

DevOps

Les deux parties du TP Docker et Jenkins

Nom : SOULEIMAN MAHAMOUD

Prénom : Choukri

- Nous allons créer nos deux fichiers :

Fichier Sum.py :

```
C: > Users > chouk > Desktop > DevOps > Jenkins > sum.py
1  import sys
2
3  # Vérifier si deux arguments sont fournis
4  if len(sys.argv) != 3:
5      print("Erreur : Deux arguments sont nécessaires.")
6      sys.exit(1)
7
8  # Interpréter les arguments
9  try:
10     arg1 = float(sys.argv[1])
11     arg2 = float(sys.argv[2])
12 except ValueError:
13     print("Erreur : Les arguments doivent être des nombres.")
14     sys.exit(1)
15
16 # Calculer la somme
17 resultat = arg1 + arg2
18
19 # Afficher le résultat
20 print(resultat)
```

Fichiers Dockerfile sans extension :

```
C: > Users > chouk > Desktop > DevOps > Jenkins > Dockerfile
1  # Utiliser une image de base Python alpine
2  FROM python:3.13.0-alpine3.20
3
4  # Définir le répertoire de travail
5  WORKDIR /app
6
7  # Copier le fichier sum.py dans le conteneur
8  COPY sum.py /app/sum.py
9
10 # Rendre le script exécutable
11 ENTRYPOINT ["python", "/app/sum.py"]
12
13
14 # Commande par défaut pour garder le conteneur actif
15 CMD ["tail", "-f", "/dev/null"]
```

- Nous allons créer un dossier sur GitHub et pousser nos deux fichiers sur ce dossier :

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk (*).

Owner *  ChoukriSOULEIMAN / Repository name * TP-Jenkins
TP-Jenkins is available.

Great repository names are short and memorable. Need inspiration? How about [expert-octo-goggles](#) ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

- Voici les commandes recommander pour pousser nos fichiers sur GitHub :

```
Command Prompt

C:\Users\chouk\Desktop>cd devops
C:\Users\chouk\Desktop\DevOps>cd jenkins
C:\Users\chouk\Desktop\DevOps\Jenkins>git init
Initialized empty Git repository in C:/Users/chouk/Desktop/DevOps/Jenkins/.git/
C:\Users\chouk\Desktop\DevOps\Jenkins>git add .
C:\Users\chouk\Desktop\DevOps\Jenkins>git commit -m "Initial commit pour ajouter un fichiers sum.py et Dockerfile"
[master (root-commit) a219fc2] Initial commit pour ajouter un fichiers sum.py et Dockerfile
 2 files changed, 35 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 sum.py
C:\Users\chouk\Desktop\DevOps\Jenkins>git remote add origin https://github.com/ChoukriSOULEIMAN/TP-Jenkins.git
C:\Users\chouk\Desktop\DevOps\Jenkins>git push -u origin main
error: src refspec main does not match any
error: failed to push some refs to 'https://github.com/ChoukriSOULEIMAN/TP-Jenkins.git'
C:\Users\chouk\Desktop\DevOps\Jenkins>git branch
* master
C:\Users\chouk\Desktop\DevOps\Jenkins>git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 797 bytes | 265.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChoukriSOULEIMAN/TP-Jenkins.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
C:\Users\chouk\Desktop\DevOps\Jenkins>
```

➤ Aperçue :

The screenshot shows a GitHub repository for 'TP-Jenkins' (Public). The repository has 1 branch (master) and 0 tags. It was created by ChoukriSOULEIMAN. The commit history shows an initial commit 35 minutes ago with files Dockerfile and sum.py. The repository has no description, website, or topics provided. The right sidebar shows activity, 0 stars, 1 watching, 0 forks, and no releases published.

Q2 : Nous allons construction notre l'image Docker.

```
C:\Users\chouk\Desktop\DevOps\Jenkins>docker build -t python-sum-app .
[+] Building 15.3s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                0.9s
=> => transferring dockerfile: 403B                                              0.2s
=> [internal] load metadata for docker.io/library/python:3.13.0-alpine3.20      3.5s
=> [auth] library/python:pull token for registry-1.docker.io                   0.0s
=> [internal] load .dockerignore                                                 0.3s
=> => transferring context: 2B                                                  0.0s
=> [1/3] FROM docker.io/library/python:3.13.0-alpine3.20@sha256:fcbbecdeae71d3b77445d9144d1914df55110f825a 0.9s
=> => resolve docker.io/library/python:3.13.0-alpine3.20@sha256:fcbbecdeae71d3b77445d9144d1914df55110f825a 0.8s
=> [internal] load build context                                                0.6s
=> => transferring context: 481B                                              0.1s
=> CACHED [2/3] WORKDIR /app                                                    0.0s
=> [3/3] COPY sum.py /app/sum.py                                               1.1s
=> exporting to image                                                            5.7s
=> => exporting layers                                                         2.6s
=> => exporting manifest sha256:ea61a83c234c76777f7c83d31781b02674cad7ea6e9946dff6a9024bb46ee2f 0.4s
=> => exporting config sha256:75a21893cfbc0201e93ab2e331a5ca915b1811b8e01b717512b38a9b33c7c593 0.3s
=> => exporting attestation manifest sha256:95ee1f81c6ae82077b3a0c786d86441bd8d90cc301265b8b3249bc04137b20b0 0.8s
=> => exporting manifest list sha256:cd216b7208cdbba932a660a68403d0f91bbc11d1862c6f223dbd618ad444a68e 0.4s
=> => naming to docker.io/library/python-sum-app:latest                      0.1s
=> => unpacking to docker.io/library/python-sum-app:latest                   0.7s

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\chouk\Desktop\DevOps\Jenkins>
```

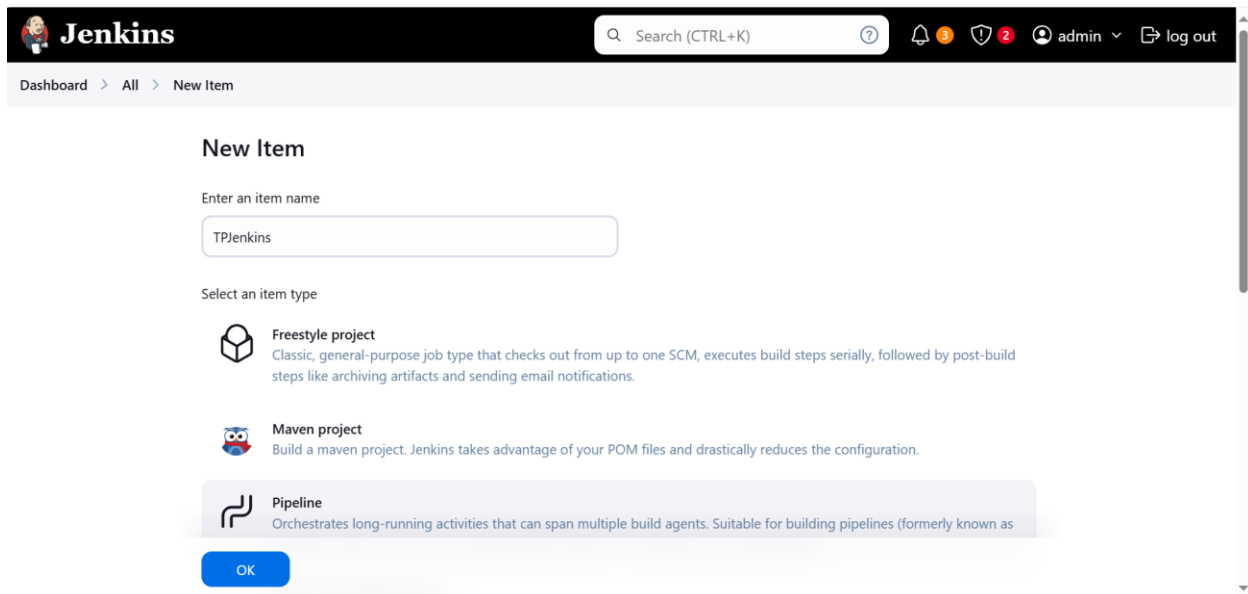
Q3 : Exécution du conteneur Docker

```
C:\Users\chouk\Desktop\DevOps\Jenkins>docker run -it --entrypoint python python-sum-app /app/sum.py 5 10
15.0

C:\Users\chouk\Desktop\DevOps\Jenkins>
```

Partie 2 : Création du Jenkinsfile

- Dans Jenkins, on crée un nouveau fichier :



Jenkins

Search (CTRL+K)

admin log out

Dashboard > All > New Item

New Item

Enter an item name

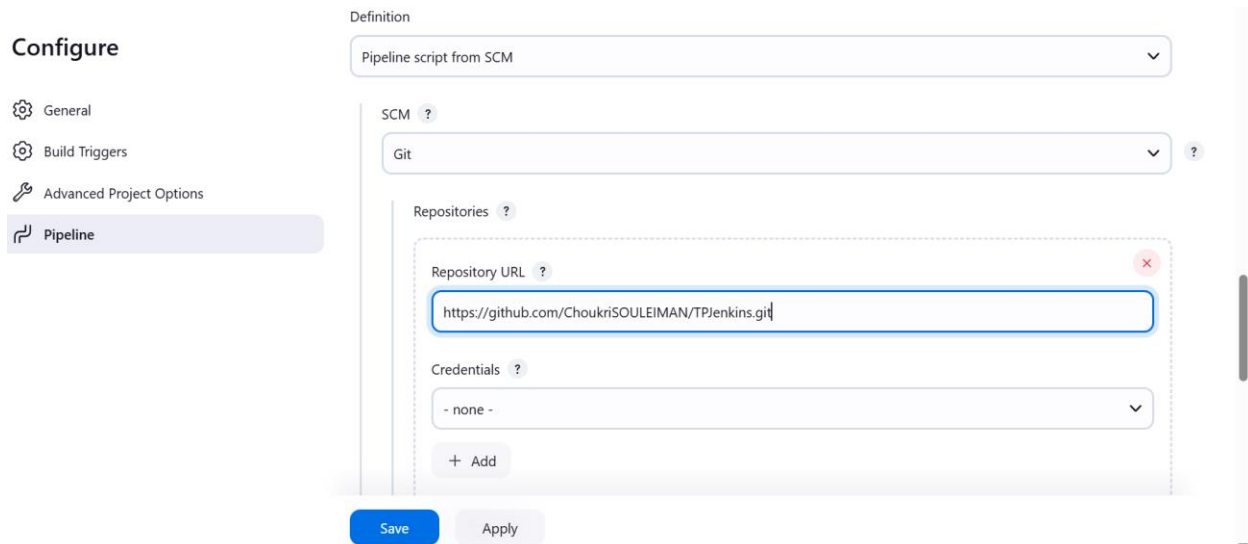
TPJenkins

Select an item type

- Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as

OK

- Puis on le configure comme suite :



Configure

- General
- Build Triggers
- Advanced Project Options
- Pipeline**

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/ChoukriSOULEIMAN/TPJenkins.git

Credentials ?

- none -

+ Add

Save Apply

➤ Et puis après on le pousse sur GitHub :

```
C:\Users\chouk\Desktop\DevOps\Jenkins>git init
Initialized empty Git repository in C:/Users/chouk/Desktop/DevOps/Jenkins/.git/

C:\Users\chouk\Desktop\DevOps\Jenkins>git add .

C:\Users\chouk\Desktop\DevOps\Jenkins>git commit -m "Ajoute de fichiers"
[master (root-commit) decd23a] Ajoute de fichiers
 4 files changed, 96 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 Jenkinsfile
 create mode 100644 sum.py
 create mode 100644 teste_variables.txt

C:\Users\chouk\Desktop\DevOps\Jenkins>git remote add origin https://github.com/ChoukriSOULEIMAN/TPJenkins.git

C:\Users\chouk\Desktop\DevOps\Jenkins>git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.39 KiB | 285.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ChoukriSOULEIMAN/TPJenkins.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

C:\Users\chouk\Desktop\DevOps\Jenkins>
```

➤ Dossier bien déposer :

The screenshot shows the GitHub interface for a repository named 'TPJenkins' owned by 'ChoukriSOULEIMAN'. The repository is public and has 1 branch (master) and 0 tags. The commit history shows a single commit 'Ajoute de fichiers' by 'ChoukriSOULEIMAN' 1 minute ago, with 1 commit in total. The commit details show four files added: 'Dockerfile', 'Jenkinsfile', 'sum.py', and 'teste_variables.txt', each with a message 'Ajoute de fichiers' and a timestamp of '1 minute ago'. Below the commit list, there is a section for 'README' with a button 'Add a README'. On the right side, there are sections for 'About' (no description), 'Releases' (no releases published), 'Packages' (no packages published), and 'Languages' (Python 55.0%, Dockerfile 45.0%).


- Pour la 1^{ère} étape on a rencontré une erreur donc à chaque fois qu'on modifier nos fichiers on le repousse sur GitHub :

```
C:\Users\chouk\Desktop\DevOps\Jenkins>git add .


C:\Users\chouk\Desktop\DevOps\Jenkins>git commit -m "fichiers modifier"
[master 155ed28] fichiers modifier
1 file changed, 11 insertions(+), 7 deletions(-)





C:\Users\chouk\Desktop\DevOps\Jenkins>git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 444 bytes | 222.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ChoukriSOULEIMAN/TPJenkins.git
   decd23a..155ed28  master -> master

C:\Users\chouk\Desktop\DevOps\Jenkins>
```

 **TPJenkins** Public Pin Unwatch 1

🔗 master 1 Branch 0 Tags Add file <> Code

 **ChoukriSOULEIMAN** fichiers modifier 155ed28 · 1 minute ago 2 Commits

 Dockerfile	Ajoute de fichiers	30 minutes ago
 Jenkinsfile	fichiers modifier	1 minute ago
 sum.py	Ajoute de fichiers	30 minutes ago
 teste_variables.txt	Ajoute de fichiers	30 minutes ago

1. Définition des variables d'environnement :

```
C: > Users > chouk > Desktop > DevOps > Jenkins > Jenkinsfile

1 pipeline {
2     agent any
3     environment {
4         SUM_PY_PATH = '/app/sum.py'
5         DIR_PATH = '.'
6         TEST_FILE_PATH = 'teste_variables.txt'
7         CONTAINER_NAME = 'python-container'
8         IMAGE_NAME = 'python-sum-app'
9         DOCKER_USERNAME = 'choukri34'
10        DOCKER_PASSWORD = 'choukri34'
11    }
12    stages {
13        stage('Build') {
14            steps {
15                script {
16                    echo "Construction de l'image Docker"
17                    bat "docker build -t ${IMAGE_NAME} ${DIR_PATH}"
18                }
19            }
20        }
21    }
22 }
```

➤ Résultat fonctionne bien maintenant :



2. Stage "Build":

Cette étape construit l'image Docker en utilisant le Dockerfile dans le répertoire spécifié.

```
12     stages {
13         stage('Build') {
14             steps {
15                 script {
16                     echo "Construction de l'image Docker"
17                     bat "docker build -t ${IMAGE_NAME} ${DIR_PATH}"
18                 }
19             }
20         }
21     }
```

3. Stage "Run" :

Cette étape lance un conteneur Docker en arrière-plan, et récupère son ID pour l'utiliser dans les étapes suivantes.

```
24     // L'étape d'exécution du conteneur Docker
25     stage('Run') {
26         steps {
27             script {
28                 // Exécution du conteneur Docker
29                 echo "Running Docker container..."
30                 def output = sh(script: "docker run -d python-sum-app", returnStdout: true).trim()
31                 CONTAINER_ID = output
32                 echo "Container ID: ${CONTAINER_ID}"
33             }
34         }
35     }
36 }
```


4. Stage "Test" :

Cette étape lit le fichier `teste_variable.txt` ligne par ligne, exécute le script Python `sum.py` dans le conteneur Docker avec les arguments appropriés, et compare le résultat avec la somme attendue. Si la somme est correcte, le test est passé. Sinon, l'exécution échoue et génère une erreur.

```
37 // Ici l'étape de test
38 stage('Test') {
39     steps {
40         script {
41             // Lecture du fichier test_variables.txt
42             def testLines = readFile(TEST_FILE_PATH).split('\n')
43             testLines.each { line ->
44                 def vars = line.split(' ')
45                 def arg1 = vars[0]
46                 def arg2 = vars[1]
47                 def expectedSum = vars[2].toFloat()
48
49                 // Exécution du script sum.py dans le conteneur Docker
50                 def output = sh(script: "docker exec ${CONTAINER_ID} python /app/sum.py ${arg1} ${arg2}", returnStdout: true).trim()
51                 def result = output.toFloat()
52
53                 // Vérification si la somme est correcte
54                 if (result == expectedSum) {
55                     echo "Test passed for ${arg1} + ${arg2}: expected ${expectedSum}, got ${result}"
56                 } else {
57                     error "Test failed for ${arg1} + ${arg2}: expected ${expectedSum}, got ${result}"
58                 }
59             }
60         }
61     }
62 }
```

5. Stage "Cleanup" :

Cette étape assure l'arrêt et la suppression du conteneur Docker après l'exécution des tests.

```
96     post {
97         always {
98             // Cette section est exécutée après chaque pipeline, qu'il réussisse ou échoue
99             echo 'Pipeline completed!'
100         }
101     }
102 }
103 }
```

6. Stage "Deploy to DockerHub" :

Cette étape se connecte à DockerHub (assurez-vous que vous avez configuré les identifiants DockerHub comme variables d'environnement dans Jenkins), tague l'image Docker et la pousse vers DockerHub.

```
76 // L'étape de déploiement sur DockerHub
77 stage('Deploy to DockerHub') {
78     steps {
79         script {
80             // Connexion à DockerHub
81             echo "Logging in to DockerHub..."
82             sh "docker login -u ${DOCKER_USERNAME} -p ${DOCKER_PASSWORD}"
83
84             // Taguer l'image Docker
85             echo "Tagging Docker image..."
86             sh "docker tag python-sum-app ${DOCKER_USERNAME}/python-sum-app:latest"
87
88             // Pousser l'image vers DockerHub
89             echo "Pushing Docker image to DockerHub..."
90             sh "docker push ${DOCKER_USERNAME}/python-sum-app:latest"
91         }
92     }
93 }
94 }
```

Partie 3 : Analyse de Performance avec Docker Stats

- Pour cette partie, nous allons surveiller l'utilisation des ressources du conteneur Docker à l'aide de la commande docker stats.

```
C:\Users> choux > Desktop > DevOps > Jenkins > Jenkinsfile
1 pipeline {
2     agent any
3     environment {
4         SUM_PY_PATH = '/app/sum.py'
5         DIR_PATH = '.'
6         TEST_FILE_PATH = 'teste_variables.txt'
7         CONTAINER_NAME = 'python-container'
8         IMAGE_NAME = 'python-sum-app'
9         DOCKER_USERNAME = 'choukri34'
10        DOCKER_PASSWORD = 'choukri34'
11    }
12    stages {
13        stage('Build') {
14            steps {
15                script {
16                    echo "Construction de l'image Docker"
17                    bat "docker build -t ${IMAGE_NAME} ${DIR_PATH}"
18                }
19            }
20        }
21    }
22    stage('Performance Analysis') {
23        steps {
24            echo 'Analyzing container performance...'
25            // Capturer les statistiques d'utilisation des ressources
26            sh "docker stats ${DOCKER_CONTAINER} --no-stream"
27        }
28    }
29 }
```

En un mot, la commande docker stats --no-stream capture un instantané de l'utilisation des ressources (CPU, mémoire). Cette étape nous aide à surveiller et identifier d'éventuels goulets d'étranglement.

Partie 4 : Génération Automatique de Documentation avec Sphinx

Étape 1 : Configuration de Sphinx dans l'Environnement Docker

- Modifions le Dockerfile pour installer Sphinx.

```
C: > Users > chouk > Desktop > DevOps > Jenkins > Dockerfile
1 FROM python:3.9-slim
2
3 # Installation des dépendances Python
4 RUN pip install sphinx
5
6 # Copier le script Python dans le conteneur
7 COPY sum.py /app/sum.py
8 WORKDIR /app
9
10 # Ici nous allons initialiser le projet Sphinx
11 RUN sphinx-quickstart -q -p "TPJenkins" -a "Auteur" -v "1.0" --sep -r 1.0
12
13 # Et enfin installer sphinx-autodoc pour la documentation automatique
14 RUN pip install sphinx-autodoc-typehints
15
16 CMD ["python3", "sum.py"]
17
```

Notre projet Sphinx est initialisé avec sphinx-quickstart. Et sphinx-autodoc-typehints est installé pour générer la documentation à partir des annotations de type Python.

Étape 2 : Écriture de la Documentation avec Sphinx

- Nous allons ajouter des docstrings dans le script sum.py en utilisant le format reStructuredText compatible avec Sphinx.

```
C: > Users > chouk > Desktop > DevOps > Jenkins > sum.py
1 def additionner(arg1, arg2):
2     """
3     Additionne deux nombres et retourne le résultat.
4
5     :param arg1: Premier nombre à additionner (float)
6     :param arg2: Deuxième nombre à additionner (float)
7     :return: La somme de arg1 et arg2 (float)
8     """
9     return arg1 + arg2
10
11
12 if __name__ == "__main__":
13     result = additionner(5, 10)
14     print(f"Le résultat est : {result}")
15
```

Étape 3 : Génération Automatique de la Documentation dans le Pipeline Jenkins

- Ajoutez un stage pour la génération de documentation dans votre Jenkinsfile.

```
29     stage('Documentation') {
30         steps {
31             echo 'Generating Sphinx documentation...'
32             // Générer la documentation avec Sphinx
33             sh 'docker exec ${DOCKER_CONTAINER} sphinx-apidoc -o source .'
34             sh 'docker exec ${DOCKER_CONTAINER} make html'
35             // Archiver la documentation
36             archiveArtifacts artifacts: 'build/html/**', allowEmptyArchive: true
37         }
38     }
```

Les commandes utiliser va nous servir à :

- **sphinx-apidoc -o source .** : permet de générer automatiquement les fichiers **.rst** pour la documentation.
- **make html** : construit la documentation en format HTML.
- **archiveArtifacts** : archive les fichiers générés pour consultation dans Jenkins.

Conclusion :

On a compris que ce pipeline Jenkins nous permet de :

- Surveiller l'utilisation des ressources du conteneur Docker pour détecter d'éventuels problèmes de performance.
- Générer automatiquement une documentation lisible et bien structurée pour votre projet Python avec Sphinx.
- Archiver la documentation pour une consultation facile dans Jenkins.

Cette approche améliore la qualité et la maintenabilité de notre projet, tout en fournissant des outils pour l'analyse de performance et la documentation.