TP5 : Jenkins avec Docker et Maven pour builder un projet depuis GitHub

A- Intégration Continue CI

Prérequis :

- Jenkins déjà configuré (fournit par le formateur)
- Créer un compte github : https://github.com
- Un projet Maven sur GitHub avec un Jenkinsfile :
  forkez le projet https://github.com/BAliani/simple-java-maven-app.git
- Docker installé sur la machine où Jenkins est déployé (déjà fait)

Étape 1 : Créer un pipeline Jenkins

- Créer un nouveau job Jenkins :
  - Depuis l'interface Jenkins, cliquez sur **New Item**.
  - Choisissez **Pipeline**, donnez un nom à votre job (ex: Maven-Docker-GitHub-Build), puis cliquez sur **OK**.
- Configurer le pipeline :
  - Sous la section **Pipeline**, sélectionnez **Pipeline script from SCM**.
  - Dans **SCM**, sélectionnez **Git**.
  - Dans **Repository URL**, entrez l'URL de votre dépôt GitHub (par exemple, https://github.com/<votre-namespace>/simple-java-maven-app.git).
  - Indiquez la branche à suivre (ex: main ou master).
  - Définissez le chemin du Jenkinsfile (Jenkinsfile).

Étape 2 : Créer un Jenkinsfile pour l'exécution dans Docker

Voici un exemple de Jenkinsfile qui exécute Maven dans un conteneur Docker basé sur l'image

https://github.com/BAliani/simple-java-maven-app/blob/master/jenkinsfile

Explication :

- Le bloc agent spécifie que le job sera exécuté dans un conteneur Docker utilisant l'image maven:3.9.2.
- Le volume -v /root/.m2:/root/.m2 est monté pour partager le cache Maven local afin de ne pas retélécharger les dépendances à chaque build.

- Les étapes **Build** et **Test** exécutent les commandes Maven dans le conteneur.

Étape 3 : Exécution du Job

- Une fois le job configuré, lancer un build en cliquant sur **Build Now**.
- Le job va cloner le dépôt depuis GitHub, puis builder et tester le projet dans le conteneur Docker.

Étape 4 : Visualisation des résultats

- Les logs du build seront visibles dans la **Console Output**.
- En cas de succès, le fichier .jar généré sera archivé et accessible dans Jenkins.

Étape 5 : Configurer un déclencheur automatique sur push GitHub

- Accédez à **Manage Jenkins** > **Manage Plugins** et **installer le plugin GitHub Integration Plugin (déjà fait)**
- Configurer le job Jenkins pour répondre aux notifications GitHub
    - Accédez à la configuration de votre job Jenkins.
    - Dans l'onglet **Build Triggers**, cochez la case **GitHub hook trigger for GITScm polling**.
      Cela permettra à Jenkins de réagir aux événements push envoyés depuis GitHub via un Webhook.
- Créer un Webhook GitHub
    - Sur votre dépôt GitHub, allez dans Settings > Webhooks.
    - Cliquez sur Add webhook.
    - Dans la section Payload URL, entrez l'URL de votre serveur Jenkins suivi de /github-webhook/ http://w.x.y.z:8080/github-webhook/
    - Pour Content type, sélectionnez application/json.
    - Dans la section Which events would you like to trigger this webhook?, sélectionnez Just the push event.
    - Cliquez sur Add webhook.
- Tester le Webhook
    - Effectuez un commit et un push dans votre dépôt GitHub.
    - GitHub enverra une notification à Jenkins via le Webhook, et cela déclenchera automatiquement le build.

B- Livraison continue CD (publication d'un image Docker)

Prérequis :

- **Compte Docker Hub : Créez un compte sur Docker Hub : https://hub.docker.com/**
- **Dépôt Docker Hub** : Créez un nouveau dépôt Docker pour stocker vos images (nommez-le par exemple my-maven-app).
- **Générer un token Docker Hub** : Suivez les étapes ci-dessous pour générer un token d'accès.
- **Token GitHub** : Si vous travaillez avec un dépôt privé, assurez-vous d'avoir un token GitHub pour les accès.

Générer un token Docker Hub

Pour éviter de stocker vos mots de passe directement dans Jenkins, il est recommandé d'utiliser un **token d'accès Docker Hub**. Voici comment le générer :

1. Connectez-vous à votre compte Docker Hub.
2. Allez dans votre profil en cliquant sur votre avatar en haut à droite, puis sélectionnez **Account Settings**.
3. Dans le menu de gauche, cliquez sur **Security**.
4. Dans la section **Access Tokens**, cliquez sur **New Access Token**.
5. Donnez un nom à votre token (par exemple : Jenkins-Token).
6. Choisissez le niveau d'accès que vous souhaitez (généralement **Read/Write** pour permettre le push des images).
7. Cliquez sur **Generate**.
8. Copiez le token généré, car vous ne pourrez plus le voir par la suite. Utilisez ce token pour vous connecter à Docker Hub dans Jenkins.

Étape 1 : Créer un nouveau job Jenkins

- Depuis l'interface Jenkins, cliquez sur **New Item**.
- Sélectionnez **Freestyle Project**, nommez votre job (ex: Build-and-Push-Docker-Image), puis cliquez sur **OK**.

Étape 2 : Configurer le déclencheur automatique du nouveau job

- Dans la section Build Triggers, cochez Build after other projects are built.
    - Dans le champ qui apparaît, entrez le nom du premier job (celui qui build le projet Maven), par exemple : Maven-Docker-GitHub-Build.

Étape 3 : Injecter des variables d'environnement pour Docker Hub

- Allez dans la section Build Environment et cochez Inject environment variables to the build process.
- Dans le champ Properties Content, ajoutez les deux variables Docker :

Properties

```
DOCKER_LOGIN=your-docker-username
DOCKER_PASS=your-docker-token
```

DOCKER_LOGIN est votre identifiant Docker Hub.
DOCKER_PASS est le token que vous avez généré.

Étape 4 : Ajouter le script Shell pour builder et pousser l'image Docker

- Dans la section Build, cliquez sur Add build step et sélectionnez Execute shell.
- Ajoutez le script suivant dans le champ de commande Shell :

```
# Construire l'image Docker avec le BUILD_NUMBER de Jenkins
docker build -t <docker-id>/my-maven-app:$BUILD_NUMBER .
# Se connecter à Docker Hub en utilisant le token d'accès
docker login -u $DOCKER_LOGIN -p $DOCKER_PASS
# Pousser l'image sur Docker Hub
docker push <docker-id>/my-maven-app:$BUILD_NUMBER
```

Remplacez <docker-id> par votre identifiant Docker Hub.

Étape 5 : Exécuter et tester le pipeline

- Une fois tout configuré, un build du premier job (le build Maven) déclenchera automatiquement le deuxième job.
- Vérifiez dans les logs de console que l'image Docker est construite et poussée avec succès sur Docker Hub.

Résultats attendus :

- Build réussi : L'image Docker est générée avec le tag correspondant au numéro de build ($BUILD_NUMBER).
- Image poussée sur Docker Hub : L'image est disponible dans votre dépôt Docker Hub avec le bon tag.