

CI/CD pipeline with GitHub, Maven, Jenkins, SonarQube and docker

SonarQube and Jenkins are installed in Amazon Linux

Pre-requis:

- install **java** in Jenkins and SonarQube servers,
- install **docker** (yum install docker), **maven** and **git** in Jenkins master or Jenkins slave (where the pipeline jobs are running),
- If you run job on master: add Jenkins in docker group (**usermod -aG docker jenkins**)
- *If you run jobs in a node: add the user that you have used to install docker in the docker group and change the owner of docker.sock to the current user like `sudo chown ec2-user:docker /var/run/docker.sock` instead you'll get some error like: **Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock***
- Allow 9000 and 8080 ports to your Security group or in your firewall if you use your localhost machine

Sonarqube and Jenkins installation

- `wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.1.69595.zip`
- `unzip sonarqube-9.9.1.69595.zip`
- `mv sonarqube-9.9.1.69595 /opt/sonarqube`
- `useradd "new_user"`
- Modify the owner cause root user cannot run sonarqube:
`chown -R new_user: new_user /opt/sonarqube/`
- Connect to the user ---> `su new_user`
- `cd /opt/sonarqube/bin/linux-x86-64`
- run sonarqube with: `./sonar start`,
Others option--- > `status`, `restart`, `stop`
- `http://ip_address:9000`

- Installation of Jenkins follow this link---> [Linux \(jenkins.io\)](https://linux.jenkins.io)

Add Sonarqube to Jenkins

- From the Jenkins Dashboard of jenkins, navigate to **Manage Jenkins > Manage Plugins** and install the **SonarQube Scanner plugin**.
- Back at the Jenkins Dashboard, navigate to **Credentials > System** from the left navigation.
- Click **the Global credentials (unrestricted)** link in the System table.
- Click **Add credentials** in the left navigation and add the following information:
- Kind: **Secret Text**
- Scope: **Global**
- Secret: Generate a token at **My Account > Security > User > Token** in SonarQube, and copy and paste it to the secret text.
- From the Jenkins Dashboard, navigate to **Manage Jenkins > Configure System**.
- From the **SonarQube Servers section**, click **Add SonarQube**. Add the following information:
- **Name**: Give a unique name to your SonarQube instance.
- **Server URL**: Your SonarQube instance URL.
- **Credentials**: Select the credentials created during step 4.
- Click Save

Build with maven, analyze with sonarqube, build docker image and push to Docker hub

Add maven it in the **Global Tool** in Manage jenkins. Give a name like Maven3 and the

path of maven [/usr/share/maven](#)

NB: nom de l'image doit être tout en minuscule

Si on veut supprimer une image docker et que ya un conteneur qui s'exécute avec cette image, il faut d'abord stop le conteneur (docker stop id_conteneur), supprimer le conteneur (docker rm id_conteneur) et ensuite supprimer l'image (docker rmi non_image ou id_image)

Adding node to run our pipeline

- Go to **Dashboard --> Manage Jenkins --> Nodes --> New node**. Give the node name and select **Permanent Agent**

The screenshot shows the 'New node' configuration page in Jenkins. The 'Name' field is filled with 'mynode'. The 'Description' field is empty, with a '[Plain text] Preview' link below it. The 'Number of executors' field is filled with '1'. The 'Remote root directory' field is filled with '/home/ec2-user'. The 'Labels' field is filled with 'groupe1'. There are help icons (?) next to the 'Description', 'Number of executors', 'Remote root directory', and 'Labels' labels.

RESTART THE SERVICE IF JENKINS IS RUNNING ON WINDOWS BEFORE WRITING THE COMMAND TO CONNECT

Number of processor: The maximum number of **concurrent** builds that Jenkins may perform on this node. A good value to start with would be the number of CPU cores on the machine. Setting a higher value would cause each build to take longer.

Remote root directory: An agent needs to have a directory dedicated to Jenkins. Specify the path to this directory on the agent.

Labels: Labels (or tags) are used to group multiple agents into one logical group.

For example, if you have multiple Windows agents and you have a job that must run on Windows, then you could configure all your Windows agents to have the label `windows`, and then tie that job to this label.

This would ensure that your job runs on one of your Windows agents, but not on any agents without this label.

Usage ?

Use this node as much as possible

Launch method ?

Launch agent by connecting it to the controller

☐ Disable WorkDir ?

Custom WorkDir path ?

Internal data directory ?

remoting

☐ Fail if workspace is missing ?

☒ Use WebSocket ?

Advanced ▾

☐ Fail if workspace is missing ?

☒ Use WebSocket ?

Advanced ▾

Availability ?

Keep this agent online as much as possible

Node Properties

☐ Disable deferred wipeout on this node ?

☐ Environment variables

☐ Tool Locations

Save

- Click on save and click on the **name of the node to show command to run in the slave**
- Run the following based on your OS. Add '&' to the last command to run the agent on the background.

Run from agent command line: (Unix)

```
curl -sO http://34.221.197.154:8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl http://34.221.197.154:8080/computer/kk/jenkins-agent.jnlp -secret
4726d7b8e64c4ea3f67dc38d2f924bd245ec7986ab53aaa9c9ca3c9cefba9270 -workDir "/home/ec2-user"
```

Run from agent command line: (Windows)

```
curl.exe -sO http://34.221.197.154:8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl http://34.221.197.154:8080/computer/kk/jenkins-agent.jnlp -secret
4726d7b8e64c4ea3f67dc38d2f924bd245ec7986ab53aaa9c9ca3c9cefba9270 -workDir "/home/ec2-user"
```

Or run from agent command line, with the secret stored in a file: (Unix)

```
echo 4726d7b8e64c4ea3f67dc38d2f924bd245ec7986ab53aaa9c9ca3c9cefba9270 > secret-file
curl -sO http://34.221.197.154:8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl http://34.221.197.154:8080/computer/kk/jenkins-agent.jnlp -secret @secret-file -workDir "/home/ec2-user"
```

Or run from agent command line, with the secret stored in a file: (Windows)

```
echo 4726d7b8e64c4ea3f67dc38d2f924bd245ec7986ab53aaa9c9ca3c9cefba9270 > secret-file
curl.exe -sO http://34.221.197.154:8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl http://34.221.197.154:8080/computer/kk/jenkins-agent.jnlp -secret @secret-file -workDir "/home/ec2-user"
```

NB: songer à remplacer l'@ IP par l'@ IP de votre Jenkins Server lorsque vous copiez ces commandes.

Or install jar file manually with: **wget** `http://<JENKINS_URL>/jnlpJars/agent.jar`

If curl does not work "Error: Invalid or corrupt jarfile agent.jar" and continue the other command

Adding Docker hub Credential

- Generate a dockerhub token
- Go to pipeline syntax and choose **withCredentials** -- > give a **variable_name** -- > add credential (choose secret text) -- > **generate pipeline**
- **Name of image be the same as the docker repository to avoid errors**

```
stage('Building docker image'){
    steps{
        sh 'docker build -t ngomansible/my_private_repo:latest .'
    }
}

stage('Push image to DockerHub'){
    steps{
        withCredentials([string(credentialsId: 'DockerToken', variable: 'docker_cred')]) {
            sh 'docker login -u ngomansible -p ${docker_cred}'
        }
        sh 'docker push ngomansible/my_private_repo:latest'
    }
}
```

Deploy to Kubernetes

1. Installer kubectl

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\"$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
sudo yum install -y kubectl
```

2. Install minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm
```

```
sudo rpm -Uvh minikube-latest.x86_64.rpm
```

```
sudo usermod -aG docker $USER && newgrp docker
```

```
chown ec2-user:docker /var/run/docker.sock
```

```
minikube start
```

3. Install the plugin **Kubernetes CLI plugin** in jenkins

4. In the node master type **cat ./kube/config** and save the whole content of the file to a new file

5. In **pipeline syntax**, choose **withKubeConfig** -- > **add a credential** --> **upload the file** that you created earlier **and give ID** -- > **generate pipeline** -- > **copy/paste to your pipeline script**.

```
stage('Deploy to kubernetes'){
  steps{
    withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s-id',
      sh 'kubectl apply -f myDeployment.yml'
    }
  }
}
```

The deployment file must be in the git repository.

Access your app by typing: **minikube service service_name --url &**

It will give the address to access your app