Jenkins pipelines : Intégration et Livraison Continues

Partie 01

Notions Abordées :

- Pipeline entant que séquence de jobs

Objectif

L'objectif de ce TP est d'utiliser le serveur jenkins pour compiler, tester et déployer régulièrement notre application. Nous allons suivre un processus complet de l'intégration continue.

(A) Partie I : Séquence de Jobs

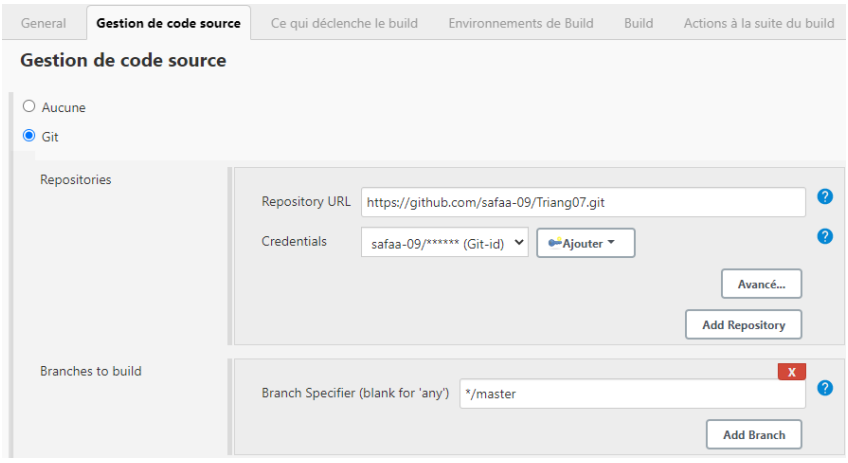
Dans cette partie, nous verrons comment créer des jobs, les lancer, faire des tests unitaires avec JUnit.

(a) Vous allez commencer par mettre le projet sous le serveur GITHub. (voir TP git).

(b) Par la suite, sur Jenkins, on va créer une séquence de Jobs permettant de compiler et de tester un projet maven.

i. Créer un premier job comme étant un ***projet freestyle***. Ce job à pour objectif de compiler un projet maven.

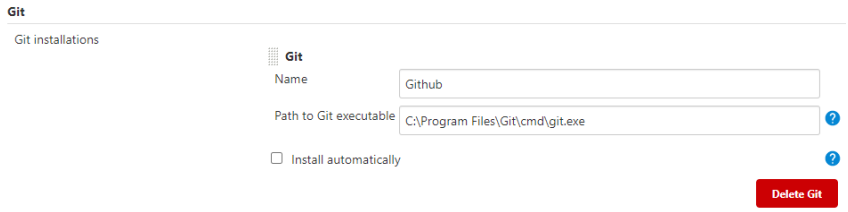
- Créer le job sur ***nouveau item > Créer un projet freestyle***
- Placez vous sur la section *"Gestion de code source"* puis cochez l'option ***git***, et entrez votre dépôt (repository).



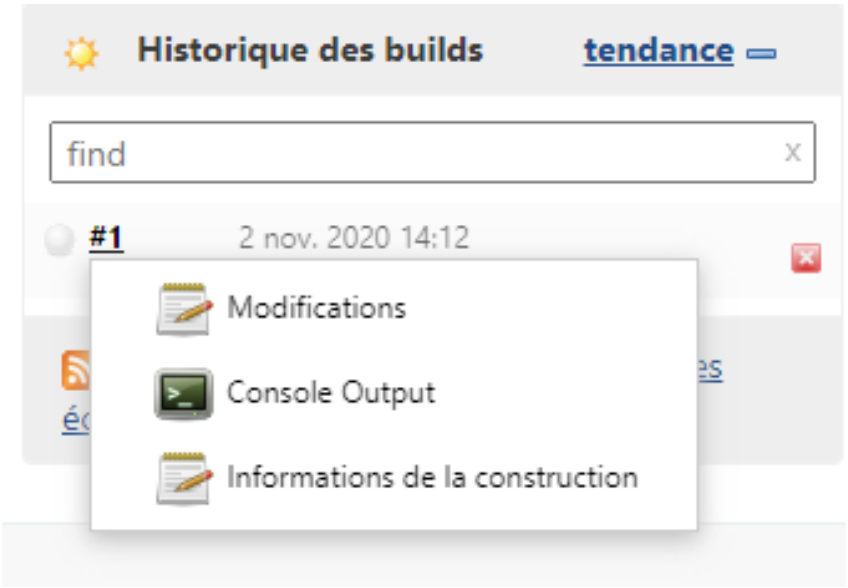
Note:

– Le plugin git doit être déjà installé sur Jenkins.

– Si votre build génère une erreur de connexion Git, vous allez devoir lui indiquer le chemin de ***git.exe*** sur votre machine. Pour ce faire, placez vous sur :
Jenkins > Administrer Jenkins > Configuration Globale des outils > Git > Path to Git executable.



- Sur la section ***Build***, sélectionner ***Ajouter une étape du build > Invoquer les cibles Maven de haut niveau***
- Cibles maven, ***compile*** (puisqu'on veut compiler notre application maven)
- Sauvegarder et lancer le job. Vous pouvez voir ce qui ce passe durant le build du job, avec la console qui vous informe des actions qui se déroulent.

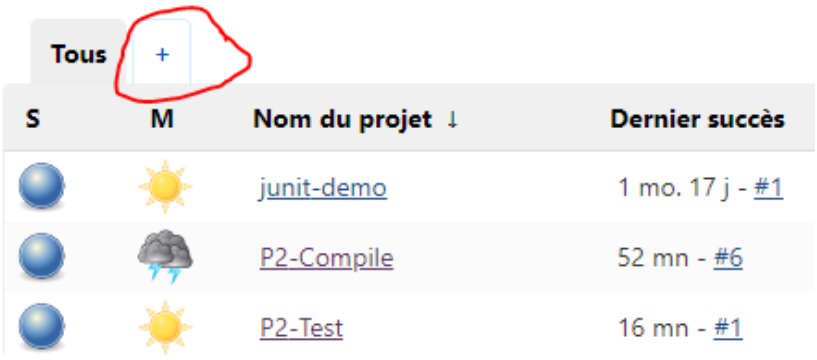


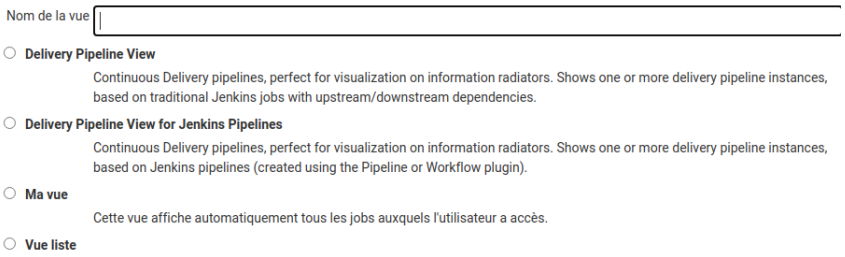
ii. Créer un deuxième job comme étant un ***projet freestyle***. Ce job a pour objectif de tester un projet maven.
De la même façon
- Créer le job sur ***nouveau item > Créer un projet freestyle***
- Placez vous sur la section *"Gestion de code source"* puis cochez l'option ***git***, et entrez votre dépôt (repository).
- Dans "Cibles maven", entrez l'option ***test*** (puisqu'on veut lancer les tests unitaires de notre application maven)
- Dans ce job, dans la section "Ce qui déclanche le build", Cocher l'option "Construire après le build sur d'autres projets", et choisissez le job précédent.
- Sauvegarder et lancer le job.

iii. De la même façon, créer un troisième job, permettant de mettre l'application dans un jar. Ce job doit se déclencher après que les deux premiers aient terminé.

(c) Télécharger le plugin "Delivery Pipeline View"

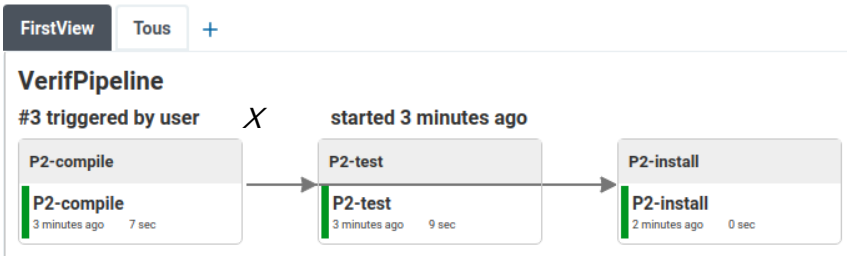
(d) Créer une delivery pipeline view





(e) Dans la section *Pipeline>Composant*, sélectionnez le premier job à déclencher dans votre pipeline et sauvegardez.

(f) Relancez le premier job, sur votre pipeline vous devez avoir la représentation graphique suivante :



Jenkins pipelines : Intégration et Livraison Continues

Partie 02

Notions Abordées :

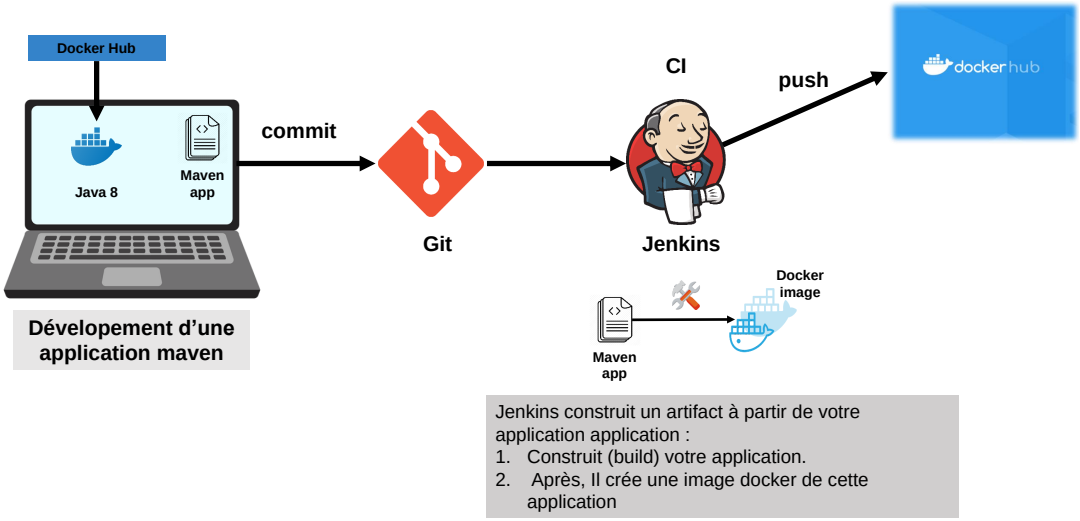
- Intégration continue avec git Jenkins et Maven.
- pipeline , Jenkinsfile
- Jenkins et github webhook

Objectif

L'objectif de ce TP est d'utiliser le serveur jenkins pour compiler, tester et déployer régulièrement notre application. Nous allons suivre un processus complet de l'intégration continue.

Nous verrons particulièrement la notion de pipeline dans Jenkins.

workflow.pdf



Dans le cadre de ce travail, Nous allons utiliser un projet Maven et son script de compilation et tests, ainsi que son Dockerfile. Vous pouvez prendre une applications Java et le script Maven permettant de la compiler. Vous pouvez aussi prendre une application trouvée sur le net, du moment qu'elle soit accompagnée d'un script de compilation, des tests unitaire, et d'un Dockerfile.

(A) Pipeline comme un code

Rappel
Un pipeline Jenkins consiste en deux types d'éléments : *stages* et *steps*.

- *stage* : une séparations logiques entre les étapes (steps), permettant ainsi de distinguer les séquences d'étapes, par exemple, Build, Unit Test, Deploy ... Ceci aidera à visualiser le progrès d'une pipeline Jenkins
- *step* : consiste en une opération, demandant à Jenkins la tâche à faire, par exemple, le check-out du code à partir d'un dépôt, exécuter un script, etc.

Pré-requis

- S'assurer que le serveur Jenkins a accès à Internet. (nécessaire pour télécharger et installer les plugins).
- S'assurer que Jenkins dispose des plugins nécessaires pour le bon fonctionnement de votre code

(a) Ajouter "Git Credentials" dans Jenkins Pour que Jenkins puisse communiquer avec GitHub, on a besoin d'ajouter, sur Jenkins, les identifiants (Credentials) du compte Github.(Assurez vous que Jenkins Credentials Plugin est installé).

Une façon de faire, pour ajouter les identifiants de github sur Jenkins, est de suivre les étapes suivantes :

- Placez vous sur *"Crendentials > Users > Global Credentials"*
- Dans le menu (à gauche), Cliquez sur Ajoutez Credentials (Identifiants)
- Choisissez l'option **Username with password**
- Ajoutez le **username et le mot de passe de votre compte GitHub**
- Donnez à vos identifiants un mot de passe unique (exp: Github-Credentials)
- Ajoutez une description (optionnel), et sauvegardez

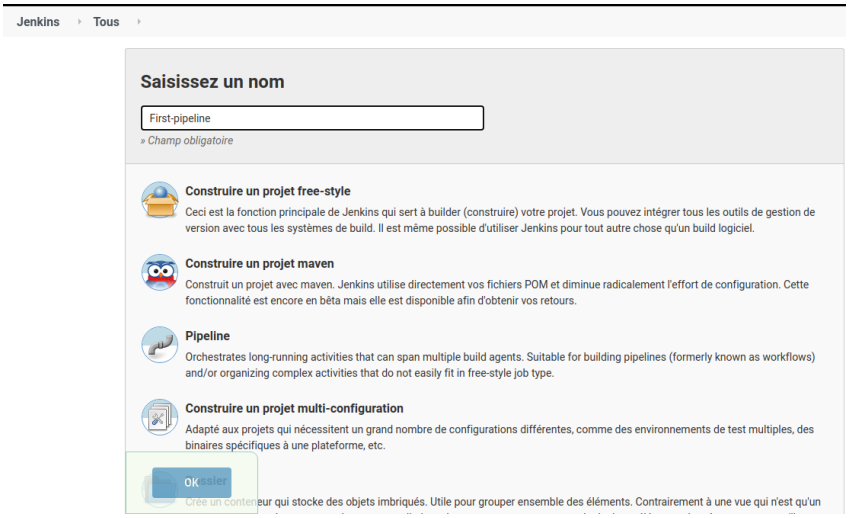
(b) "Global Tool Configuration
Avant l'exécution de votre pipeline, tâchez à voir la page "Global Tools". C'est dans cette dernière que vous configurez des outils que vous pouvez utiliser tout au long de votre pipeline tel que : Java, Maven, Git

- Sur la section *"Manage Jenkins>Global Tools Configuration"*, défiler vers le bas, jusqu'à arriver à la section Jenkins
- Ajouter Maven, donnez lui un nom unique, et indiquez la version souhaitée.

De même pour Git.

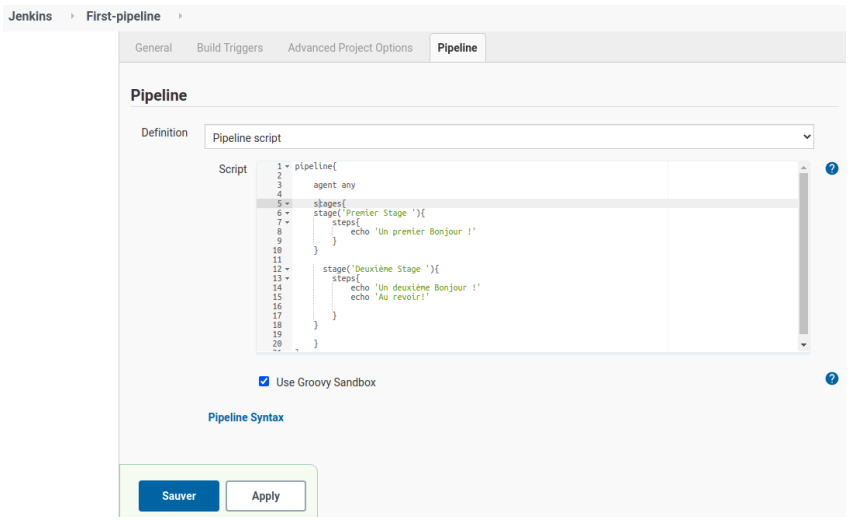
Note : Si Jenkins ne reconnait pas Maven ou Git, assurez vous également de l'installation d'un plugin Git et d'un plugin Maven ou Maven Integration Plugin.

Un Premier Exemple: Sur votre tableau de bord Jenkins, cliquez sur nouveau item → pipeline :

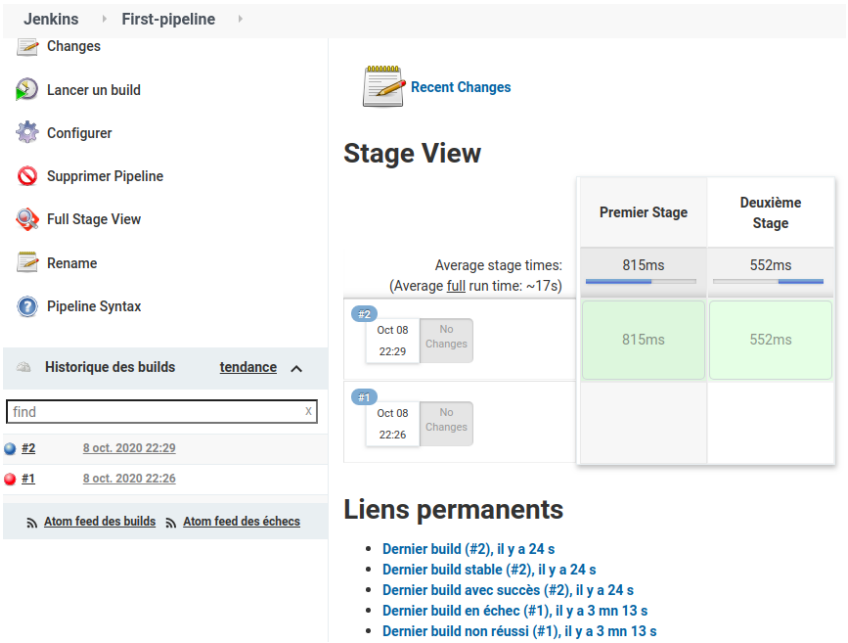


Si vous n'arrivez pas à trouver l'icône du pipeline, vous devrez d'abord installer le plugin pipeline.

Par exemple, soit une simple pipeline contenant deux *"stages"* :



Sauvegardez votre pipeline et lancer un build. Une représentation visuelle devrait apparaître.



Jenkins vous permet de voir l'historique des builds, et vous pouvez aussi voir les détails d'exécution en cliquant sur le numéro du build et puis sur console. Si l'une des étapes échoue, le processus du build s'arrête, et aucune autre étape ne s'exécute. La pipeline nous permet aussi de visualiser le point d'échec.

Pipeline pour un projet Maven

1) Première étape : Avoir un projet avec des tests sur Github

Envoyez le code de votre projet accompagné des tests sur Github.

Rappel :
Initialiser votre dépôt avec *git init* Commitez votre code dans le repo, en ajoutant d'abord le code dans l'index.

- git add .
- git commit -m "v1"
- git remote add origin votre_githubrepo
- git push -u origin master

On peut créer maintenant une nouvelle pipeline nommée *pipeline-triangle*, entant qu'un script, et y mettra un premier "*stage*"
qui nous permettra de charger le code de notre projet à partir de github
Création d'un premier "*stage*"
qu'on nommera checkout, nous permettant de charger le code du projet à partir de github.

```
pipeline{
  agent any
    stages{
      stage('git checkout'){
        steps{
          git credentialsId: 'git_credentials', url: 'https://github.com/badre-09/Triang07.git'
        }
      }
    }
}
```

Le pipeline peut être exécuté sur n'importe quel agent, jusqu'ici notre pipeline ne contient qu'une seule étape (chargement du projet à partir de github). Après exécution, on devrait obtenir la vue suivante :

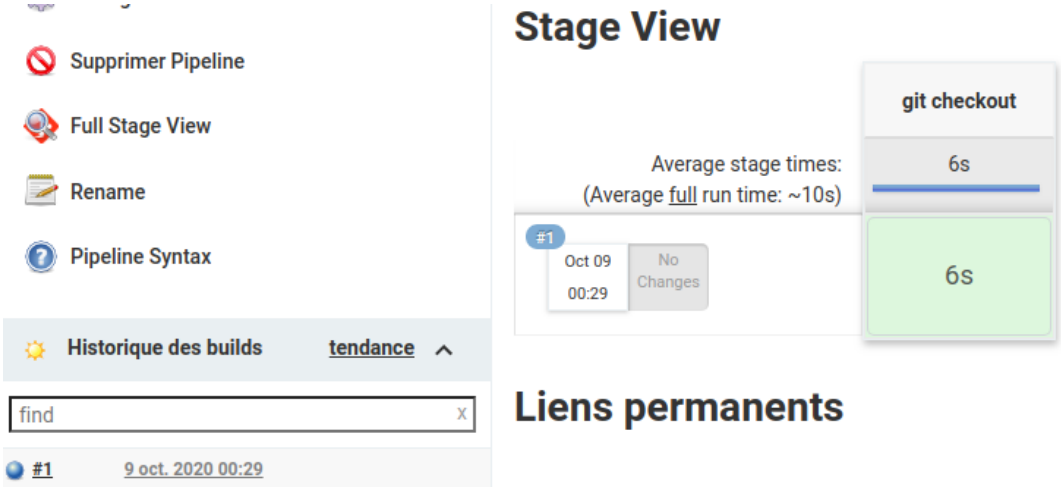


Figure 1:

2) Deuxième étape : Compilation de votre application

La deuxième étape du pipeline consiste en la compilation de l'application avec l'outil de build maven. **Notons, qu'il faut installer le plugin maven sur jenkins.**

Figure 2:

Sauvegardons le script du pipeline, et exécutons-le. La vue apparaît, mais cette fois-ci avec deux étapes :

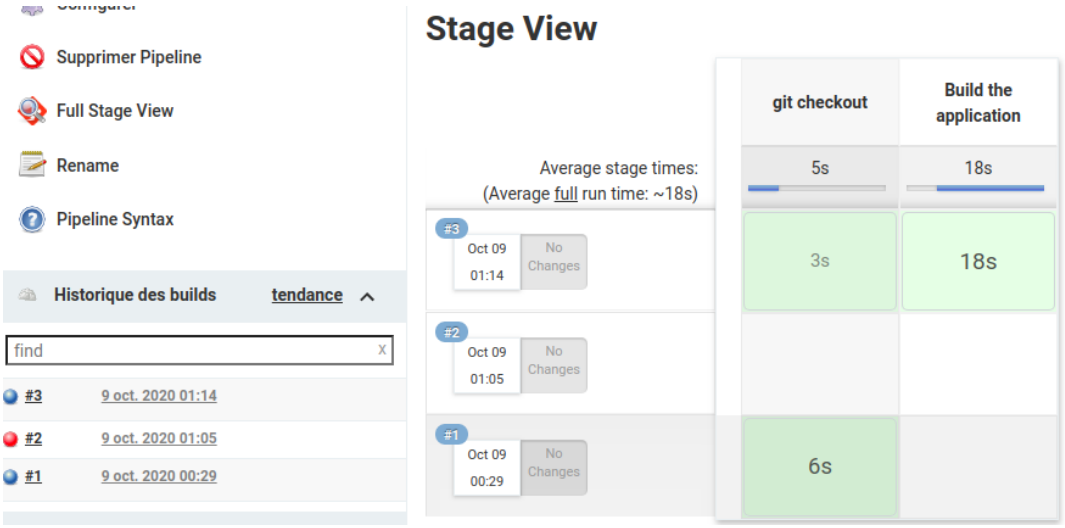


Figure 3:

Note ("Tools attribute") : Pour utiliser l'outil maven, on doit avoir cet outil pré-installé ou pré-configuré. Ainsi, Si le build génère une erreur par rapport à l'outil Maven, c'est qu'il n'arrive pas à le trouver. Dans ce cas, il faudra le définir dans l'attribut d'outils (tools attribute) dans votre script :

```
    tools{
Maven : Maven_nom_sur_Jenkins
}
    }
```

Maven "Maven_nom_sur_Jenkins"
Maven_nom_sur_Jenkins : le nom de l'outil pré-installé ou pré-configurer dans Jenkins résidant à : **manage>Jenkins>Global tool configuration>maven>name**

3) **Troisième étape : Exécution des test unitaires en ajoutant une autre étape à notre pipeline :**

```
    stage('Unit Test Execution') {
     steps{
       sh 'mvn test'
     }
    }
```

4) **Quatrième étape : Build de l'image docker**

Notre projet devrait contenir un Dockerfile permettant de transformer notre application sous forme d'image Docker. Le build de l'image docker sera effectué à travers jenkins, en ajoutant au script de notre pipeline l'étape suivante :

```
    stage('Build the docker image') {
     steps{
       sh "docker build -tag badre09/triang7:1.0.0."
     }
    }
```

5) **Cinquième étape : Mettre l'image de le dépot DockerHub**

Pour ce, il faut d'abord se connecter à votre compte dockerhub.

Avant d'envoyer l'image docker à DockerHub, on va d'abord commencer par **définir les identifiants (Credentials)**.
Note : les identifiants sont d'abord définis dans l'interface graphique.
Vous aurez besoin du plugin : "Credentials Binding".
En utilisant la fonction withCredentials(..), Nous permet de passer le mot de passe en secret.

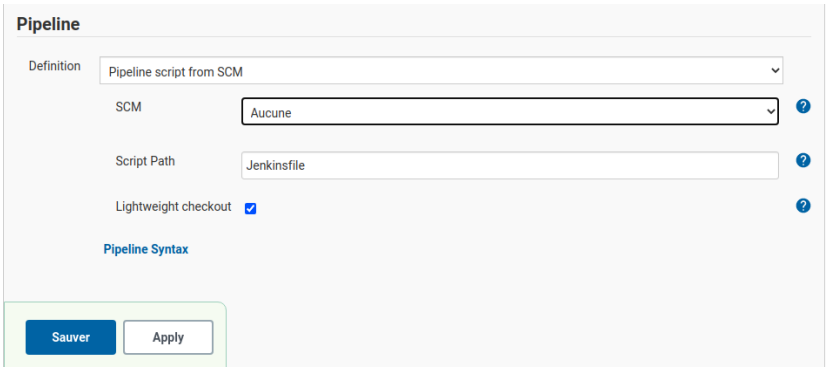
```
    stage('Build the docker image') {
     steps{
       withCredentials([string(credentialsId: 'dockerhubpass', variable: 'dockerHubPass')]) {
       sh "docker login -u badre09 -p $dockerHubPass"
      }
       sh 'docker push badre09/triang7:1.0.0'
     }
    }
```

(B) Partie III : Pipeline avec Jenkinsfile

Le Jenkinsfile est un fichier à la racine du dépôt git. Il décrit l'exécution du pipeline avec la même syntaxe qu'un pipeline normal.

- Copiez le code de votre pipeline dans un fichier nommé Jenkinsfile.

- Poussez le fichier dans le git repository
- Sur Jenkins, configuration du pipeline, chosissez l'option *Pipeline script from SCM*



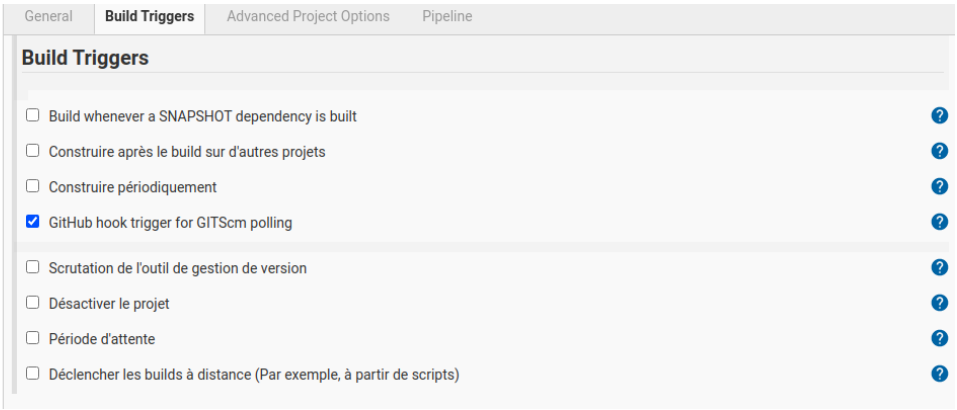
- Sur SCM, choisissez git et entrez l'adresse du dépôt contenant votre Jenkinsfile
- Sauvegarder et lancer votre build.

(C) Jenkins et Webhook

Dans ce TP, l'utilisation des webhooks nous permettra de déclencher automatiquement un build Jenkins à chaque fois qu'on réalise un commit sur github.

Pour que Jenkins puisse recevoir un trigger lui indiquant qu'un commit a été effectué sur github, il faut lui indiquer quel comportement suivre :

Placer vous dans la configuration de votre pipeline, dans la section *Build Triggers* puis cocher la case *"GitHub hook trigger for GITScm polling"*



Par la suite, placez-vous sur votre projet existant sur github repository, puis sur Configuration>webhook et ajouter un webhook.

Note : Pour pouvoir ajouter un webhook, il faut que votre adresse Jenkins soit public. Pour ce faire, on va transformer notre adresse local Jenkins en une adresse public en utilisant un logiciel appelé ngrok. Suivez les étapes suivantes :

- Téléchargez ngrok.zip
- Puis unzippez-le : *unzip /path/to/ngrok.zip*

- Exécutez cette instruction sur la ligne de commande :
  ./ngrok authtoken 1if7mcXXzxHImP3EjKOxS8ut7a_6b12Bn8Ghxdjkf6CjgdzZ
- Exécuter l'instruction : *./ngrok http 8080* (8080 étant mon Localhost)
- et enfin sur votre navigateur connectez vous avez le http ou le https généré.

```
ngrok by @inconshreveable

Session Status                online
Account                       safau.achour@gmail.com (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://43f332d1b16f.ngrok.io -> http://localhost:8080
Forwarding                    https://43f332d1b16f.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              245     0       0.00    0.01    5.01    9.00

HTTP Requests
```

Effectuez une modification dans votre projet sur github pour vérifier si le build automatique sur Jenkins a été effectué

(D) Post-action : notification par email

(a) La première étape est de configurer votre serveur smtp pour envoyer un mail sur : manage Jenkins > configure system > extended email notification.
Pour le cas de Gmail : smtp.gmail.com
Port : 465
cocher la case *SSL*

(b) Dans votre pipeline, on peut ajouter une post-action afin de pouvoir envoyer un email lors de l'échec d'un build

```
post{

   failure{
     emailext body: 'Ce Build $BUILD_NUMBER a échoué',
    recipientProviders:[requestor()], subject: 'build', to:
     'badre.bousalem@enpc.fr'
   }
}
```

Jenkins pipelines : Intégration et Livraison Continues

Partie 03

Notions Abordées :

- Multibranches pipeline
- Jenkinsfile

Rappel

Dans le premier TP de Jenkins, on a vu une façon classique de créer une pipeline sur Jenkins en utilisant une séquence de jobs de type freestyle, où chaque étape (stage) de la pipeline est représentée par un job Jenkins.

(A) Pipeline comme un code

Rappel du Cours :
Le concept de "Pipeline as code" reconsidère la façon de créer un pipeline CI. L'idée est d'écrire tout le pipeline d'Intégration Continue comme étant un code qui offre quelques niveaux de programmation et qui peut être déposer sous forme d'un Jenkinsfile sur un système de contrôle de version (dans notre cas git/github).

(B) Multibranch Pipeline

Cette partie Concerne les jobs Jenkins "Multibranch pipeline"

Prérequis

- Assurez vous que le plugin "Pipeline Multibranch" est installé

(a) Créer un nouveau job (item) de type "Multibranch Pipeline"

(b) Donnez un nom à votre job "Multibranch Pipeline"

(c) Déplacez vous sur la section "Branch Source", où nous allons configurer le repository GitHub choisi.
Note : Sur votre Repo GitHub, vous devez disposer du projet contenant au moins de branches.

(d) Cliquez sur Ajouter, et choisissez votre repository Github.
- Pour les identifiants GitHub, choisissez les identifiants déjà Créés dans la section précédentes.

(e) Sauvegarder votre "Multibranch Pipeline", et l'exécution s'exécutera automatiquement. (Une pipeline est exécutée par branche).

(C) Variables d'environnement

- Créer un pipeline avec deux étapes (stage)
  - stage Lister les variables
  - stage Utilisation des variables
- Créer deux variables d'environnement : nom d'un utilisateur et sa couleur préférée
- Affichez ces deux variables sur le "stage Utilisation des variables"
- Dans ce même "stage", créer une nouvelle variable d'environnement décrivant un loisir de l'utilisateur
- Dans le même stage redéfinissez la couleur préférée de la personne.
- Qu'est-ce que vous remarquez?