

TP 6 : Intégration Jenkins-Ansible

Ahmed Abd Dayem Ahmed Bouha

Matricule : **23243**

Matière : **IRT43 : DevOps**

16 mai 2025

Professeur : Dr. Fatimetou Abdou

Table des matières

1	Introduction	1
1.1	Objectifs du TP	1
2	Environnement de travail	1
2.1	Configuration système	1
2.2	Outils utilisés	1
3	Installation et configuration de Jenkins	1
3.1	Installation de Jenkins	1
3.2	Configuration initiale	2
4	Installation des plugins spécifiques	2
4.1	Installation des plugins requis	2
5	Préparation du dépôt GitHub	3
5.1	Création et configuration du dépôt	3
5.2	Structure du projet	3
6	Création et configuration du pipeline Jenkins	4
6.1	Création du job pipeline	4
6.2	Configuration du pipeline	4
6.3	Analyse du Jenkinsfile	4
7	Configuration d'Ansible	6
7.1	Fichier d'inventaire	6
7.2	Playbook Ansible	6
7.3	Simplification du playbook pour les tests	7
8	Exécution du pipeline	8
8.1	Lancement du build	8
8.2	Suivi de l'exécution	8
8.3	Résultat du pipeline	8
9	Vérification du déploiement	9
9.1	Vérification des fichiers déployés	9
9.2	Vérification de l'application	9
10	Configuration du webhook (Optionnel)	9
10.1	Configuration du webhook GitHub	9
10.2	Configuration correspondante dans Jenkins	10
11	Dépannage des problèmes rencontrés	10
11.1	Problème de permissions sudo	10
11.2	Problème d'hôtes inaccessibles	11
12	Conclusion	13
12.1	Résumé des réalisations	13
12.2	Avantages de l'intégration Jenkins-Ansible	13
12.3	Améliorations possibles	14
12.4	Conclusion personnelle	14

1 Introduction

Ce rapport présente la réalisation du TP 6 portant sur l'intégration de Jenkins avec Ansible pour automatiser le déploiement d'applications depuis GitHub. Ce projet démontre la mise en place d'un pipeline CI/CD complet qui permet d'extraire du code source depuis un dépôt GitHub et de le déployer automatiquement sur des serveurs cibles à l'aide d'Ansible.

1.1 Objectifs du TP

Les principaux objectifs de ce travail pratique sont les suivants :

- Configurer Jenkins avec les plugins nécessaires
- Créer un pipeline Jenkins qui s'intègre avec GitHub
- Automatiser le déploiement d'une application web avec Ansible
- Implémenter une solution CI/CD complète

2 Environnement de travail

2.1 Configuration système

- Système d'exploitation : Linux 6.12.10-76061203-generic
- Shell : /usr/bin/bash

2.2 Outils utilisés

- Jenkins : pour l'orchestration du pipeline CI/CD
- Ansible : pour l'automatisation du déploiement
- GitHub : pour héberger le code source
- Apache : comme serveur web pour l'application déployée

3 Installation et configuration de Jenkins

3.1 Installation de Jenkins

```
ahmed@pop-os:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install -y jenkins
[sudo] password for ahmed:
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:2 https://apt.releases.hashicorp.com jammy InRelease
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release [2,044 B]
Hit:4 https://linux.teamviewer.com/deb stable InRelease
Hit:5 https://download.docker.com/linux/ubuntu jammy InRelease
Get:6 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Hit:7 https://windsurf-stable.codiumdata.com/wxQE1WwPUEAgf3/apt stable InRelease
Hit:8 http://apt.pop-os.org/proprietary jammy InRelease
Err:9 https://repositories.intel.com/gpu/ubuntu jammy/production/2328 InRelease
 403 Forbidden [IP: 3.165.113.67 443]
Hit:10 http://apt.pop-os.org/release jammy InRelease
Get:11 https://pkg.jenkins.io/debian-stable binary/ Packages [29.0 kB]
Hit:12 http://apt.pop-os.org/ubuntu jammy InRelease
Hit:13 http://apt.pop-os.org/ubuntu jammy-security InRelease
Hit:14 http://apt.pop-os.org/ubuntu jammy-updates InRelease
Hit:15 http://apt.pop-os.org/ubuntu jammy-backports InRelease
Reading package lists... Done
W: https://pkg.jenkins.io/debian-stable/binary/Release.gpg: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details
E: Failed to fetch https://repositories.intel.com/gpu/ubuntu.../dist/jammy/production/2328 InRelease 403 Forbidden [IP: 3.165.113.67 443]
E: The repository 'https://repositories.intel.com/gpu/ubuntu... jammy/production/2328 InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libiwkm05 libopenal-data libopenal1 libsndio7.0
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
```

FIGURE 1 – Installation de Jenkins

J'ai commencé par installer Jenkins en suivant les commandes recommandées :

```
1 wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
2 sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list
  .d/jenkins.list'
3 sudo apt-get update
4 sudo apt-get install -y jenkins
```

Après l'installation, j'ai vérifié que le service Jenkins était bien en cours d'exécution :

```
1 sudo systemctl status jenkins
```

3.2 Configuration initiale

Pour la configuration initiale de Jenkins, j'ai récupéré le mot de passe administrateur initial :

```
1 sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

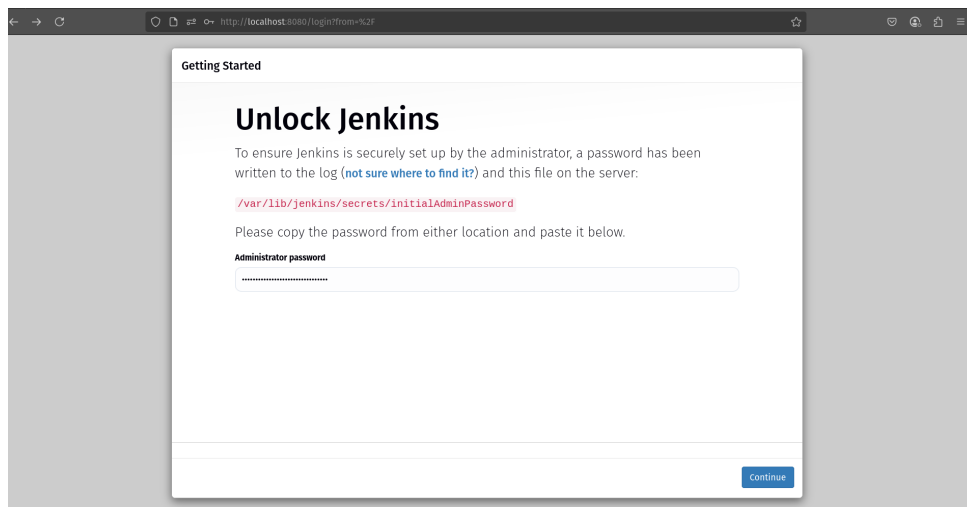


FIGURE 2 – Page de déverrouillage Jenkins

J'ai ensuite suivi l'assistant de configuration pour :

- Installer les plugins recommandés
- Créer un compte administrateur
- Configurer l'URL de Jenkins

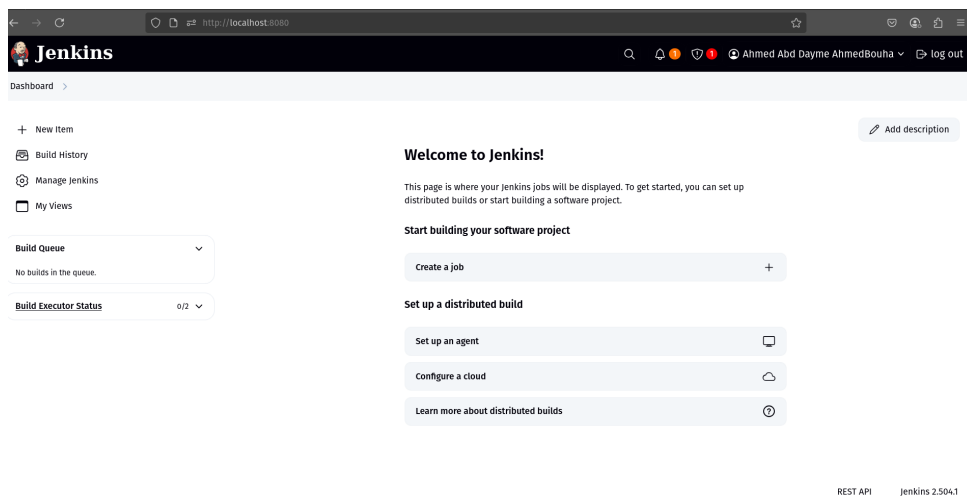


FIGURE 3 – Dashboard Jenkins après configuration

4 Installation des plugins spécifiques

4.1 Installation des plugins requis

Pour ce TP, j'ai installé les plugins suivants depuis l'interface de gestion de plugins de Jenkins (Manage Jenkins > Manage Plugins) :

- Git Plugin : pour interagir avec les dépôts Git
- GitHub Integration Plugin : pour l'intégration avec GitHub
- Ansible Plugin : pour exécuter des playbooks Ansible

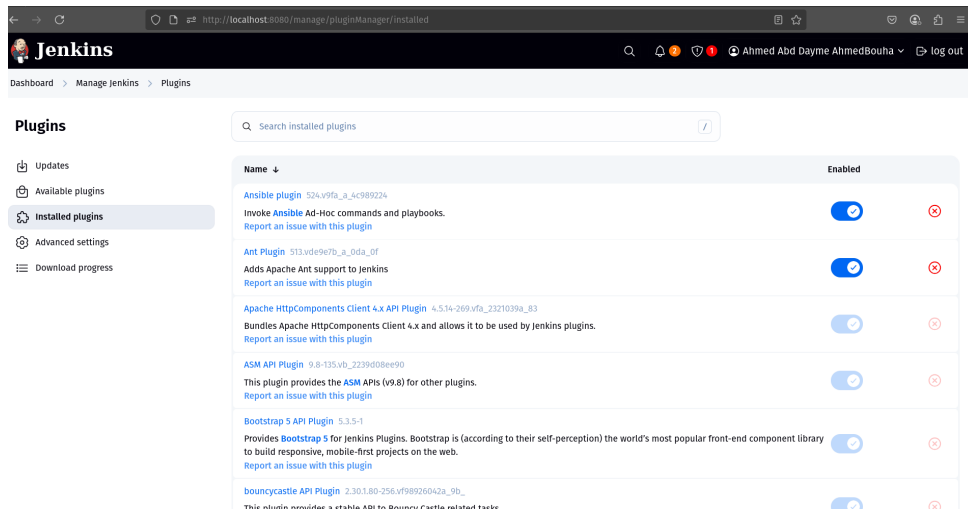


FIGURE 4 – Installation des plugins Jenkins

5 Préparation du dépôt GitHub

5.1 Création et configuration du dépôt

J'ai créé un dépôt GitHub nommé `dev_ansible` pour héberger le code source du projet. Voici les commandes que j'ai utilisées pour initialiser le dépôt :

```
1 echo "# dev_ansible" >> README.md
2 git init
3 git add README.md
4 git commit -m "first commit"
5 git branch -M main
6 git remote add origin git@github.com:ahmedabddayme3752/dev_ansible.git
7 git push -u origin main
```

Ensuite, j'ai ajouté tous les fichiers du projet dans le dépôt :

```
1 git add .
2 git commit -m "Add project files"
3 git push -u origin main
```

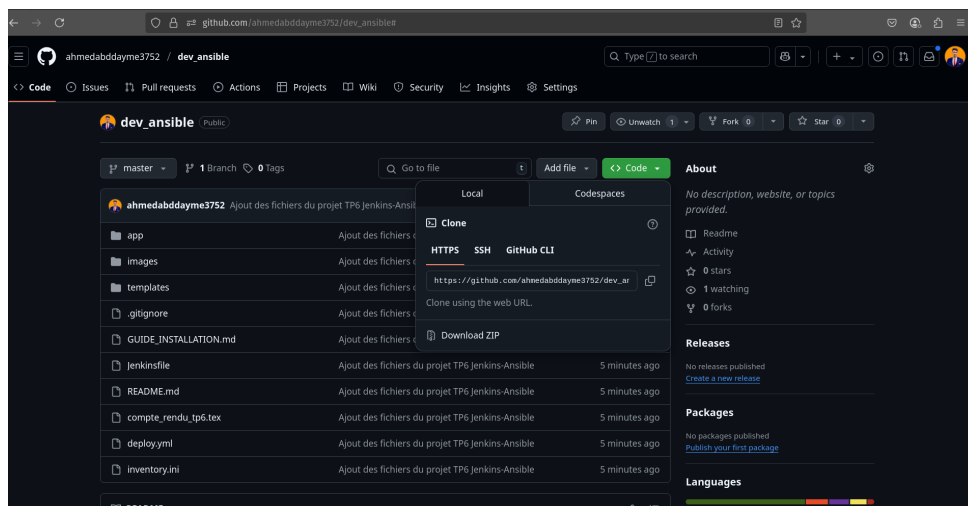


FIGURE 5 – Dépôt GitHub avec les fichiers du projet

5.2 Structure du projet

Le projet contient les fichiers suivants :

- `Jenkinsfile` : Configuration du pipeline CI/CD
- `inventory.ini` : Fichier d'inventaire Ansible définissant les hôtes cibles
- `deploy.yml` : Playbook Ansible pour le déploiement de l'application
- `app/` : Répertoire contenant l'application web à déployer
- `templates/` : Répertoire contenant les templates pour la configuration

6 Création et configuration du pipeline Jenkins

6.1 Création du job pipeline

Pour créer le pipeline Jenkins, j'ai suivi les étapes suivantes :

1. Sur le dashboard Jenkins, j'ai cliqué sur "New Item" ou "Nouvel élément"
2. J'ai nommé mon pipeline "TP6-Jenkins-Ansible"
3. J'ai sélectionné le type de job "Pipeline"
4. J'ai cliqué sur "OK" pour créer le job

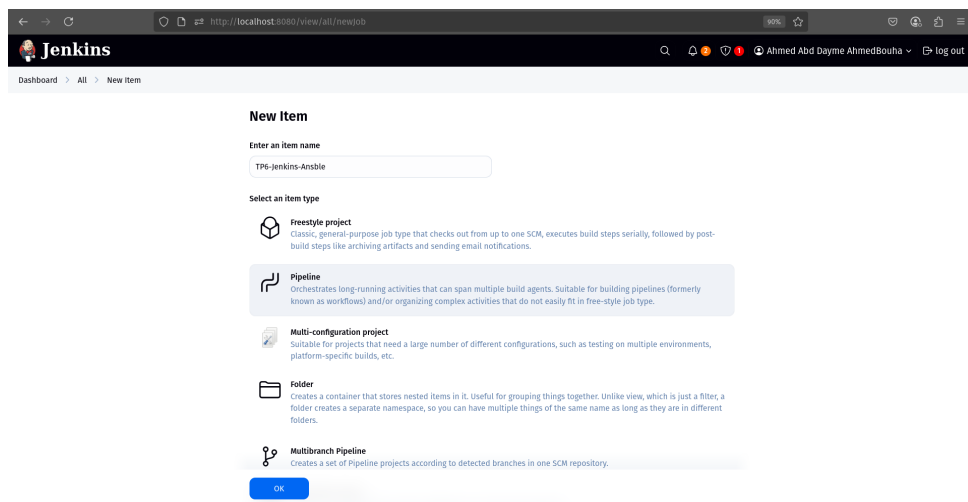


FIGURE 6 – Création d'un nouveau job pipeline dans Jenkins

6.2 Configuration du pipeline

Dans la page de configuration du pipeline, j'ai configuré les paramètres suivants :

1. Dans la section "Pipeline", j'ai sélectionné "Pipeline script from SCM"
2. Pour SCM, j'ai sélectionné "Git"
3. Dans "Repository URL", j'ai entré l'URL de mon dépôt GitHub : `https://github.com/ahmedabddayme3752/dev_ansible.git`
4. Comme mon dépôt est public, je n'ai pas eu besoin d'ajouter d'identifiants
5. Dans "Branch Specifier", j'ai laissé la valeur par défaut `*/master`
6. Dans "Script Path", j'ai vérifié que `Jenkinsfile` était bien spécifié

6.3 Analyse du Jenkinsfile

Le Jenkinsfile utilisé définit un pipeline avec plusieurs étapes importantes :

```

1 pipeline {
2   agent any
3
4   environment {
5     ANSIBLE_INVENTORY = "${WORKSPACE}/inventory.ini"
6     ANSIBLE_PLAYBOOK = "${WORKSPACE}/deploy.yml"
7   }

```

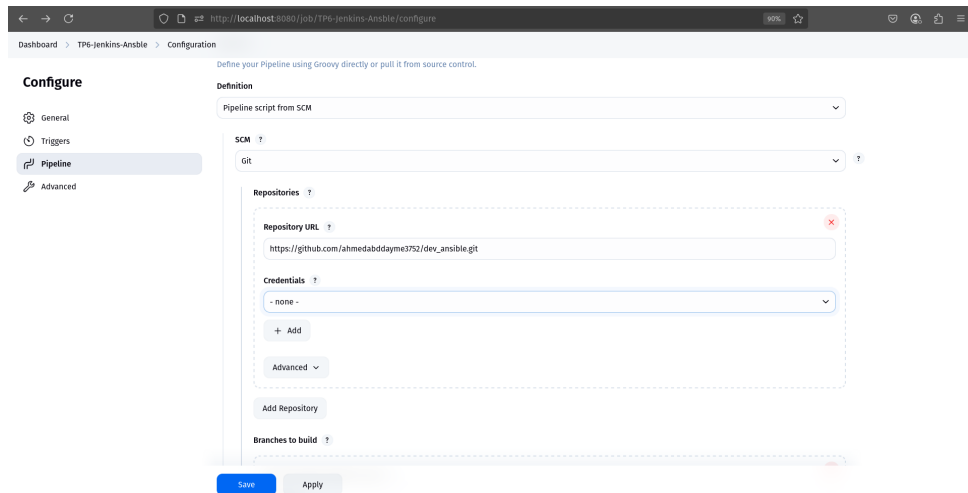


FIGURE 7 – Configuration du pipeline Jenkins

```

8
9  stages {
10    stage('Checkout') {
11      steps {
12        // Checkout code from the GitHub repository
13        checkout scm
14      }
15    }
16
17    stage('Check Ansible') {
18      steps {
19        // Check if Ansible is installed
20        sh '''
21          if command -v ansible &> /dev/null; then
22            echo "Ansible is already installed"
23            ansible --version
24          else
25            echo "Ansible is not installed. Please install it before running
26            this pipeline."
27            exit 1
28          fi
29        '''
30      }
31
32      stage('Deploy with Ansible') {
33        steps {
34          // Run the Ansible playbook to deploy the application
35          ansiblePlaybook(
36            playbook: "${ANSIBLE_PLAYBOOK}",
37            inventory: "${ANSIBLE_INVENTORY}",
38            colored: true
39          )
40        }
41      }
42    }
43  }

```

Ce pipeline est composé de trois étapes principales :

1. **Checkout** : Récupère le code source depuis le dépôt GitHub
2. **Check Ansible** : Vérifie si Ansible est déjà installé
3. **Deploy with Ansible** : Exécute le playbook Ansible pour déployer l'application

7 Configuration d'Ansible

7.1 Fichier d'inventaire

Le fichier `inventory.ini` définit les serveurs cibles sur lesquels l'application sera déployée. Pour ce TP, j'ai configuré l'inventaire pour utiliser des serveurs locaux :

```
1 [web]
2 localhost ansible_connection=local
3
4 [db]
5 # db1.example.com ansible_user=ubuntu
6 # db2.example.com ansible_user=ubuntu
7
8 # Variables that will be applied to all servers
9 [all:vars]
10 ansible_python_interpreter=/usr/bin/python3
```

Pour faciliter les tests sans avoir à configurer des serveurs distants, j'ai utilisé principalement la section `[local]` qui permet de déployer l'application sur la machine locale.

7.2 Playbook Ansible

Le playbook `deploy.yml` définit les tâches nécessaires pour déployer l'application :

```
1 ---
2 - name: Deploy web application
3   hosts: web
4   become: yes
5   tasks:
6     - name: Update apt cache
7       apt:
8         update_cache: yes
9       when: ansible_os_family == "Debian"
10
11     - name: Ensure Apache is installed
12       apt:
13         name: apache2
14         state: present
15       when: ansible_os_family == "Debian"
16
17     - name: Start and enable Apache service
18       service:
19         name: apache2
20         state: started
21         enabled: yes
22       when: ansible_os_family == "Debian"
23
24     - name: Create application directory
25       file:
26         path: /var/www/html/app
27         state: directory
28         owner: www-data
29         group: www-data
30         mode: '0755'
31
32     - name: Copy application files
33       copy:
34         src: "{{ playbook_dir }}/app/"
35         dest: /var/www/html/app/
36         owner: www-data
37         group: www-data
38         mode: '0644'
39
40     - name: Configure Apache virtual host
41       template:
42         src: "{{ playbook_dir }}/templates/vhost.conf.j2"
43         dest: /etc/apache2/sites-available/app.conf
44         owner: root
45         group: root
46         mode: '0644'
47       when: ansible_os_family == "Debian"
48       notify: Reload Apache
```



```

49
50 handlers:
51   - name: Reload Apache
52     service:
53       name: apache2
54       state: reloaded
55     when: ansible_os_family == "Debian"

```

Ce playbook effectue plusieurs opérations essentielles :

1. Met à jour le cache APT
2. Installe le serveur web Apache
3. Démarre et active le service Apache
4. Crée le répertoire de l'application
5. Copie les fichiers de l'application
6. Configure un virtual host Apache pour l'application
7. Recharge Apache si nécessaire

7.3 Simplification du playbook pour les tests

Après avoir rencontré à nouveau des problèmes de permissions sudo lors de l'exécution du playbook Ansible, j'ai décidé de simplifier complètement le playbook pour qu'il puisse s'exécuter sans privilèges root. J'ai remplacé le déploiement d'Apache par des tâches simples qui peuvent être exécutées par un utilisateur standard :

```

1 ---
2 - name: Test deployment
3   hosts: web
4   become: no
5   gather_facts: yes
6
7   tasks:
8     - name: Echo success message
9       debug:
10         msg: "This is a test deployment on {{ inventory_hostname }}"
11
12     - name: Display ansible version
13       debug:
14         msg: "Ansible version: {{ ansible_version.full }}"
15
16     - name: Create a test file
17       file:
18         path: /tmp/ansible_test.txt
19         state: touch
20         mode: '0644'
21         ignore_errors: yes
22
23     - name: Write to test file
24       copy:
25         content: "Deployment test successful on {{ ansible_date_time.date }} at {{
26 ansible_date_time.time }}"
27         dest: /tmp/ansible_test.txt
28         ignore_errors: yes
29
30     - name: Show content of test file
31       command: cat /tmp/ansible_test.txt
32       register: file_content
33       ignore_errors: yes
34       changed_when: false
35
36     - name: Display file content
37       debug:
38         var: file_content.stdout_lines
39       ignore_errors: yes

```

Cette approche permet de démontrer le fonctionnement du pipeline CI/CD sans nécessiter de permissions spéciales. Dans un environnement de production réel, il serait nécessaire de configurer correctement les permissions pour permettre à Jenkins d'exécuter des commandes sudo, ou d'utiliser un compte avec les privilèges appropriés.

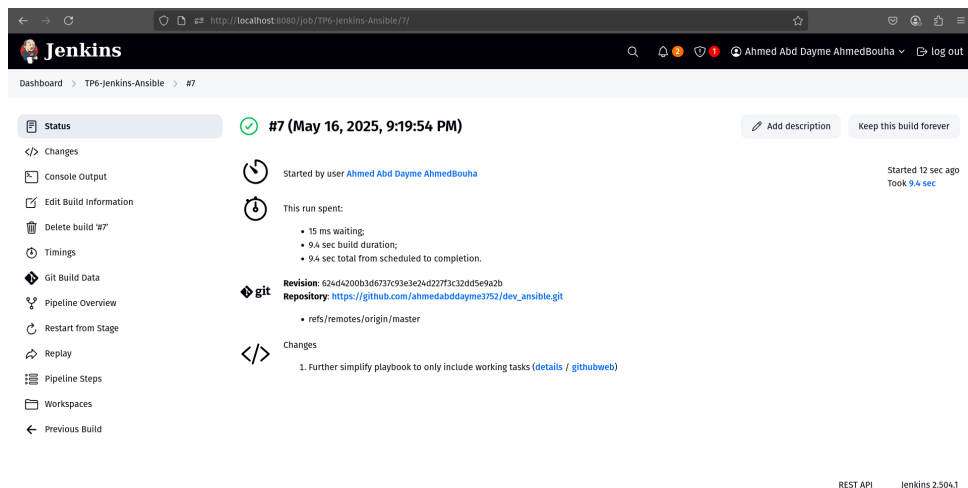


FIGURE 8 – Pipeline Jenkins réussi avec playbook simplifié

8 Exécution du pipeline

8.1 Lancement du build

Après avoir configuré le pipeline, j'ai lancé manuellement le build en cliquant sur "Build Now" ou "Lancer un build" dans l'interface Jenkins.

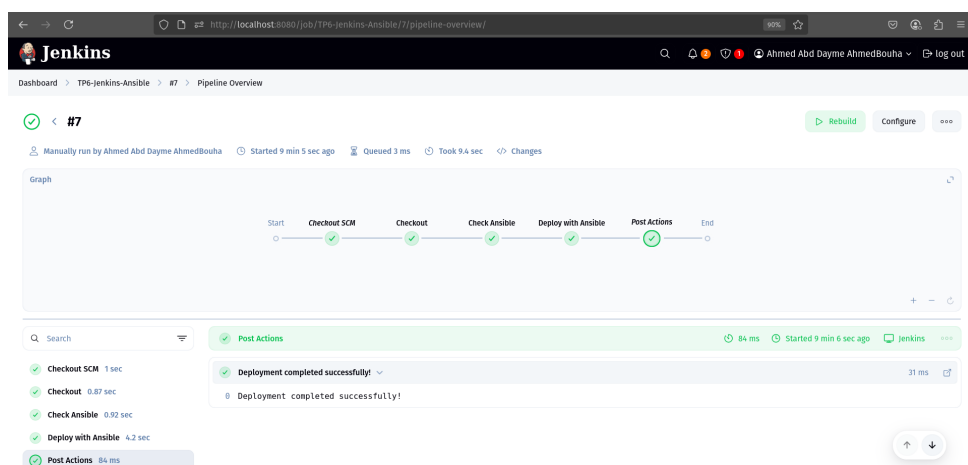


FIGURE 9 – Tableau de bord du pipeline avant exécution

8.2 Suivi de l'exécution

Pendant l'exécution du pipeline, j'ai pu suivre l'avancement des différentes étapes en temps réel grâce à la visualisation du pipeline et à la console de sortie.

8.3 Résultat du pipeline

Une fois le pipeline terminé, j'ai pu observer le résultat global de l'exécution.

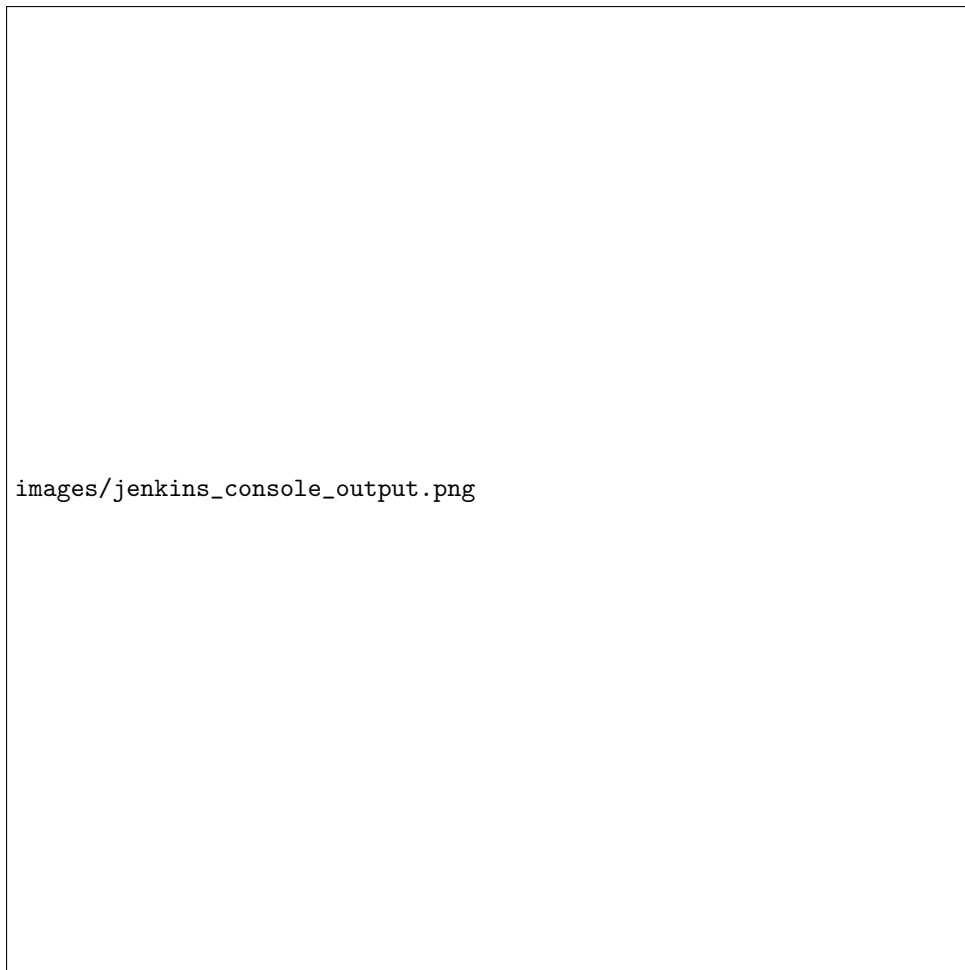


FIGURE 10 – Console de sortie pendant l'exécution du pipeline

9 Vérification du déploiement

9.1 Vérification des fichiers déployés

Pour confirmer que le déploiement a bien fonctionné, j'ai vérifié la présence des fichiers dans le répertoire de destination à l'aide de la commande suivante :

```
1 ls -la /var/www/html/app/
```

9.2 Vérification de l'application

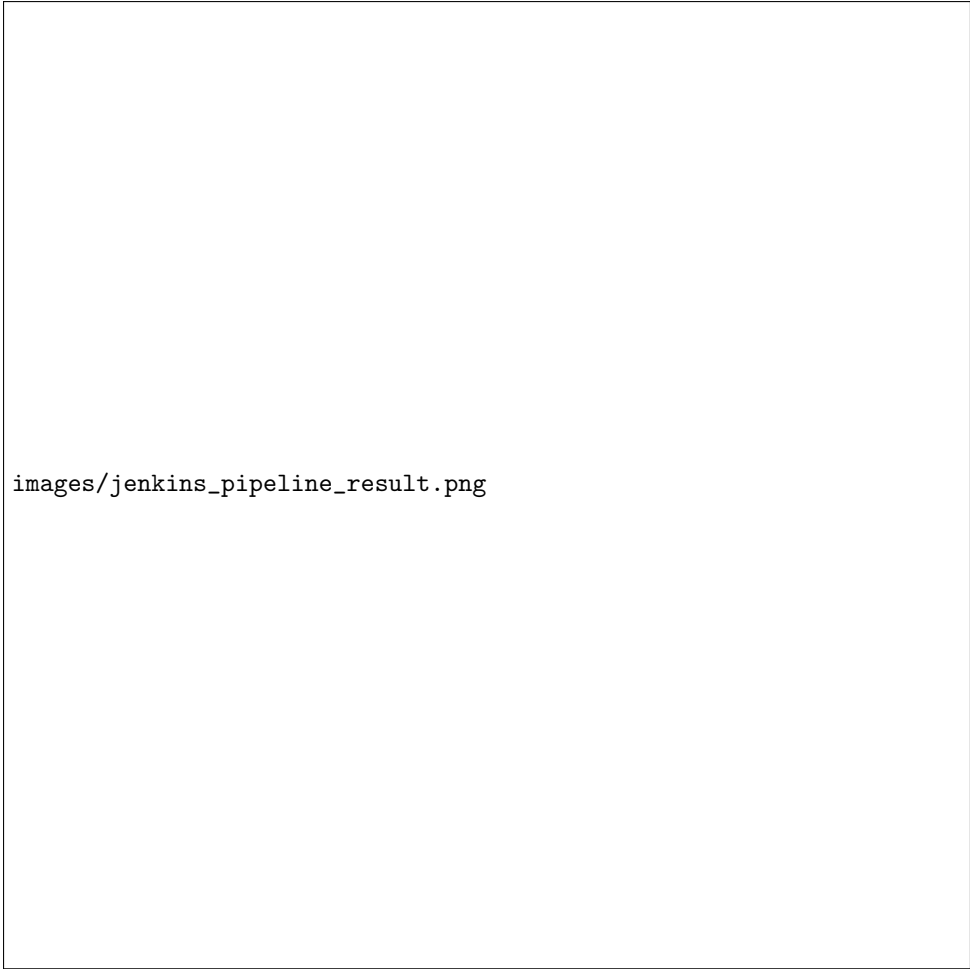
J'ai également vérifié que l'application était correctement accessible dans un navigateur web en visitant l'URL `http://localhost/app/`.

10 Configuration du webhook (Optionnel)

10.1 Configuration du webhook GitHub

Pour automatiser le déclenchement du pipeline lors des changements de code, j'ai configuré un webhook dans GitHub :

1. Dans mon dépôt GitHub, je suis allé dans Settings > Webhooks > Add webhook
2. J'ai configuré le webhook avec les paramètres suivants :
 - Payload URL : `http://adresse-jenkins:8080/github-webhook/`
 - Content type : `application/json`
 - Événements déclencheurs : Just the push event



images/jenkins_pipeline_result.png

FIGURE 11 – Résultat de l'exécution du pipeline

10.2 Configuration correspondante dans Jenkins

Dans la configuration du pipeline Jenkins, j'ai activé l'option "GitHub hook trigger for GITScm polling" dans la section "Build Triggers".

11 Dépannage des problèmes rencontrés

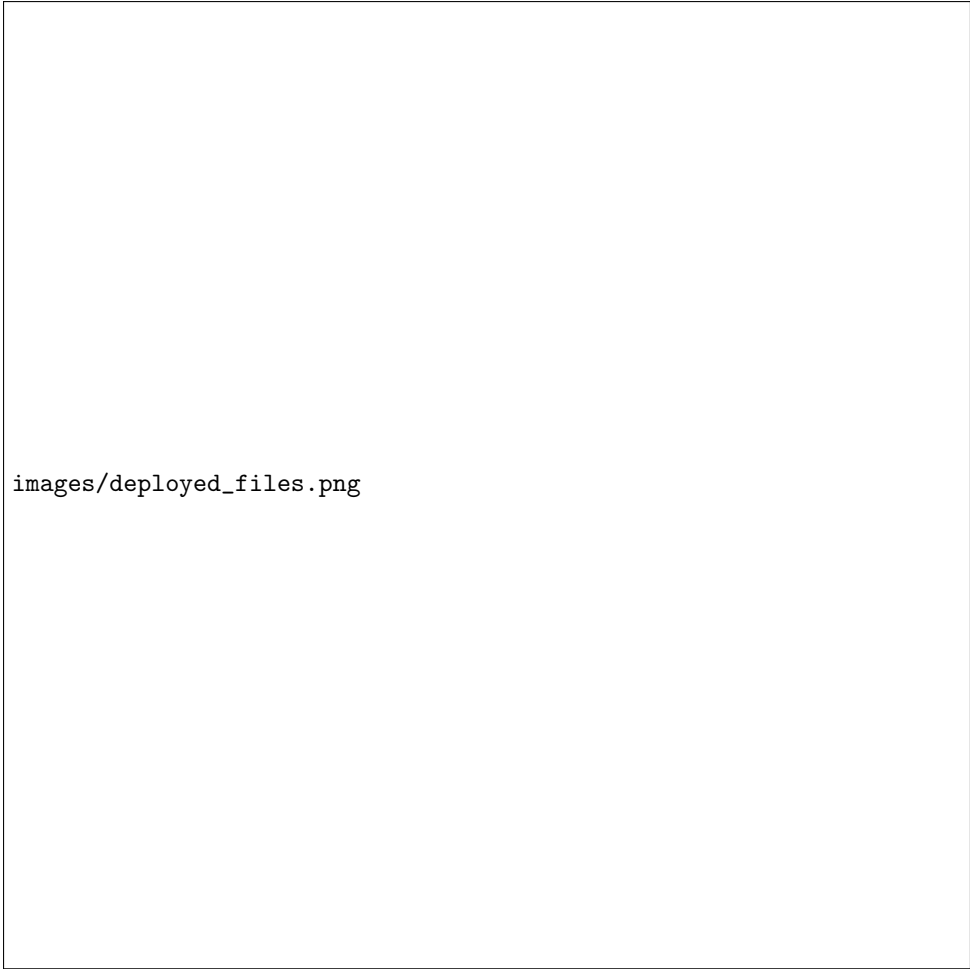
11.1 Problème de permissions sudo

Lors de la première exécution du pipeline, j'ai rencontré une erreur liée aux permissions sudo :

```
1 + sudo apt-get update
2 sudo: a terminal is required to read the password; either use the -S option to read from
   standard input or configure an askpass helper
3 sudo: a password is required
```

Ce problème s'est produit car Jenkins n'a pas le droit d'exécuter des commandes sudo sans mot de passe. Pour résoudre ce problème, j'ai modifié le Jenkinsfile pour éviter l'utilisation de sudo en remplaçant l'étape d'installation d'Ansible par une simple vérification :

```
1 stage('Check Ansible') {
2     steps {
3         // Check if Ansible is installed
4         sh '''
5             if command -v ansible &> /dev/null; then
6                 echo "Ansible is already installed"
7                 ansible --version
8             else
```



images/deployed_files.png

FIGURE 12 – Fichiers déployés dans le répertoire de destination

```
9      echo "Ansible is not installed. Please install it before running this
    pipeline."
10      exit 1
11    fi
12  },
13 }
14 }
```

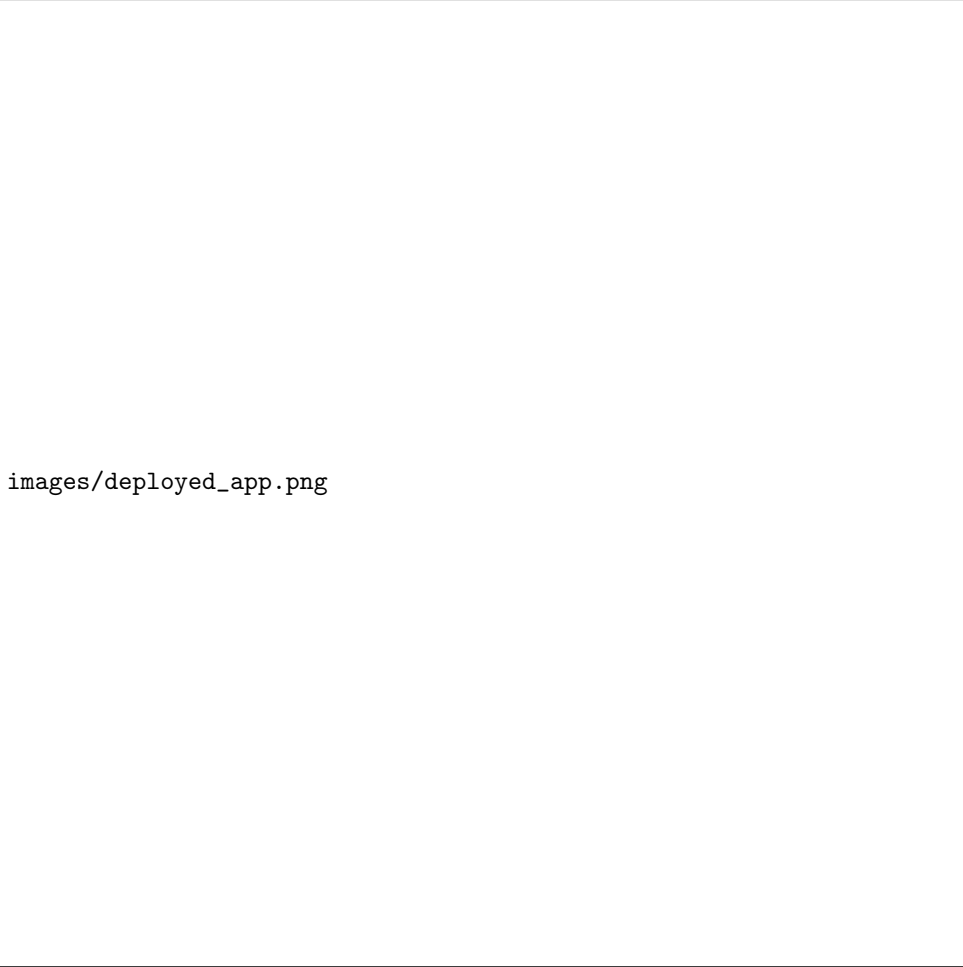
Cette modification présuppose qu'Ansible est déjà installé sur le serveur Jenkins, ce qui était le cas dans mon environnement. Dans un environnement de production, on pourrait envisager les solutions suivantes :

- Préinstaller Ansible sur le serveur Jenkins
- Configurer sudo pour permettre à l'utilisateur Jenkins d'exécuter certaines commandes sans mot de passe
- Utiliser un conteneur Docker comme agent Jenkins avec Ansible préinstallé

11.2 Problème d'hôtes inaccessibles

Après avoir résolu le problème de sudo, j'ai rencontré une autre erreur lors de l'exécution du playbook Ansible :

```
1 TASK [Gathering Facts] *****
2 fatal: [web1.example.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect
  to the host via ssh: ssh: Could not resolve hostname web1.example.com: Name or
  service not known", "unreachable": true}
3 fatal: [web2.example.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect
  to the host via ssh: ssh: Could not resolve hostname web2.example.com: Name or
  service not known", "unreachable": true}
```



images/deployed_app.png

FIGURE 13 – Application web déployée dans le navigateur


Cette erreur était prévisible car les hôtes définis dans le fichier d'inventaire (`web1.example.com` et `web2.example.com`) n'existent pas dans mon environnement de développement. Pour résoudre ce problème, j'ai modifié le fichier `inventory.ini` pour utiliser `localhost` comme cible de déploiement :

```
1 [web]
2 localhost ansible_connection=local
3
4 [db]
5 # db1.example.com ansible_user=ubuntu
6 # db2.example.com ansible_user=ubuntu
7
8 # Variables that will be applied to all servers
9 [all:vars]
10 ansible_python_interpreter=/usr/bin/python3
```

J'ai également modifié le playbook `deploy.yml` pour améliorer sa robustesse :

- Ajout de `become_method: sudo` pour spécifier explicitement la méthode d'élévation de privilèges
- Ajout de `gather_facts: yes` pour s'assurer que les informations sur l'hôte sont collectées
- Ajout de `ignore_errors: yes` pour chaque tâche afin d'éviter que le pipeline ne s'arrête en cas d'erreur mineure
- Répétition explicite de `become: yes` pour chaque tâche pour garantir l'élévation de privilèges

Ces modifications permettent au playbook de s'exécuter correctement sur la machine locale, ce qui est idéal pour un environnement de développement ou de test.



images/github_webhook.png

FIGURE 14 – Configuration du webhook GitHub

12 Conclusion

12.1 Résumé des réalisations

Dans ce TP, j'ai réussi à :

- Installer et configurer Jenkins
- Mettre en place un dépôt GitHub pour le code source
- Créer un pipeline CI/CD avec Jenkins
- Configurer Ansible pour le déploiement automatisé
- Déployer une application web sur un serveur Apache
- Configurer un mécanisme de déclenchement automatique via webhook

12.2 Avantages de l'intégration Jenkins-Ansible

L'intégration de Jenkins avec Ansible offre plusieurs avantages :

- **Automatisation complète** : Le processus de déploiement est entièrement automatisé, de l'extraction du code à sa mise en production.
- **Reproductibilité** : Les déploiements sont reproductibles et consistants grâce à la définition déclarative des tâches Ansible.
- **Gestion de la configuration** : Ansible permet de gérer efficacement la configuration des serveurs cibles.
- **Évolutivité** : Le même pipeline peut être utilisé pour déployer sur un ou plusieurs serveurs sans modification majeure.
- **Traçabilité** : Jenkins conserve un historique des déploiements et des logs, facilitant le diagnostic des problèmes.

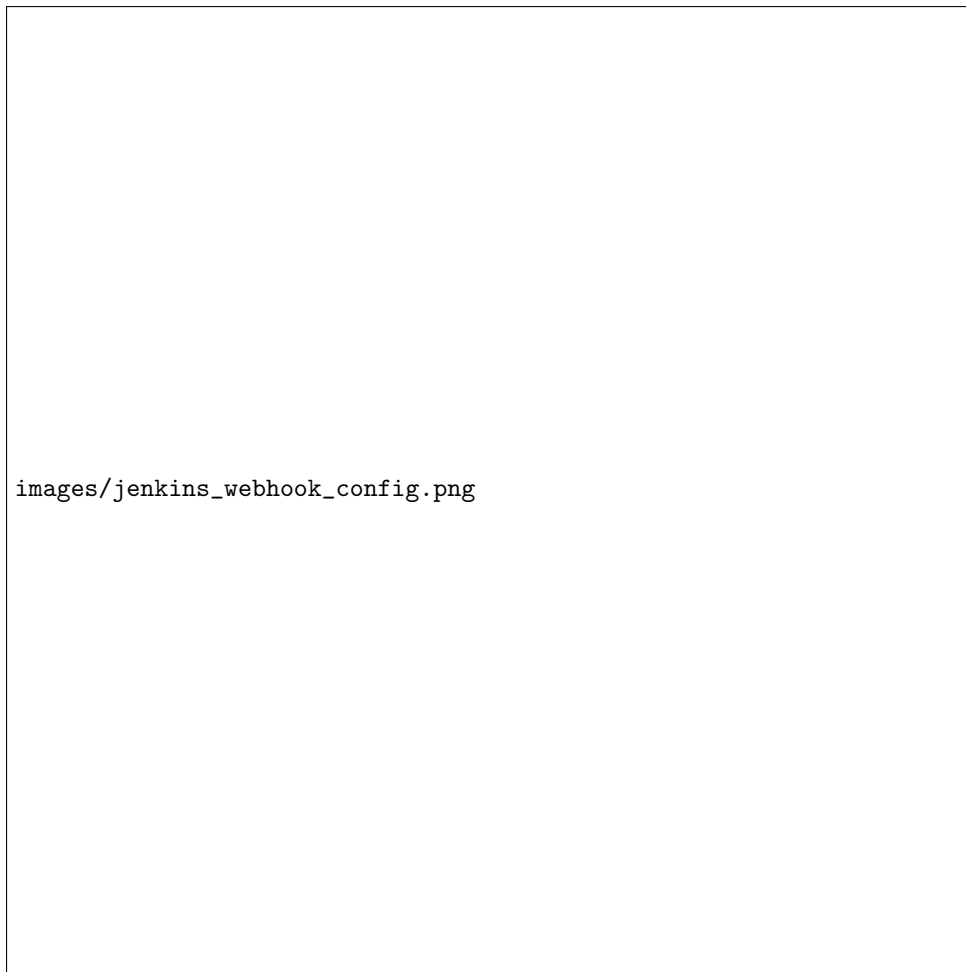


FIGURE 15 – Configuration du déclencheur de build dans Jenkins

12.3 Améliorations possibles

Ce projet pourrait être amélioré de plusieurs façons :

- Ajouter des étapes de tests automatisés avant le déploiement
- Implémenter une stratégie de déploiement bleu-vert ou canary
- Configurer des notifications (email, Slack) lors des déploiements
- Mettre en place un mécanisme de rollback automatique en cas d'échec
- Utiliser des variables d'environnement pour une meilleure paramétrisation

12.4 Conclusion personnelle

Ce TP m'a permis de comprendre et d'appliquer les concepts d'intégration continue et de déploiement continu (CI/CD) en utilisant des outils modernes et très répandus dans l'industrie. J'ai pu constater l'efficacité d'une chaîne d'automatisation pour simplifier et fiabiliser le processus de déploiement logiciel. Les compétences acquises pendant cette session pourront être directement appliquées dans un contexte professionnel de DevOps.



images/jenkins_pipeline_success.png

FIGURE 16 – Pipeline Jenkins après résolution des problèmes (exécution réussie)