TP 3 : Ajout de variables, de credentials et gestion des artefacts

Objectifs

- Définir et utiliser des variables dans un pipeline Jenkins.
- Gérer des credentials de manière sécurisée.
- Archiver et récupérer des artefacts générés lors de l'exécution du pipeline.

Prérequis :

- Jenkins déjà configuré (voir TP précédent).
- Un projet GitHub contenant un `Jenkinsfile`.
- Un compte Docker Hub (optionnel, si on veut pousser un artefact sur un registre).

Étape 1 : Ajout de variables d'environnement

1. **Modifier le `Jenkinsfile` pour inclure des variables**
   - Ajouter des variables d'environnement directement dans le pipeline :

```
pipeline {
    agent any
    environment {
        APP_NAME = 'my-app'
    }
    stages {
        stage('Build') {
            steps {
                script {
                    def buildVersion = "1.0.${env.BUILD_NUMBER}"
                    echo "Building ${APP_NAME} version ${buildVersion}"
                }
            }
        }
        stage('Test') {
            steps {
                echo "Testing ${APP_NAME}..."
            }
        }
        stage('Deploy') {
            steps {
                script {
                    def buildVersion = "1.0.${env.BUILD_NUMBER}"
                    echo "Deploying ${APP_NAME} version ${buildVersion}"
                }
            }
        }
    }
}
```

2. **Committer et pousser les modifications**

```
git add Jenkinsfile
git commit -m "Ajout de variables d’environnement"
git push origin main
```

3. **Exécuter le pipeline dans Jenkins**
   - Vérifier que les logs affichent bien les variables (`APP_NAME` et `BUILD_VERSION`).

Étape 2 : Gestion des credentials dans Jenkins

1. **Ajouter des credentials dans Jenkins**
   - Aller dans **Jenkins → Manage Jenkins → Manage Credentials**.
   - Ajouter une **credential de type secret text** :
     - **ID** : `DOCKER_PASSWORD`
     - **Secret** : (mot de passe Docker Hub ou token d’API)
2. **Modifier le `Jenkinsfile` pour utiliser un credential**
   - Ajouter la gestion des credentials avec `withCredentials` :

```
pipeline {
    agent any
    environment {
        DOCKER_USER = 'mon-utilisateur-docker'
    }
    stages {
        stage('Login Docker') {
            steps {
                withCredentials([string(credentialsId: 
'DOCKER_PASSWORD', variable: 'DOCKER_PASS')]) {
                    sh 'echo $DOCKER_PASS | docker login -u 
$DOCKER_USER --password-stdin'
                }
            }
        }
    }
}
```

3. **Committer et pousser les modifications**

```
git add Jenkinsfile
git commit -m "Ajout de la gestion des credentials"
git push origin main
```

4. **Exécuter le pipeline dans Jenkins**
   - Vérifier que Jenkins récupère et utilise le credential sans l’afficher en clair dans les logs.

Étape 3 : Gestion des artefacts

1. **Modifier le `Jenkinsfile` pour générer un artefact**
   - Simuler un fichier généré lors du build et l'archiver :

```
pipeline {
    agent any
    environment {
        ARTIFACT_NAME = 'app.tar.gz'
    }
    stages {
        stage('Build') {
            steps {
                sh 'echo "Contenu de l\'application" > app.txt'
                sh 'tar -czf ${ARTIFACT_NAME} app.txt'
                archiveArtifacts artifacts: ARTIFACT_NAME,
fingerprint: true
            }
        }
        stage('Deploy') {
            steps {
                echo "Déploiement de ${ARTIFACT_NAME}"
            }
        }
    }
}
```

2. **Committer et pousser les modifications**

```
git add Jenkinsfile
git commit -m "Ajout de la gestion des artefacts"
git push origin main
```

3. **Exécuter le pipeline dans Jenkins**
   - Aller dans l'onglet **"Artifacts"** pour voir le fichier `app.tar.gz` généré.
   - Vérifier les empreintes (`fingerprint`) dans Jenkins.