

République Du Sénégal  
Un Peuple – Un But – Une Foi



Ministère de l'Enseignement Supérieur et de la Recherche

Université Alioune Diop de Bamby

UFR : SATIC

Spécialité : SI / SR

### THEMATIQUE:

## Automatisation CI/CD Microservices



Encadrant: Mr. Diop

Présente par

Mamadou Absa GUEYE

Mor Laye Cisse

Mourtalla Fall

Ndeye Rokhaya Sylla

Mourtalla Gueye

Année Académique

2026

Contexte et problématique

Dans le cadre de notre formation en Master 2 Systèmes d'Information et Systèmes Réseaux, nous avons été amenés à concevoir et mettre en œuvre une infrastructure DevOps complète. Le projet consiste à développer une application de gestion d'événements baptisée "Magal", en adoptant une approche moderne basée sur les micro services et l'automatisation des processus de développement et de déploiement.

La problématique centrale de ce projet repose sur plusieurs défis techniques :

- Comment garantir la qualité et la fiabilité du code à chaque modification ?
- Comment automatiser le déploiement sans intervention manuelle ?
- Comment assurer la sécurité des composants applicatifs ?
- Comment orchestrer efficacement plusieurs services interdépendants ?

Technologies et outils utilisés

Notre stack technologique s'articule autour des éléments suivants :

#### **Développement applicatif :**

- Backend : Django (Framework Python pour API REST)
- Frontend : React (Bibliothèque JavaScript pour interface utilisateur)
- Base de données : PostgreSQL

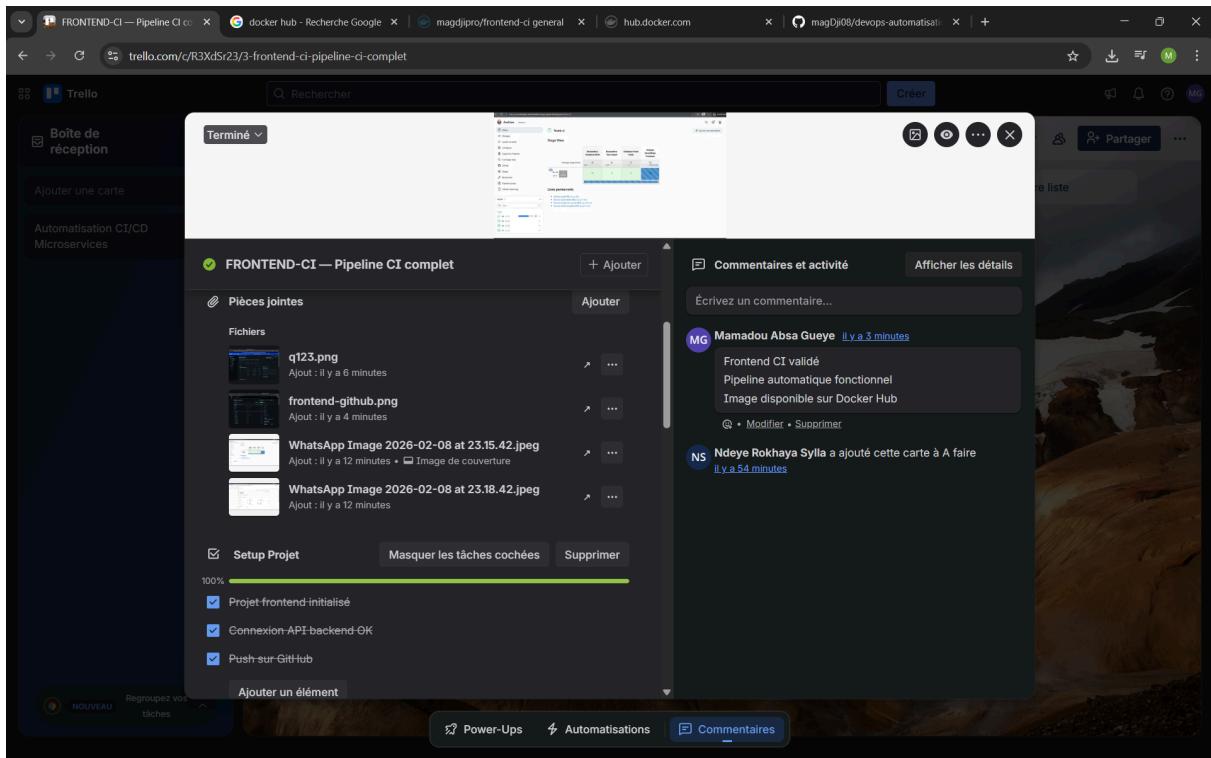
## **1. Organisation du projet avec Trello (Étape 1)**

Pour ce projet, nous avons adopté la méthodologie Agile. Un board Trello a été mis en place pour segmenter les tâches.

**Découpage :** Crédit de cartes pour la CI Backend, CI Frontend, le serveur SonarQube et le déploiement CD.

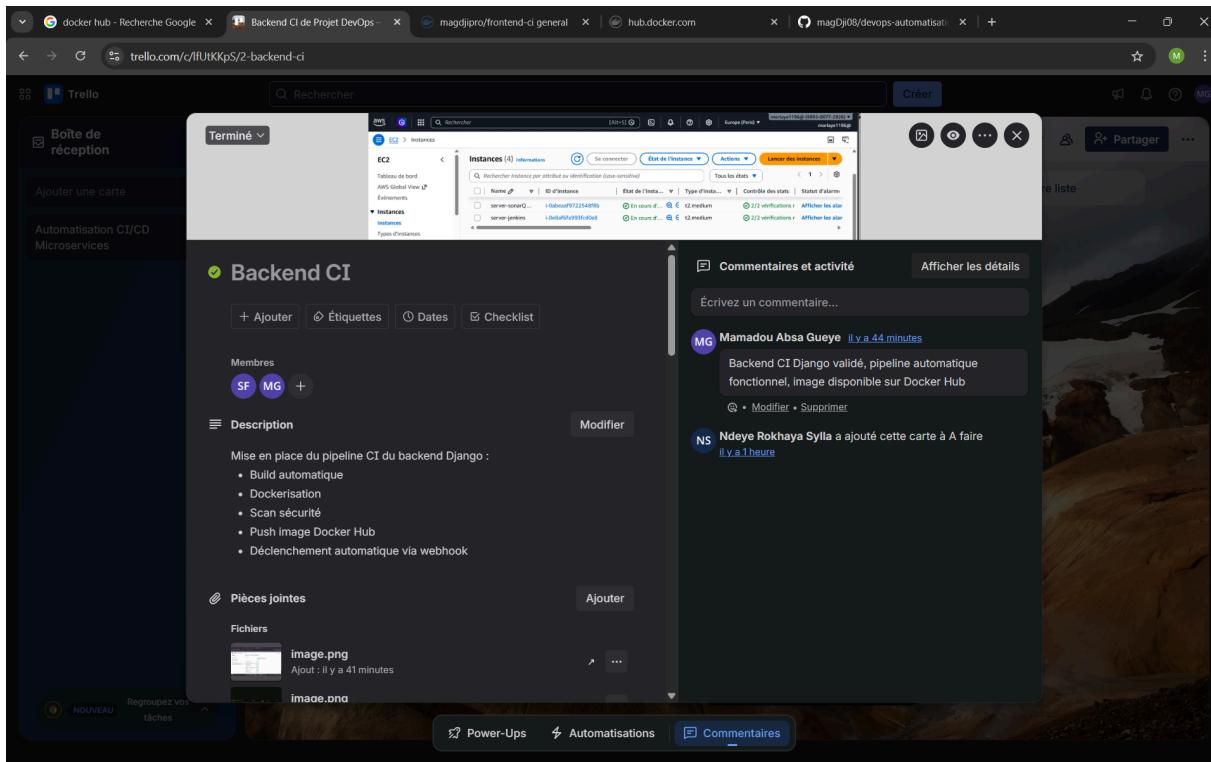
**Collaboration :** Attribution des rôles pour une visibilité totale de l'avancement.

 **CAPTURE :** Vue d'ensemble **tout ce qui doit apparaître dans la partie "Frontend CI" de mon board Trello.**



accessible via le lien: <https://trello.com/c/R3XdSr23/3-frontend-ci-pipeline-ci-complet> pour plus de clarté .

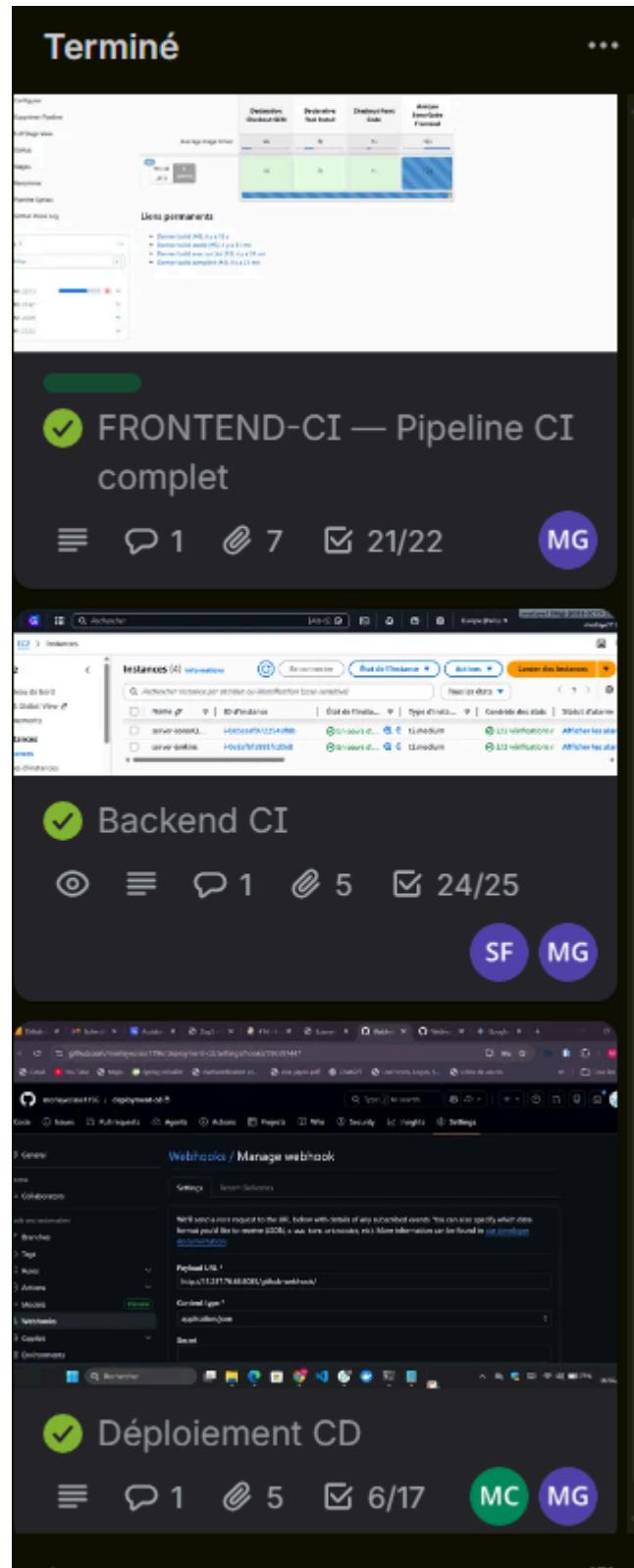
**CAPTURE :** Vue d'ensemble de tout ce qui doit apparaître dans la partie "backend CI" de mon board Trello.



accessible via le lien: <https://trello.com/c/lfUtKKpS/2-backend-ci> pour plus de clarté .



**CAPTURE :** Vue d'ensemble mon board Trello.



📸 **CAPTURE** : Vue d'ensemble via: <https://trello.com/b/g9YSOhvq/projet-devops-master-2-si-sr-uadb>

## 2.1 Initialisation Locale et Git

L'application se compose d'un backend Django et d'un frontend React. Pour les ajouter sur GitHub depuis le local :

1. `git init` dans chaque dossier.
2. `git remote add origin [URL_REPO]`.
3. `git add . && git commit -m "Initial commit"`.
4. `git push -u origin main`.

## 2.2 Gestion des Branches (Workflow d'équipe)

Nous avons instauré une branche par développeur pour isoler les fonctionnalités :

- **Branche `frontend-dev`** : Gérée par Mamadou Absa Gueye.
- **Branche `backend-dev`** : Gérée par Mourtada.
- **Branche `deployment-cd`** : Gérée par Morlaye Cisse pour l'orchestration finale.

The screenshot shows the GitHub 'Branches' page for the repository 'morlayecisse1196 / devops-projet'. The 'Default' branch is 'main', which was updated '1 minute ago'. The 'Your branches' section lists three developer branches: 'deployment-cd', 'backend-dev', and 'frontend-dev', all of which were updated 'now'. A green 'New branch' button is visible in the top right corner of the search bar area.

Branch	Updated	Check status	Behind	Ahead	Pull request
main	1 minute ago	Default	0	0	...

Branch	Updated	Check status	Behind	Ahead	Pull request
deployment-cd	now	0   0	0	0	...
backend-dev	now	0   0	0	0	...
frontend-dev	now	0   0	0	0	...

## 3. Infrastructure Cloud AWS & Services

Nous avons provisionné deux instances **Ubuntu 24.04** sur AWS EC2.

The screenshot shows the AWS Management Console with the EC2 service selected. The Instances page displays two instances:

Name	ID d'instance	État de l'instance	Type d'instance	Contrôle des statut	Statut d'alarme
server-sonarQ...	i-0abeaa9722548f8b	En cours d'...	t2.medium	2/2 vérifications r	Afficher les alar
server-jenkins	i-0e8af6fa993fc0e8	En cours d'...	t2.medium	2/2 vérifications r	Afficher les alar

## 3.1 Accès et Configuration Docker

L'accès se fait par SSH : `ssh -i "votre_cle.pem" ubuntu@[IP_PUBLIQUE]`.

Sur chaque instance, Docker a été installé pour conteneuriser nos services.

Accéder à distance au server-sonar ou server-jenkins grâce à leurs adresse IP publique :

```
PS C:\Users\USER\Desktop\DOC\CLOUD\DEVOPS\PROJET\DEMO-AWS> ssh -i .\demo-key.pem ubuntu@13.38.54.58
```

## 3.2 Installation de SonarQube et Jenkins

- SonarQube** : Déployé via un conteneur Docker sur le port 9000 pour l'analyse de la qualité du code.

Après installation dans docker :

```
ubuntu@ip-172-31-6-29:~$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
39b7c62ae45f sonarqube:9.9-community "/opt/sonarqube/dock..." 2 days ago Up 44 hours 0.0.0.0:9000->9000/tcp, [::]:9000->9000/
tcp sonarqube-sonarqube-1
56d0f6a79782 postgres:14 "docker-entrypoint.s..." 2 days ago Up 44 hours 0.0.0.0:5432->5432/tcp, [::]:5432->5432/
tcp sonarqube-sonar_db-1
ubuntu@ip-172-31-6-29:~$
```

- Création d'un projet SonarQube:

All fields marked with \* are required

**Project display name \***

Up to 255 characters. Some scanners might override the value you provide.

**Project key \***

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '\_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

**Main branch name \***

The name of your project's default branch [Learn More](#)

**Set Up**

- **Jenkins** : Déployé sur le port 8085. Jenkins est un outil d'automatisation "Open Source" permettant de créer des pipelines CI/CD.

Apres installation dans docker:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
bcc31c833a9a	morlayecisse1196/react-frontend:latest	"./docker-entrypoint...."	10 hours ago	Up 10 hours	
22f2a1540b58	morlayecisse1196/django-app:latest	"/app/docker-entryp...."	11 hours ago	Up 10 hours (healthy)	8000/tcp
354f5c3f37fc	postgres:15	"docker-entrypoint.s...."	11 hours ago	Up 10 hours	5432/tcp
13b5186ea06b	sonarqube:9.9-community	"/opt/sonarqube/dock...."	42 hours ago	Up 10 hours	0.0.0.0:9000-9000/tcp
28eea598829b	postgres:14	"docker-entrypoint.s...."	42 hours ago	Up 10 hours	0.0.0.0:5432-5432/tcp
14e9b9d7c4ee	jenkins/jenkins:lts	"/usr/bin/tini -- /u...."	2 days ago	Up 10 hours	0.0.0.0:50000-50000/tcp, 0.0.0.0:8085->8080/tcp, 0.0.0.0:8080->8080/tcp
<b>ubuntu@ip-172-31-0-110:</b> \$					

## 4. Configuration CI : Jenkins & Qualité

### 4.1 Configuration Globale

- **Plugins installés** : Docker Pipeline, SonarQube Scanner, GitHub Integration.

**Progression du téléchargement**

- Checking internet connectivity
- Checking update center connectivity
- Success

Plugin	Statut
SonarQube Scanner	Succès
Loading plugin extensions	Success
Apache HttpComponents Client 5.x API	Succès
Sonar Quality Gates	Succès
Loading plugin extensions	Success
JavaMail API	Succès
Oracle Java SE Development Kit Installer	Succès
SSH server	Succès
Command Agent Launcher	Succès
Quality Gates	Succès
Loading plugin extensions	Success

- **Agent Jenkins :** Création d'un nœud distant (agent-ubuntu) pour exécuter les builds hors du serveur maître mais on voit bien que l'agent est désactivé.

**Agent agent-ubuntu**

! La connexion est cassée

Exécuter depuis la ligne de commande de l'agent : (Unix)

```
curl -s0 http://51.44.14.31:8085/jnlpJars/agent.jar
java -jar agent.jar -url http://51.44.14.31:8085/ -secret
c66d9ab70f42393941ca3796d374e91312c2bf9ca26f32ec32c4c308ff39a7ee -name "agent-ubuntu" -webSocket -workDir
"/home/ubuntu/agent-ubuntu"
```

Exécuter à partir de la ligne de commande de l'agent : (Windows)

Activons notre agent-ubuntu :

```

ubuntu@ip-172-31-5-106:~/agent-ubuntu$ ls -ld /home/ubuntu/agent-ubuntu
drwxr-xr-x 2 root root 4096 Feb 14 16:21 /home/ubuntu/agent-ubuntu
ubuntu@ip-172-31-5-106:~/agent-ubuntu$ sudo chown -R ubuntu:ubuntu /home/ubuntu/agent-ubuntu
ubuntu@ip-172-31-5-106:~/agent-ubuntu$ chmod -R 755 /home/ubuntu/agent-ubuntu
ubuntu@ip-172-31-5-106:~/agent-ubuntu$ java -jar agent.jar -url http://13.38.43.49:8083/ \
-secret af9016907e5a35b87d768b4ea5db9d90ee5337ba7089e5182cc4106f7a7ab405 \
-name "agent-ubuntu" -webSocket -workDir "/home/ubuntu/agent-ubuntu"
Feb 14, 2026 4:22:49 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ubuntu/agent-ubuntu/remoting as a remoting work directory
Feb 14, 2026 4:22:49 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/ubuntu/agent-ubuntu/remoting
Feb 14, 2026 4:22:49 PM hudson.remoting.Launcher createEngine
INFO: Setting up agent: agent-ubuntu
Feb 14, 2026 4:22:49 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3352.v17a_fb_4b_2773f
Feb 14, 2026 4:22:49 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ubuntu/agent-ubuntu/remoting as a remoting work directory
Feb 14, 2026 4:22:49 PM hudson.remoting.Launcher$CuiListener status
INFO: WebSocket connection open
Feb 14, 2026 4:22:49 PM hudson.remoting.Launcher$CuiListener status
INFO: Connected

```

S	Nom	Architecture	Différence entre les horloges	Espace disque disponible	Espace de swap disponible	Free Temp Space	Temps de réponse
	agent-ubuntu	Linux (amd64)	Synchronisé	34.67 GiB	0 B	34.67 GiB	92ms
	contrôleur	Linux (amd64)	Synchronisé	34.67 GiB	0 B	34.67 GiB	0ms
Données obtenues		27 s	27 s	27 s	27 s	27 s	27 s

Icône: S M L Legend

## 4.2 Pipeline CI Backend (Django)

- Creation d'un projet jenkins :

Saisissez un nom  
backend-jenkin

Select an item type

Pipeline  
Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

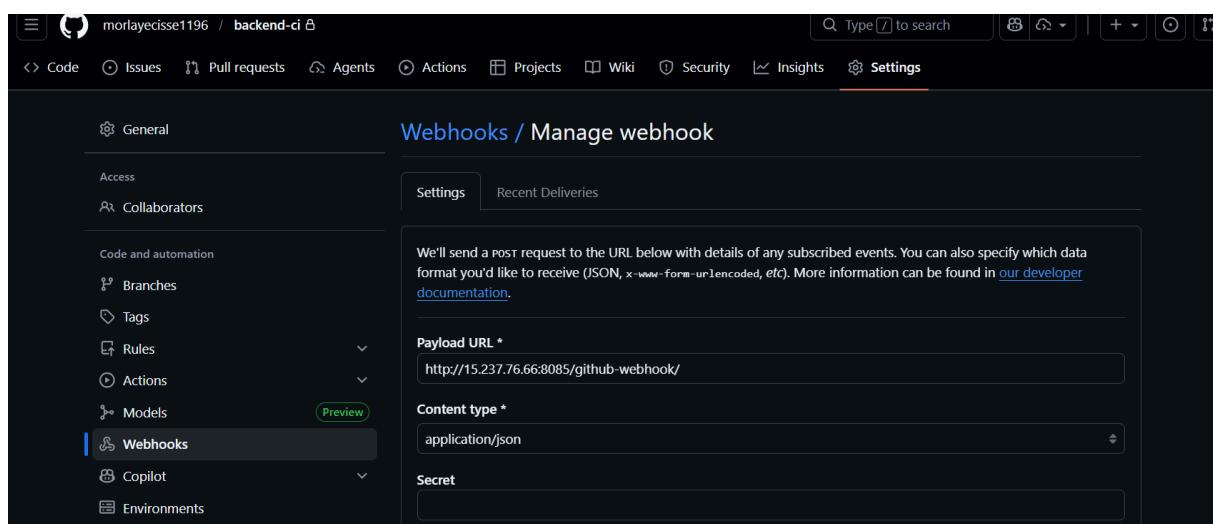
Construire un projet free-style  
Job legacy polyvalent qui récupère l'état depuis un outil de gestion de version au plus, exécute les étapes de build en série, suivi d'étapes post-construction telles que l'archivage d'artefacts et l'envoi de notifications par e-mail.

Construire un projet maven  
Construit un projet avec maven. Jenkins utilise directement vos fichiers POM et diminue radicalement l'effort de configuration. Cette fonctionnalité est encore en bêta mais elle est disponible afin d'obtenir vos retours.

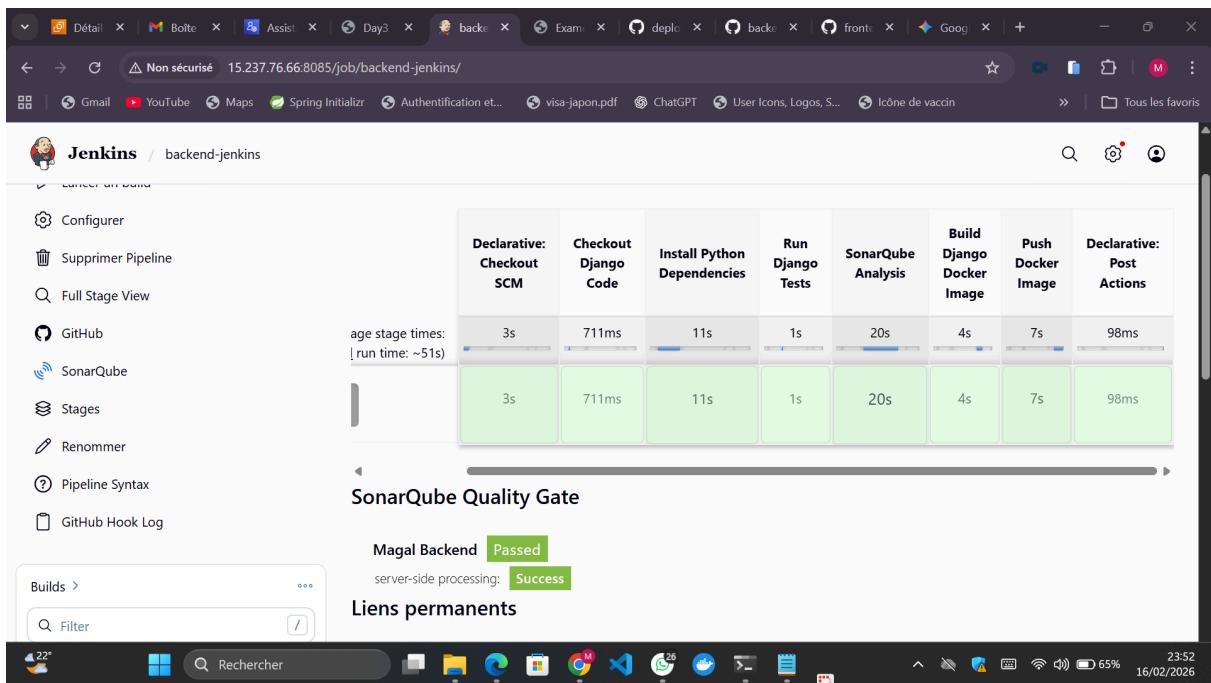
- Liaison GitHub : Configuration de l'URL du dépôt et des "Credentials".

## Configuration du Webhook GitHub :

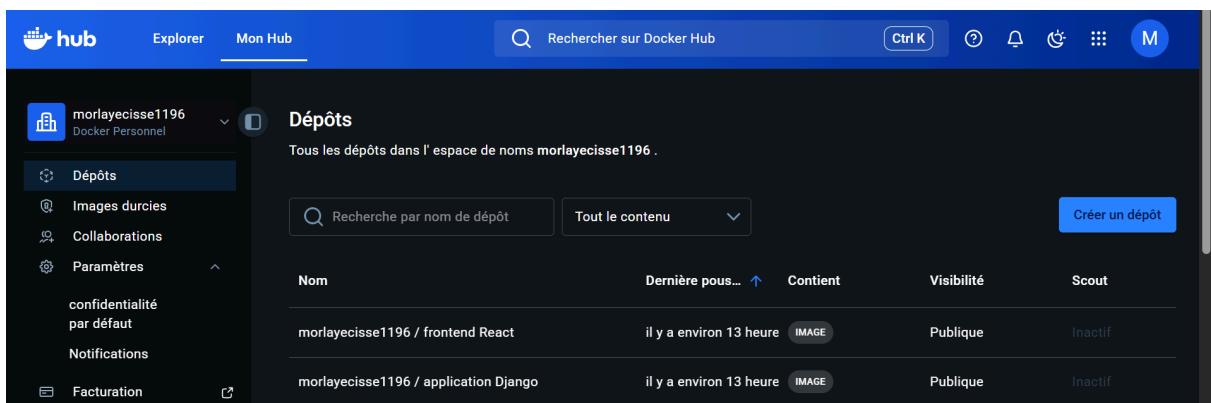
Dans les paramètres du dépôt GitHub (Settings > Webhooks), nous avons configuré un Webhook pointant vers l'URL de notre serveur Jenkins pour automatiser le déclenchement des builds à chaque push. **Étant donné que notre instance Jenkins est hébergée sur une adresse IP publique directement accessible**, l'utilisation d'un tunnel de connexion via ngrok n'a pas été nécessaire. Nous avons donc renseigné directement l'adresse IP du serveur dans l'URL de charge utile (Payload URL), garantissant ainsi une communication directe et performante entre GitHub et notre pipeline CI/CD.



- **Jenkinsfile :** Contient les étapes : *Checkout, SonarQube Analysis, Docker Build et Push vers Docker Hub* voici les detail apres build du projet *backend*.



- Apres push de l'image docker :



Voici quelque detail de l'analyse du code sonarQube realiser

The screenshot shows the SonarQube interface for a Django project. A specific security hotspot is highlighted in the code editor. The code snippet is as follows:

```

from django.http import JsonResponse
from django.db import connection
def bilan_de_sante(demande):

```

Annotations above the code include:

- 3 points d'accès de sécurité à examiner**
- Priorité de révision : HAUT**
- Falsification de requête intersite (CSRF)**
- Assurez-vous que l'autorisation des méthodes HTTP sécurisées et non sécurisées est bien prise en compte ici.**
- Priorité de révision : MOYEN**

The screenshot shows the SonarQube interface for a Django project, displaying a list of issues. The interface includes a sidebar for filtering problems and a main area for viewing specific issues. One issue is highlighted:

**Dockerfile**  
Remplacez 'as' par le format en majuscules 'AS'.

Code Odeur - Majeur - Ouvrir - Non attribué - 5 minutes d'effort - Commentaire

**gestion/views/health\_views.py**  
Supprimez le paramètre de fonction inutilisé « request ».

Code Odeur - Majeur - Ouvrir - Non attribué - 5 minutes d'effort - Commentaire

Statistics at the bottom:

- 1/2 numéros
- 10 minutes d'effort

## 4.3 Pipeline CI Frontend (React)

- Liaison GitHub :** Configuration de l'URL du dépôt et des "Credentials".

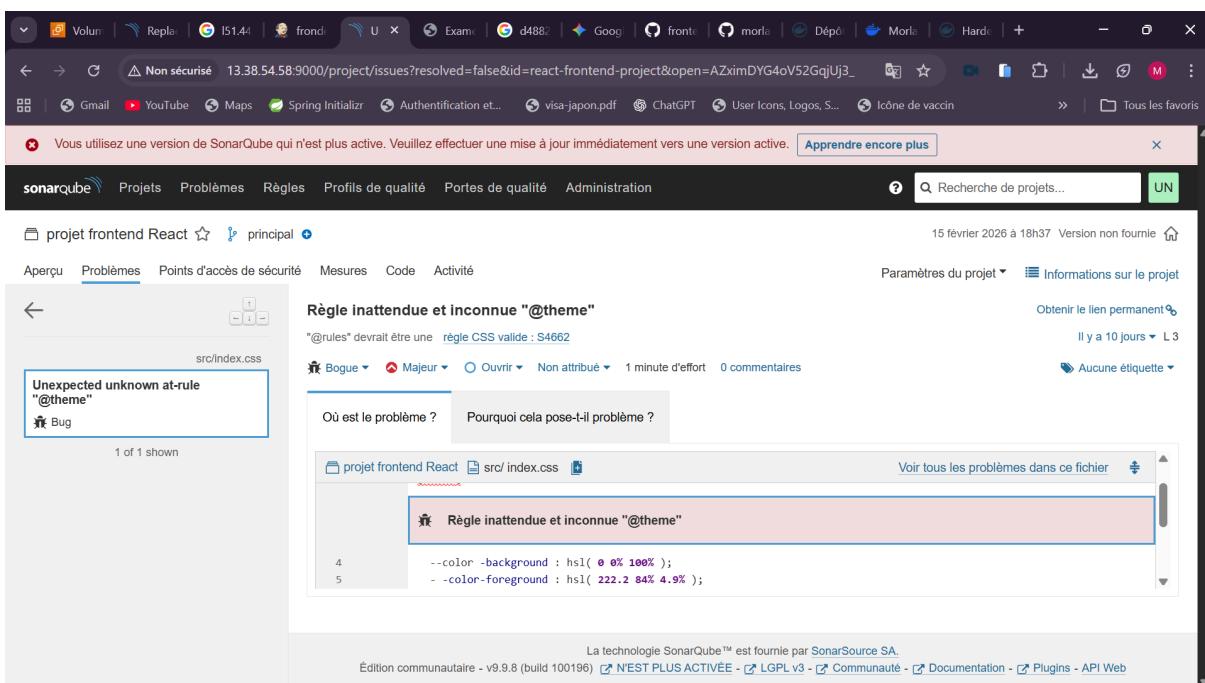
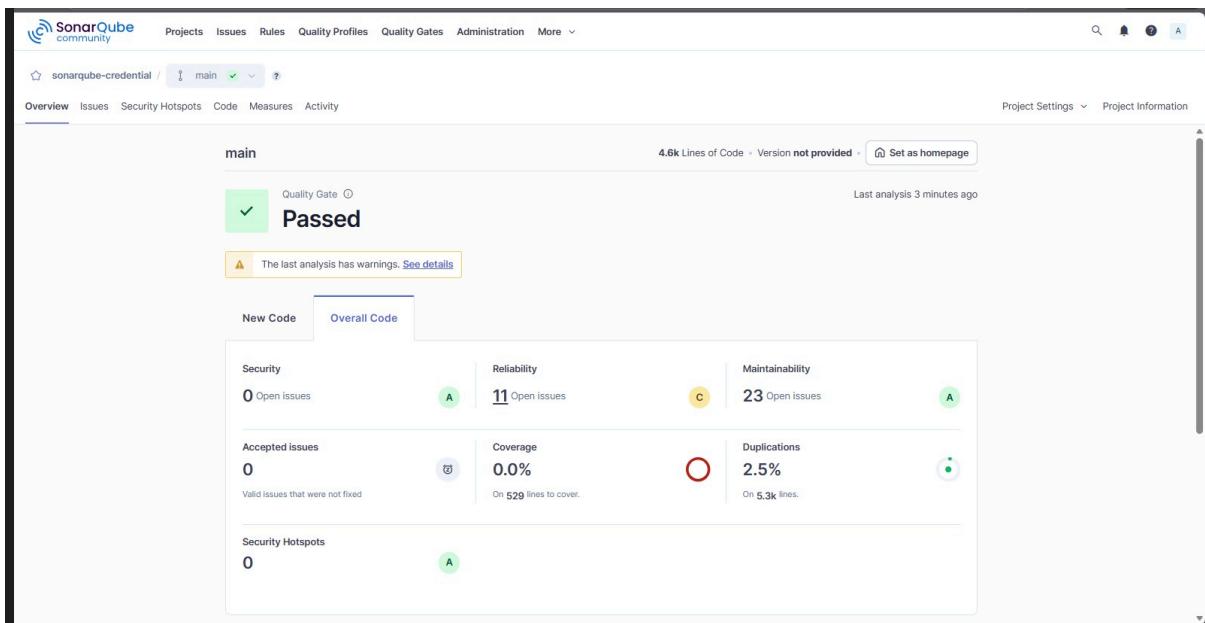
**Webhook :** Dans GitHub (Settings > Webhooks), nous avons ajouté l'URL de Jenkins pour déclencher le build automatiquement à chaque `push`.

The screenshot shows the GitHub settings page for the repository `frontend-ci`. The left sidebar has a 'Webhooks' section selected under 'Code and automation'. The main area is titled 'Webhooks / Manage webhook' and contains fields for 'Payload URL' (set to `http://15.237.76.66:8085/github-webhook/`), 'Content type' (set to `application/json`), and 'Secret'. A note explains that a POST request will be sent to the URL with event details.

**Jenkinsfile :** Contient les étapes : *Checkout, SonarQube Analysis, Docker Build et Push vers Docker Hub* voici les detail apres build du projet frontend.

The screenshot shows the Jenkins pipeline details for the job `frondend-jenkins`. The left sidebar lists pipeline management options like 'Changes', 'Lancer un build', 'Configurer', etc. The main area displays a timeline of pipeline stages: 'Declarative: Checkout SCM' (998ms), 'Checkout React Code' (656ms), 'Install Node Dependencies' (3s), 'SonarQube Analysis' (20s), 'Build React Production' (17s), 'Build Docker Image' (53s), 'Push Docker Image' (7s), and 'Declarative: Post Actions' (93ms). Below the timeline, the 'SonarQube Quality Gate' is shown as 'Passed' with a green button. The pipeline also includes a 'GitHub Hook Log' and a 'Builds' section.

Quelque detail de l'analyse du code realiser dans sonarQube



## 5. Déploiement Continu (CD) & Sécurité (Étape 4 & 5)

### 5.1 Pipeline de Déploiement

- Structure du projet :

The screenshot shows a GitHub repository page for 'deployment-cd' under the 'main' branch. The repository contains several files: 'scripts', '.env', '.env.example', '.gitignore', 'Jenkinsfile', 'README.md', and 'docker-compose.yml'. The 'Jenkinsfile' is highlighted, showing its content:

```

#!/bin/bash
# Stage 1: Build
echo "Building Docker image"
docker build -t myapp .
# Stage 2: Deploy
echo "Deploying Docker image to AWS Lambda"
aws lambda update-function-code --function-name myfunction --zip-file fileb://dist.zip

```

The 'README.md' file contains the following text:

**Déploiement - Gestion Événement Magal (CD)**

- **Liaison GitHub :** Configuration de l'URL du dépôt et des "Credentials".

**Webhook :** Dans GitHub (Settings > Webhooks), nous avons ajouté l'URL de Jenkins pour déclencher le build automatiquement à chaque `push`.

The screenshot shows the GitHub settings page for the 'deployment-cd' repository. The 'Webhooks' tab is selected. A new webhook is being configured with the following details:

- Payload URL :** `http://15.237.76.66:8085/github-webhook/`
- Content type :** `application/json`
- Secret :** (empty field)

**Jenkinsfile :** Contient les étapes

```
steps {
    echo "Récupération des images depuis Docker Hub..."
    sh "docker compose pull"
}

stage('Security Scan (Trivy)') {
    steps {
        echo "Scan des vulnérabilités des images..."
        // Scan de l'image Backend
        sh "trivy image morlayecisse1196/django-app:latest"
        // Scan de l'image Frontend
        sh "trivy image morlayecisse1196/react-frontend:latest"
    }
}

stage('Deploy Application') {
    steps {
}
```

Le projet `deployment-jenkins` centralise le déploiement final.

**Scan de Sécurité (Trivy)** : Avant le déploiement, nous utilisons **Trivy** pour scanner les images Docker et identifier les vulnérabilités (CVE) qu'on doit installer.

Pourquoi trivy ?

Bien que Kubernetes soit mentionné dans l'énoncé comme outil cible pour le scan de vulnérabilités, nous avons implémenté cette exigence de sécurité via **Trivy** au sein de notre pipeline Jenkins. Cette approche permet de sécuriser la chaîne d'approvisionnement logicielle (Software Supply Chain) en analysant les images Docker dès leur récupération depuis Docker Hub, garantissant ainsi que seuls des conteneurs vérifiés sont déployés par Docker Compose.

```
ubuntu@ip-172-31-0-110:~$ sudo snap install trivy
trivy 0.52.2 from James Luther (b34rd) installed
ubuntu@ip-172-31-0-110:~$ trivy --version
Version: 0.52.2
ubuntu@ip-172-31-0-110:~$
```

The screenshot shows a terminal window on a Windows desktop. The terminal content is as follows:

```
ubuntu@ip-172-31-0-110:~$ sudo snap install trivy
trivy 0.52.2 from James Luther (b34rd) installed
ubuntu@ip-172-31-0-110:~$ trivy --version
Version: 0.52.2
ubuntu@ip-172-31-0-110:~$
```

The desktop taskbar at the bottom shows various icons for applications like File Explorer, Edge, and Google Chrome.

**Orchestration : Utilisation de Docker Compose pour lancer simultanément le Backend, le Frontend et la base de données PostgreSQL avec un volume persistant**

The screenshot shows a GitHub browser interface displaying a `docker-compose.yml` file. The file defines three services: `db`, `backend`, and `frontend`. The `db` service uses the `postgres:15` image, restarts always, and has environment variables `POSTGRES_DB`, `POSTGRES_USER`, and `POSTGRES_PASSWORD`. It also maps the `/var/lib/postgresql/data` volume. The `backend` service uses the `morlayecisse1196/django-app:latest` image, restarts always, depends on `db`, and maps port `80:80`. The `frontend` service uses the `morlayecisse1196/react-frontend:latest` image, restarts always, and maps port `80:80`.

```
version: '3.8'
services:
  db:
    image: postgres:15
    container_name: magal_db_prod
    restart: always
    environment:
      POSTGRES_DB: magal_db
      POSTGRES_USER: magal_user
      POSTGRES_PASSWORD: changeme_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
  backend:
    image: morlayecisse1196/django-app:latest
    container_name: django_backend_prod
    restart: always
    env_file: .env
    depends_on:
      - db
  frontend:
    image: morlayecisse1196/react-frontend:latest
    container_name: react_frontend_prod
    restart: always
    ports:
      - "80:80"
```

Apres build du projet ont vois ici le graphique du pipeline réussi dans Jenkins.

The screenshot shows the Jenkins Stage View page for a pipeline named "deployment-jenkins". On the left, there's a sidebar with options like "Lancer un build", "Configurer", "Supprimer Pipeline", "Full Stage View", "GitHub", "Stages", "Renommer", "Pipeline Syntax", and "GitHub Hook Log". Below that is a "Builds >" section with a "Filter" input. The main area is titled "Stage view" and displays three stages:

#9	#8	#7	Declarative: Checkout SCM	Checkout Deployment Code	Pull Latest Images	Security Scan (Trivy)	Deploy Application	Declarative: Post Actions
fevr. 17 01:42 No Changes	fevr. 17 01:27 1 commit	fevr. 17 01:15 1 commit	2s	700ms	2s	15s	1s	132ms
			4s	820ms	2s	10s	1s	132ms
			1s	651ms	2s	34s	2s	failed
			1s	630ms	2s	424ms	87ms	failed

Average stage times: (full run time: ~24s)

**Liens permanents**

- Dernier build (#8), il y a 9 mn 16 s

Après déploiement de l'application, voici les logs après l'analyse effectuée par Trivy sur les images Docker du projet :

The screenshot shows the Jenkins console output for build #9. The output is a terminal window displaying the results of a Trivy security scan:

```

morlayecisse1196/django-app:latest (debian 13.3)
=====
Total : 104 (INCONNU : 0, FAIBLE : 84, MOYEN : 18, ÉLEVÉ : 2, CRITIQUE : 0)

Bibliothèque | Vulnérabilité | Gravité | Statut | Version installée | Version corrigée | Titre |
apt | CVE-2011-3374 | FAIBLE | affecté | 3.0.3 | Il a été constaté que la clé apt dans apt, toutes versions confondues, ne fonctionne pas correctement.
| | | | | correctement... |
| | | | | https://avd.aquasec.com/nvd/cve-2011-3374 |
bash | TEMP-0841856-B18BAF | 5.2.37-2+b7 | [élévation de priviléges possible pour un utilisateur autre que root] |
| | | | | https://security-tracker.debian.org/tracker/TEMP-0841856-B1- |
| | | | | 8BAF |

```

Voici quelques commentaires et analyse de Sécurité :

L'analyse de sécurité effectuée par Trivy révèle la présence de **2 vulnérabilités de gravité ÉLEVÉE** (CVE-2026-0861) liées à la bibliothèque système **glibc**. Ces failles concernent la gestion de la mémoire et pourraient entraîner une corruption du tas (heap corruption).

## **Correctifs recommandés :**

- 1. Mise à jour de l'image de base** : Modifier le Dockerfile pour utiliser une version plus récente ou plus sécurisée de Debian/Python (ex: `python:3.11-slim-bookworm`).
- 2. Réduction de la surface d'attaque** : Utiliser une image **Alpine Linux** (`python:3.11-alpine`) qui, comme démontré lors du scan de notre Frontend, présente généralement 0 vulnérabilité car elle ne contient que le strict nécessaire.
- 3. Audit régulier** : Maintenir ce scan Trivy dans le pipeline CD pour bloquer tout déploiement si une faille 'CRITIQUE' est détectée à l'avenir.

## **Conclusion**

Ce projet démontre une maîtrise complète de la chaîne DevOps, de la gestion de tâches à la sécurité des conteneurs, assurant une mise à jour sans interruption du service