**TP - Construction d’un pipeline JENKINS CI/CD**

Le but de ce TP est de construire un pipeline JENKINS CI/CD (Continuous Integration/Continuous Delivery) à partir d’un projet Java présent dans un repository GitHub. Les technologies utilisées dans ce TP sont Jenkins, Maven, Git, SonarQube, Docker, Cloud AWS, Ansible.

Ci-dessous se trouve un schéma détaillant les technologies utilisées ainsi que le pipeline CI /CD JENKISN à réaliser.

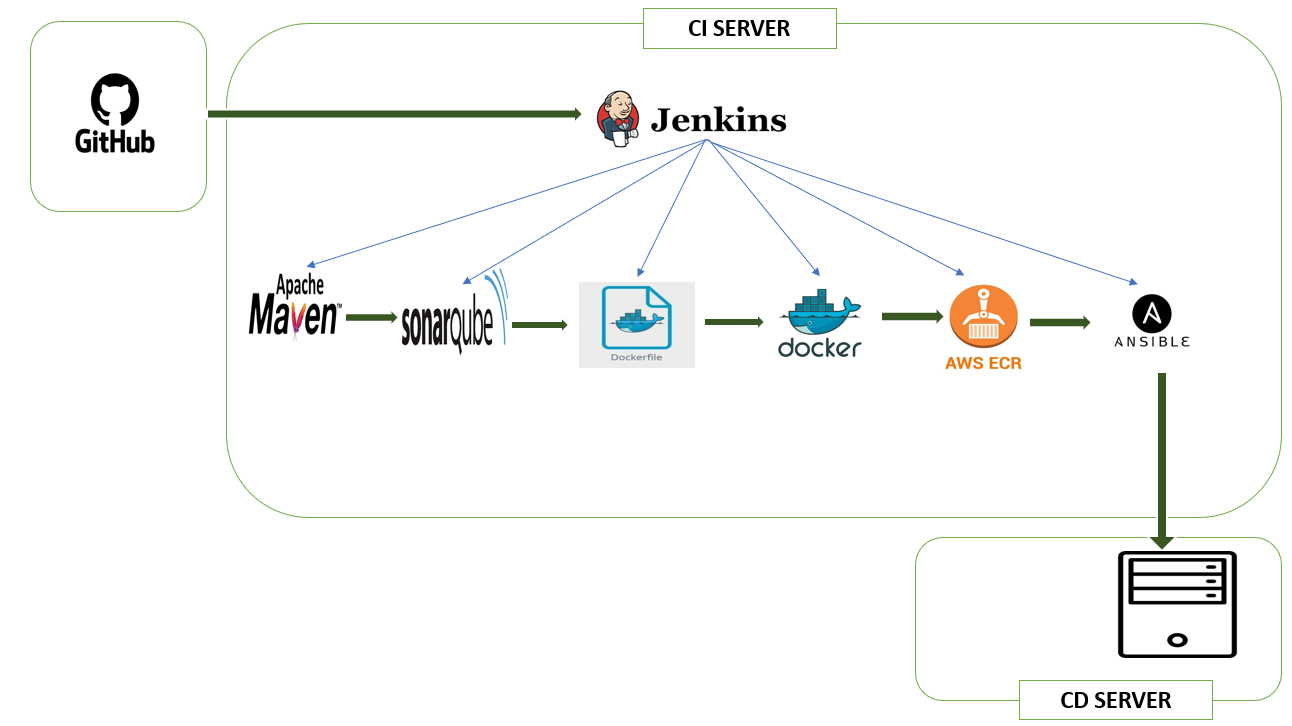


Fig.1 Schéma global de la chaîne CI/CD

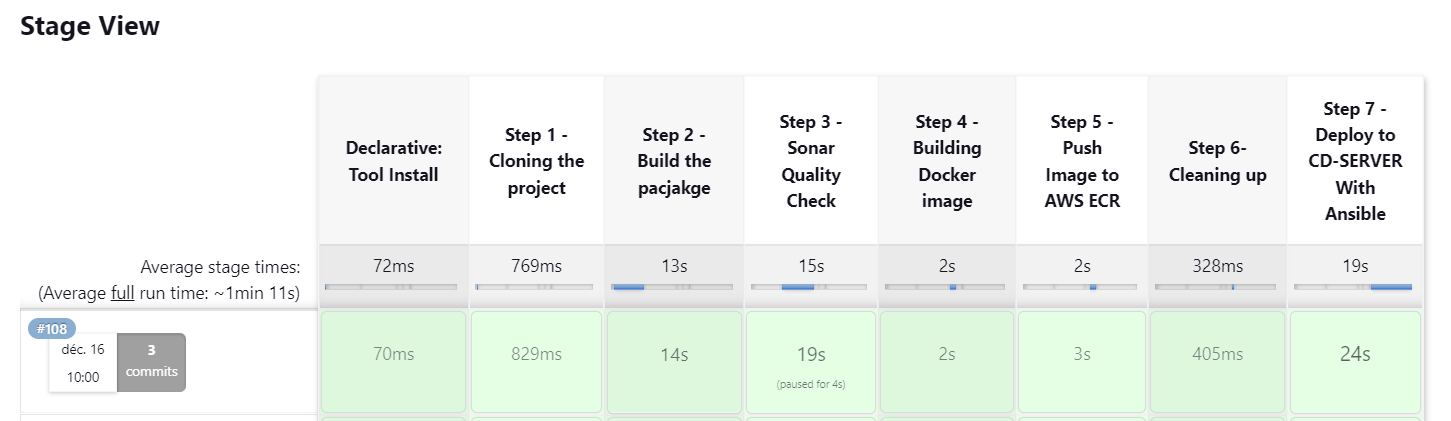


Fig.2 Pipeline CI/CD JENKINS

1. **Création des instances** **EC2 dans AWS**

Dans la première étape de ce TP nous allons tout d’abord créer l’infrastructure AWS qui sera composée de deux instances EC2 AWS avec les caractéristiques suivantes :

* Les instances seront nommées CI-SERVER et CD-SERVER
* Dans la région d’Irlande « eu-west-1»
* Instances de type t2.large
* Système d’explantation Amazon Linux
* Security Group SG-SSM
* Dans le sous-réseau LAB-IAC-subnet-public1-eu-west-1a
* Paire de clés SSH sélectionnées LAB-IAC

1. **Construction du pipeline CI/CD sous Jenkins**

**II.1 Installation de Jenkins**

Afin de construire le pipeline Jenkins CICD nous allons tout d’abord installer les librairies suivantes :

* **Java 11**

Installez Java et vérifiez la version avec la commande « java -- version ».

Java est indispensable car le projet source est un projet java mais également car il est un prérequis pour installer Jenkins, Maven et SonarQube.

* **Jenkins**

Installez Jenkins afin de configurer le pipeline CICD. Jenkins écoute par défaut sur le port 8080, pensez à rajouter ce port dans les inbound rules du Security Group de l’instance EC2 CI-SERVER.

Accédez à Jenkins à adresse **< JENKINS-IP>:8080** et suivez les instructions d’installation.

**II.2 Création et configuration du job pipeline CI/CD**

**II.2.1 Création du job de type pipeline**

Nous allons ensuite créer un job JENKINS de type « Pipeline » nommé « **Pipeline-CI-CD**». Paramétrez ce job afin que le paramètre d’entré soit la branche à builder et nommez ce paramètre « BRANCH ».

Créez ensuite un webhook dans Github qui déclenchera le lancement du job à chaque modification du code. L’url du webhook est sous la forme : **http://<JENKINS-IP>:8080/github-webhook/**

**II.2.2 Configuration du pipeline CI-CD**

Nous allons construire le pipeline Jenkins à l’aide de la syntaxe [**Declarative Pipeline**](https://www.jenkins.io/doc/book/pipeline/syntax/)**.**

Pour les différentes étapes du pipeline, penser à utiliser le générateur de script pipeline **« Pipeline Syntax »** proposé par Jenkins.

**Step 1 : Cloner le projet avec Git**

Installez tout d’abord la librairie Git dans le serveur CI-SERVER afin de permettre à Jenkins de cloner le projet.

Le repository git est à l’adresse suivante : **https://github.com/oadya/ci-cd-pipeline.git**

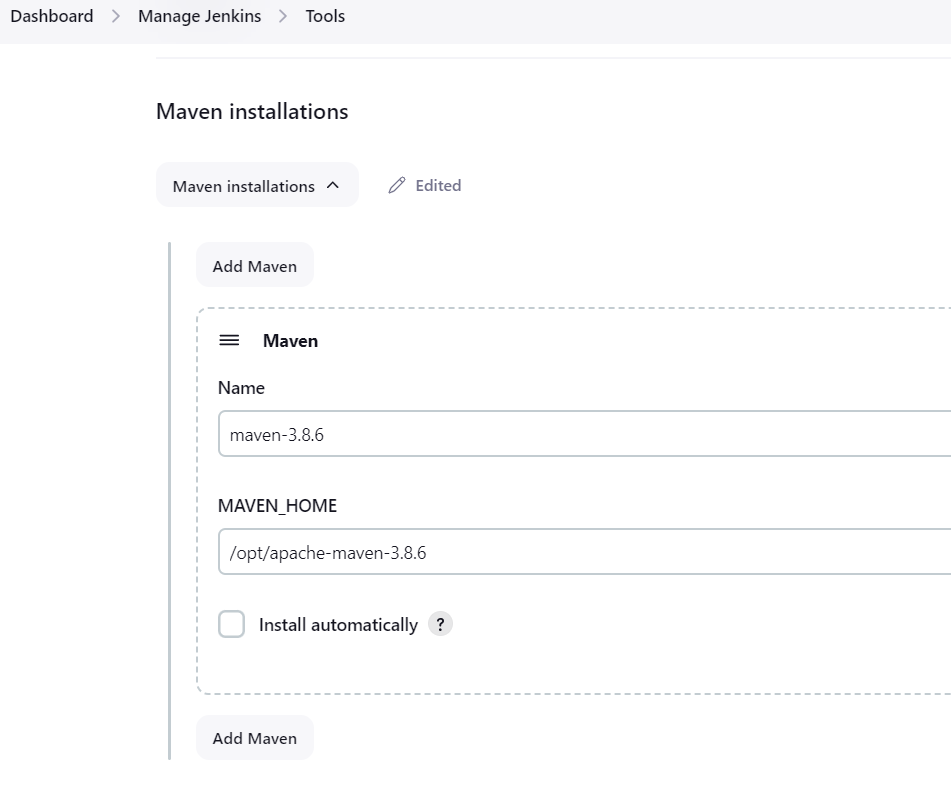
La première étape va consister à ajouter un script pipeline afin de cloner le projet Git. Vérifier ensuite que le projet a bien été cloné dans le workspace par défaut de Jenkins qui est /var/lib/jenkins/workspace.

Lancez le Job et vérifiez que le projet a bien été cloné.

**Step 2 : Builder le projet avec Maven**

Installez tout d’abord Maven dans le serveur CI-SERVER selon la documentation fournie en annexe.

Il faut ensuite indiquer à Jenkins où se trouve l’installation de Maven comme ceci :



Rajoutez un script pipeline qui va permettre de builder le projet java.

Lancez le Job et vérifiez que le build maven est ok.

**Step 3 : Vérifier la qualité du code avec SonarQube**

Installez tout d’abord SonarQube dans le serveur CI-SERVER à l’aide de la documentation fournie en annexe.

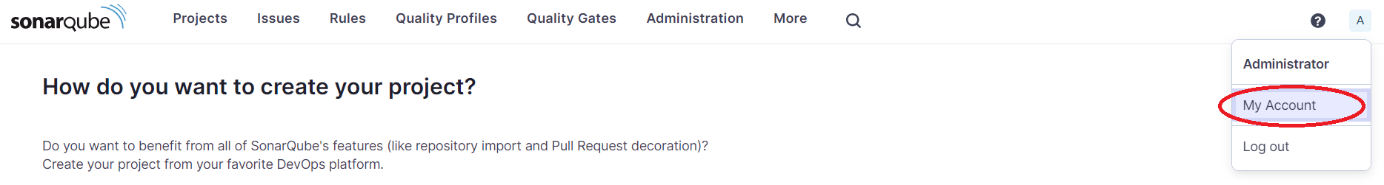
Vérifiez la bonne installation de SonarQube en accédant l’adresse : **<SONARQUEBE-IP>:9000**.

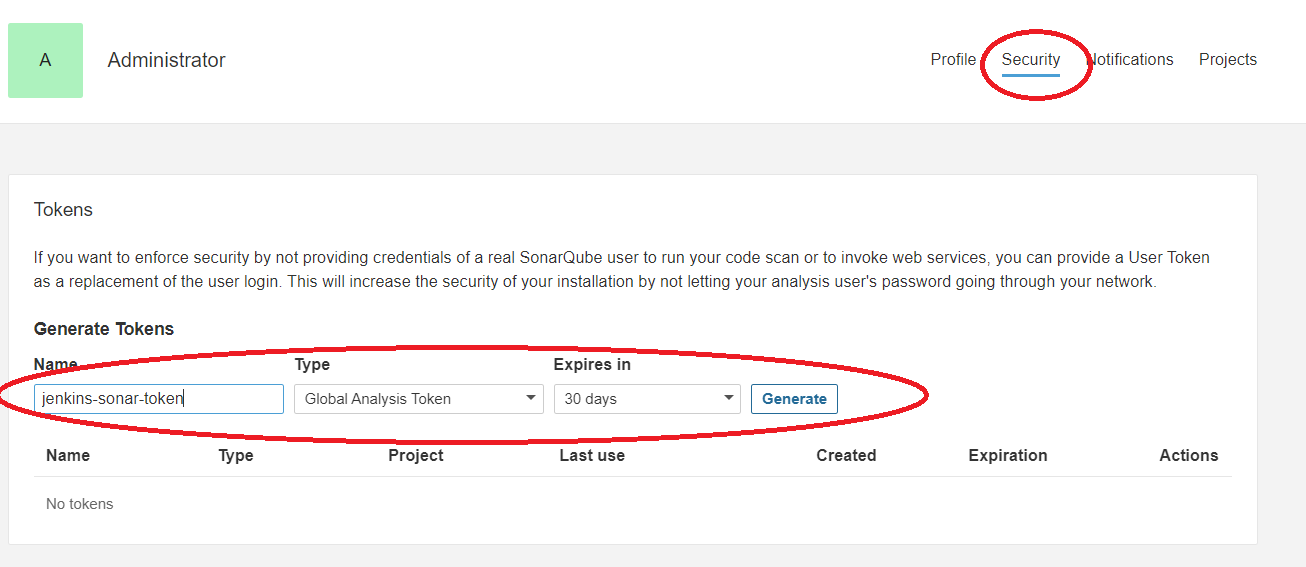
Les identifiants par défaut à SonarQube sont admin/amin.

Dans cette étape, nous allons tout d’abord réaliser une série de configuration afin que Jenkins puisque communiquer avec SonarQube.

* Création d’un token de connexion à SonarQube

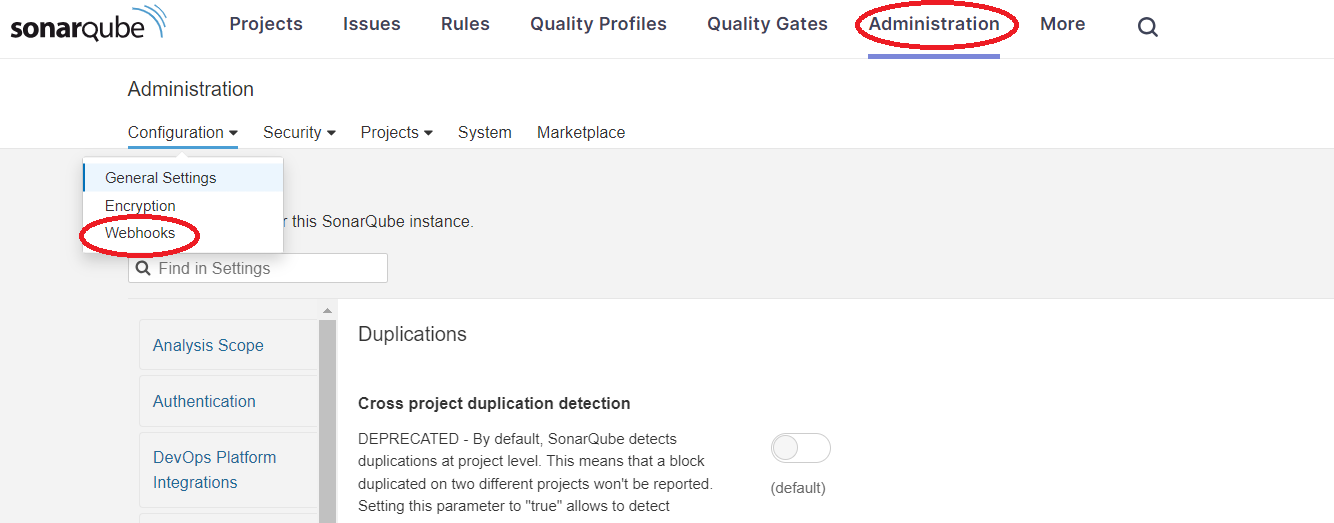
Le token va permettre à Jenkins de s’authentifier auprès de SonarQube



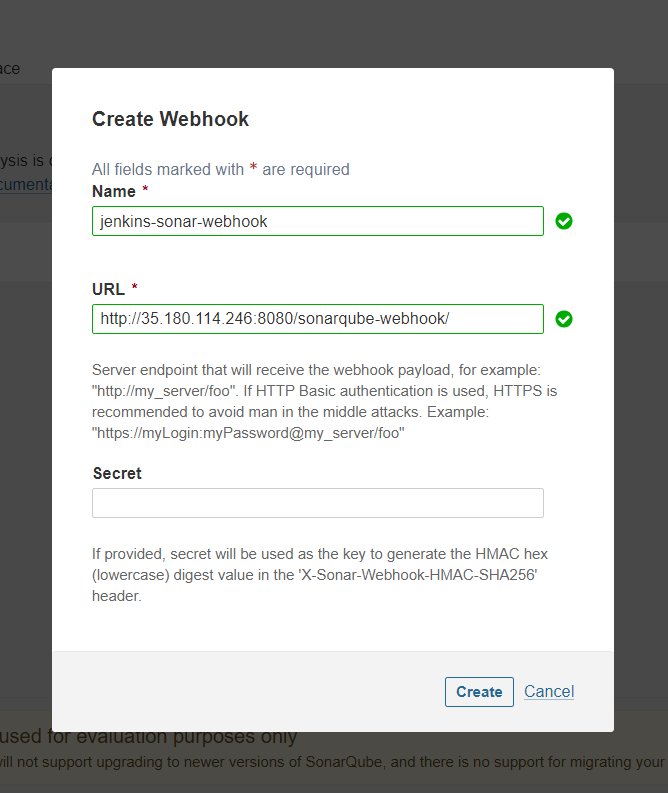


* Création d’un webhook dans SonarQube

Le webhook va permettre à SonarQube d’envoyer à Jenkins le résultat de l’analyse de la qualité.

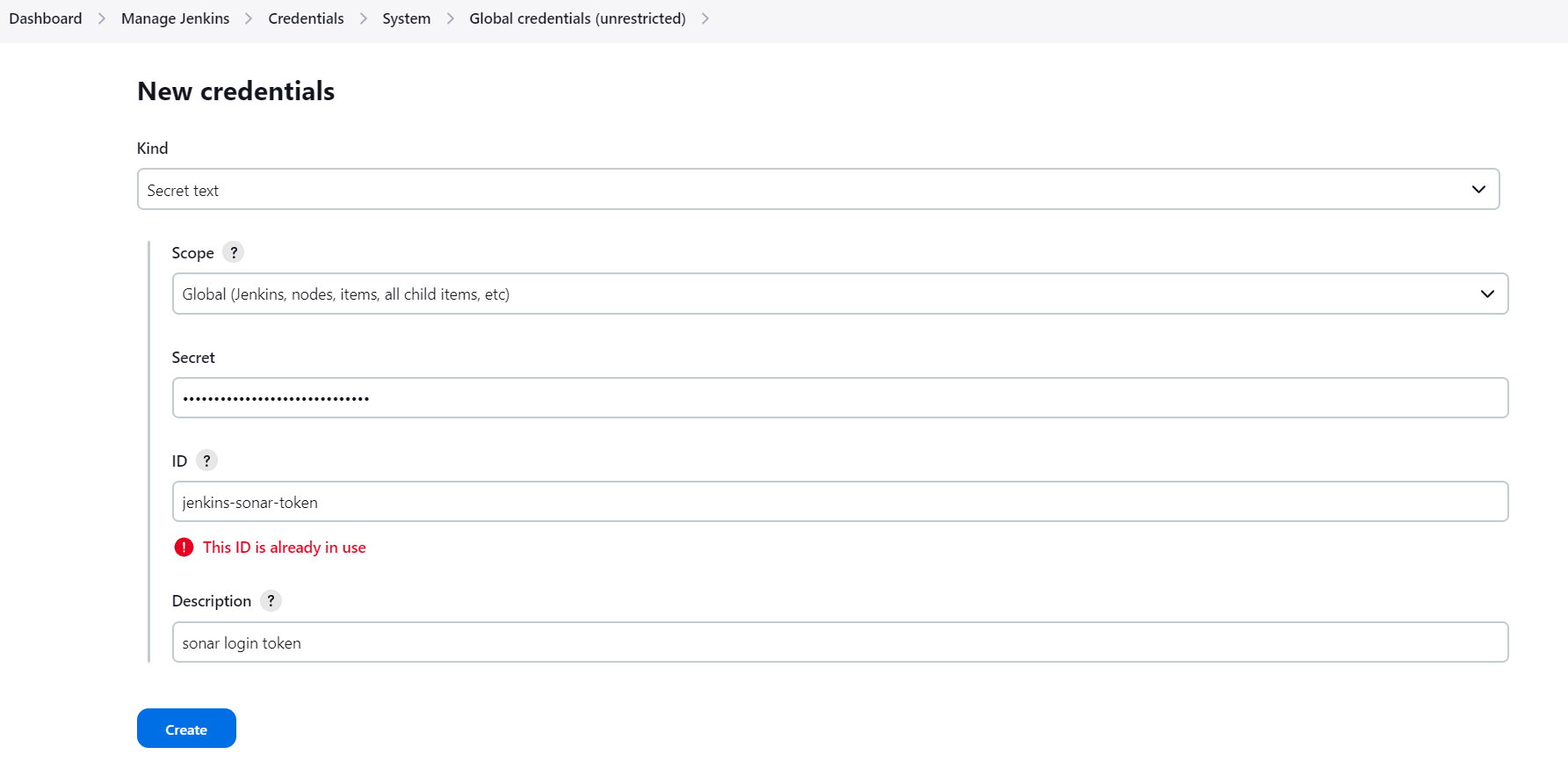


Renseignez l’url du webhook comme ceci : **http://<JENKINS-IP>:8080/sonarqube-webhook/**



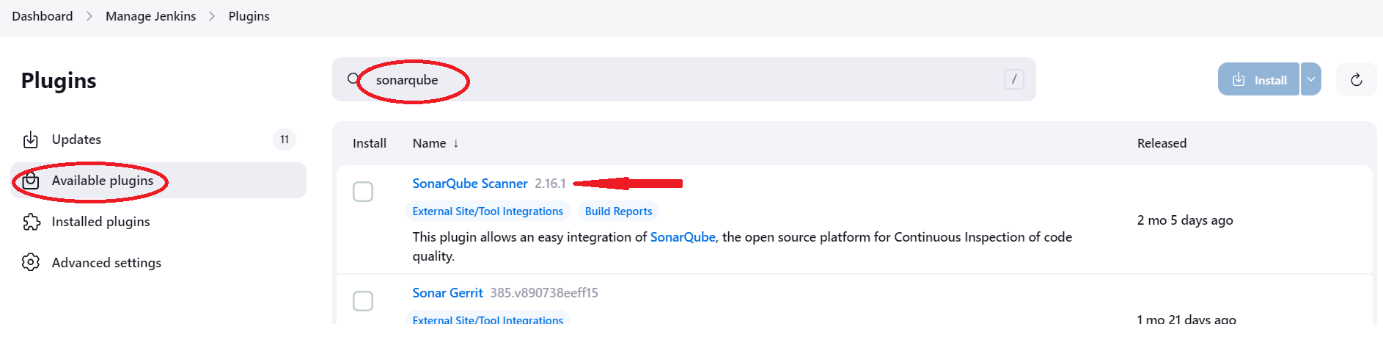
* Ajoutez le token SonarQube comme secret dans Jenkins

Renseignez le token généré dans SonarQube afin que Jenkins puisse s’authentifié auprès de SonarQube.



* Renseignez l’url du serveur SonarQube dans jenkins

Afin de renseigner l’url de sonarQube dans Jenkins, il faut tout d’abord installer le plugin « SonarQuabe Scanner » dans Jenkins comme illustré ci-dessous :



Indiquez l’url de SonarQube à Jenkins



Rajoutez un script pipeline qui va permettre à SonarQube de vérifier la qualité du code en vous aidant de la documentation fournie en annexe.

Lancez le Job et vérifiez que l’étape d’analyse du code par SonarQube est ok.

**Step 4 : Build de l’image docker**

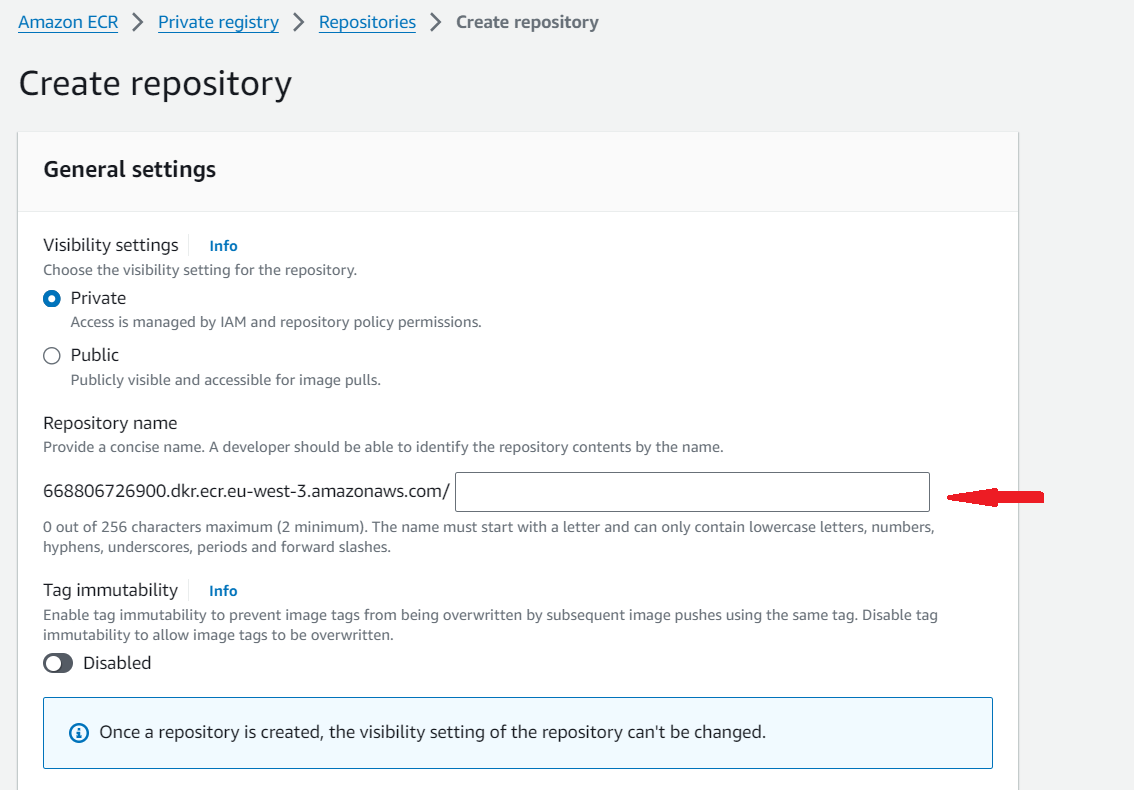
Installez tout d’abord Docker dans le serveur CI-SERVER à l’aide de la documentation fournie en annexe.

* **Création du registre docker AWS ECR**

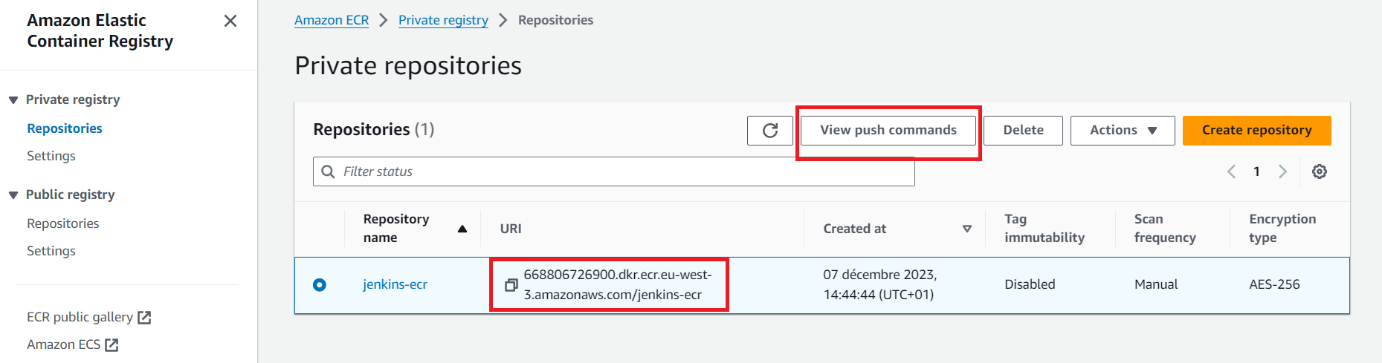
Nous allons créer un registre AWS ECR dans lequel nous allons de publier l’image Docker.

Dans la barre de recherche, recherchez le service « Elastic Container Registry ».

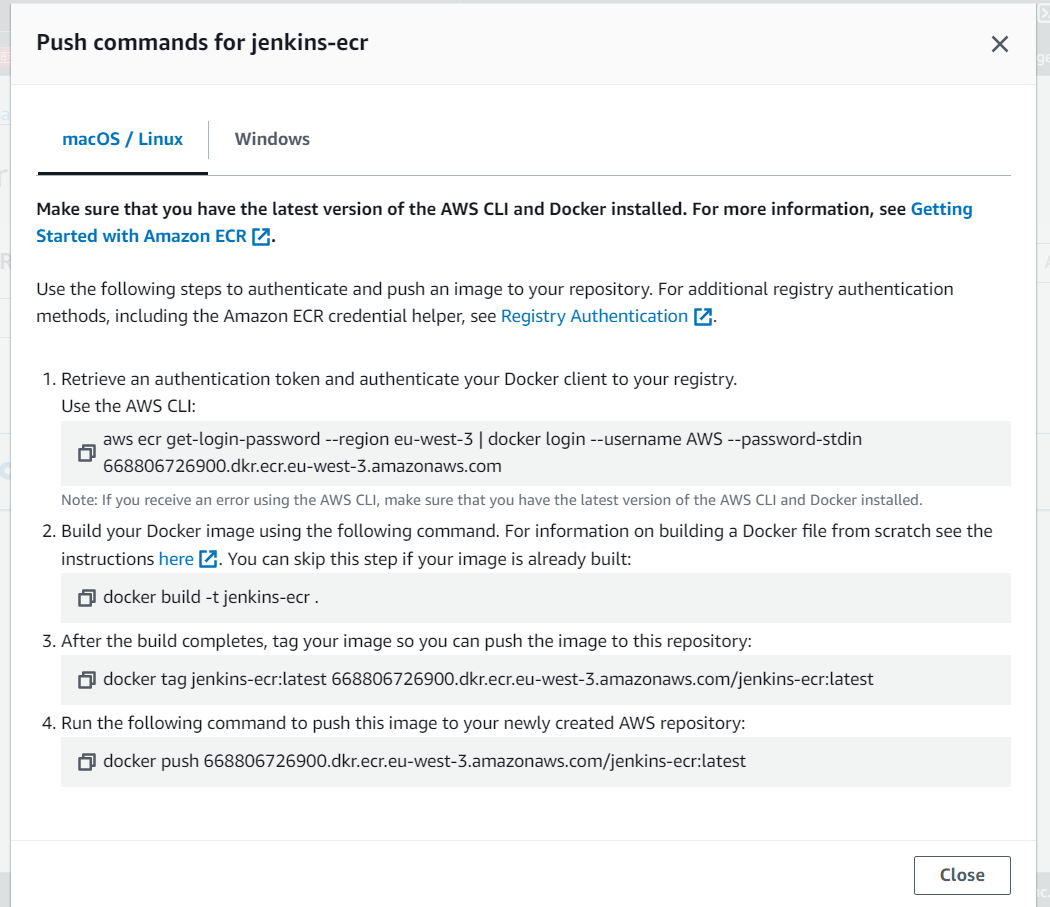
Créer un registre privé et donnez un nom à votre registre comme indiqué ci-dessous :



Une fois le registre crée, vous devez avoir un repository comme illustré ci-dessous :



En cliquant sur le bouton «View push commands », on a accès à une série d’instruction permetant de publier une image Docker dans le reposotory que nous venons de créér comme indiqué ci-dessous :



* **Création de l’image Docker**

Modifiez ensuite le fichier DockerFile présent à la racine du projet Git afin de générer une image Docker à partir du livrable Java généré dans la Step 2 du pipeline présent dans le répertoire : target/ci-cd-pipeline-0-0-1-SNAPSHOP.jar

Pour rappel, voici quelques exemples de commandes à exécuter pour lancer une application Java :

* java –jar <POROJECT\_NAME>.jar
* java –jar <POROJECT\_NAME >.jar -Dserver.port=8081 (redéfinition du port)

Générez ensuite une image Docker en prenant en compte les caractéristiques du registre AWS ECR configuré précédemment. L’image générée doit être sous la forme :

**<ACCOUNT>/<IMAGE\_NAME> :<BUILD\_NUMBER>**

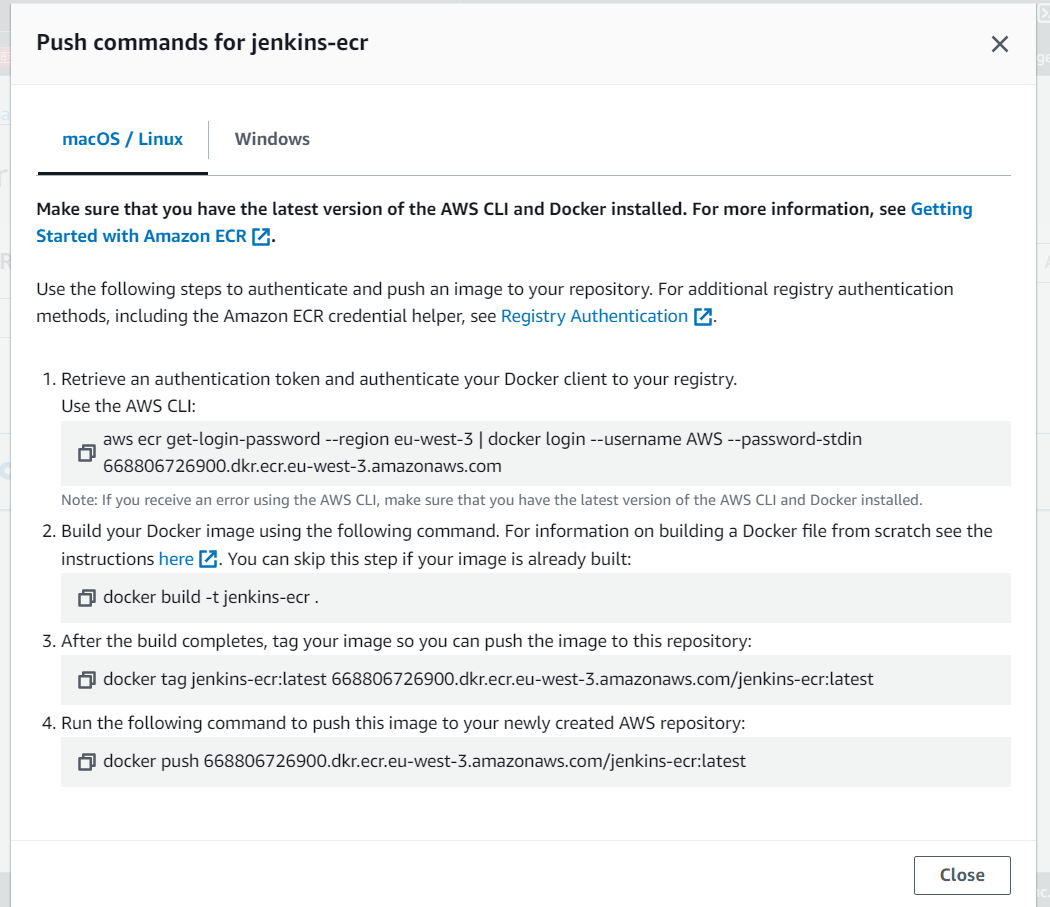
Assurezd’avoir installé tous les plugins Docker comme indiqué en annexe et ensuite rajoutez un script pipeline qui va permettre de générer l’image Docker.

Lancez le Job et vérifiez que l’image Docker est bien générée en exécutant la commande suivante : docker image ls

**Step 5 : Publiez l’image docker dans le registre AWS ECR**

L’objectif de cette étape est de publier l’image générée à l’étape précédente dans le registre Docker AWS ECR que nous avons créé.

Afin de publier l’image Docker que nous avons généré cliquez sur le bouton « view push commands » de votre repository AWS ECR comme indiqué ci-dessous :



Rajoutez un script pipeline qui va permettre de publier l’image Docker dans le registre AWS ECR.

Lancez le Job et vérifiez que l’image a bien été publiée dans le registre Docker AWS ECR.

**Step 6 : Clean up des Images**

Faire un nettoyage de l’espace de travaille en supprimant localement les images générées.

**Step 7 : Déployez l’image dans le serveur CD-SERVER**

Cette étape a pour objectif de déployer l’image Docker présente dans le registre Docker AWS ECR dans le serveur de déploiement CD-SERVER à l’aide d’Ansible.

Installez tout d’abord Ansible dans le serveur CI-SERVER afin de pouvoir exécuter des commandes Ansible.

* Génération d’une paire de clé SSH

L’authentification d’Ansible au serveur CD-SERVER doit se faire par une connexion SSH, vous devez donc posséder ou générer une paire de clés SSH et copier la clé publique dans le répertoire ~/.ssh/authorized\_keys du serveur CD-SERVER.

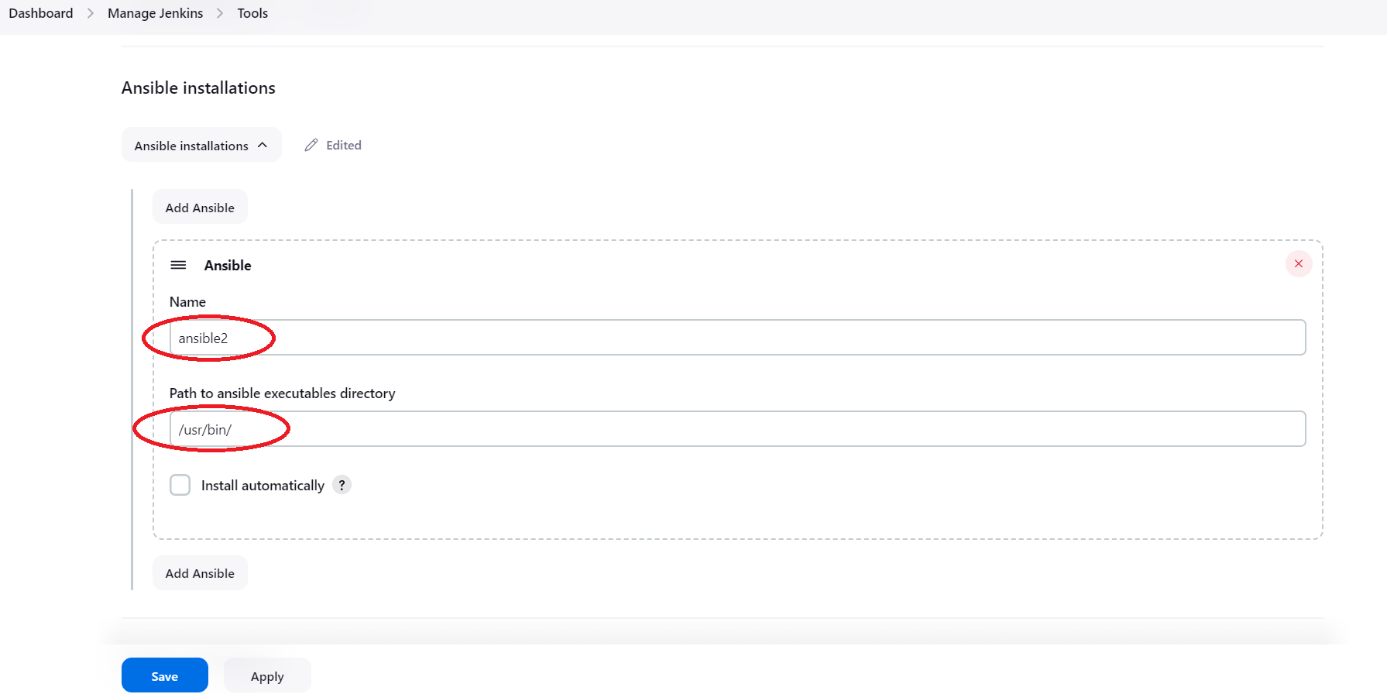
* Renseigner les fichiers d’inventaire et playbook

Modifier le playbook et l’inventaire présent à la racine du projet afin que Ansible puisse récupérer l’image Ansible présente dans le registre AWS ECR et lancer l’application packagé dans l’image Docker.

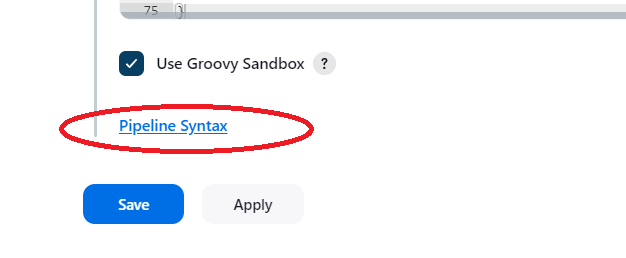
* Indiquez à Jenkins où se trouve l’installation d’Ansible

Afin de renseigner la localisation d’Ansible dans Jenkins, il faut tout d’abord installer le plugin Ansible dans Jenkins.

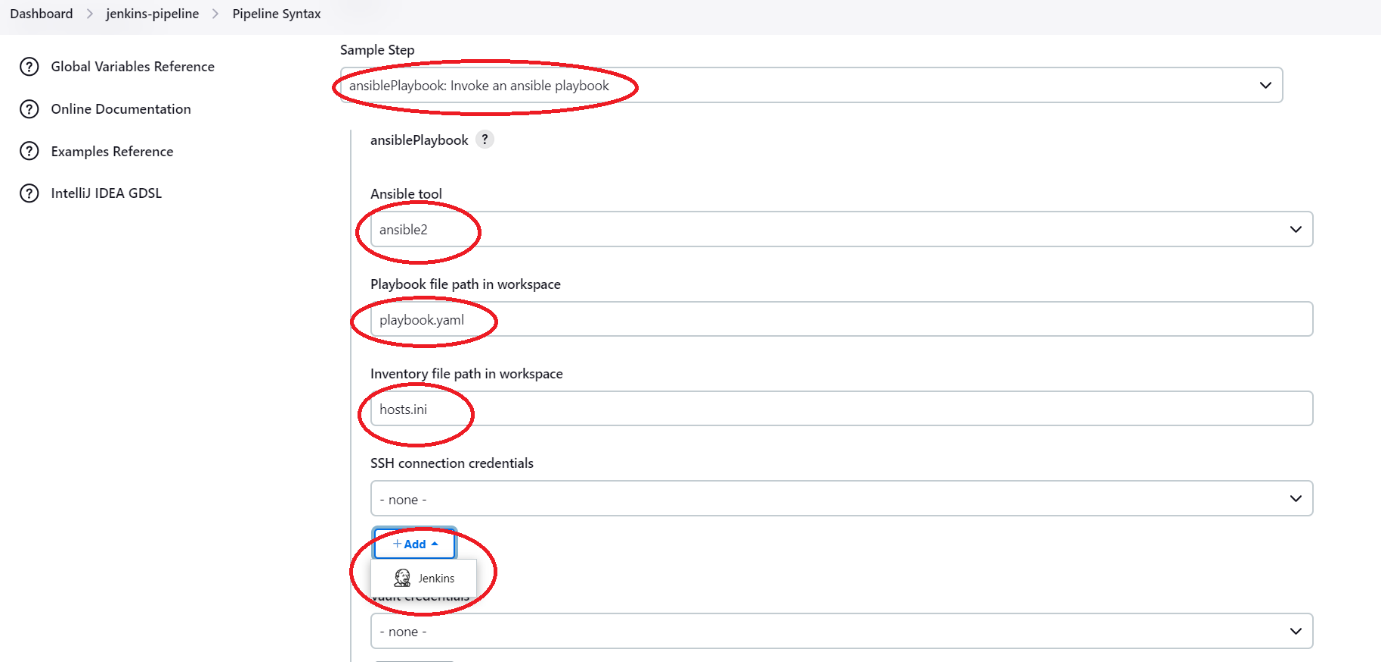
Puis indiquez à Jenkins où se trouve le chemin d’installation d’Ansible. La commande linux « which ansible » permet de savoir où se trouve l’exécutable ansible.



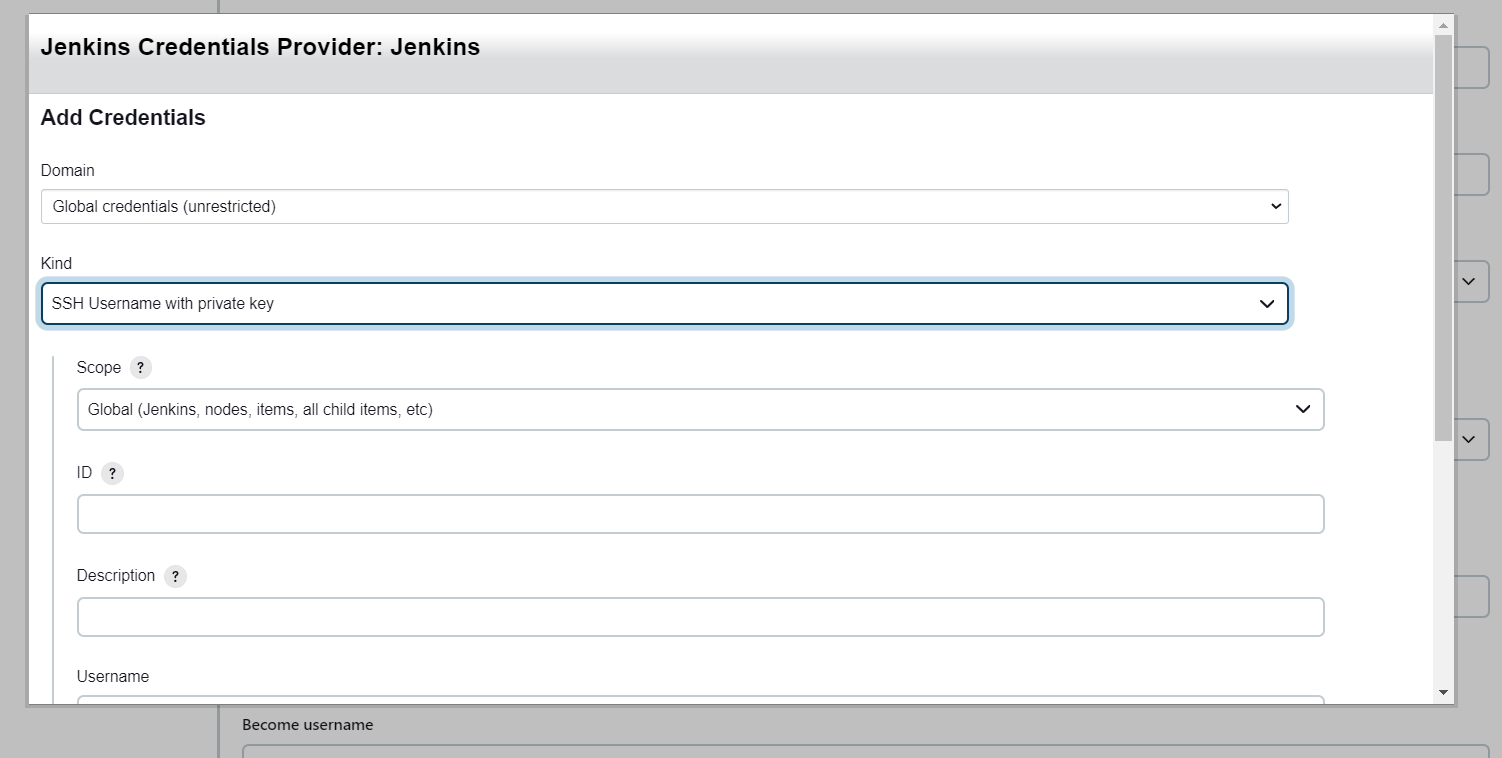
* Générez le script Ansible de déploiement à l’aide de la syntaxe « Pipeline Syntax »



Comme indiqué ci-dessous, renseignez ensuite le playbook et l’inventaire présent à la racine du projet ainsi que la clé privée pour permettre à Ansible de se connecter au serveur CD-SERVER par clé SSH.



Ajoutez la clé privée en renseignant toutes les informations de cette pop-up**:**

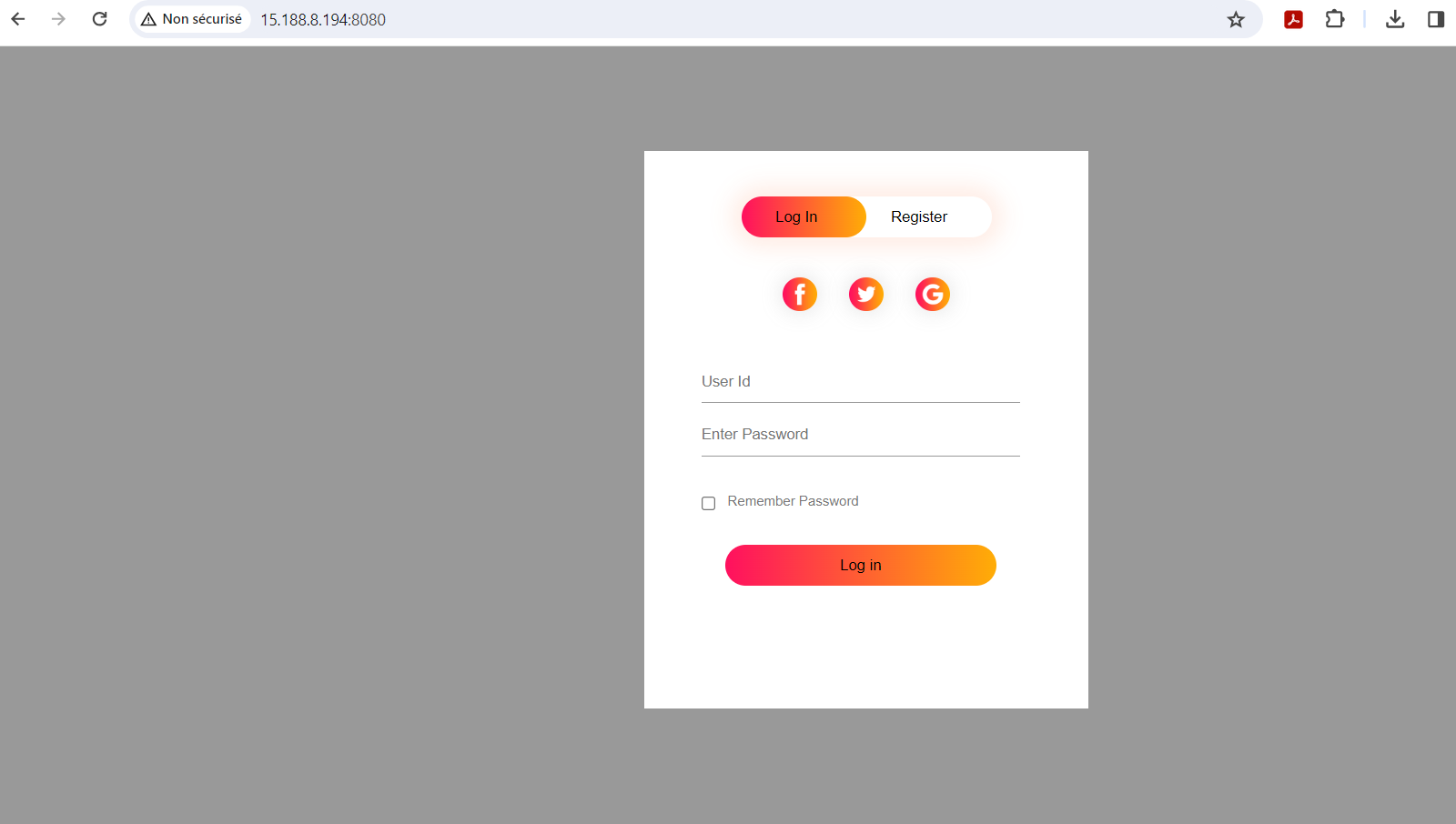


L’application étant déployée sous forme de conteneur Docker, assurez-vous que docker soit bien installé dans le serveur CD-SERVER.

Lancez le job et vérifiez que l’image à bien été déployée dans le serveur de déploiement CD-SERVER.

L’application est accessible à l’adresse : **<CD-SERVER>:8081**

Vous devriez accéder à une page de login comme indiqué ci-dessous :



**ANNEXE**

**Plugin Jenkins à installer**

* SonarQube Scanner for Jenkins
* Docker plugin
* Docker Pipeline
* Amazon EC2 plugin
* Ansible plugin

**Références :**

Jenkins

* [Jenkins on AWS](https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/)
* [jenkins linux rpm](https://get.jenkins.io/redhat/)
* [jenkins declarative pipeline](https://www.jenkins.io/doc/book/pipeline/syntax/)
* [Jenkins Variables d’environnement par défaut](https://wiki.jenkins.io/display/JENKINS/Building+a+software+project)

Java

* [install java 11 for amazon linux](https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/amazon-linux-install.html)

Maven

* [index of maven](https://archive.apache.org/dist/maven/maven-3/)

SonarQube

* [sonarqube scanner for jenkins pipeline](https://www.jenkins.io/doc/pipeline/steps/sonar/)

Docker

* [install docker for amazon linux](https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-docker.html)
* [How To Make Docker Images with Jenkins Pipelines](https://hackernoon.com/how-to-make-docker-images-with-jenkins-pipelines-xsh3uza)
* [Building Docker Image using Jenkins Pipeline & Push it to AWS](https://medium.com/@vijulpatel865/building-docker-image-using-jenkins-pipeline-push-it-to-aws-ecr-aa02cc7a295e)

**Repository GIT :**

<https://github.com/oadya/ci-cd-pipeline>

**Connexion SSH**

<https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server>

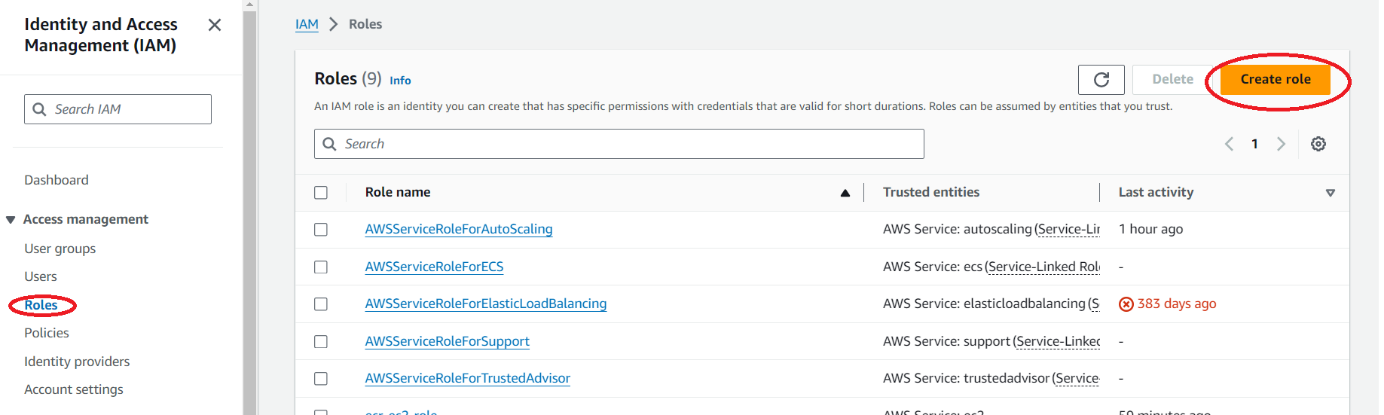
<https://linovox.com/ssh-copy-id-command/>

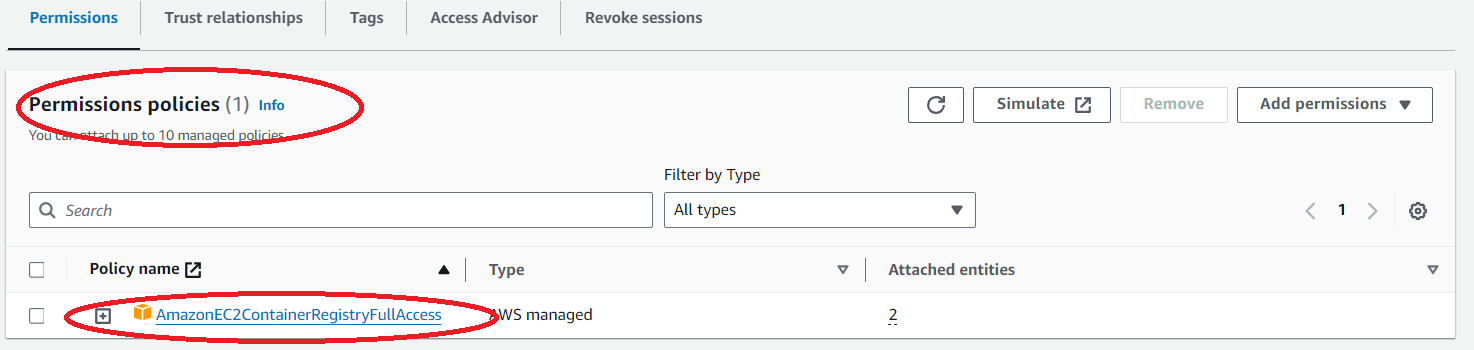
<https://linuxhint.com/use-ssh-copy-id-command/>

**Gestion des permissions avec le service IAM**

Dans AWS, afin que deux services puissent communiquer entre eux, il faut leur accorder des permissions à travers des rôles.

Dans notre cas, pour que les instances EC2 puissent communiquer avec le service AWS ECR, on peut créer un rôle auquel on va attribuer la permission « AmazonEC2ContainerRegistryFullAccess ». Ce rôle devra ensuite être attribué aux instances EC2 afin de leur donner l’accès au service AWS ECR.





**Exemple de script Pipeline dans un fichier Jenkinsfiles :**

pipeline {  
  
    agent any  
  
    environment {  
        NEW\_VERSION = '1.3.0'  
        AN\_ACCESS\_KEY = credentials('my-predefined-secret-text')  
    }  
  
    parameters {  
        string(name: 'FIST\_NAME', defaultValue: 'Mr Jenkins', description: 'Who should I say hello to?')  
    }  
  
    tools {  
        maven 'apache-maven-3.0.1'  
    }  
  
    stages {  
        stage('Step 1') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
  
       stage('Step 2') {  
            steps {  
               sh 'My Second Step'  
               sh '''  
                    echo 'Hello World'  
                   ls -al  
                '''  
            }  
        }  
  
      stage('Step 3') {  
            steps {  
       echo 'Building application'  
                echo 'Building version is ${NEW\_VERSION}'  
            }  
        }  
  
       stage('Step 4') {  
            when {  
                 expression {  
                   BRANCH\_NAME == 'dev' ||  BRANCH\_NAME == 'master'  
                 }  
            }  
            steps {  
                echo 'Testing application'  
            }  
        }  
  
  
  
   stage('Step 5') {  
            steps {  
                echo 'Hello World'  
  
                script {  
                    def browsers = ['chrome', 'firefox']  
                    for (int i = 0; i < browsers.size(); ++i) {  
                        echo "Testing the ${browsers[i]} browser"  
                    }  
                }  
            }  
        }  
  
  stages {  
        stage('Step 6') {  
            options {  
                timeout(time: 1, unit: 'HOURS')  
            }  
            steps {  
                echo 'Hello World'  
            }  
        }  
    }  
  
    }  
  
    post {  
        always {  
            echo 'I will always say Hello again!'  
        }  
       success {  
            echo 'Job success!'  
        }  
      failure {  
            echo 'Job failure'  
        }  
    }  
}