

Délais estimés :

- Mise en place BDD, comptes associés et API keys : 1 semaine
- Mise en place de l'infrastructure sur le cloud : 1 semaine
- Monitoring – Hardware : 1 semaines
- Dockerisation, CI / CD / CD : 1 semaine
- Monitoring – Production, trafic : 1 semaine

Objectifs court terme : Sécuriser l'API et permettre de s'y enregistrer qui discute sans enregistrerd'information.

3. Périmètre

Géographique : Online

- Finances : 4000€
- Temps : max 2 mois
- Organisation : Profils recherchés : 1 Développeur, 1 Delivery Manager

4. Description fonctionnelle

4.1. Formulation des fonctions

Fonctions :

- Identification

  Tâches:

  ➔ Développement de l'API

- Préparation de l'environnement

  Tâches

  ➔ Préparer l'environnement (Docker compose, back-end…)
  - ◆ Création d'instances (x3 t3.micro)
    - Une instance pour le CI/CD
      - Technologie : Docker/K8S
    - Une instance pour l'API
      - Technologie : NodeJS
    - Une instance pour le monitoring
      - Technologie : Docker, Prometheus, Grafana
    - Une instance pour la BDD
      - Ajouter un volume pour avoir de la marge
      - Technologie : Docker, MariaDB
  - ◆ Création de clefs d'authentification et de tokens AWS en créant un compte administrateur indépendant de l'administrateur racine
  - ◆ Créer les politiques de gestion des ressources, l'associer au group Administrateurs

- Configurer un modèle d'utilisateurs éphémères en passant par l'IAM Identity Center
- Création d'une AMI et d'instantanées

➔ Mise en place d'un pipeline jenkins, git repo… Pour le CI/CD

➔ Monter la BDD (penche vers MARIADB)

Liste des tâches

- Préparer l'environnement (cloud, back-end…)
- Créer le système de connexion
- Sécurisation du système de connexion et d'accès à l'API
- Routage des différents services
- Mise en place des services
- Dockerisation
- Documentation et README

USER STORIES :

En tant que jeune entreprise, je veux avoir ma propre infrastructure informatique pour avoir une indépendance de service tiers souvent bien plus chers de par leur savoir que je ne possède pas.

Les APIs sont une technologies incontournable aujourd'hui de communication entre différents services et technologies, et je veux faire partie de cette révolution

AWS is so hot right now ! Voyagant beaucoup, je veux avoir accès à mes ressources partout où je vais. C'est pourquoi je choisis la solution cloud pour construire mes applications !

Fonction : Enregistrer les annonces et les informations des clients	
Objectif	Enregistrer toute nouvelle annonce et toute nouvelle opportunité de manière sécurisée, rapidement et à moindre frais
Description	
Contraintes / règles de gestion	Nombre de connexions possibles, nombre de comptes possibles etc..
Niveau de priorité	Priorité haute

Fonction : Communication avec les autres plateformes	
Objectif	Faire le lien avec les autres sites pour pouvoir diffuser les annonces sur leurs plateformes.
Description	Amélioration de la mise à jour des biens sur les différentes annonces.
Contraintes / règles de gestion	Voir les prix que demandent les entreprises pour pouvoir diffuser des annonces chez eux
Niveau de priorité	Priorité basse

Fonction : Infrastructure fiable et sans downtime	
Objectif	Création d'une AMI pour chaque instance, mise en place du monitoring du hardware des instances, création d'instantanées
Description	
Contraintes / règles de gestion	Faire des tests sur l'infrastructure pour vérifier si les ressources ont correctement été allouées
Niveau de priorité	Priorité basse

Fonction : Automatisation des tâches	
Objectif	Récupération des données utilisateurs sous format JSON pour faciliter la distribution
Description	Plus besoin d'assigner un compte Hektor à une adresse mail d'un employé puisqu'un compte avec token peut être donnée pour que tout le monde autorisé y accède (prévoir un Token temporaire pour les employés et une clef pour les patrons). Création du pipeline Jenkins pour faciliter la transition entre le code et la production.
Contraintes / règles de gestion	Créer un frontend accessible pour que les particuliers puissent se servir de l'API

Niveau de priorité	Priorité basse

5. Enveloppe budgétaire

Poste	tjm	charge en jours homme	Budget alloué	Commentaires
Développement	30€	30 J	900€	Code pur de l'application, mise en place des bases de données, back end APIs et mappage
Delivery manager	30€	2 semaines	450€	Sécurisation, automatisation, mise en place du monitoring, mise en place des pipelines (MariaDB), vitesse d'exécution, ergonomie… système de log d'erreurs, API keys and tokens…
AWS Infrastructure	Plan offre gratuite sur la première année		**50€**	Revoir le budget 2 mois avant L'échéance
Total			**1400€**	
Total Ap. Risque		**60J**	**2850€**	

6. Délais

Livrable	Date souhaitée	Commentaires
Documentation, maquette, mise en place parcours,	1 semaine	Guide utilisateur et maquettage
Prototypage	3 semaines	Code terminé, désigné, prêt à être testé, Les requêtes fonctionnent et sont sécurisées et la BDD est en place
Produit fini	2 mois max	- Dockerisation

		- Deploiement sur le cloud
Fin de projet	**Fin octobre 2024**	

Service Level Agreement SLA
99,9%

II. Automatiser la création de serveurs à l'aide de scripts

7. Le serveur est fonctionnel

- **Repos git des différents serveurs:**
  - **Server NodeJS : https://github.com/Manianise/docker-node-api**
  - **Server Jenkins : https://github.com/Manianise/docker-jenkins-env**
  - **Server Grafana : https://github.com/Manianise/docker-prometheus-grafana**

- **Les serveurs fonctionnent :**

	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
	jenkins-agen node-ctn a642e413f195	mechameleon/nod	Running		0.11%	2 hours ago	
	docker-node-api		Running (2/2)		0.01%	3 hours ago	
	docker-env		Running (3/3)		1.84%	2 hours ago	
	docker-monitoring		Running (4/4)		0.12%	3 hours ago	

- **Voici un extrait du code qui permet de run les containers :**

```
- services:
-   prometheus-ctn:
-     image: prom/prometheus:latest
```

```
-       container_name: prometheus-ctn
-       ports:
-         - "9090:9090"
-       networks:
-         - dev-tools-network
-       volumes:
-         - ./prometheus:/etc/prometheus/
-         - prometheus-storage:/prometheus
-       command:
-         - "--config.file=/etc/prometheus/prometheus.yml"
-
-     grafana-ctn:
-       image: grafana/grafana:latest
-       container_name: grafana-ctn
-       ports:
-         - "3000:3000"
-       networks:
-         - dev-tools-network
-       environment:
-         - GF_SECURITY_ADMIN_PASSWORD=admin
-       volumes:
-         - grafana-storage:/var/lib/grafana
-
-     alertmanager-ctn:
-       image: prom/alertmanager:latest
-       container_name: alertmanager-ctn
-       ports:
-         - "9093:9093"
-       networks:
-         - dev-tools-network
-       volumes:
-         -
    ./alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.yml
-         - alertmanager-storage:/alertmanager
-
-     node-exporter-ctn:
-       image: prom/node-exporter:latest
-       container_name: node-exporter-ctn
-       ports:
-         - "9100:9100"
-       networks:
-         - dev-tools-network
-
-   networks:
-     dev-tools-network:
-       external: true
-
-   volumes:
-     grafana-storage:
```

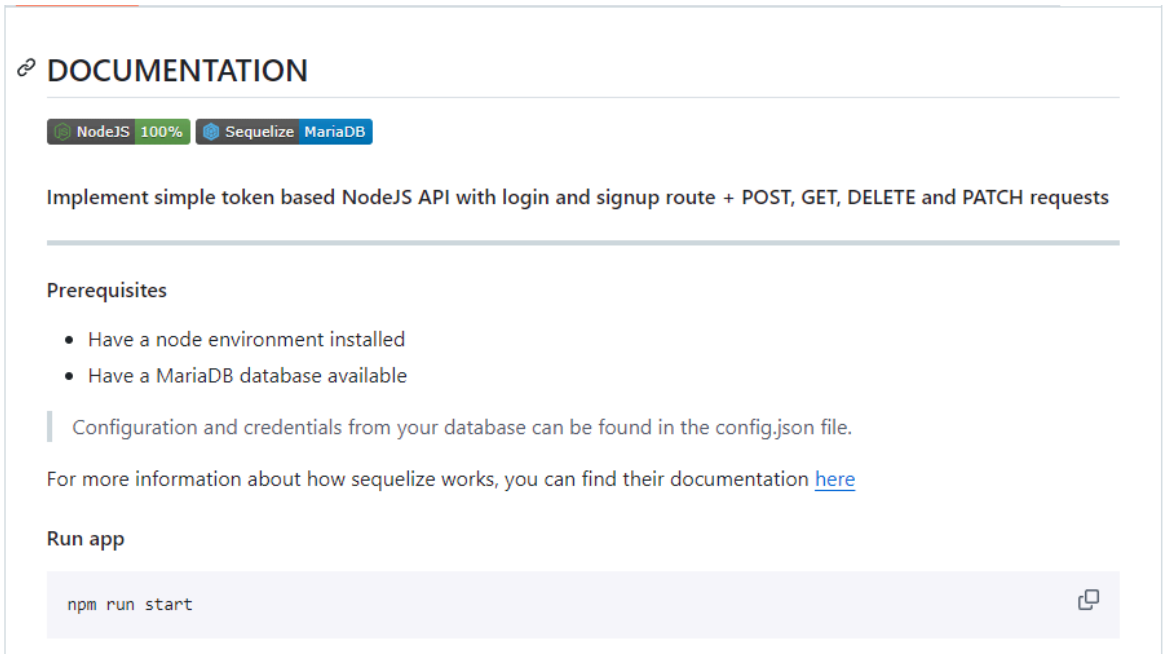
```
  -    prometheus-storage:
  -    alertmanager-storage:
  -
```

8. La configuration est conforme au cahier des charges

- Le serveur NodeJS est dans un container séparé de la base de données MariaDB
- Le serveur d'outils de monitoring est à part
- Le serveur de déploiement est à part

9. Les documentations sont mises à jour

Les documentations sont mises à jour et accessibles à toute l'équipe de développement



DOCUMENTATION

NodeJS 100% Sequelize MariaDB

Implement simple token based NodeJS API with login and signup route + POST, GET, DELETE and PATCH requests

Prerequisites

- Have a node environment installed
- Have a MariaDB database available

> Configuration and credentials from your database can be found in the config.json file.

For more information about how sequelize works, you can find their documentation here

Run app

```
npm run start
```

- Verifier que les tags sont à jour :

```
Writing objects: 100% (1/1), 202 bytes | 202.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Manianise/vagrant.git
 * [new tag]         v0.1.1 -> v0.1.1
```

```
and the repository exists.
PS C:\wamp64\www\production\NodeJS\clc-api\api-nodejs> git push origin tag v1.1.0
Everything up-to-date
PS C:\wamp64\www\production\NodeJS\clc-api\api-nodejs> git add . &&  git commit -m "Commit 04/09/20024" && git push origin tag v1.1.0
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS C:\wamp64\www\production\NodeJS\clc-api\api-nodejs> git tag --list
v1.0.1
v1.1.0
```

III. Automatiser le déploiement d'une infrastructure

10. Les serveurs sont fonctionnels

La création de l'infrastructure du client n'ayant pas demandé une configuration par script car utilisant les capacités de bases offertes par AWS lors de la création de leur compte sur le cloud et ceci tout en respecant le cahier des charges. Ci-dessous, l'extrait du code fonctionnel de l'infrastructure :

- Tree

```
.
├── README.md
├── main.tf
├── modules
│   ├── main.tf
│   ├── outputs.tf
│   ├── secrets
│   │   └── aws-key-public.pub
│   ├── variables.tf
│   └── versions.tf
├── terraform.tf
├── terraform.tfstate
└── terraform.tfstate.backup
```

- Extract :

```
provider "aws" {
  shared_config_files      = ["C:\\Users\\pierr\\.aws\\config"]
  shared_credentials_files = ["C:\\Users\\pierr\\.aws\\credentials"]
}

data "aws_vpc" "euw-3_vpc" {
  id = var.vpc-euw-3
}

# use subnet with internet access
data "aws_subnet" "public_subnet" {
  id = var.eu_subnets.subnet_a.id
}

data "aws_security_group" "aws_sg" {
  id = var.security_group_id
}

# Create the first EC2 instance (Node API) in subnet 1 (AZ 1)
resource "aws_instance" "node_api_instance" {
  ami           = var.ami.id
  instance_type = var.ami.instance_type

  key_name = "aws-key"

  vpc_security_group_ids = [data.aws_security_group.aws_sg.id]

  subnet_id  = data.aws_subnet.public_subnet.id
  associate_public_ip_address = true

  tags = {
    Name = "node-api-instance"
```

- **Variable pour le subnet**

```
variable "eu_subnets" {
  description = "This is a variable of type Map of objects"
  type = map(object({
    az = string,
    cidr = string
    id = string
  }))
  sensitive= true
```

```
}
```

Néanmoins, voici le code qui permettra de créer une nouvelle infrastructure lorsque cela sera nécessaire dans un futur plus ou moins proche, et voici l'infrastructure en question :

- Création d'un VPC avec :
  - 1 sous-réseau en NAT qui contiendra l'application ainsi que sa base de données
  - 1 sous-réseau en Bridge pour le serveur de monitoring et le serveur de CI/CD/CD
- Une table de routage
- Un groupe de sécurité (pour la sécurité) pouvant admettre les connexions en SSH

Des extraits du code, fonctionnel et prêt à l'emploi :

```
provider "aws" {
  region = "eu-west-3"
}

# VPC
resource "aws_vpc" "main_vpc" {
  cidr_block = "172.31.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = "main_vpc"
  }
}

# Subnet 1 (Public)
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.main_vpc.id
  cidr_block        = "172.31.1.0/24"
  availability_zone = "eu-west-3a"
  map_public_ip_on_launch = true
  tags = {
    Name = "public_subnet"
  }
}

# Subnet 2 (Private)
resource "aws_subnet" "private_subnet" {
  vpc_id            = aws_vpc.main_vpc.id
  cidr_block        = "172.31.2.0/24"
  availability_zone = "eu-west-3a"
  tags = {
    Name = "private_subnet"
```

```
# Internet Gateway (for public subnet)
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main_vpc.id
  tags = {
    Name = "internet_gateway"
  }
}

# NAT Gateway (for private subnet's outbound access)
resource "aws_eip" "nat_eip" {
  vpc = true
}

resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = aws_subnet.public_subnet.id
  tags = {
    Name = "nat_gateway"
  }
}

# Route Table for Public Subnet
resource "aws_route_table" "public_route_table" {
  vpc_id = aws_vpc.main_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {

# Route Table Association for Private Subnet
resource "aws_route_table_association" "private_route_table_association" {
  subnet_id      = aws_subnet.private_subnet.id
  route_table_id = aws_route_table.private_route_table.id
}

# Security Group (Optional, can customize)
resource "aws_security_group" "allow_ssh" {
  vpc_id = aws_vpc.main_vpc.id
  name   = "allow_ssh"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "allow_ssh"
  }
}
```

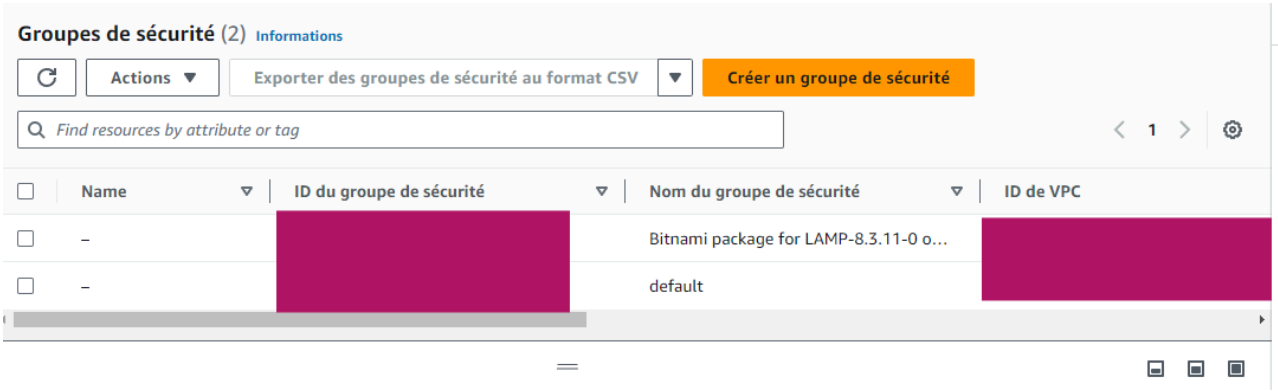
11. L'architecture est conforme au cahier des charges

Comme expliqué précédemment, L'architecture est effectivement conforme au cahier des charges, que vous pouvez retrouver en suivant ce lien.

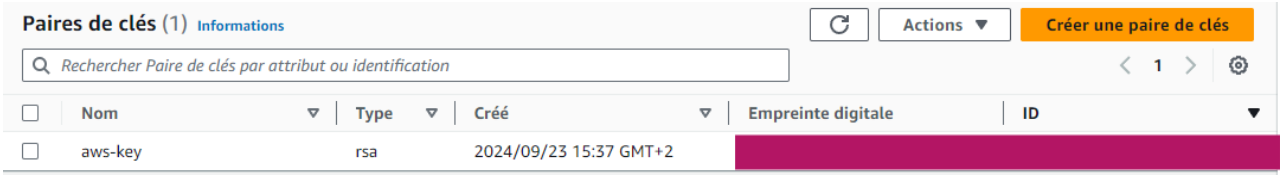
12. Mettre l'infrastructure en production dans le cloud

Comme il n'est pas possible de montrer trop de données sensibles du client, voici quelques extraits pertinents de l'infrastructure dans le cloud en production :

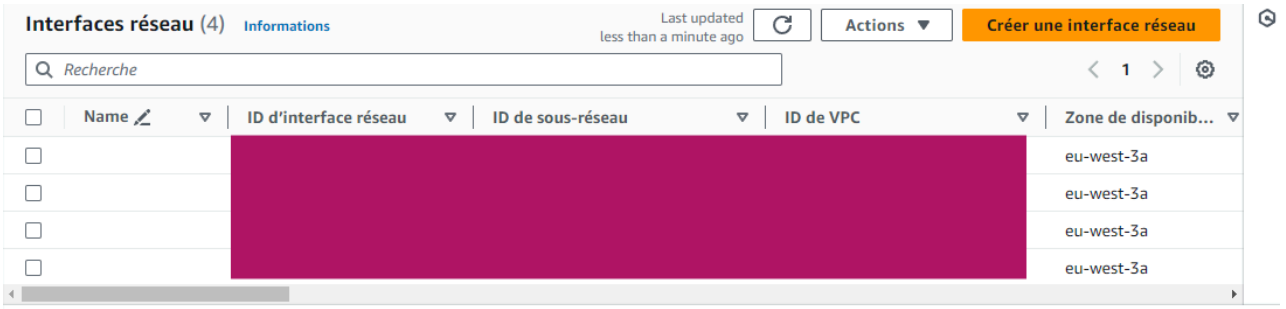
- **Groupes de sécurité**



- **Clefs d'accès SSH**



- **Subnets**



IV. Sécuriser l'infrastructure

13. Les mises à jour sont testées avant le déploiement :

Comme expliqué dans l'article d'AWS sur les environnements de tests, l'infrastructure dans le cloud est correctement configurée pour la phase de test avant le lancement :

Provisioning instances

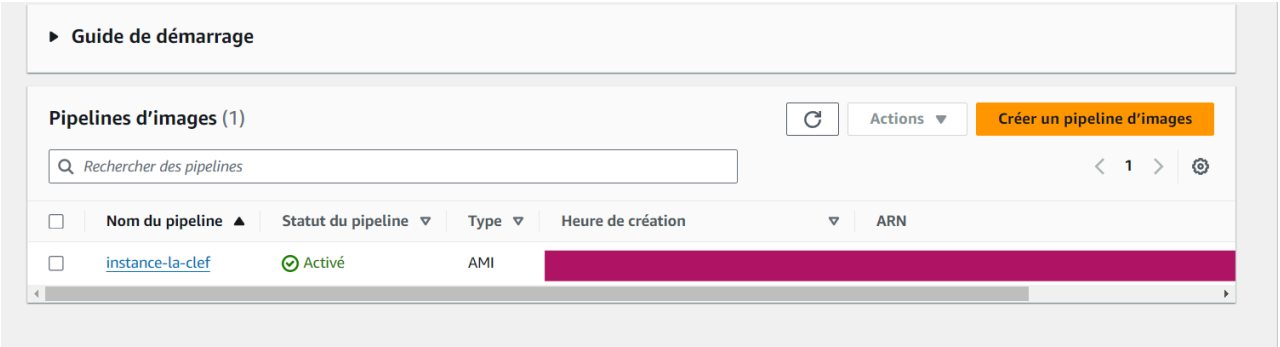
You can easily provision Amazon EC2 instances from AMIs. An AMI encapsulates the operating system and any other software or configuration files pre-installed on the instance. When you launch the instance, all the applications are already loaded from the AMI and ready to run. For information about creating AMIs, refer to the Amazon EC2 documentation. The challenge with AMI-based deployments is that each time you need to upgrade software, you have to create a new AMI. Although the process of creating a new AMI (and deleting an old one) can be completely automated using EC2 Image Builder, you must define a strategy for managing and maintaining multiple versions of AMIs.

An alternative approach is to include only components into the AMI that don't change often (operating system, language platform and low-level libraries, application server, and so on). More volatile components, like the application under development, can be fetched and deployed to the instance at runtime. For more details on how to create self- bootstrapped instances, refer to Bootstrapping.

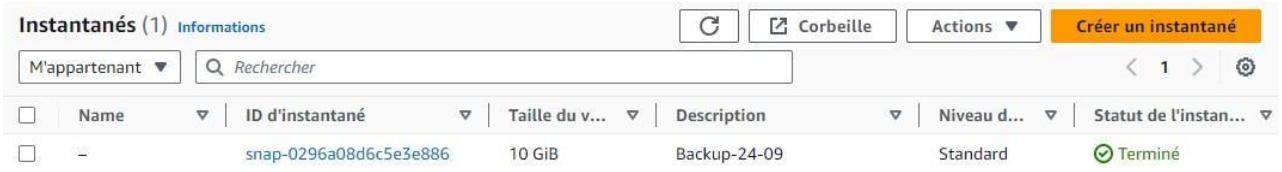
Provisioning databases

Test databases can be efficiently implemented as Amazon RDS database instances. Your test teams can instantiate a fully operational database easily, and load a test dataset from a snapshot. To create this test dataset, you first provision an Amazon RDS instance. After injecting the dataset, you create a snapshot of the instance. From that time, every time you need a test database for a test environment, you can create one as an Amazon RDS instance from that initial snapshot. Refer to Restoring from a DB snapshot. Each Amazon RDS instance started from the same snapshot will contain the same dataset, which helps ensure that your tests are consistent.

- **Un pipeline d'image a donc été généré pour l'entreprise, afin d'effectuer à une date fixe et de garantir la stabilité et l'intégrité des instances :**



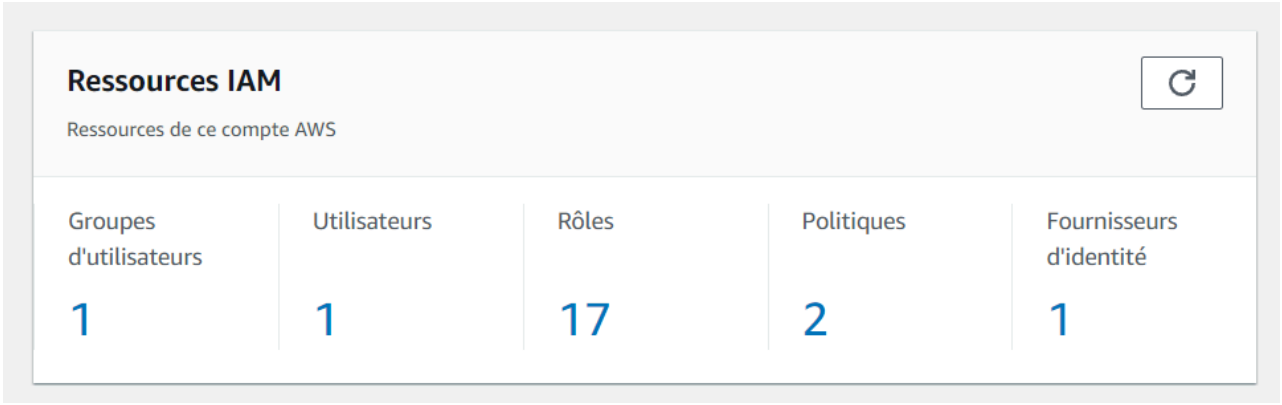
- **Les snapshots sont prêts pour l'équipe de test :**



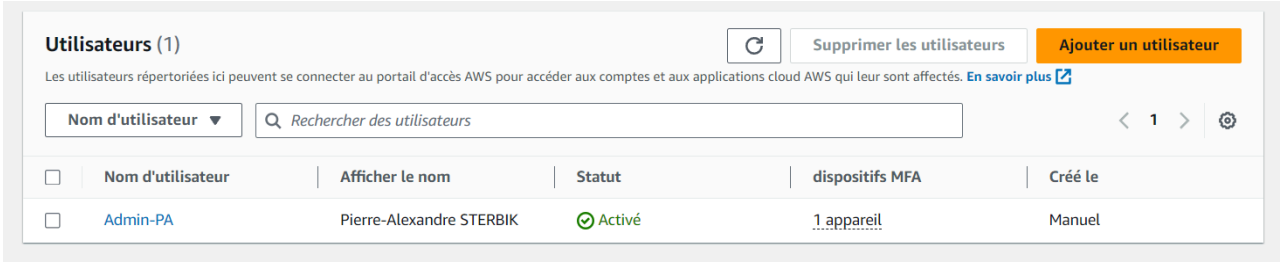
14. L'application est sécurisée

Une console administration a été préparée pour les futurs administrateurs ainsi que pour les directeurs de l'entreprise. Les rôles IAM ont été créés et l'Identity Center a été mis en place pour les sessions éphémères. Un système de facteur à double authentification a également été mis en place :

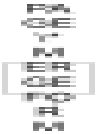
- Overview :

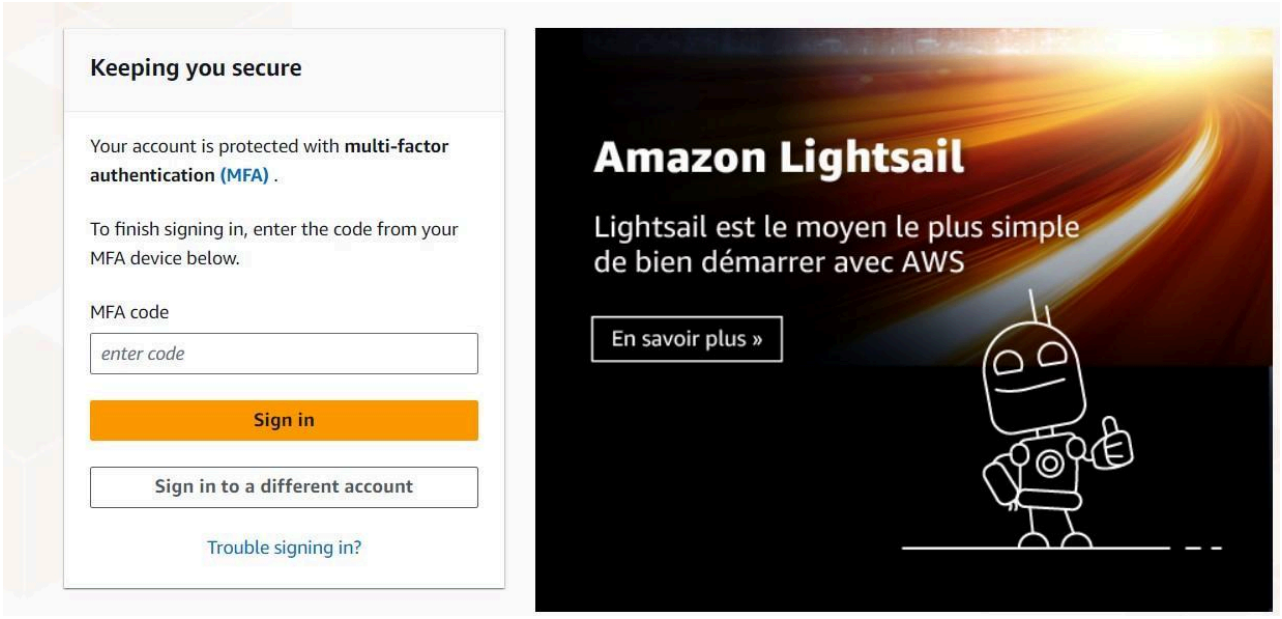


- Identity Center : compte éphémère :



- Double authentication :





L'authentification par code éphémère mobile a été utilisée. Le subnet du VPC gérant la base de données à un nombre limité d'IPs pouvant y avoir accès rentrer sur le serveur privé a été limité au port 22, protocole TCP pour des raisons de sécurité (connexion en SSH).

V. Mettre l'infrastructure en production dans le cloud

15. Le serveur est fonctionnel

- **Comme vu dans la figure ci-dessous, les serveurs dans le cloud sont fonctionnels et ont été générés à l'aide de scripts :**
- **Tree :**

```
.
├── README.md
├── main.tf
├── modules
│   ├── main.tf
│   ├── outputs.tf
│   ├── secrets
│   │   └── aws-key-public.pub
│   ├── variables.tf
│   └── versions.tf
├── terraform.tf
├── terraform.tfstate
└── terraform.tfstate.backup
```

- **Afin d'automatiser la création de serveurs, j'ai utilisé la technologie Terraform. Extrait :**

```
provider "aws" {
  shared_config_files      = ["C:\\Users\\pierr\\.aws\\config"]
  shared_credentials_files = ["C:\\Users\\pierr\\.aws\\credentials"]
}

data "aws_vpc" "euw-3_vpc" {
  id = var.vpc-euw-3
}

# use subnet with internet access
data "aws_subnet" "public_subnet" {
  id = var.eu_subnets.subnet_a.id
}

data "aws_security_group" "aws_sg" {
  id = var.security_group_id
}

# Create the first EC2 instance (Node API) in subnet 1 (AZ 1)
resource "aws_instance" "node_api_instance" {
  ami           = var.ami.id
  instance_type = var.ami.instance_type

  key_name = "aws-key"

  vpc_security_group_ids = [data.aws_security_group.aws_sg.id]

  subnet_id  = data.aws_subnet.public_subnet.id
  associate_public_ip_address = true

  tags = {
    Name = "node-api-instance"
```

Ln 28, Col 48 Spaces: 2 UTF-8 CRLF Terraform

- **Donnant ainsi le résultat suivant :**

Instances (4) Informations

Date de la dernière mise à jour Il y a less than a minute | Se connecter | État de l'instance | Actions | Lancer des instances

Rechercher Instance par attribut ou identification (case-sensitive) | Tous les états | 1

Name	ID d'instance	État de l'insta...	Type d'insta...	Contrôle des statu	Statut d'alarm	Zone
node-api-insta...	i-0ff03e4ce62873edf	En cours d'...	t2.micro	2/2 vérifications r	Afficher les alarm	eu-we
monitoring-in...	i-005e10ef6016c2f57	En cours d'...	t2.micro	Initialisation en cc	Afficher les alarm	eu-we
database-insta...	i-062dc0c24eaf9b375	En cours d'...	t2.micro	Initialisation en cc	Afficher les alarm	eu-we
delivery-insta...	i-015585981357132ca	En cours d'...	t2.micro	Initialisation en cc	Afficher les alarm	eu-we

- **Les volumes correspondants ont été également ajoutés :**

Name	ID du volume	Type	Taille	IOPS	Débit	ID de l'inst...	Créé
–	vol-0478e7076fd3d2e05	gp3	8 GiB	3000	125	snap-044d974...	2024
–	vol-048f45457bcb8327c	gp3	8 GiB	3000	125	snap-044d974...	2024
–	vol-00fa7ee1e1a0deee4	gp3	8 GiB	3000	125	snap-044d974...	2024
–	vol-0ebf01b2d152d798e	gp3	10 GiB	3000	125	-	2024

```
    + ebs_block_device {
        + delete_on_termination = true
        + device_name           = "/dev/sdh"
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + snapshot_id           = (known after apply)
        + tags_all              = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = 10
        + volume_type           = "gp3"
      }

    + root_block_device {
        + delete_on_termination = true
        + device_name           = (known after apply)
        + encrypted             = (known after apply)
        + iops                  = (known after apply)
        + kms_key_id            = (known after apply)
        + tags_all              = (known after apply)
        + throughput            = (known after apply)
        + volume_id             = (known after apply)
        + volume_size           = 20
        + volume_type           = "gp3"
      }
  }
```

Code en lien git : https://github.com/Manianise/terraform-ec2-aws (safe credentials)

16. La configuration est conforme au cahier des charges

Afin de vérifier que la configuration est conforme au cahier des charges, vous pouvez vous y rendre en cliquant sur ce **lien**.

Vous y trouverez que le besoin dans l'infrastructure cloud est de 4 instances t2.micro soumises à l'offre gratuite avec l'image AMI d'un AWS Linux classique, la mise en place de l'Autoscaling garantira un passage doux de l'offre gratuite à une offre payante en fonction du nombre de requêtes et d'utilisations croissantes de l'API.

17. La documentation est mise à jour

La documentation officielle est à jour et est disponible à tous les initiés au projet

Documentation Officielle :

Récupérer des membres de l'équipe

- Tous les membres et leur id peuvent être récupérés ici : `/members`
- Un membre seul peut être récupéré si vous connaissez id nom `/members/id`
- la requete POST a l'adresse `/members` permet de rajouter un membre.
- `Création d'un objet JSON`

```
{
    "name" : "prénom",
    "lastName" : "nom",
    "phone" : "numéro (string)",
    "email" : "string",
    "url" : "string"
}
```

- La requete UPDATE permet de modifier le membre `/members/id` en utilisant la meme methode que pour une requete POST
- La requete DELETE a `/members/id` pour supprimer un membre de l'équipe

La documentation concernant l'infrastructure cloud peut être retrouvée facilement dans le README.md du projet Terraform du repository GIT en suivant ce lien :

- **https://github.com/Manianise/terraform-ec2-aws/blob/main/README.md**

Connect to a AWS VPC and create t2.micro instances

terraform hashicorp AWS EC2

Simply run the follwing commands

/!\ You will need AWS keys to run this code :

> make sure terraform is installed in your environment

```
terraform init
terraform apply
```

Et ci-dessous, un extrait de la documentation pour installer les applications de monitoring :

```
### All in one compilation to install Grafana, Prometheus, Alertmanager cAdvisor and Node Exporter on a Linux based machine

#### Before we begin

- This playbook is set to install Prometheus and Grafana on a local machine. If you wish to use the version v0.0.1, witch has an inventory, by running
following command :

```
git checkout tag/v0.0.1
```

Requirements

- Ansible installed on local machine
- Grafana Account with credentials in group_vars/all/vault.yml

Linux

- Replace [vault.yml](group_vars/all/vault.yml) with your secret variables e.g Grafana credentials as <code>grafana_user</code> and <code>grafana_pass
code>
- It is recommended to encrypt your [vault.yml](group_vars/all/vault.yml)
- If you do not encrypt your vault.yml, add a semi-colon next to ask vault pass in ansible.cfg

- Run the following commands :
```
ansible-playbook -i inventory.ini playbook.yml
```

Windows
```

VI. Exemple concret

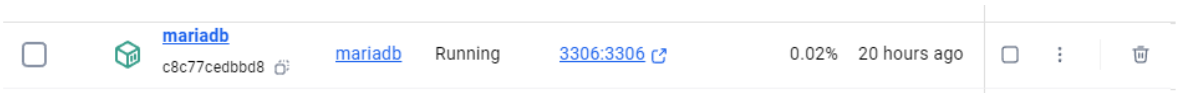
18. Introduction

Dans cet exemple concret, j'ai mis en place tout un développement de l'application elle-même dans le contexte dans lequel elle sera envoyée, jusqu'à son déploiement en 'production', c'est-à-dire la présentation d'un cycle de DevOps dans un environnement de test.

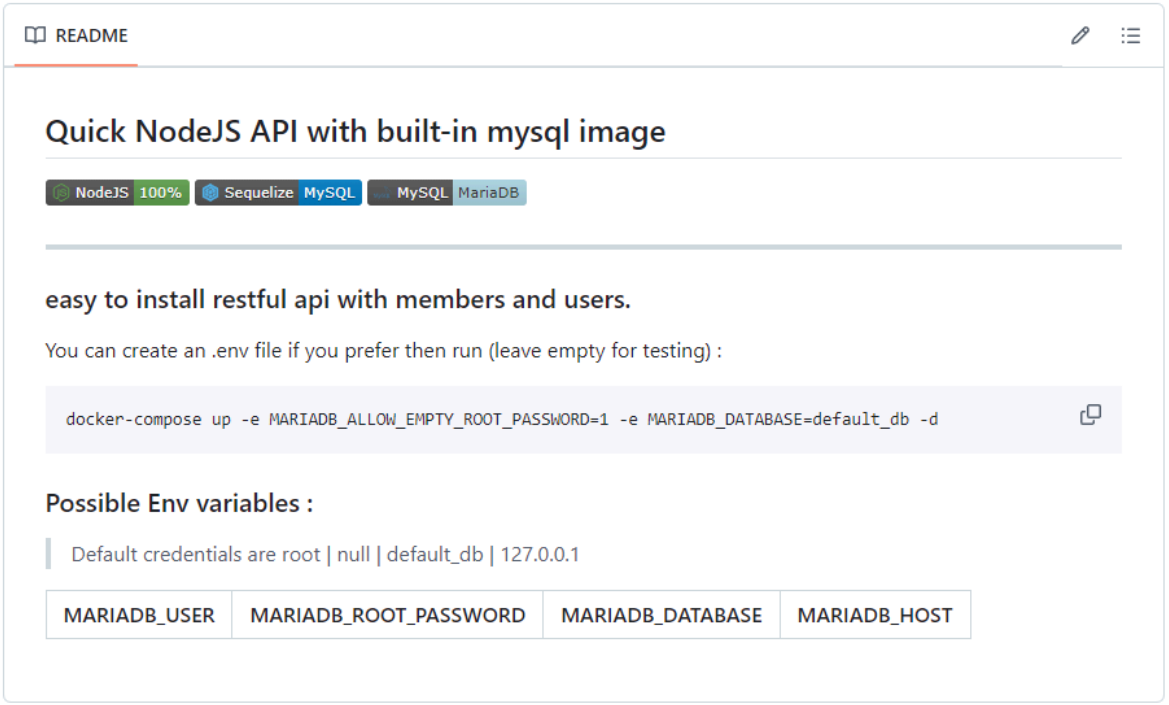
19. L'Api et sa database :

Tout d'abord la mise en place d'un container contenant une image mariadb avec une database par défaut créée :

```
PS C:\wamp64\www\production\NodeJS\clc-api\api-nodejs> docker run -d --network dev-tools-network --name mariadb -e MARIADB_ALLOW_EMPTY_ROOT
_PASSWORD=1 -e MARIADB_DATABASE=default_db -p 3306:3306 mariadb
```



On peut aussi mettre en place le docker compose avec l'image de l'api directement



README

Quick NodeJS API with built-in mysql image

NodeJS 100% | Sequelize MySQL | MySQL MariaDB

easy to install restful api with members and users.

You can create an .env file if you prefer then run (leave empty for testing) :

```
docker-compose up -e MARIADB_ALLOW_EMPTY_ROOT_PASSWORD=1 -e MARIADB_DATABASE=default_db -d
```

Possible Env variables :

> Default credentials are root | null | default_db | 127.0.0.1

MARIADB_USER	MARIADB_ROOT_PASSWORD	MARIADB_DATABASE	MARIADB_HOST

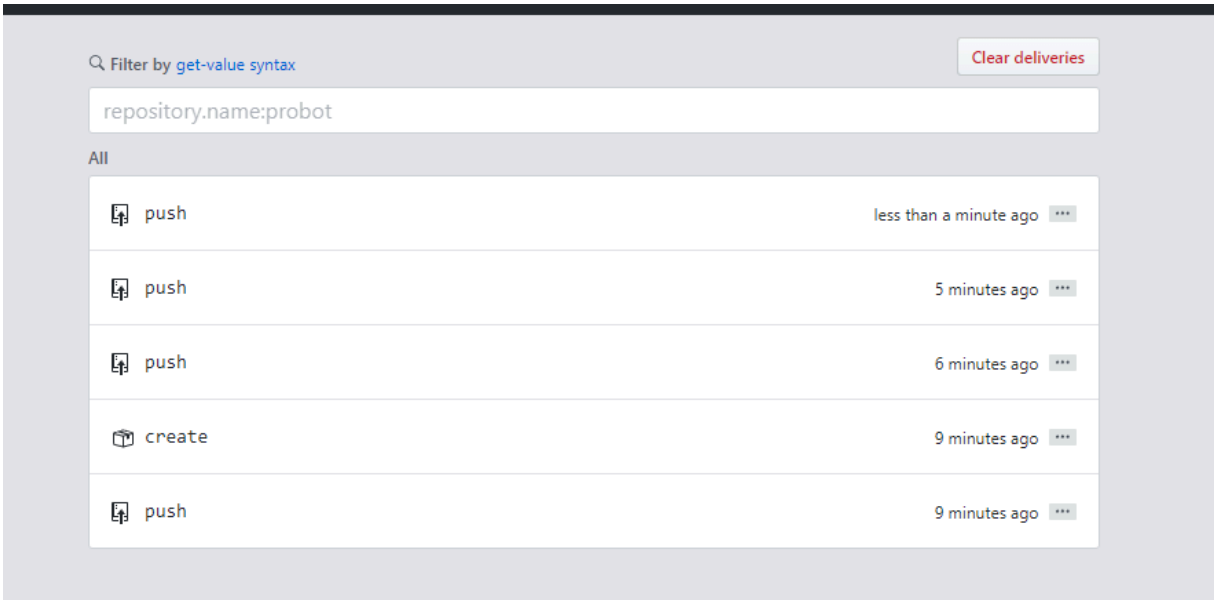
- L'application est bien mise en place :

```
[nodemon] restarting due to changes...
[nodemon] starting `node -r dotenv/config app.js`
Connexion to datbase established !
Server is running on http://localhost:5134
member was successfully created or updated
user was successfully created or updated
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Members' AND TABLE_
SCHEMA = NULL
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Users' AND TABLE_SC
HEMA = NULL
Executing (default): CREATE TABLE IF NOT EXISTS `Members` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `lastName` VARCHAR(2
55), `email` VARCHAR(255), `phone` VARCHAR(255), `url` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY
KEY (`id`)) ENGINE=InnoDB;
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `email` VARCHAR(255),
`password` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Members`
Executing (default): SHOW INDEX FROM `Users`
```

20. Le pipeline

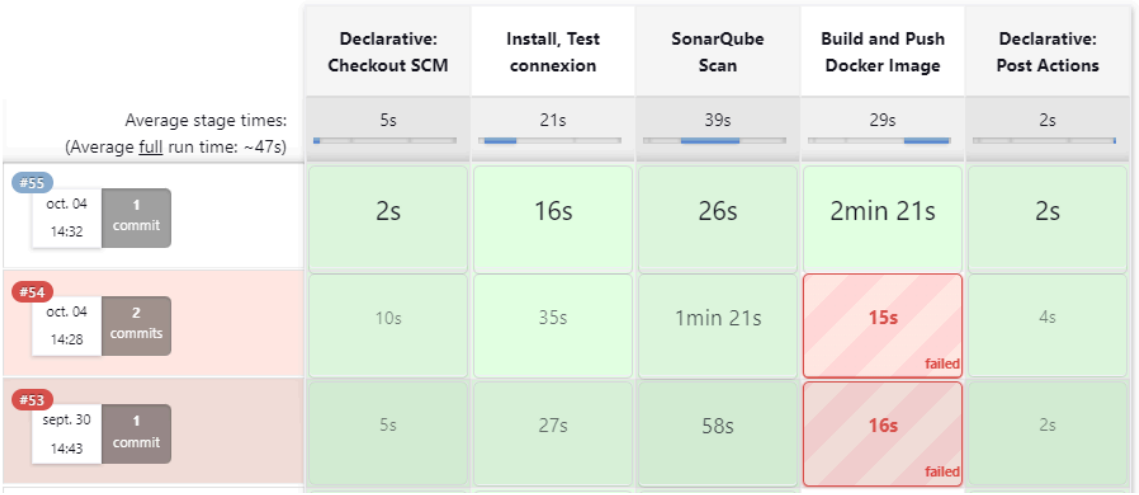
Maintenant que notre application est en 'production', nous allons déployer les changements dés lors qu'un push a été effectué à l'aide d'un Webhook et de Github et d'un Jenkinsfile. Comme nous travaillons en localhost, nous allons installer un proxy pour envoyer la data directement en local en utilisant la technologie Smee :

```
PS C:\wamp64\www\production\NodeJS\clc-api\api-nodejs> smee -u https://smee.io/6IsBYYxMGYV7u5m -t http://127.0.0.1:8080/github-webhook/
Forwarding https://smee.io/6IsBYYxMGYV7u5m to http://127.0.0.1:8080/github-webhook/
Connected https://smee.io/6IsBYYxMGYV7u5m
POST http://127.0.0.1:8080/github-webhook/ - 200
POST http://127.0.0.1:8080/github-webhook/ - 200
```



- **Résultats sur le serveur Jenkins**

```
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
using credential 38f1a338-e34a-4b47-96e6-6f2a1a083283
Fetching changes from the remote Git repository
Checking out Revision dff4e54e790ca3862dcef79f5afe2d28b4a24c30 (refs/remotes/origin/main)
Commit message: "Commit 04/09/20024"
 > git rev-parse --resolve-git-dir /home/jenkins/agent/workspace/node-api-pipeline/.git # timeout=10
 > git config remote.origin.url https://github.com/Manianise/api_nodejs # timeout=10
Fetching upstream changes from https://github.com/Manianise/api_nodejs
 > git --version # timeout=10
 > git --version # 'git version 2.39.2'
using GIT_ASKPASS to set credentials
 > git fetch --tags --force --progress -- https://github.com/Manianise/api_nodejs +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f dff4e54e790ca3862dcef79f5afe2d28b4a24c30 # timeout=10
 > git rev-list --no-walk 081a3042d62869fbdf7d93b27bb01d9617030656 # timeout=10
```



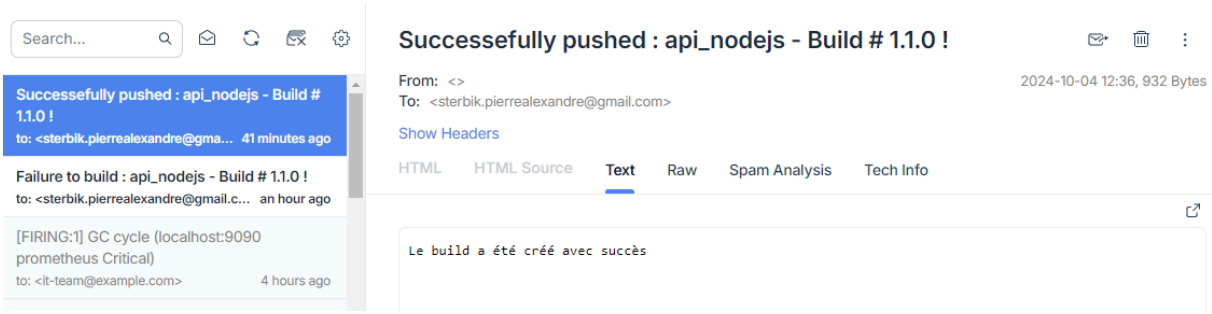
- On notera que dans les différents stages, on a le test de connexion à la database, le scan de Sonarqube qu l'on a configuré à l'aide d'un webhook (règles de bases des bonnes pratiques Javascript), le build docker afin de push l'image dans le docker Hub (à termine on créera

notre propre Registry pour l'entreprise, ainsi que la mise en place d'un système de mail. Le Jenkinsfile est écrit ci-dessous :

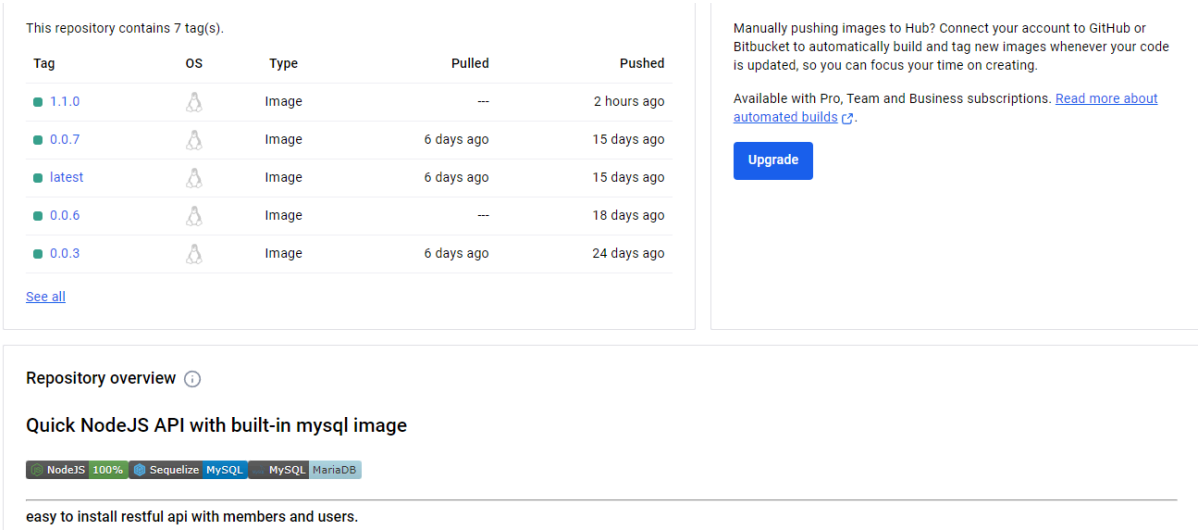
```
- pipeline {
-     agent {
-         label 'node-agent'
-     }
-
-     environment {
-         DOCKER_HUB_CREDENTIALS = credentials('jenkins-id')
-         SONARQUBE_SCANNER = tool name: 'SonarQube', type:
  'hudson.plugins.sonar.SonarRunnerInstallation'
-         SONAR_SCANNER_KEY = credentials('sonar-scanner')
-         DOCKERHUB_USERNAME = 'mechameleon'
-         DOCKER_IMAGE_NAME = 'api_nodejs'
-         ADMIN_MAIL = credentials('admin_mail')
-         LATEST_VERSION = sh(returnStdout: true, script: "git tag
  --sort version:refname |  sed 's/^v//' | tail -1").trim()
-     }
-
-     stages {
-
-         stage('Install, Test connexion') {
-             steps {
-                 script {
-                     sh '''
-                     export MARIADB_HOST=<HOST_IP>
-                     npm install
-                     npm run test
-                     nohup npm run build > output.log 2>&1 &
-                     '''
-                 }
-             }
-         }
-
-         stage('SonarQube Scan') {
-             steps {
-                 withSonarQubeEnv('SonarQube') {
-                     script {
-                         sh '''
-                         ${SONARQUBE_SCANNER}/bin/sonar-scanner \
-                         -Dsonar.projectKey=node-api-scan \
-                         -Dsonar.sources=. \
-                         -Dsonar.host.url=http://172.18.0.4:9000 \
-                         -Dsonar.login=${SONAR_SCANNER_KEY}
-                         '''
-                     }
-                 }
-             }
```

```
-          }
-
-          stage('Build and Push Docker Image') {
-              steps {
-                  script {
-
-                      sh '''
-                      docker login -u ${DOCKERHUB_USERNAME} -p
   ${DOCKER_HUB_CREDENTIALS}
-                      docker build . -t
   ${DOCKERHUB_USERNAME}/${DOCKER_IMAGE_NAME}:${LATEST_VERSION}
-                      docker push
   ${DOCKERHUB_USERNAME}/${DOCKER_IMAGE_NAME}:${LATEST_VERSION}
-                      '''
-
-                  }
-              }
-          }
-      }
-
-      post {
-          failure{
-              mail bcc: '',
-              body: 'Une erreur est survenue : Erreur dans le build',
-              cc: '', from: 'Jenkins pipelines ',
-              replyTo: '',
-              subject: "Failure to build : ${DOCKER_IMAGE_NAME} - Build
   # ${LATEST_VERSION} !",
-              to: "${ADMIN_MAIL}"
-            }
-          success{
-              mail bcc: '',
-              body: 'Le build a été créé avec succès', cc: '',
-              from: 'Jenkins pipelines',
-              replyTo: '',
-              subject: "Successefully pushed : ${DOCKER_IMAGE_NAME} -
   Build # ${LATEST_VERSION} !",
-              to: "${ADMIN_MAIL}"
-             }
-          }
-  }
-
-
```

- **La notification a bien été reçue (mailtrap) :**



- **L'image est bien mise à jour sur dockerhub**



VII. (Extra) : La supervision

21. Définir et mettre en place les statistiques de service

Afin de mettre en place les statistiques de services, les technologies suivantes ont été utilisées :

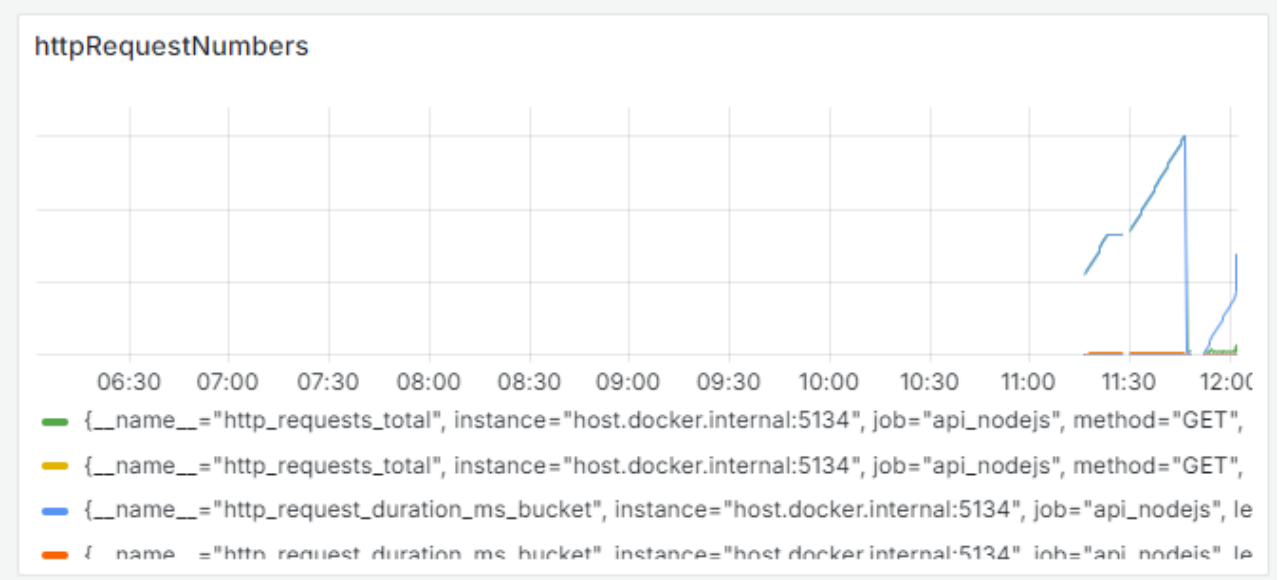
- Prometheus
  - Node Exporter
  - Alertmanager
- Grafana

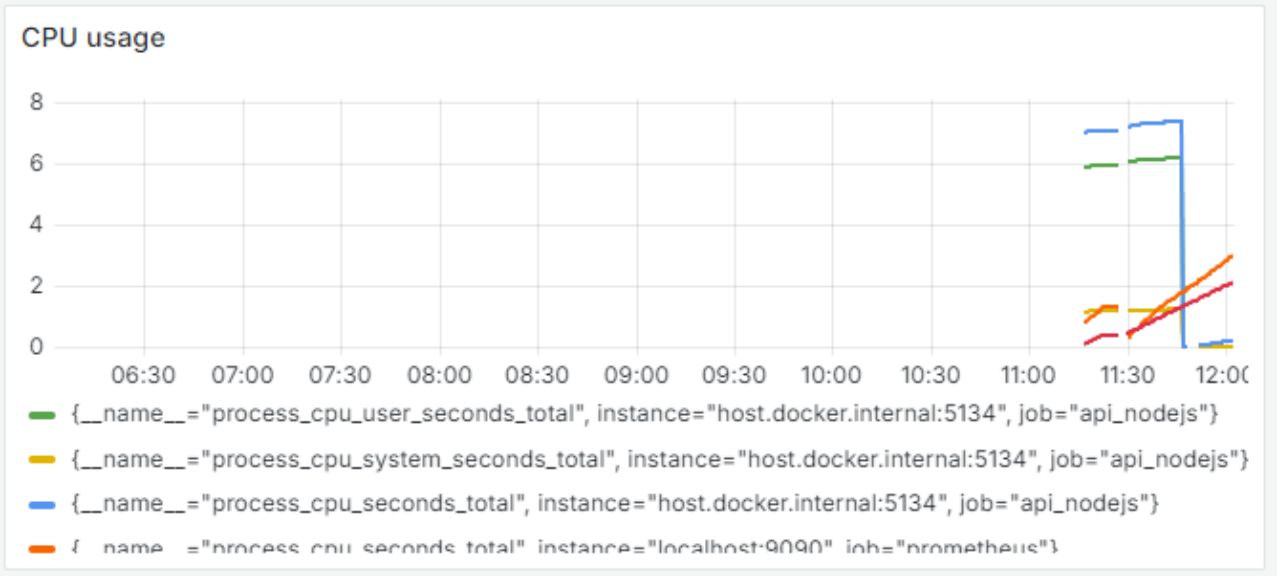
La configuration all in one peut être retrouvée ici :

- https://github.com/Manianise/ansible-prometheus-grafana

- **Les indicateurs choisis sont pertinents**

Comme le projet est une API, ce qui va être déterminent c'est la vitesse d'exécution des requêtes et les temps de réponses entre le serveur et les différents clients. Ainsi deux alertes ont été créées dans ce sens, une pour évaluer la qualité des échanges et la stabilité du serveur, l'autre pour déterminer l'utilisation des composants du serveur :





- **Ils tiennent compte des SLA :**

Le SLA indique qu'un utilisateur/client ne peut pas envoyer plus de 200 requêtes en une session. Une alerte a été créée pour prévenir lorsque la barre des 100 a été atteinte

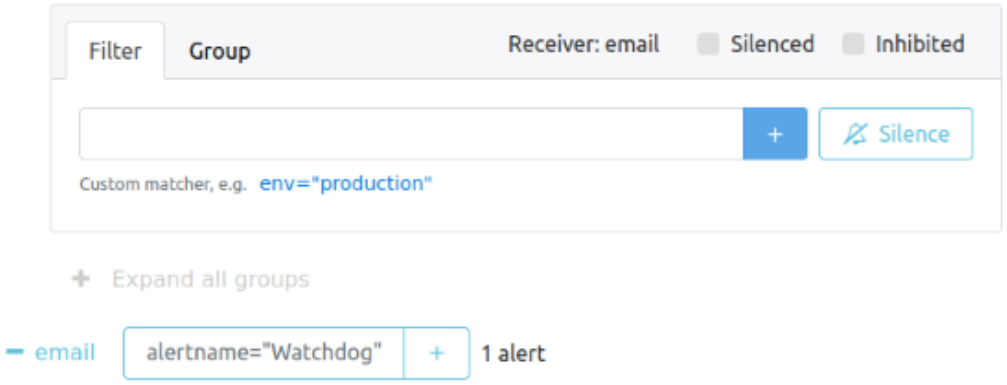
- **Ils sont exhaustifs :**

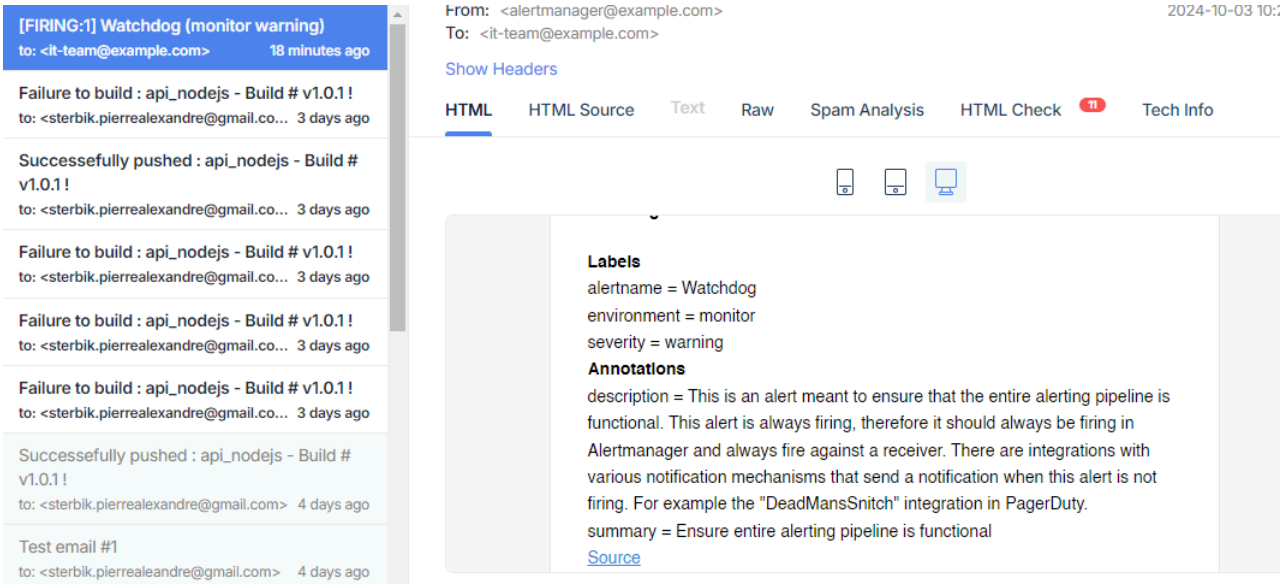
Dans l'exemple ci-dessus, on peut voir que tous les processus d'utilisation du CPU sont pris en compte : l'utilisateur, le système, et le total des deux. Les Dashboards créés sur Grafana sont tous agencé pour prendre en compte le maximum d'information pertinentes.

I. Exploiter une solution de supervision

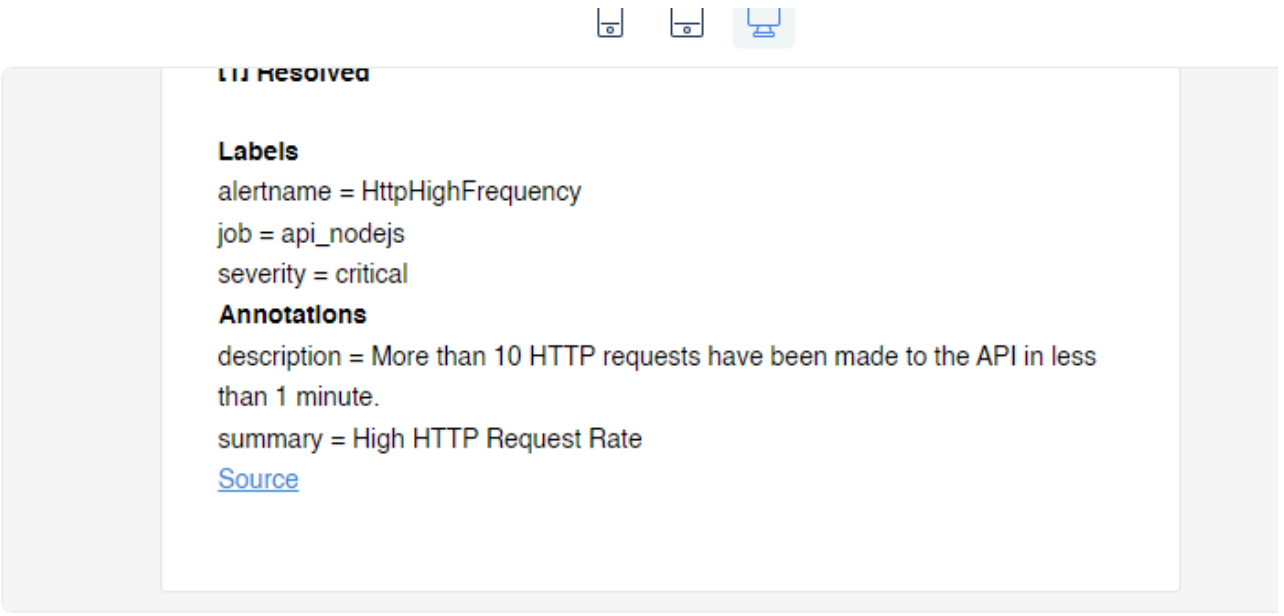
- **Les indicateurs sont pertinents :**

Une alerte pour vérifier que l'ensemble du système d'alerte fonctionne **:**





- **Exemple d'une alerte lorsque le threshold est atteint :**



VIII. Exemple de recherche effectuée

22. Choisir la bonne technologie

Plusieurs questions ont germé lors de la mise en place du projet :

- **Quelle manière de concevoir l'API ?**
- **De combien de ressources aura-t-on besoin ?**
- **Quel système de déploiement utiliser ?**
- **Quel système de stockage est pertinent ?**
- **Comment répondre à toutes ces questions tout en essayant d'avoir le plus petit budget possible ?**

L'API, dans l'ensemble, va reposer sur deux systèmes :

- Un système de stockage de noms/mots de passe et de toute information sensible qui doit être stockée en toute sécurité
- Un système de stockage éphémère sur toutes les informations publiques qui peuvent être librement diffusées sur Internet (dans le cadre du projet, les informations sur les biens de l'agence)

Pour la sécurité, voici ce que répondent les sondages et ce pourquoi la technologie a été utilisée :

2. MariaDB

- **Security**: MariaDB offers built-in data encryption at rest (tablespace encryption, log encryption), fine-grained access control, and support for SSL/TLS connections.
- **Resource Use**: Though it is a full relational database management system (RDBMS), MariaDB is optimized for performance and can be fine-tuned to run efficiently on low-resource systems.
- **Free**: MariaDB is an open-source fork of MySQL and is free to use.
- **Use Case**: Suitable for small to medium-sized applications that need more advanced database features than SQLite but still want to keep resource consumption low.

Pour la mise en place d'une base de donnée fiable et concurrencielle en terme de transfert de données, voici ce que donnent les recherches :

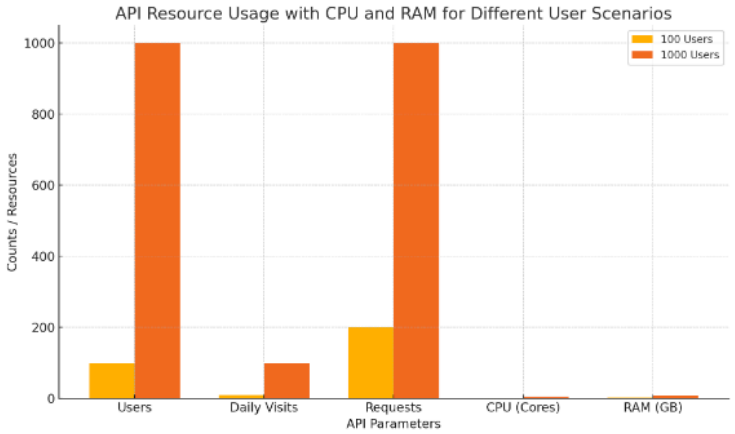
Comparison

Database	Type	Setup Complexity	Resource Use	Best Use Case
SQLite	Relational	Very Simple	Very Low	Simple APIs with low concurrency
TinyDB	Document	Very Simple	Very Low	Simple, file-based APIs
LevelDB	Key-Value	Simple	Low	APIs with key-value needs
BoltDB	Key-Value	Simple	Low	Go-based APIs
NeDB	Document	Simple	Low	Node.js APIs with small datasets
Redis	Key-Value	Moderate	Moderate	High-speed, in-memory needs
LowDB	Document	Simple	Low	Node.js-based APIs

Le choix se penchera donc sur LowDB, qui utilise peu de ressources, basée sur du documents et complètement adapté à une API NodeJS

Pour répondre à la question des ressources, il a été vu dans le cahier des charges que le projet serait scalable. Concrètement, l'API ne sera utilisée que par le cabinet en début de production, il n'y aura donc pas besoin dans l'immédiat de grande configuration, surtout si l'on sépare les serveurs en des serveurs plus petit comme sur AWS, qui offre aussi un nombre de services gratuits pendant un an. C'est pourquoi le choix d'utiliser AWS a été retenu. Facile d'utilisation, populaire et avec un nombre gigantesque de service, ils sont leaders du marché. Dans le cadre du projet à court terme, ce sera donc les images AWS Linux et des instances T3.micro.

Lorsque l'application complète sera terminée, en revanche, il faudra réfléchir à une plus grande infrastructure, puisqu'elle ne s'arrêtera pas simplement à une entreprise, mais à d'autres, ce qui augmentera le nombre de requêtes et donc l'utilisation des ressources du serveur. Voici l'étude menée pour estimer les ressources nécessaires selon le nombre d'utilisateurs (et de requêtes). Il est estimé que le nombre de cœurs et de RAM pour cent utilisateurs est d'environ 1 cœur pour 2gb RAM (minimum config), 4 cœurs pour 8GB RAM pour 1000 users etc…



Il sera alors impératif de passer à la technologie de clusters, pour éviter tout downtime en cas de panne de l'un des serveurs. Une technologie très connue, Kubernetes, sera alors employée.

IX. Conclusion

Le Delivery Manager, un atout de poids. Le choix des technologies de déploiement pour ce projet a été déterminé par les technologies vues en classe bien entendu !

Blague à part, la révolution que représente Docker comme système de déploiement et de facilité de transfert est simplement incontournable dans le monde IT moderne. La qualité de transferts sécurisés grâce aux différentes technologies comme OpenSSH, Ansible et Ansible-Galaxy rendent le développement et le déploiement d'applications hyper rapides et stables.

Certaines documentations sont plus difficiles à comprendre que d'autres. Dans le cas d'Ansible ou de Prometheus, il y a par certains côtés, une marge d'apprentissage difficile parce qu'il y a, pour configurer des infrastructures ou transférer des données ou encore organiser son code et les fichiers yaml, une multitude de manières différentes de parvenir à ce résultat, et tout le monde le fait un peu à sa manière. Donc parfois, lorsqu'on essaye de recouper les exemples des uns et des autres, on se retrouve noyé d'informations qui semblent contradictoires. La qualité de la documentation est pour moi la difficulté la plus haute à appréhender lorsqu'on s'intéresse à une technologie.

La quantité de savoir à accumuler pour n'arriver à faire que des petits pas a un côté décourageant. Mais c'est aussi la raison qui dicte une volonté sous-jacente et que je résumerai ainsi : dans la vie, rien de ce qui en vaut la peine ne s'obtient facilement. Alors on s'accroche.

Grâce à l'IA, les portes de l'informatique se sont ouvertes à un plus grand nombre d'acteurs encore, ce qui aboutira, j'en suis sûr, à une explosion d'idées favorables au monde et favorables à l'avenir. Travailler sur ce projet m'en a fait prendre conscience. Il m'a aussi fait prendre conscience à quel point la pérennité de la fourmilière qu'est l'informatique de notre monde repose sur les épaules de tant de fourmis ouvrières, qui œuvrent depuis maintenant plus de 50 ans à son bon fonctionnement et à faciliter la vie d'autrui.

La communauté informatique, vaste et ouverte, peut sembler dure et impitoyable de prime abord, surtout lorsqu'on est néophyte. Mais elle repose sur une vertu, une simple règle, qui m'a fait choisir cette ligne d'emploi :

Si tu travailles dur pour arriver à ce qu'on sait faire, si tu montres que tu fais tout ton possible pour creuser, alors nous t'aiderons du mieux que nous pouvons. Ce savoir, qui permet de contribuer à l'avancée humaine, te sera alors complètement accessible, et dans la majorité des cas, gratuite, car chacun a mis une pierre à cet édifice, de sa volonté, pour l'avenir de tous.

La route est longue, et ce projet est une brique que je taille pour combler une aile de plus.

- ANNEXE 1 : EXEMPLE DE DEPLOIEMENT RAPIDE AVEC VAGRANT

- Tree

```
.
├── README.md
├── Vagrantfile
└── ansible-prometheus-grafana
    ├── README.md
    ├── ansible.cfg
    ├── group_vars
    │   └── all
    │       └── vault.yml
    ├── playbook.yml
    └── requirements.yml
```

- Documentation

README

All in one compilation to install Grafana, Prometheus, Alertmanager cAdvisor and Node Exporter on a Linux based machine

ubuntu v22.04.4 grafana latest prometheus latest

Works on Debian/Ubuntu, untested elsewhere

Before we begin

- This playbook is set to install Prometheus and Grafana on a local machine. If you wish to use the version v0.0.1, witch has an inventory, by running the following command :

```
git checkout tag/v0.0.1
```

Requirements

- Ansible installed on local machine
- Grafana Account with credentials in group_vars/all/vault.yml

Linux

- Replace vault.yml with your secret variables e.g Grafana credentials as `grafana_user` and `grafana_password`
- It is recommended to encrypt your vault.yml
- If you do not encrypt your vault.yml, add a semi-colon next to ask vault pass in ansible.cfg
- Run the following commands :

```
ansible-galaxy install -r requirements.yml
ansible-playbook -i inventory.ini playbook.yml
```

With no vault

```
ansible-galaxy install -r requirements.yml
ansible-playbook playbook.yml
```

Windows

- You will first need to configure WSL environment
- Once done, enter your distribution and follow the instructions (Linux) :

- **Code**

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

Vagrant.configure("2") do |config|

  config.vm.box = "generic/ubuntu2204"

  if Vagrant.has_plugin?("vagrant-vbguest")
    config.vbguest.auto_update = false
  end

  # config monitoring server

  config.vm.define "monitor", primary:true do |monitor|

    monitor.vm.hostname = "monitor"
    monitor.vm.synced_folder "./ansible-prometheus-grafana", "/vagrant", type: "smb"
    monitor.vm.provision "ansible_local" do |ansible|
      ansible.provisioning_path = "/vagrant"
      ansible.playbook = "playbook.yml"
      ansible.galaxy_command = "sudo ansible-galaxy install -r requirements.yml"
    end

    monitor.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get upgrade
    sudo apt install -yf ubuntu-desktop-minimal
    sudo snap install firefox
    sudo reboot
  SHELL

  end

  # config api server

  config.vm.define "client" do |client|

    client.vm.hostname = "client"
    client.vm.provision "shell", inline: <<-SHELL
      sudo apt-get update
      sudo apt-get upgrade
      sudo apt install -y mariadb-server
      sudo systemctl start mariadb
      sudo systemctl enable mariadb
      sudo mysql_secure_installation <<EOF
```

```
Y
n
Y
Y
Y
EOF
        sudo mysql -u root -e "SET PASSWORD FOR 'root'@'localhost' = PASSWORD('');"
        sudo mysql -u root -e "FLUSH PRIVILEGES;"
        sudo mariadb -u root -e "CREATE DATABASE default_db;"
        sudo curl -fsSL https://fnm.vercel.app/install | bash
        sudo source ~/.bashrc
        sudo fnm use --install-if-missing 22
        sudo git clone https://github.com/Manianise/api_nodejs.git
        sudo chown -hR vagrant:vagrant api_nodejs/
        sudo apt install -yf ubuntu-desktop-minimal
        sudo snap install firefox
        sudo reboot
      SHELL

  end


  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # NOTE: This will enable public access to the opened port
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine and only allow access
  # via 127.0.0.1 to disable public access
  # config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  # config.vm.synced_folder "../data", "/vagrant_data"
```

```
  # Disable the default share of the current code directory. Doing this
  # provides improved isolation between the vagrant box and your host
  # by making sure your Vagrantfile isn't accessible to the vagrant box.
  # If you use this you may want to enable additional shared subfolders as
  # shown above.
  # config.vm.synced_folder ".", "/vagrant", disabled: true

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  # config.vm.provider "virtualbox" do |vb|
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
  #   vb.memory = "1024"
  # end
  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Enable provisioning with a shell script. Additional provisioners such as
  # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
  # documentation for more information about their specific syntax and use.
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

- ## ANNEXE 2 : CREATION DE SERVEURS LINUX

- Serveur SSH pour l'accès distant sécurisé.
- Serveur Web Apache pour l'hebergement de sites web. Héberger un site web de votre choix où vous utiliserez le site web fournit.
- Serveur FTP (vsftpd) pour le transfert de fichiers.
- GLPI pour la gestion des actifs informatiques et du service d'assistance.
- Zabbix pour la surveillance du réseau et des serveurs.
- Serveur DHCP pour attribuer automatiquement des adresses IP aux clients
- Serveur DNS pour la résolution des noms de domaine.

Configuration SSH :

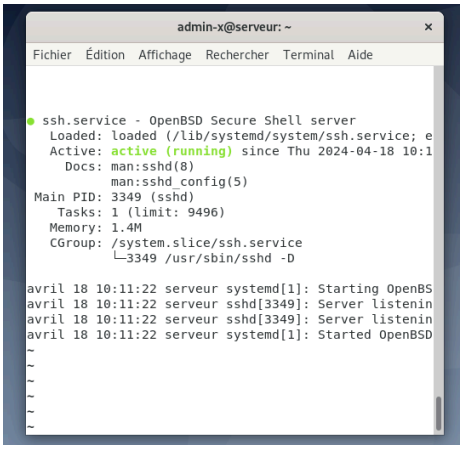
- Installer le serveur SSH en utilisant la commande

sudo apt-get install openssh-server.

- Vérifier le statut du service SSH en utilisant

sudo systemctl status ssh.

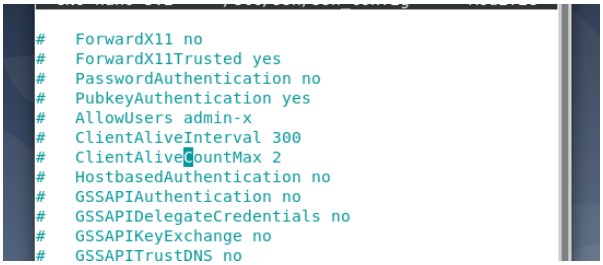
- Configurer SSH pour une sécurité renforcée en éditant le fichier de configuration /etc/ssh/sshd_config.



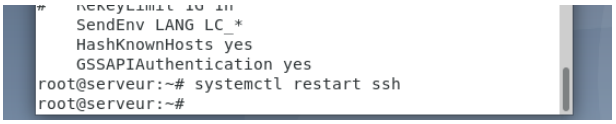
Pour ce qui est de renforcer le fichier conf :

1. **Disable root login**:
   - Change the line `**PermitRootLogin yes**` to `**PermitRootLogin no**` to prevent direct root logins.
2. **Use SSH key-based authentication**:
   - Ensure that `**PubkeyAuthentication**` is set to `**yes**` to allow SSH key-based authentication.
3. **Disable password authentication** (if using SSH keys):
   - Set `**PasswordAuthentication no**` to disallow password-based authentication.
4. **Restrict SSH protocol versions**:
   - Use `**Protocol 2**` to enforce SSH protocol version 2, which is more secure than SSH protocol version 1.

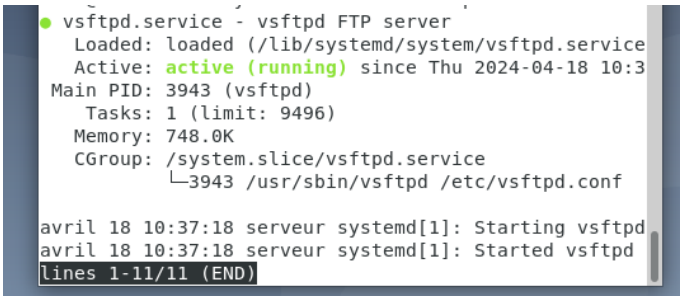
Using nano /etc/ssh/ssh_config



Restart ssh :



Installation et verification du service vsftpd :

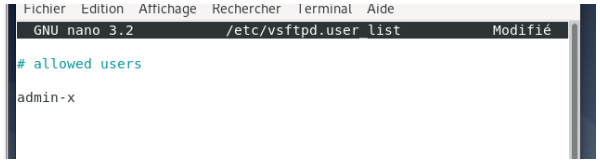


Le sécuriser :

- Débloquer le userlist :

```
#
# Allow userlist
userlist_enable=YES
userlist_file=/etc/vsftpd.user_file
userlist_deny=NO
#
```

- Dans vsftpd.user_list :



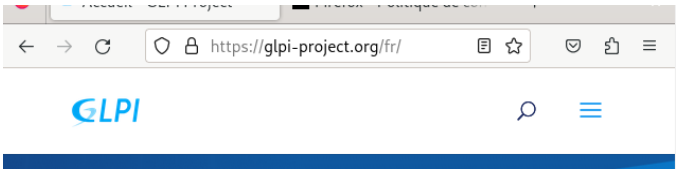
- Empêcher l'accès anonyme :

```
#
# Allow anonymous FTP? (Disabled by default).
anonymous_enable=NO
```

- Restart le service :

Systemctl reload vsftpd.service

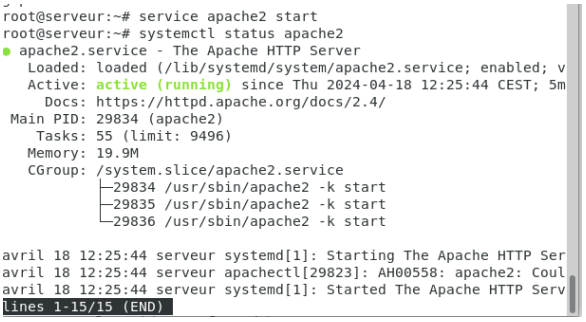
Installation GLPI :



Pour installer et configurer GLPI, il y a plusieurs pré-requis :

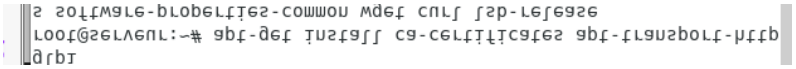
- Installer un web server apache2 et le lancer :

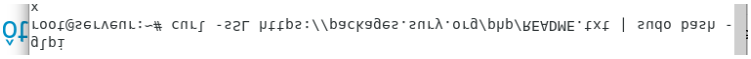
Apt install apache2



- **Installer PHP**

Apt-get update





- **On va installer plusieurs versions de PHP au cas où :**

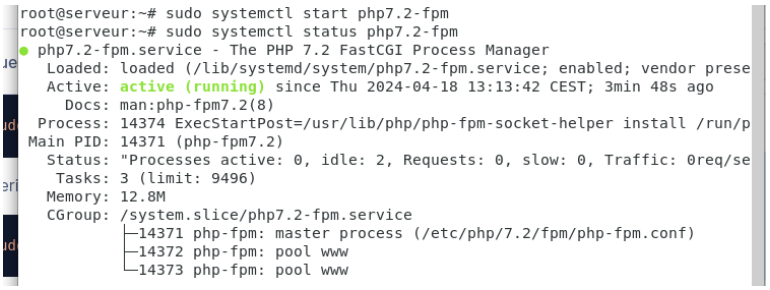
sudo apt-get install software-properties-common -y

sudo add-apt-repository ppa:ondrej/php

sudo apt-get update -y

sudo apt-get install php7.0 php7.0-fpm php7.0-mysql libapache2-mod-php7.0 libapache2-mod-fcgid -y

- Puis on démarre le service php



- Pour continuer à configurer le serveur , il faut qu'on ajoute les paquets UFW pour le firewall, tout en y ajoutant quelques modifications :

apt install ufw

- Setting default policies

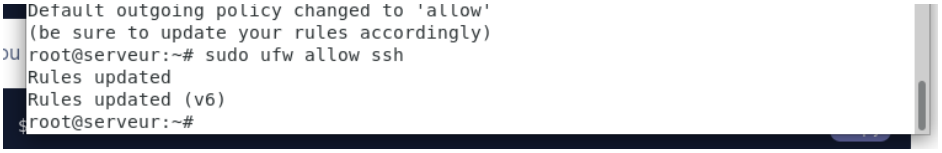
Sudo ufw default deny incoming

Sudo ufw default allow outgoing

- Allow SSH connection and openssh

sudo ufw allow ssh

Sudo ufw allow 'WWW'



Sudo ufw start, status, disable, enable et devrait donner un résultat comme :

Additionnal commands (pas utilisées dans l'exercice, mais l'exercice sert également de pense-bête) :

sudo ufw status verbose

sudo ufw allow http

sudo ufw allow 6000:6007/tcp

sudo ufw allow 6000:6007/udp

sudo ufw allow from 203.0.113.0/24 to any port 22

ip addr

sudo ufw allow in on eth0 to any port 80

hostname -I : connaître l'ip de son serveur

- **Maintenant que tout est prêt, on va set notre domaine sur le serveur apache :**

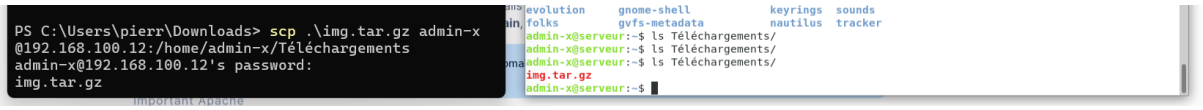
sudo mkdir -p /var/www/your_domain

sudo chown -R $USER:$USER /var/www/your_domain

- **Et y installer notre site web que j'ai transféré de ma machine physique à ma machine virtuelle en utilisant le protocole SSH par masochisme**

Ssh-keygen

Ssh IP_server (suivre les steps, entrer la passphrase, la confirmer)



- **Extraire dans notre serveur Apache, supprimer le tar**

```
admin-x@serveur:/var/www/esprit-prof.com$ tar -jxvf ~/Téléchargements/img.tar.bz2
img/
img/devops-recherche-stage-grey.webp
index.html
favicon.ico
admin-x@serveur:/var/www/esprit-prof.com$ ls
favicon.ico  img  index.html
admin-x@serveur:/var/www/esprit-prof.com$ cd ~
admin-x@serveur:~$ ls
Bureau  Documents  Images  Modèles  Musique  Public  Téléchargements  Vidéos
admin-x@serveur:~$ rm Téléchargements/img.tar.bz2
admin-x@serveur:~$
```

- **Faire en sorte que notre domaine soit le défault du port 80 en créant un autre fichier que dans le default :**

sudo nano /etc/apache2/sites-available/esprit-prof.conf

Et dans le .conf esprit-prof :

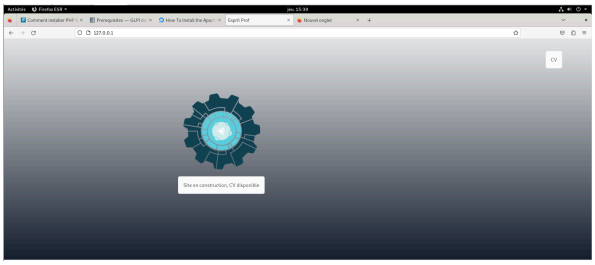
```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName esprit-prof
    ServerAlias www.esprit-prof
    DocumentRoot /var/www/esprit-prof
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- Retirer le default

sudo a2ensite esprit-prof.conf

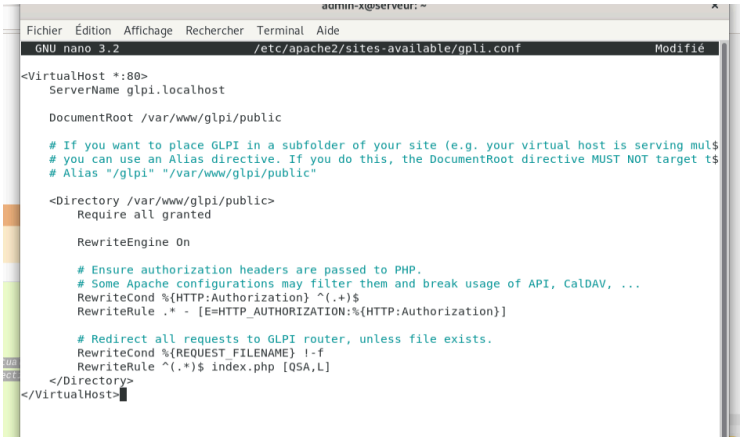
sudo a2dissite 000-default.conf

- Success !!

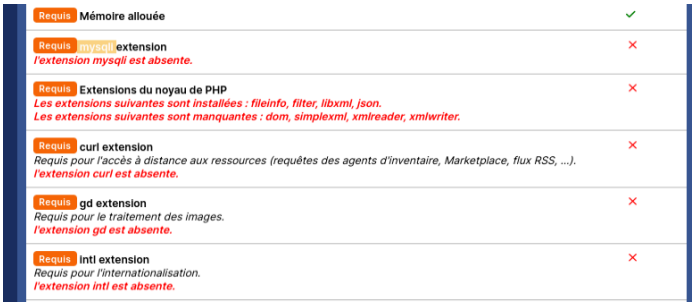


On va faire un autre domaine pour installer GLPI après avoir déplacé le dossier dans le /var/www et on le mettra sur le port 81 puisqu'on a déjà un autre site sur le port 80

sudo nano /etc/apache2/sites-available/glpi.conf



- Et on l'enable et on reload mais pas avant d'avoir enable le mod_rewrite
- Temporairement donner les accès à www-data
- Enable les extensions suivantes : mysqli
- Dom, simplexml, xmlreader, xmlwriter, curl, intl
- Mettre le session.cookie_httponly = ON dans le php.ini dans le /etc/php/8.3/apache2




```
# Save existing php package list to packages.txt file
sudo dpkg -l | grep php | tee packages.txt

# Add Ondrej's repo source and signing key along with dependencies
sudo apt install apt-transport-https
sudo curl -sSLo /usr/share/keyrings/deb.sury.org-php.gpg https://packages.sury
sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/deb.sury.org-php.gpg] htt
sudo apt update

# Install new PHP 8.3 packages
sudo apt install php8.3 php8.3-cli php8.3-{bz2,curl,mbstring,intl}

# Install FPM OR Apache module
sudo apt install php8.3-fpm
# OR
# sudo apt install libapache2-mod-php8.3

# On Apache: Enable PHP 8.3 FPM
sudo a2enconf php8.3-fpm
# When upgrading from an older PHP version:
sudo a2disconf php8.2-fpm

# Remove old packages
sudo apt purge php8.2*
```

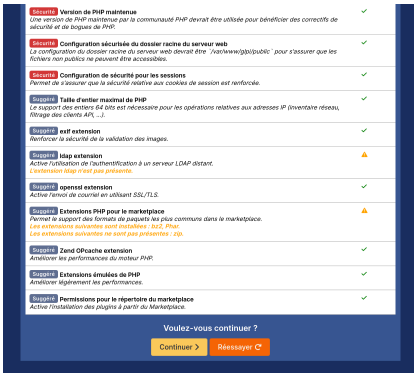
sudo apt install php8.3 php8.3-cli php8.3-{bz2, curl,mbstring, intl,gd,mysqli,simplexml,xmlreader,xmlwriter}

Sudo a2enmod rewrite

Sudo chown -R www-data /var/www/glpi

sudo a2ensite dlpi.conf ; sudo systemclt reload apache2

- Ce qui donne quelque chose comme ceci :



- On remet les droits au admin-x

Sudo chown -R admin-x /var/www/glpi

- Pour pouvoir avancer, il va falloir installer mySQL

sudo apt install gnupg

mkdir tmp ; cd /tmp (or any dir to write in)

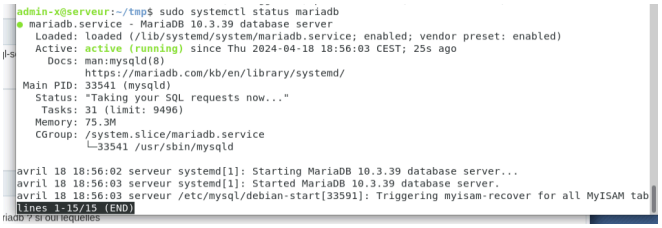
wget https://dev.mysql.com/get/mysql-apt-config_0.8.22-1_all.deb

sudo dpkg -i mysql-apt-config*

sudo apt update

sudo apt install mariadb-server (mysql-server obsolete)

sudo systemctl status mariadb



Sudo mysql_secure_installation

Configuration (optionnelle) :

sudo mysql

GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED BY 'password' WITH GRANT OPTION;

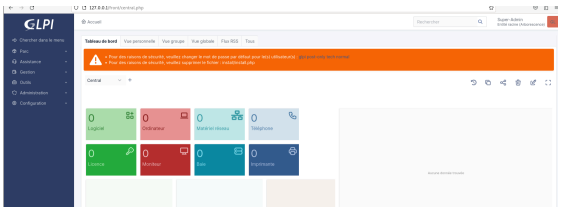
FLUSH PRIVILEGES;

exit

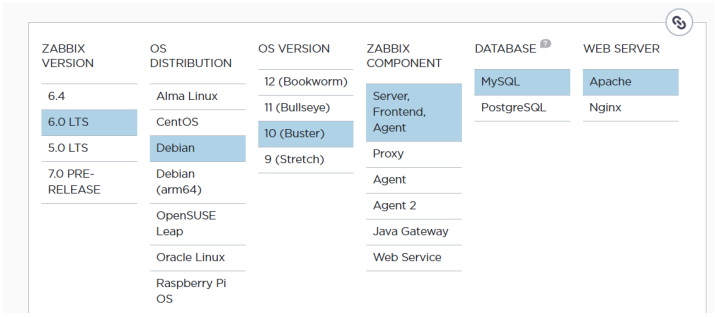
- On va ensuite lier GLPI et mariadb

- Si on obtient un « permission denied », on peut flush les privilèges et reset un password pour root@localhost jusqu'à se connecter :

- J'ai créé une table_test base de donnée



- On va maintenant installer Zabbix en supposant que ce qui suit est la bonne configuration :

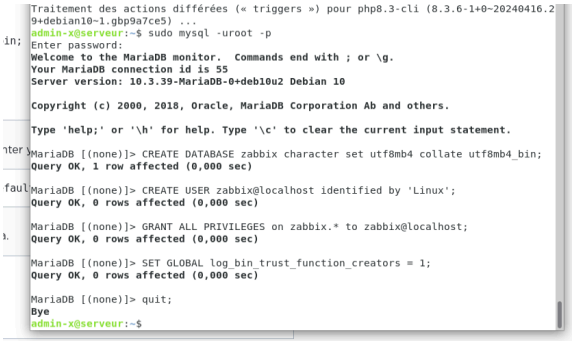


wget https://repo.zabbix.com/zabbix/6.0/debian/pool/main/z/zabbix-release/zabbix-release_6.0-4+debian10_all.deb
dpkg -i zabbix-release_6.0-4+debian10_all.deb
apt update

apt install zabbix-server-mysql zabbix-frontend-php zabbix-apache-conf zabbix-sql-scripts zabbix-agent

mysql -uroot -p
password
mysql> create database zabbix character set utf8mb4 collate utf8mb4_bin;
mysql> create user zabbix@localhost identified by 'password';
mysql> grant all privileges on zabbix.* to zabbix@localhost;

mysql> set global log_bin_trust_function_creators = 1;
mysql> quit;

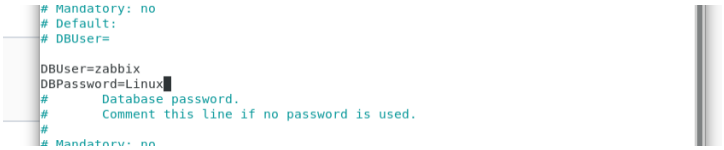


zcat /usr/share/zabbix-sql-scripts/mysql/server.sql.gz | mysql --default-character-set=utf8mb4 -uzabbix -p zabbix

mysql -uroot -p
password
mysql> set global log_bin_trust_function_creators = 0;
mysql> quit;

- Editer /etc/zabbix/zabbix_server.conf

Sudo nano /et/zabbix/zabbix_server.conf DBPassword=password



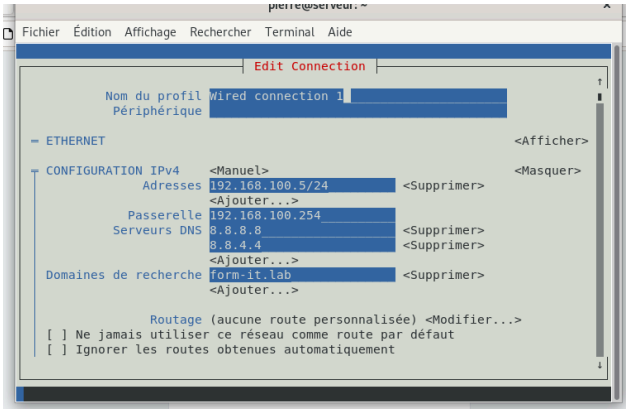
systemctl restart zabbix-server zabbix-agent apache2
systemctl enable zabbix-server zabbix-agent apache2

- Et on le lance sur http://localhost/zabbix puis on configure son compte zabbix

- Première connexion : ID : Admin PWD : zabbix
- Installer le DHCP

- Configurer notre /etc/network/interfaces : ens33 comme suit :

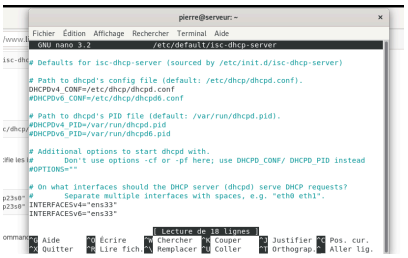
Sudo nmtui



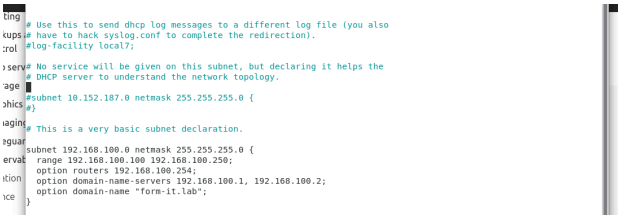
Sudo apt intall isc-server-dhcp

- On vérifie que dans le isc-server-dhcp notre netadapter est bien celui prit en compte

Sudo nano /etc /default/isc-server-dhcp



- Puis dans /etc/dhcp/dhcpd.conf on va créer notre dhcp conformément à l'ip de notre netadapter (on décommente **authoritative** pour que ce soit le dhcp principal de notre réseau)

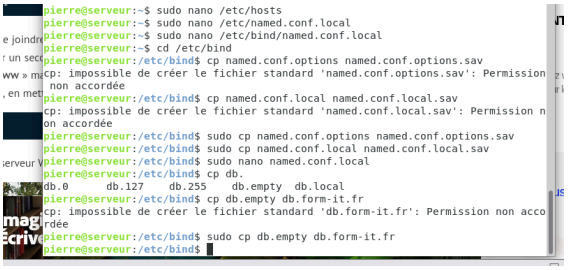


- On vérifie s'il est en service :

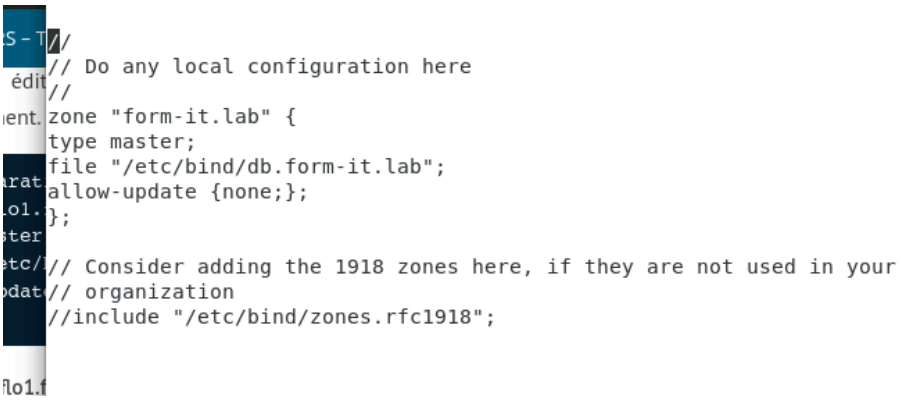
- Il ne reste plus qu'à configurer le DNS

Sudo apt-get install bind 9

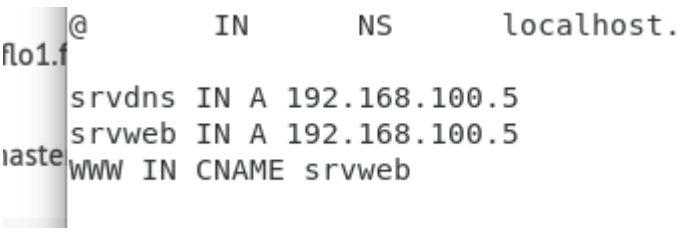
- Préparer les fichiers



Sudo nano /etc/bind/named.conf.local



Sudo nano /etc/bind/db.form-it.lab



- On check si ca a fonctionné :

```
srvdns IN A 192.168.100.5
srvweb IN A 192.168.100.5
WWW IN CNAME srvweb
pierre@serveur:/etc/bind$ sudo systemctl restart bind9
pierre@serveur:/etc/bind$ sudo named-checkzone form-it.lab db.form-it.lab
zone form-it.lab/IN: loaded serial 1
OK
pierre@serveur:/etc/bind$
```

- On configure le resolve.conf

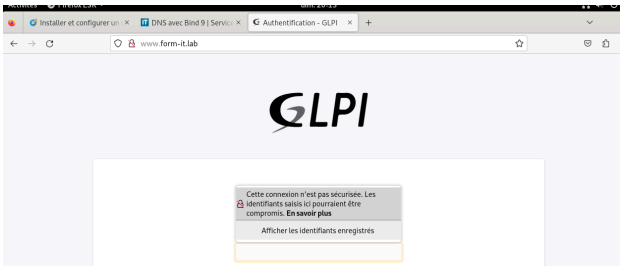
```
pierre@serveur:/etc/bind$ sudo nano /etc/
Display all 210 possibilities? (y or n)
pierre@serveur:/etc/bind$ sudo nano /etc/re
reportbug.conf  resolv.conf
pierre@serveur:/etc/bind$ sudo nano /etc/resolv.conf
pierre@serveur:/etc/bind$ /etc/init.d/network
networking       network-manager
pierre@serveur:/etc/bind$ /etc/init.d/networking restart
[ ok ] Restarting networking (via systemctl): networking.service.
pierre@serveur:/etc/bind$
```

```
pierre@serveur:/etc/bind$ nslookup srvdns.form-it.lab
Server:         192.168.100.5
Address:        192.168.100.5#53

Name:   srvdns.form-it.lab
Address: 192.168.100.5

pierre@serveur:/etc/bind$
```

Success !



- On configure la zone inversée :

```
//
// Do any local configuration here
// Zone principale
zone "form-it.lab" {
type master;
file "/etc/bind/db.form-it.lab";
allow-update {none;};
};

// Zone inversée

zone "1.168.192.in-addr.arpa" {
type master;
file "db.inverse.form-it.lab";
allow-update { none; };
};

// Consider adding the 1918 zones here, if they are not used in your
```

- Dans le fichier recréé db.invers.form-it.lab :

```
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                          86400 )       ; Negative Cache TTL
;
@       IN      NS      localhost.

100 IN PTR srvdns
100 IN PTR srvweb
```

- Redemarrez le service DNS pour appliquer les modifications.

Sudo systemctl restart bind9