Intégration de GitHub avec Joget, SonarQube, Nexus et Automatisation via Docker

Étape 1 : Installation d'OpenSSH Server pour l'utilisation de Terminus

1. **Ouvrir un terminal** sur votre machine virtuelle Ubuntu.
2. **Mettre à jour la liste des paquets** :

   ```
   sudo apt update
   ```

3. **Installer OpenSSH Server** :

   ```
   sudo apt install openssh-server -y
   ```

4. **Vérifier l'état du service SSH** :

   ```
   sudo systemctl status ssh
   ```

   - Si le service est **actif (running)**, l'installation est réussie.
   - Si SSH n'est pas en cours d'exécution, le démarrer :

     ```
     sudo systemctl start ssh
     ```

   - Activer le démarrage automatique de SSH au boot :

     ```
     sudo systemctl enable ssh
     ```

Étape 2 : Prérequis - Installation de Docker et Docker Compose

Ajout du dépôt Docker

1. **Ajouter la clé GPG officielle de Docker** et configurer le dépôt :

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

2. **Ajouter le dépôt Docker aux sources APT** :

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Installation des paquets Docker

Pour installer la dernière version de Docker, exécutez :

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y
```

Vérification des versions installées

```
docker --version
docker compose version
```

Étape 3 : Déploiement des Conteneurs Jenkins, SonarQube et Nexus avec Docker Compose

1. **Créer un fichier** `docker-compose.yml` et y ajouter la configuration suivante :

```
version: '3'
services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    ports:
      - "8081:8080"
      - "50000:50000"
    volumes:
      - jenkins_home:/var/jenkins_home

  sonarqube:
    image: sonarqube:latest
    container_name: sonarqube
    ports:
      - "9000:9000"
    environment:
      - SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true

  nexus:
    image: sonatype/nexus3
    container_name: nexus
    ports:
      - "8082:8081"
      - "8085:8085"

volumes:
  jenkins_home:
```

2. **Lancer les conteneurs en arrière-plan** :

```
docker compose up -d
```

Récupération des mots de passe par défaut

- **Jenkins** :(Chercher la ligne indiquant le mot de passe initial)

```
docker logs jenkins
```

- **SonarQube** :Utilisateur par défaut : `admin`, Mot de passe par défaut : `admin`
- **Nexus** :

```
docker exec -it nexus cat /nexus-data/admin.password
```

Étape 4 : Gestion des Branches GitHub dans un Projet Joget

1. Initialisation du Projet et Configuration GitHub

1.1 Création du projet et du référentiel

1. Créer un projet **Joget** nommé `my_app`.
2. Créer un dépôt **GitHub** du même nom (`my_app`).
3. Partager un **Token d'Authentification** avec l'équipe via un canal privé (ex: Slack, Teams).

1.2 Configuration de Git dans Joget

Dans Joget, aller dans :

Settings → Git Config

Remplir les champs suivants :

- **URL du dépôt GitHub** : `https://github.com/mon-org/my_app.git`
- **Nom d'utilisateur GitHub**

- **Token d’accès personnel** (dans le champ mot de passe)

---

2. Mise en place des branches Git

2.1 Création de la branche principale (`main`)

Vérifier que la branche `main` existe bien :

```
git checkout -b main
git push origin main
```

Si elle n’existe pas encore, la renommer :

```
git branch -M main
git push -u origin main
```

2.2 Structuration des branches

Chaque développeur doit travailler sur une branche spécifique issue de `main` :

```
git checkout -b my_app_2 main
git push origin my_app_2

git checkout -b my_app_3 main
git push origin my_app_3
```

Étape 5 : Extraction des Codes

Script 1 : Extraction des Codes Java et SQL

Objectif

Ce script est conçu pour extraire les morceaux de code Java et SQL des fichiers JSON et XML d'une application **Joget**.

Il analyse les fichiers de formulaires JSON ainsi que le fichier `appDefinition.xml` pour détecter et extraire tout code pertinent, puis les enregistre dans des fichiers distincts.

Explication du Code

1️⃣ Définition des Répertoires et Chemins

```
BASE_DIR = "/opt/joget/wflow/scripts/output"
FORMS_DIR = "/opt/joget/wflow/app_src/vente_voitures/vente_voitures_1/forms"
APPDEF_FILE = "/opt/joget/wflow/app_src/vente_voitures/vente_voitures_1/appDefinition.xml"
JAVA_DIR = os.path.join(BASE_DIR, "java")
SQL_DIR = os.path.join(BASE_DIR, "sql")
```

- **BASE_DIR** : Dossier où les fichiers extraits seront stockés.
- **FORMS_DIR** : Contient les formulaires de l'application au format JSON.
- **APPDEF_FILE** : Contient la définition de l'application avec du code Java et SQL.
- **JAVA_DIR** et **SQL_DIR** : Sous-dossiers où seront stockés respectivement les codes **Java** et **SQL**.

2️⃣ Création des Dossiers de Sortie

```
os.makedirs(JAVA_DIR, exist_ok=True)
os.makedirs(SQL_DIR, exist_ok=True)
```

- Vérifie si les dossiers existent, sinon les crée pour stocker les fichiers extraits.

3️⃣ Définition des Expressions Régulières

```
JAVA_PATTERN = re.compile(r"\b(import java|public class|void main|System\.out\.println|PreparedStatement|Connection)\b")
```

```
SQL_PATTERN = re.compile(r"\b(SELECT|INSERT|UPDATE|DELETE|CREATE|ALTER|DROP|TRUNCATE|FROM|WHERE)\b", re.IGNORECASE)
```

- **JAVA_PATTERN** : Identifie les scripts Java en recherchant des mots-clés Java.
- **SQL_PATTERN** : Identifie les requêtes SQL en détectant des mots-clés SQL comme `SELECT`, `INSERT`, etc.

4️⃣ Suppression des Fichiers Anciens

```
def clear_output_directory():
    for directory in [JAVA_DIR, SQL_DIR]:
        if os.path.exists(directory):
            for file in os.listdir(directory):
                file_path = os.path.join(directory, file)
                try:
                    os.remove(file_path)
                except Exception as e:
                    print(f"⚠️ Impossible de supprimer {file_path}: {e}")
```

- Supprime les fichiers précédemment générés pour éviter les doublons.

5️⃣ Extraction et Enregistrement des Codes

```
def save_code(directory, base_name, extension, content, index):
    file_name = f"{base_name}_{index}.{extension}"
    file_path = os.path.join(directory, file_name)

    with open(file_path, "w", encoding="utf-8") as f:
        f.write(content)
    print(f"✅ Fichier généré : {file_path}")
```

- **Crée un fichier** pour chaque extrait de code trouvé et l'enregistre sous un nom unique.

6️⃣ Extraction des Codes des Fichiers JSON

```
def extract_code_from_json(file_path):
    with open(file_path, "r", encoding="utf-8") as f:
        try:
            data = json.load(f)
        except json.JSONDecodeError:
            print(f"❌ ERREUR: Impossible de lire {file_path}")
            return
```

- Ouvre et charge le fichier JSON.
- Gère les erreurs en cas de JSON mal formé.

```
    extracted_java = []
    extracted_sql = []

    def search_code(element):
        if isinstance(element, dict):
            for key, value in element.items():
                if key == "script" and isinstance(value, str):
                    if JAVA_PATTERN.search(value):
                        extracted_java.append(value.strip())
                    elif SQL_PATTERN.search(value):
                        extracted_sql.append(value.strip())
                elif isinstance(value, (dict, list)):
                    search_code(value)
```

- **Recherche récursive** dans le JSON pour extraire les scripts Java et SQL.

7️⃣ Extraction des Codes de `appDefinition.xml`

```
def extract_code_from_appdefinition():
    if not os.path.exists(APPDEF_FILE):
        print(f"❌ ERREUR : Fichier {APPDEF_FILE} introuvable")
```

```
        return

    tree = ET.parse(APPDEF_FILE)
    root = tree.getroot()

    extracted_java = []
    extracted_sql = []

    for elem in root.iter("pluginProperties"):
        raw_code = elem.text
        if raw_code:
            if JAVA_PATTERN.search(raw_code):
                extracted_java.append(raw_code.strip())
            elif SQL_PATTERN.search(raw_code):
                extracted_sql.append(raw_code.strip())
```

- Parcourt **appDefinition.xml** pour trouver les propriétés contenant du code Java ou SQL.

8️⃣ Exécution du Script

```
clear_output_directory()

for filename in os.listdir(FORMS_DIR):
    if filename.endswith(".json"):
        extract_code_from_json(os.path.join(FORMS_DIR, filename))

extract_code_from_appdefinition()
```

- Nettoie les anciens fichiers, puis exécute l'extraction sur tous les fichiers.

Script 2 : Extraction des Scripts JavaScript

Objectif

Ce script analyse les fichiers JSON des formulaires **Joget** pour extraire les scripts **JavaScript** intégrés.

Explication du Code

1️⃣ Définition des Chemins

```
FORMS_DIR = "/opt/joget/wflow/app_src/vente_voitures/vente_voitures_1/forms"
OUTPUT_JS_DIR = "/opt/joget/wflow/scripts/output/js"
```

- **FORMS_DIR** : Dossier contenant les fichiers JSON des formulaires.
- **OUTPUT_JS_DIR** : Dossier où seront stockés les fichiers JS extraits.

2️⃣ Création du Dossier de Sortie

```
os.makedirs(OUTPUT_JS_DIR, exist_ok=True)
```

- **Crée le dossier** `js` si inexistant.

3️⃣ Suppression des Anciens Scripts

```
for file in os.listdir(OUTPUT_JS_DIR):
    file_path = os.path.join(OUTPUT_JS_DIR, file)
    try:
        os.remove(file_path)
        print(f"🗑️ Supprimé : {file_path}")
    except Exception as e:
        print(f"⚠️ Impossible de supprimer {file_path}: {e}")
```

- Supprime tous les anciens fichiers JS.

4️⃣ Extraction des Scripts JavaScript

```
def extract_js_from_json(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            data = json.load(f)

        scripts = []

        def search_elements(elements):
            if isinstance(elements, list):
                for element in elements:
                    search_elements(element)
            elif isinstance(elements, dict):
                if elements.get("className") == "org.joget.apps.form.lib.CustomHT
ML":
                    js_code = elements.get("properties", {}).get("value", "")
                    if "<script>" in js_code:
                        clean_js = re.sub(r"<\/?script.*?>", "", js_code, flags=re.DOTAL
L).strip()
                        scripts.append(clean_js)
                for key, value in elements.items():
                    if isinstance(value, (dict, list)):
                        search_elements(value)

        search_elements(data.get("elements", []))

        return scripts
    except json.JSONDecodeError as e:
        print(f"❌ Erreur JSON dans {file_path}: {e}")
        return []
```

- Recherche **récursivement** dans le JSON des scripts JavaScript, en les nettoyant des balises `<script>`.

5️⃣ Sauvegarde des Scripts JS

```
for form_file in os.listdir(FORMS_DIR):
    if form_file.endswith(".json"):
        form_path = os.path.join(FORMS_DIR, form_file)
        extracted_scripts = extract_js_from_json(form_path)

        if extracted_scripts:
            for index, script_content in enumerate(extracted_scripts, start=1):
                script_filename = f"{form_file.replace('.json', '')}_script_{index}.js"
                script_path = os.path.join(OUTPUT_JS_DIR, script_filename)

                with open(script_path, "w", encoding="utf-8") as script_file:
                    script_file.write(script_content)

                print(f"✅ Fichier JS extrait : {script_path}")
```

- **Stocke chaque script extrait** sous forme de fichier `.js`.

Structure des Répertoires

Avant d'exécuter les scripts, assurez-vous que l'arborescence suivante est respectée dans votre conteneur **Joget**.

```
/opt/joget/wflow/
├── app_src/
│   └── vente_voitures/
│       └── vente_voitures_1/
│           ├── forms/              # Contient les fichiers JSON des formulaires
│           │   ├── form1.json
│           │   ├── form2.json
│           │   └── ...
│           ├── appDefinition.xml     # Fichier contenant la définition de l'applica
tion
│
└── scripts/
    ├── extract_code.py             # Script pour extraire Java et SQL
```

```
├── extract_js.py          # Script pour extraire JavaScript
├── output/                # Dossier contenant les fichiers générés
│   ├── java/
│   ├── sql/
│   ├── js/
│   └── ...
```

Déploiement des Scripts

1. **Copiez les fichiers `extract_code.py` et `extract_js.py` dans le répertoire `/opt/joget/wflow/scripts/`.**

   ```
   docker exec -it jogetapp
   cd /opt/joget/wflow/scripts
   ```

extract_code.py

extract_js.py

Étape 6 : Configuration du Conteneur Joget (`jogetapp`)

Avant d'exécuter les scripts et d'automatiser l'extraction des codes, il est essentiel d'installer certains outils et de configurer l'environnement dans le conteneur **Joget (`jogetapp`)**.

1️⃣ Accéder au Conteneur Joget

Tout d'abord, ouvrez un terminal et entrez dans le conteneur **Joget** en exécutant :

```
docker exec -it jogetapp
```

Vous serez maintenant dans l'environnement du conteneur.

2️⃣ Mise à Jour et Installation des Outils Nécessaires

Dans le conteneur, exécutez les commandes suivantes :

Mettre à jour les paquets existants

```
apt update && apt upgrade -y
```

Installer les outils de base

```
apt install -y wget unzip 3 3-pip nano vim
```

- `wget` : Permet de télécharger des fichiers à partir d'Internet.
- `unzip` : Utilisé pour extraire des fichiers compressés.
- `3` : Langage requis pour exécuter les scripts d'extraction.
- `3-pip` : Gestionnaire de paquets (utile si des dépendances doivent être installées).
- `nano` et `vim` : Éditeurs de texte pour modifier des fichiers si nécessaire.

Étape 7 : Analyse de Code avec SonarQube

L'objectif de cette étape est d'installer et configurer **SonarQube Scanner**, d'activer l'analyse de code pour Java, SQL et JavaScript, et d'intégrer le plugin SQL pour SonarQube.

1️⃣ Générer un Token d'Authentification dans SonarQube

Avant de lancer l'analyse, **SonarQube nécessite un token d'authentification**. Suivez ces étapes pour générer un **token personnel** :

1. **Se connecter à SonarQube**
   - Accédez à votre instance **SonarQube** (exemple : `http://localhost:9000`).
   - Connectez-vous avec un compte administrateur (`admin` / `admin` par défaut).
2. **Créer un token**
   - Dans le menu en haut à droite, cliquez sur **"Mon compte"**.
   - Allez dans l'onglet **"Sécurité"**.
   - Entrez un nom pour le token (ex: `jenkins-sonar-token`).
   - Cliquez sur **"Générer"**.
   - **Copiez et enregistrez** le token affiché (⚠️ Vous ne pourrez plus le voir après).
3. **Utilisation dans Jenkins**
   - Dans **Jenkins → Gestion des Credentials**, ajoutez une **nouvelle Credential** :
     - **Type** : Secret Text
     - **ID** : `SONAR_TOKEN`
     - **Valeur** : (collez le token généré).

2️⃣ Installation de SonarQube Scanner

Explication du Stage : `Prepare SonarQube Scanner`

Ce stage vérifie si SonarQube Scanner est installé sur la machine Jenkins et l'installe si nécessaire.

```
stage('Prepare SonarQube Scanner') {
    steps {
        script {
            echo "🔍 Checking if SonarQube Scanner is installed..."
        }
        sh '''
```

```
            if [ ! -d "sonar-scanner" ]; then
                echo "🚀 Installing SonarQube Scanner..."
                apt-get update && apt-get install -y wget unzip
                wget -q https://binaries.sonarsource.com/Distribution/sonar-scanne
r-cli/sonar-scanner-cli-5.0.1.3006-linux.zip -O sonar-scanner.zip
                unzip -q sonar-scanner.zip
                mv sonar-scanner-5.0.1.3006-linux sonar-scanner
                chmod +x sonar-scanner/bin/sonar-scanner
            else
                echo "✅ SonarQube Scanner is already installed."
            fi
        '''
    }
}
```

Explication des commandes

- Vérifie si le répertoire `sonar-scanner` existe.
- Si **non**, il :
  - Met à jour les paquets.
  - Installe `wget` et `unzip` (au cas où ils ne sont pas installés).
  - Télécharge **SonarQube Scanner CLI v5.0.1.3006**.
  - Décompresse et renomme le dossier en `sonar-scanner`.
  - Rend l'exécutable `sonar-scanner/bin/sonar-scanner` utilisable.

3️⃣ Installation du Plugin SQL pour SonarQube

Par défaut, SonarQube **ne supporte pas l'analyse des fichiers SQL**. Il faut donc ajouter un plugin **tiers** qui permet de scanner les fichiers SQL.

Installation du Plugin SQL

1. **Vérifier et mettre à jour Java**

```
java -version
```

- SonarQube **requiert Java 17+**.
- Si vous avez une version inférieure, mettez à jour Java :

```
sudo apt update
sudo apt install openjdk-17-jdk
java -version
```

- **Définir Java 17 comme version par défaut** :

```
sudo update-alternatives --config java
```

2. **Cloner le plugin SQL SonarQube**

```
git clone https://github.com/gretard/sonar-sql-plugin.git
cd sonar-sql-plugin
```

3. **Compiler et installer le plugin**

```
mvn versions:display-dependency-updates spotless:check spotless:apply
install
```

4. **Déployer le plugin dans SonarQube**
   - Copier le fichier `.jar` généré vers le dossier des plugins SonarQube :

```
docker cp sonar-sql-plugin/src/sonar-sql-plugin/target/sonar-sql-plug
in-1.1.0.jar sonarqube:/opt/sonarqube/extensions/plugins/
```

   - **Redémarrer SonarQube** :

```
docker restart sonarqube
```

4️⃣ Exécution de l’Analyse avec SonarQube

Une fois SonarQube Scanner installé et le plugin SQL ajouté, nous pouvons analyser notre code.

Explication du Stage : `SonarQube Analysis`

```
stage('SonarQube Analysis') {
    steps {
        sh '''
            if [ -d "./output/java" ] || [ -d "./output/sql" ] || [ -d "./output/js" ]; then
                echo "✅ Files found, proceeding with SonarQube analysis..."
                sonar-scanner/bin/sonar-scanner \
                  -Dsonar.projectKey=voitures \
                  -Dsonar.projectName=voitures \
                  -Dsonar.host.url="${SONAR_HOST_URL}" \
                  -Dsonar.login="${SONAR_TOKEN}" \
                  -Dsonar.sources=./output \
                  -Dsonar.inclusions="**/*.java,**/*.sql,**/*.js" \
                  -Dsonar.java.binaries=. \
                  -Dsonar.scm.disabled=true \
                  -Dsonar.sourceEncoding=UTF-8 \
                  -Dsonar.verbose=true
            else
                echo "🚨 ERROR: No Java, SQL, or JS files found in ./output!"
                exit 1
            fi
        '''
    }
}
```

Explication des paramètres

- **Vérifie la présence des fichiers** à analyser (`Java, SQL, JS`).
- **Lance SonarQube Scanner** en définissant les options :

- `Dsonar.projectKey=voitures` : Nom du projet dans SonarQube.
- `Dsonar.projectName=voitures` : Nom affiché dans SonarQube.
- `Dsonar.host.url="${SONAR_HOST_URL}"` : URL du serveur SonarQube.
- `Dsonar.login="${SONAR_TOKEN}"` : Utilise le token d'authentification.
- `Dsonar.sources=./output` : Spécifie le répertoire contenant les fichiers extraits.
- `Dsonar.inclusions="**/*.java,**/*.sql,**/*.js"` : Spécifie les fichiers à analyser.
- `Dsonar.java.binaries=.` : Indique où trouver les fichiers binaires Java.
- `Dsonar.scm.disabled=true` : Désactive l'analyse du gestionnaire de versions (Git).
- `Dsonar.sourceEncoding=UTF-8` : Définit l'encodage des fichiers source.
- `Dsonar.verbose=true` : Active le mode **verbeux** pour plus de logs.

Étape 8 : Exportation et Upload du fichier JWA vers Nexus

L'objectif de cette étape est d'exporter l'application **Joget** sous forme de fichier **JWA** et de l'envoyer dans un **repository Nexus** pour stockage et gestion.

1️⃣ Création d'un Repository Nexus

Avant de pouvoir uploader le fichier JWA, il est nécessaire de **configurer un repository Nexus**.

Accéder à Nexus

1. Ouvrir **Nexus Repository Manager** via l'URL :

   ```
   http://192.168.193.128:8082
   ```

2. Se connecter avec :
   - **Utilisateur** : `admin`
   - **Mot de passe** : (retrouvé via `docker exec -it nexus cat /nexus-data/admin.password`)

Créer un Repository RAW Hosted

Nous allons créer un **repository Raw Hosted**, car nous voulons stocker un fichier **JWA** qui ne correspond pas aux formats standard comme Maven ou npm.

1. **Aller dans l'interface Nexus**
   - Se connecter à **Nexus Repository Manager**.
   - Aller dans **"Repositories"**.
   - Cliquer sur **"Create repository"**.
2. **Choisir le type de repository**
   - Sélectionner **"raw (hosted)"**.
3. **Configurer le repository**
   - **Name** : `vente_voitures-repo`
   - **Version policy** : `Release`
   - **Deployment policy** : `Allow redeploy`
4. **Enregistrer la configuration**.

2️⃣ Explication du Stage `Export & Upload JWA to Nexus`

Ce stage effectue **trois actions principales** :

1. **Exporter l'application Joget** sous forme de fichier `.jwa`.
2. **Vérifier la validité du fichier exporté**.
3. **Uploader le fichier sur Nexus**.

```
stage('Export & Upload JWA to Nexus (joget-repo)') {
    steps {
        script {
            echo "📦 Exporting application RSU from Joget..."
        }
        sh '''
        # Définition des variables
```

```
        APP_NAME="vente_voitures"
        APP_ID="1"
        TIMESTAMP=$(date +"%Y%m%d%H%M%S")
        JWA_FILE="APP_${APP_NAME}-${APP_ID}-${TIMESTAMP}.jwa"
        EXPORT_PATH="/opt/joget/wflow/export"

        echo "📦 Exporting: Name=$APP_NAME, ID=$APP_ID, File=$JWA_FILE"

        # Installation de zip dans le conteneur Joget si nécessaire
        docker exec -u root jogetapp  -c "apt-get update && apt-get install -y zi
p"

        # Création du dossier d'exportation et modification des permissions
        docker exec -u root jogetapp  -c "mkdir -p ${EXPORT_PATH} && chmod -
R 777 ${EXPORT_PATH}"

        # Exporter l'application en tant que fichier JWA
        docker exec jogetapp  -c "cd /opt/joget/wflow/app_src && zip -r ${EXPO
RT_PATH}/${JWA_FILE} ${APP_NAME} -x \"${APP_NAME}/.git/*\""

        # Vérifier la taille du fichier exporté
        FILE_SIZE=$(docker exec jogetapp  -c "stat -c %s ${EXPORT_PATH}/${J
WA_FILE}")

        if [ "$FILE_SIZE" -lt 1024 ]; then
            echo "❌ ERROR: Exported JWA file seems empty (size: ${FILE_SIZE}
bytes)."
            exit 1
        fi

        echo "✅ Export successful! File: ${JWA_FILE} (Size: ${FILE_SIZE} byte
s)"

        # Copier le fichier JWA exporté de Joget vers Jenkins
        docker cp jogetapp:${EXPORT_PATH}/${JWA_FILE} .
```

```
        if [ ! -f "${JWA_FILE}" ]; then
            echo "❌ ERROR: JWA file was not copied correctly to Jenkins."
            exit 1
        fi

        echo "📤 Uploading ${JWA_FILE} to Nexus..."

        # Upload du fichier vers Nexus
        curl -u admin:Mouhamed2000** --upload-file ${JWA_FILE} \
        "http://192.168.193.128:8082/repository/vente_voitures-repo/com/voitures/jwa/${APP_ID}/${JWA_FILE}"

        echo "✅ Upload completed successfully to joget-repo!"
        '''
    }
}
```

3️⃣ Explication du Code

1. Définition des Variables

```
APP_NAME="vente_voitures"
APP_ID="1"
TIMESTAMP=$(date +"%Y%m%d%H%M%S")
JWA_FILE="APP_${APP_NAME}-${APP_ID}-${TIMESTAMP}.jwa"
EXPORT_PATH="/opt/joget/wflow/export"
```

- `APP_NAME` : Nom de l'application.
- `APP_ID` : Identifiant de l'application.
- `TIMESTAMP` : Génère un timestamp pour avoir un fichier unique (`YYYYMMDDHHMMSS`).
- `JWA_FILE` : Nom du fichier `.jwa` qui sera exporté.

- `EXPORT_PATH` : Chemin où le fichier sera stocké dans le conteneur **Joget**.

---

2. Installation et Préparation du Conteneur Joget

```
docker exec -u root jogetapp  -c "apt-get update && apt-get install -y zip"
docker exec -u root jogetapp  -c "mkdir -p ${EXPORT_PATH} && chmod -R 77
7 ${EXPORT_PATH}"
```

- **Installe** `zip` si ce n'est pas déjà installé.
- **Crée le dossier d'exportation** et donne les permissions nécessaires.

---

3. Export de l'Application en `.jwa`

```
docker exec jogetapp  -c "cd /opt/joget/wflow/app_src && zip -r ${EXPORT_P
ATH}/${JWA_FILE} ${APP_NAME} -x \"${APP_NAME}/.git/*\""
```

- **Compresse l'application Joget** située dans `/opt/joget/wflow/app_src` en excluant le dossier `.git`.
- **Stocke le fichier compressé dans** `${EXPORT_PATH}`.

---

4. Vérification de l'Exportation

```
FILE_SIZE=$(docker exec jogetapp  -c "stat -c %s ${EXPORT_PATH}/${JWA_
FILE}")

if [ "$FILE_SIZE" -lt 1024 ]; then
    echo "❌ ERROR: Exported JWA file seems empty (size: ${FILE_SIZE} byte
s)."
    exit 1
fi
```

- **Vérifie la taille du fichier exporté**.

- **Si la taille est inférieure à 1 Ko**, on considère que le fichier est vide et on arrête le processus.

---

5. Copie du Fichier JWA vers Jenkins

```
docker cp jogetapp:${EXPORT_PATH}/${JWA_FILE} .
```

- **Copie le fichier depuis le conteneur vers Jenkins** pour l'envoyer à Nexus.

```
if [ ! -f "${JWA_FILE}" ]; then
    echo "❌ ERROR: JWA file was not copied correctly to Jenkins."
    exit 1
fi
```

- Vérifie si le fichier **a bien été copié**.

---

6. Upload du Fichier JWA vers Nexus

```
curl -u admin:Mouhamed2000** --upload-file ${JWA_FILE} \
"http://192.168.193.128:8082/repository/vente_voitures-repo/com/voitures/jwa/${APP_ID}/${JWA_FILE}"
```

- **Utilise `curl` pour envoyer le fichier vers Nexus**.
- **Authentification avec l'utilisateur `admin`** et mot de passe.
- **Chemin Nexus :**

  ```
  http://192.168.193.128:8082/repository/vente_voitures-repo/com/voitures/jwa/1/APP_vente_voitures-1-<timestamp>.jwa
  ```

- Une fois le fichier uploadé, un message `✅ Upload completed successfully to joget-repo!` est affiché.

Étape 9 : Explication Complète du Pipeline Jenkins

pipeline.sh

Ce pipeline Jenkins est conçu pour :
✅
Récupérer le code source depuis GitHub.

✅ **Exporter et stocker l'application Joget** sous forme de fichier `.jwa` dans Nexus.

✅ **Configurer et exécuter SonarQube Scanner** pour analyser les fichiers Java, SQL et JavaScript.

✅ **Exécuter des scripts** pour extraire du code de Joget.

✅ **Envoyer une notification par e-mail** en cas de succès ou d'échec du pipeline.

📌 Structure Générale

```
pipeline {
    agent any

    environment {
        SONAR_HOST_URL = 'http://192.168.193.128:9000'
        SONAR_TOKEN = 'squ_b6986d26e43b9a5c9fe5c8247fb84abdb22995a
4'
    }
```

Explication :

- `agent any` : Le pipeline s'exécute sur n'importe quel agent disponible dans Jenkins.
- `environment` : Définit les variables d'environnement pour SonarQube :
  - `SONAR_HOST_URL` : URL du serveur SonarQube.
  - `SONAR_TOKEN` : Token d'authentification pour SonarQube.

1️⃣ Stage : Checkout Code

```
stage('Checkout Code') {
    steps {
        script {
            echo "📥 Cloning repository: main"
        }
        git branch: 'main', url: 'https://github.com/motrabelsi10/testpipeline.git'
    }
}
```

Explication :

- **Récupère le code source** depuis GitHub.
- Affiche un message de confirmation.

2️⃣ Stage : Export & Upload JWA to Nexus

```
stage('Export & Upload JWA to Nexus (joget-repo)') {
    steps {
        script {
            echo "📦 Exporting application RSU from Joget..."
        }
        sh '''
        APP_NAME="vente_voitures"
        APP_ID="1"
        TIMESTAMP=$(date +"%Y%m%d%H%M%S")
        JWA_FILE="APP_${APP_NAME}-${APP_ID}-${TIMESTAMP}.jwa"
        EXPORT_PATH="/opt/joget/wflow/export"

        echo "📦 Exporting: Name=$APP_NAME, ID=$APP_ID, File=$JWA_FILE"

        docker exec -u root jogetapp  -c "apt-get update && apt-get install -y zi
p"
```

```
        docker exec -u root jogetapp  -c "mkdir -p ${EXPORT_PATH} && chmod -R 777 ${EXPORT_PATH}"
        docker exec jogetapp  -c "cd /opt/joget/wflow/app_src && zip -r ${EXPORT_PATH}/${JWA_FILE} ${APP_NAME} -x \"${APP_NAME}/.git/*\""

        FILE_SIZE=$(docker exec jogetapp  -c "stat -c %s ${EXPORT_PATH}/${JWA_FILE}")

        if [ "$FILE_SIZE" -lt 1024 ]; then
            echo "❌ ERROR: Exported JWA file seems empty (size: ${FILE_SIZE} bytes)."
            exit 1
        fi

        echo "✅ Export successful! File: ${JWA_FILE} (Size: ${FILE_SIZE} bytes)"

        docker cp jogetapp:${EXPORT_PATH}/${JWA_FILE} .

        if [ ! -f "${JWA_FILE}" ]; then
            echo "❌ ERROR: JWA file was not copied correctly to Jenkins."
            exit 1
        fi

        echo "📤 Uploading ${JWA_FILE} to Nexus..."

        curl -u admin:Mouhamed2000** --upload-file ${JWA_FILE} \
        "http://192.168.193.128:8082/repository/vente_voitures-repo/com/voitures/jwa/${APP_ID}/${JWA_FILE}"

        echo "✅ Upload completed successfully to joget-repo!"
        '''
    }
}
```

Explication :

- **Exporte l'application Joget** sous format `.jwa`.
- **Vérifie la validité du fichier** (taille > 1 Ko).
- **Copie le fichier de Joget vers Jenkins**.
- **Upload vers Nexus**.

3️⃣ Stage : Installation du Scanner SonarQube

```
stage('Prepare SonarQube Scanner') {
    steps {
        script {
            echo "🔍 Checking if SonarQube Scanner is installed..."
        }
        sh '''
            if [ ! -d "sonar-scanner" ]; then
                echo "🚀 Installing SonarQube Scanner..."
                apt-get update && apt-get install -y wget unzip
                wget -q https://binaries.sonarsource.com/Distribution/sonar-scanne
r-cli/sonar-scanner-cli-5.0.1.3006-linux.zip -O sonar-scanner.zip
                unzip -q sonar-scanner.zip
                mv sonar-scanner-5.0.1.3006-linux sonar-scanner
                chmod +x sonar-scanner/bin/sonar-scanner
            else
                echo "✅ SonarQube Scanner is already installed."
            fi
        '''
    }
}
```

Explication :

- Vérifie si **SonarQube Scanner est déjà installé**, sinon il l'installe.

4️⃣ Stage : Préparation de Joget

```
stage('Prepare Joget Environment') {
    steps {
        sh 'docker exec -i jogetapp  -c "mkdir -p /opt/joget/wflow/scripts/output
&& chmod -R 777 /opt/joget/wflow/scripts/output"'
    }
}
```

Explication :

- **Crée le répertoire** `output` pour stocker les fichiers extraits.
- **Assigne les permissions nécessaires**.

5️⃣ Stage : Exécution des Scripts

```
stage('Run  Script') {
    steps {
        sh '''
            docker exec -i jogetapp  -c "3 /opt/joget/wflow/scripts/extract_code.p
y"
            docker exec -i jogetapp  -c "3 /opt/joget/wflow/scripts/extract_js.py"
        '''
    }
}
```

Explication :

- Exécute les **scripts** pour **extraire** le code Java, SQL et JavaScript de Joget.

6️⃣ Stage : Copie des Fichiers Extraits

```
stage('Copy Extracted Files to Jenkins') {
    steps {
```

```
        sh '''
            rm -rf output
            mkdir -p output/java output/sql output/js
            docker cp jogetapp:/opt/joget/wflow/scripts/output/java ./output/
            docker cp jogetapp:/opt/joget/wflow/scripts/output/sql ./output/
            docker cp jogetapp:/opt/joget/wflow/scripts/output/js ./output/
        '''
    }
}
```

Explication :

- Supprime les anciens fichiers.
- Copie les nouveaux fichiers extraits vers Jenkins.

7️⃣ Stage : Vérification des Fichiers

```
stage('Verify Extracted Files in Jenkins') {
    steps {
        sh 'ls -R ./output || echo "🚨 No extracted files found!"'
    }
}
```

Explication :

- **Affiche les fichiers extraits** ou une alerte si aucun fichier n'a été trouvé.

8️⃣ Stage : Analyse SonarQube

```
stage('SonarQube Analysis') {
    steps {
        sh '''
            if [ -d "./output/java" ] || [ -d "./output/sql" ] || [ -d "./output/js" ]; then
                echo "✅ Files found, proceeding with SonarQube analysis..."
```

```
            sonar-scanner/bin/sonar-scanner \
              -Dsonar.projectKey=voitures \
              -Dsonar.projectName=voitures \
              -Dsonar.host.url="${SONAR_HOST_URL}" \
              -Dsonar.login="${SONAR_TOKEN}" \
              -Dsonar.sources=./output \
              -Dsonar.inclusions="**/*.java,**/*.sql,**/*.js" \
              -Dsonar.sourceEncoding=UTF-8 \
              -Dsonar.verbose=true
          else
            echo "🚨 ERROR: No Java, SQL, or JS files found in ./output!"
            exit 1
          fi
        '''
    }
}
```

Explication :

- Vérifie la présence de fichiers.
- Exécute l'analyse SonarQube.

📩 Notifications par E-mail

```
post {
    success {
        script {
            mail to: 'mouhamedtrabelsi.28@gmail.com',
                 subject: "Succès du build: ${env.JOB_NAME} #${env.BUILD_NUMB
ER}",
                 body: "Le build a réussi ! Consultez le lien suivant : ${env.BUILD_UR
L}"
        }
    }
    failure {
```

```
        script {
            mail to: 'mouhamedtrabelsi.28@gmail.com',
                 subject: "Échec du build: ${env.JOB_NAME} #${env.BUILD_NUMBE
R}",
                 body: "Le build a échoué ! Consultez le lien suivant : ${env.BUILD_U
RL}"
        }
    }
}
```

Explication :

- Envoie **une notification par e-mail** en cas de succès ou d’échec.

🚀 Ce pipeline assure une automatisation complète du cycle CI/CD avec Joget, Nexus et SonarQube !