**JENKINS**

**Build Tool installation:**

**Make sure you have installed Maven and JDK8 in Jenkins. Follow below steps to install JDK8**

1. **SSH to Jenkins instance**
2. **Run commands**

* **Sudo su -**
* **Sudo apt update**
* **Sudo apt install openjdk-8-jdk -y**
* **ls /usr/lib/jvm**
* **copy the path like - /usr/lib/jvm/java-1.8.0-openjdk-amd64**

1. **Go to Jenkins dashboard**
2. **Click on “Manage Jenkins” – Click on “Global Tool Configuration”**
3. **Go to “JDK Installations” – Click Add – Give name as “OracleJDK8”**
4. **Paste the JDK8 installation path which we have copied to “JAVA\_HOME”**
5. **Now go down and select “Maven Installations”**
6. **Give name as “MAVEN3” – select the version as 3.\* and Click Save**

**Create Our First Job**

1. **Click on “Create Job”**
2. **Give Job name as “Build”**
3. **Select template “Freestyle project” and click ok**
4. **Give description as “sscademy maven build project”**
5. **Select “Git” and paste the GitHub link**
6. **Mention the Branch name as “ss-rem”**
7. **Go to “Build Steps” option – Add build step – Select “Invoke top-level Maven targets”**
8. **Select Maven version – “MAVEN3”**
9. **Type in Goals – “install”**
10. **Click on “Save”**
11. **Go to Dashboard**
12. **Go to Build Job and click on “Build Now”**
13. **You can go to “Console output” to check the logs**
14. **Go to Workspace to see the complete files Jenkins has used to Build the artifact**
15. **To save artifact we have to add Post Build action**
16. **Now Click on “Configure”**
17. **Click on “Add post-build action” – Select “Archive the artifact”**
18. **Type “ \*\*/\*.war” in Files to archive**
19. **Click Save**
20. **Run Build now again**

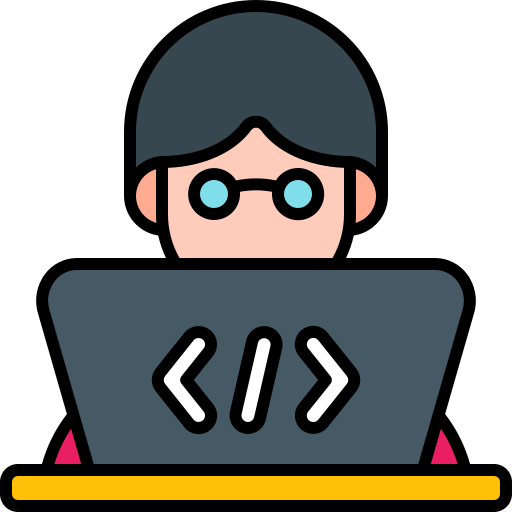
**What is MAVEN**

**Maven is a Build tool. (Refer PPT)**

**Versioning the Artifact**

**Since in each build, the artifact is getting replaced with the new artifact. Just consider if you want to keep each artifact ran on every build. To achieve this, we have to version the artifact.**

**Flow of Continuous Integration**



**Developer(git)**



**GitHub**

Fetch Code

Build

Unit Test

Code Analysis

Upload Artifact



**Steps for Continuous Integration**

1. Jenkins Setup
2. Nexus Setup
3. SonarQube Setup
4. Security Setup
5. Plugins
6. Integrate

* Nexus
* SonarQube

1. Write pipeline script
2. Set notifications

Jenkins Setup:

1. Create an EC2 instance

* Give name as you wish “Jenkins-server”
* Select AMI “Ubuntu 20.04 LTS”
* Select instance type : t2.small
* Create key pair
* Edit incoming securing group –
  + - Allow port 22 from anywhere
    - Allow port 8080 from anywhere
    - Allow port 80 from anywhere
* Copy the Jenkins setup shell script in the user data field
* Launch instance

Nexus setup

1. Create an EC2 instance for Nexus server

* Give name as “Nexus server”
* Select AMI from marketplace – CentOS 7
* Select instance type – t2.medium
* Create key pair – nexus key
* Create security group
  + - Name – nexusSG
    - Allow port 22 from anywhere
    - Allow port 8081 (nexus runs on port 8081)
* Copy paste Nexus shell script into user data field

SonarQube setup

1. Launch instance for Sonar Server

* Give name as “SonarServer”
* Use AMI as “Ubuntu 18.04 LTS”
* Instance type: t2.medium
* Create key pair – sonar key
* Create security group
  + - Name – sonarSG
    - Allow port 22 from anywhere
    - Allow port 80 from anywhere
    - Allow port 9000 from anywhere
* Copy pastes the Sonar shell script into user data field
* Launch instance

Check the services up and running

1. Login to Jenkins server and check Jenkins service is running using below command

Systemctl status Jenkins

Access from browser: Public\_ip:8080

1. Login to Sonar server and check the sonar service is running

Systemctl status sonarqube

Access from browser: Public\_ip

1. Login to Nexus server and check the nexus service is running

Systemctl status nexus

Access from browser: Public\_ip:8081

**Plugins Installation**

* Nexus
* Sonarqube
* Git
* Pipeline Maven Integration
* BuildTimeStamp

Go to Manage Jenkins 🡪 Manage Plugins 🡪 Click on Available 🡪 Search for “Nexus” and

Select “Nexus artifact uploader” 🡪 Search for “Sonarqube scanner” 🡪 Search for “Build timestamp” 🡪 Search for “Pipeline Maven Integration” 🡪 last one “Pipeline utility steps”

Click on “Install without restart”

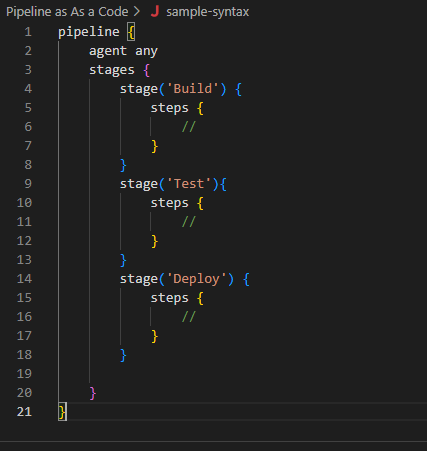
**Pipeline As a Code**

* Automate pipeline setup with Jenkinsfile
* Jenkinsfile defines stages in CI/CD Pipeline
* Jenkinsfile is a text file with Pipeline DSL Syntax
* Similar to Groovy
* Two Syntax
  + Scripted
  + Declarative

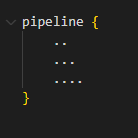
**Pipeline Concept**

* Pipeline
* Node/Agent
* Stage
* Step

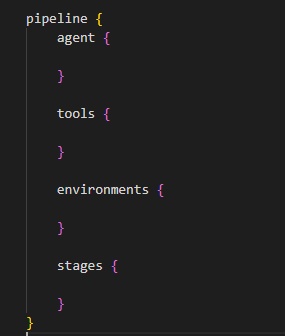
**Sample pipeline syntax**



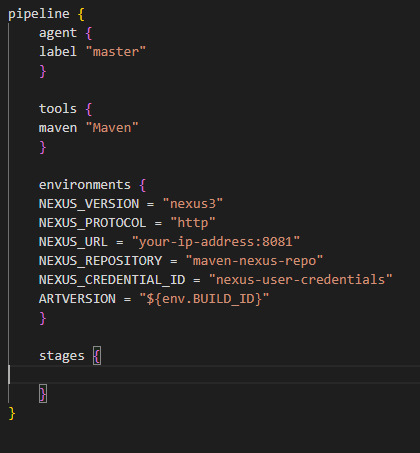
**Pipeline block**



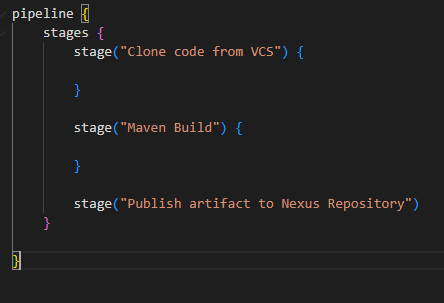
**Pipeline components**

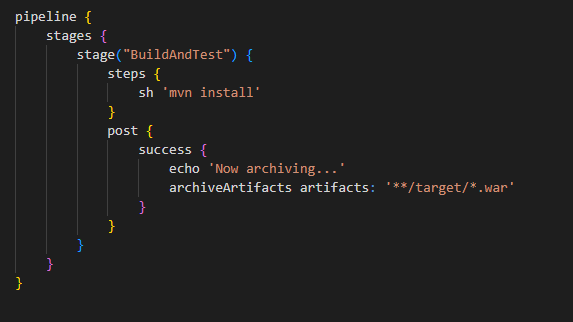


**Pipeline components with details**



**Stages**





<https://www.jenkins.io/doc/book/pipeline/>

**Code Analysis**

* Detects vulnerabilities and functional errors
* Best practices
* Vulnerabilities in the code (top 10 OWASP vulnerabilities)
* Functional Errors before deployment

**Tools in the market for Code analysis**

* Checkstyle
* Cobertura
* Mstest
* Owasp
* SonarQube Scanner
* Etc.…

**SonarQube Tools installation in Jenkins**

* Go to “Manage Jenkins” 🡪 “ go to “Global Tool Configuration”
* Go to ‘Sonarqube scanner” option 🡪 Click “Add Sonarqube scanner”
* Give name “sonar4.7” 🡪 save it

**Integrate SonarQube with Jekins**

* Go to “Manage Jenkins” 🡪 Click on “Configure system” 🡪 Go to “SonarQube Servers” 🡪 Select “Add SonarQube server” 🡪 check mark “Environmet variables” 🡪 give a name “sonar” 🡪 type the sonarqube server IP address 🡪

<http://172.88.90.22> & save it

**Now create a token for authentication**

* Go to SonarQube server URL
* Login there
* Click on Profile logo
* Go to “My Account”
* Go to security
* Give a name for token
* Click on generate
* Copy the generated token

Now go to “Configure system” in Jenkins and go to SonarQube server & add the credentials by selecting “secret text” & give the token ID as “sonartoken”

<https://www.jenkins.io/doc/pipeline/steps/sonar/>

**Quality Gates in SonarQube**

* Lets create gate for our SonarQube project
* Login to SonarQube server
* Go to Quality Gates option on the top
* Create a Quality Gate – give any name e.g: “sscademy-QG”
* Click “Add condition” 🡪 select “On overall Code”
* Select metrics 🡪 “Bugs” 🡪 Give “60”
* Click Add condition
* Come to “Project” option on SonarQube
* Click on our project
* Go to “Project settings”
* Click on “Quality gate”
* Select the newly created Quality Gate
* Click again on “Project settings”
* Click on “Webhooks”
* Click “Create”
* Give name as “Jenkins-ci-webhooks”
* Type URL as 🡪 <http://jenkins_public_ip:8080/sonarqube-webhook>
* Click Create

**Software Repository (Nexus)**

* Nexus sonatype is a software repository. It is the place where we keep the software and retrieve from it.

**Software repositories**

* Maven (Used for maven dependencies)
* Apt (Package for Debian based systems)
* Yum (Packages for Redhat based systems)
* Nuget (Package manager for .NET)
* Npm (Package manager for JavaScript)
* Docker (Registry to store Docker Images)

**Setup Nexus Repository**

* Login to Nexus server using URL : <http://public_ip_of_nexus_server:8081>
* Click on Settings button
* Go to Repositories
* Click create Repository
* Select “maven2 (hosted)”
* Give a name like “sscademy-repo”
* Go down and click on “Create Repository”
* Now we will set the credentials
* Go to Jenkins
* Click on “Manage Jenkins”
* Click on “Manage credentials”
* Click on “Jenkins symbol”
* Click on “Global credentials”
* Click on “Add Credentials”
* Kind “Username and Password”
* Give username and password for Nexus server
* Give ID name as “nexuslogin”

Now setup the pipeline for Nexus repository

<https://github.com/jenkinsci/nexus-artifact-uploader-plugin>

* Set value “Build Timestamp” plugin 🡪 go to Jenkins 🡪 Click on “Manage Jenkins” 🡪 “configure system” 🡪 Go to “Build TimeStamp”
* Give the pattern as “yy-MM-dd-HH-mm”
* Now create a pipeline
* Run the pipeline – on success go to Nexus server
* Click on “Browse”
* Go to sscademy-repo
* There we should see our Artifact uploaded

**Pipeline Automate Trigger**

**Popular Triggers**

1. Git Webhooks
2. Poll SCM
3. Scheduled JOB
4. Remote Triggers
5. Build after other projects are built

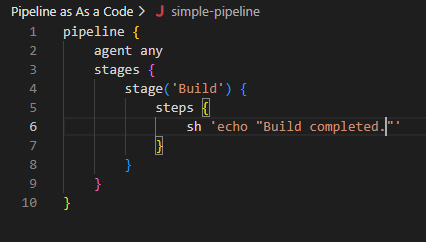
**Steps**

1. Create a git repository
2. Ssh auth
3. Create a Jenkinsfile in GIT repo & commit
4. Create Jenkins job to access Jenkinsfile from GIT repo
5. Test triggers
6. Create a GIT repository

* Login to github account
* Create a new repository – give name as “jenkinstrigger”
* Select “Private” repo
* Create repository

1. Create SSH key

* Create SSH key 🡪 Open gitbash
* Type “ssh-keygen.exe”
* Create a SSH key
* You can see the SSH keys in “ **ls ~/.ssh ”**
* Copy the content of the public key
* Come to GITHUB account 🡪 go to GITHUB settings
* Go to “SSH and GPG key”
* Click on “New ssh key”
* Give a title “projectkey”
* Go to repository and copy the ssh link not http link
* Clone the repo
* Create a simple pipeline as below – give name as “**Jenkinsfile**”



* Commit and push it to the repository

1. Create Jenkins job to access Jenkinsfile from GIT repo

* Go to Jenkins dashboard
* Go to “Manage Jenkins”
* Click on “Configure Global Security”
* Go to “Git Host Key Verification Configuration” option
* Select option as “Accept first connection”
* Save
* Now create a pipeline
* In the pipeline section select “Pipeline script from SCM”
* Select SCM as “git”
* Give the Repository URL of SSH
* Click Add credential 🡪 Select kind as “SSH username with private key”
* Give ID as “gitsshkey” 🡪 username, type github account name
* Select private key 🡪 enter directly
* Copy the SSH private from which we have generated
* Click add
* Select the credential which we added
* Click save
* Click on Build now

**WEBHOOKS**

* Copy the Jenkins server URL ( http://public\_ip:8080)
* Go to GitHub repository 🡪 go to GitHub Repository settings (not account settings)
* Go to option “Webhooks”
* Click on “Add webhook”
* Paste the Jenkins URL along with /github-webhook/
* Like this <http://Jenkins_public_ip:8080/github-webhook/>
* In Content type 🡪 select “application/json”
* Select “Just the push event”
* Click Add webhook
* Now to go to Jenkins job
* Click on Configure
* In the “Build Triggers” 🡪 Select “GitHub hook trigger or GITScm polling”
* Click Save
* Now do the commit to the repository

**POLL SCM**

* Go to Jenkins job
* Click Configure
* Click on “Poll SCM”
* Type “ \* \* \* \* \* “ 🡪 which means it which check for every minute, every hour, every day of the month, every month, every day of the week.
* Click save
* Click on “GIT Polling log”
* Create new commit to the repository

**SCHEDULED JOB**

* Go to Jenkins job
* Click Configure
* Click on “Build periodically”
* It will be same cron job format
* Type “ 30 20 \* \* 1-5 “ 🡪 Monday to Friday 8:30pm
* Click save

**REMOTE TRIGGER**

**Generate JOB URL**

1. Job Configure => Build Triggers

2. Check mark on “Trigger builds remotely”

3. Give a token name

4. Generate URL & save in a file

**Generate Token for User**

1. Click your username drop down button (Top right corner of the page)

2. configure => API Token => Generate

3. Copy token name and save username:tokenname in a file

**Generate CRUMB**

1. wget command is required for this, so download wget binary for git bash

2. Extract content in c:/program files/Git/mingw64/bin

3. Run below command in Git Bash, (replace username,password,Jenkins URL) wget -q --auth-no-challenge --user username --password password --output-document - 'http://JENNKINS\_IP:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

4. Save the token in a file

**Build Job from URL**

By now we should have below details

1. JENKINS Job URL with token

E:g http://52.15.216.180:8080/job/vprofile-Code-Analysis/build?token=testtoken

1. API Token

USERNAME:API\_TOKEN E:g admin:116ce8f1ae914b477d0c74a68ffcc9777c

1. Crumb

E:g Jenkins-Crumb:8cb80f4f56d6d35c2121a1cf35b7b501

Fill all the above details in below URL and Execute

curl -I -X POST http://username:APItoken @Jenkins\_IP:8080/job/JOB\_NAME/build?token=TOKENNAME -H "Jenkins-Crumb:CRUMB"

e:g curl -I -X POST http://admin:110305ffb46e298491ae082236301bde8e@52.15.216.180:8080/job/ vprofile-Code-Analysis/build?token=testtoken -H "Jenkins-Crumb:8cb80f4f56d6d35c2121a1cf35b7b501”

**Jenkins MASTER and SLAVE concept**

**Use Cases**

* **Load Distribution**
* **Cross-platform build**
* **Software testing**

Master Server

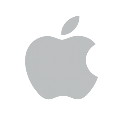
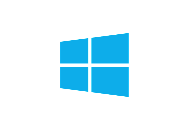


Software Testing Server

Node  
 MAC

Node  
 Linux

Node  
Windows



**Pre-requisites for Node Setup**

* **Any OS**
* **Network Access from Master to Slave**
* **Check firewall rules**
* **JAVA, JRE, JDK**
* **User**
* **Directory with user ownership**
* **Tools as required by the Jenkins job** 
  + **E.g., Maven, Docker, Git etc.**

**Setup Slave/Node Machine:**

* **Login to AWS cloud**
* **Launch an EC2 Instance. Here we will add Linux node for example**
  + **Give name as “Jenkins-slave01”**
  + **Create a new security group like – “SlaveSG”**
  + **Allow port 22 from anywhere**
* **Login to the Slave instance and install the pre-requisites**
  + **$ apt update && apt install openjdk-11-jdk**
  + **$ adduser devops**
  + **$ mkdir /opt/Jenkins-slave**
  + **$ chown devops.devops /opt/Jenkins-slave -R**
  + **$ vim /etc/ssh/sshd\_config**
    - **Find the entry with “PasswordAuthentication”**
    - **Uncomment it and give “yes”**
    - **Save and quit**
  + **$ systemctl restart ssh**
* **Login to Jenkins dashboard**
  + **Go to Manage Jenkins**
  + **Under System Configuration 🡪 Go to “Manage Nodes and Clouds”**
  + **Click on “New Node” 🡪 give a name like “Node1”**
  + **Select “Permanent Agent” 🡪 Click ok**
    - **# of executors 🡪 type “5” (5 parallel jobs)**
    - **Remote root directory 🡪 “/opt/Jenkins-slave”**
    - **Label 🡪 Node1**
    - **Launch method 🡪 “Launch agents via SSH”**
      * **Host 🡪 ip\_address\_of\_slave**
      * **Credentials 🡪 Click Add 🡪 Jenkins**
      * **Username with password 🡪 username 🡪 devops**
      * **Password 🡪 type the given password for devops user**
      * **ID 🡪 Node1**
      * **Click Add**
      * **If you want to have Key based login then**
      * **Go to Credentials again 🡪 Select Jenkins**
      * **Select kind 🡪 SSH Username with private key**
      * **Give the username as “ubuntu”**
      * **Copy the .pem key for slave machine and paste it**
      * **ID 🡪 Node1-login-key**
    - **Use password based login as we don’t have directory for ubuntu user with ownership**
    - **Host Key Verification Strategy: “Non verifying verification strategy”**
    - **Click save**
* **Create a Jenkins job**
  + - **Give name as “test”**
    - **Select “Freestyle project”**
    - **Under “Build” 🡪 “Execute shell” 🡪 give some commands like “pwd, whoami, ls-ltr”**
    - **Save it**
    - **Now if you Build the job there is a chance that the job will run on the slave machine**
  + **Login to Node1 and check in the directory /opt/Jenkins-slave**
  + **You will the related file for Jenkins job**
* **Now if you want that job should always runs on that slave only**
  + **Go to Job 🡪 click “Configure”**
  + **Click on “Restrict where this project can be run”**
  + **Type the slave label 🡪 Node1**
  + **Now it will be guaranteed that this job will run only on this slave**

**Jenkins Security (Authentication/Authorization)**

**Methods**

1. **User login**
2. **Jenkins own database**
   1. **Sign up**
3. **LDAP Integration**

**Control Permissions on Jenkins**

* **Admin**
* **Read**
* **Jobs**
* **Credentials**
* **Plugins etc.**

**Control Permissions on Jobs**

* **Job**
* **Build**
* **Delete**
* **Configure**
* **Etc.**

**Implementation:**

**Approach 1: options available Jenkins**

* **Login to Jenkins dashboard**
* **Go to Manage Jenkins**
* **Under Security 🡪 go to “Configure Global Security”**
* **Under Authentication 🡪 Select “Jenkins own user database”**
* **Go to Authorization section**
* **We will see two options**
  + **Matrix -based security**
  + **Project-based Matrix authorization Strategy**
* **Project-based**
  + **Create a user and give minimum read access**
  + **Go to any Jenkins job**
  + **Click on Configure 🡪 select “Enable project-based security”**
  + **Click Add user and check the permissions**

**Approach 2: Installing Plugin**

* **Go to Manage Plugins**
* **Click on Available**
* **Search for “Role-based Authorization Strategy”**
* **Select and install without restart**
* **Go to “Configure Global Security” under Security section**
* **Under Authorization you will get new option call “Role-Based Strategy”**
* **Select it**
* **Save it**
* **Now under manage Jenkins 🡪 go to “Manage and Assign Roles”**
* **Click on “Manage Roles”**
* **Add a role**
* **Now you can give permissions to the added role**
* **Save it**
* **Go to Assign Roles**
* **Add a user**
* **Select the Role**

**Manage User option**

* **Go to Manage Jenkins**
* **Go to ‘Manage Users”**
* **You can add your users from here**
* **Create user**