**Before Devops :**developer and operational engineer working independently without knowing what each other do .

* Developer will be writing the source code of the product and operational engineer will process the source code and deploy it to test envi. And make sure that the application is running. And later deploy it to the production.
* Since there is no collaboration between developer and operation team , if there are issues in processing the source code , it takes a while to figure out the problem and connect with a developer and it delays the product delivery .
* **Devops** process is introduced to overcome these challenges and fasten the software delivery process.
* Devops Engineer is the combined role of developer and operational engineer
* Devops makes life easier as it supports complete automation solution for application deployments
* by integrating various tools .
* These tools help us to process the source code towards the production deployments.
* We need to understand that Devops is not a tool, Devops is a process /culture that the engineer has to follow to deploy the source code into the production env. And operate it normally .

The source code has to go through different stages to make it production ready .

**WHAT IS DEVOPS?**

Devops is a set of practices that combines software development and IT operations.

**DEVOPS PRINCIPELS:** Devops process primarily works on 3 principals

1. Communication

2. Collaboration

3. Integration

Devops provides communication between developer and operation team and it also provides continous collaboration between teams to fasten the software delivery process & also Devops provides pipeline integration between the tools that are required to process the source code for the production deployments

**Devops Architecture**

**Git-** As local version control system

**Github-** As a Distributed Version Control System .

**Maven –** As a Build Automation tool ,to Build the Java Project.

**Docker-** To create and Manage the containers

**Tomcat –** As an Application server to deploy the java applications

**Kubernetes-** As a container Orchestration Tool to manage the deployment of containers as a pods.

**SONARQUBE:**  for code analysis

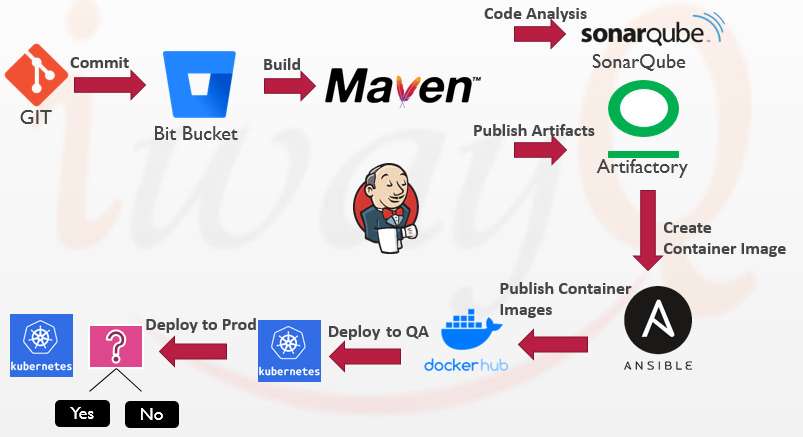
**Jfrog:**  as a artifactory management to store the artefacts that are generated by the maven build process

**ANSIBLE:**  As a configuration management and deployment automation tool

These tools can be run independently to fulfil certain requirements of the application and deployment process .However these tools must be integrated to have a complete automated solution for application deployments. So we use Jenkins as a continuous integration and delivery tool which makes the integration process easier to integrate all these Devops tools

**Jenkins :**As a continuous integration and delivery tool.

We have manual and Automation Deployements



**MANUAL VS AUTOMATION DEPLOYMENTS**

1.Manual deployments

2.Automated Deployments using Jenkins CI/CD Pipeline Code

**Pre-requsites for Devops**

**LINUX BASICS**

**User management-** How to create a user,setup passwords.

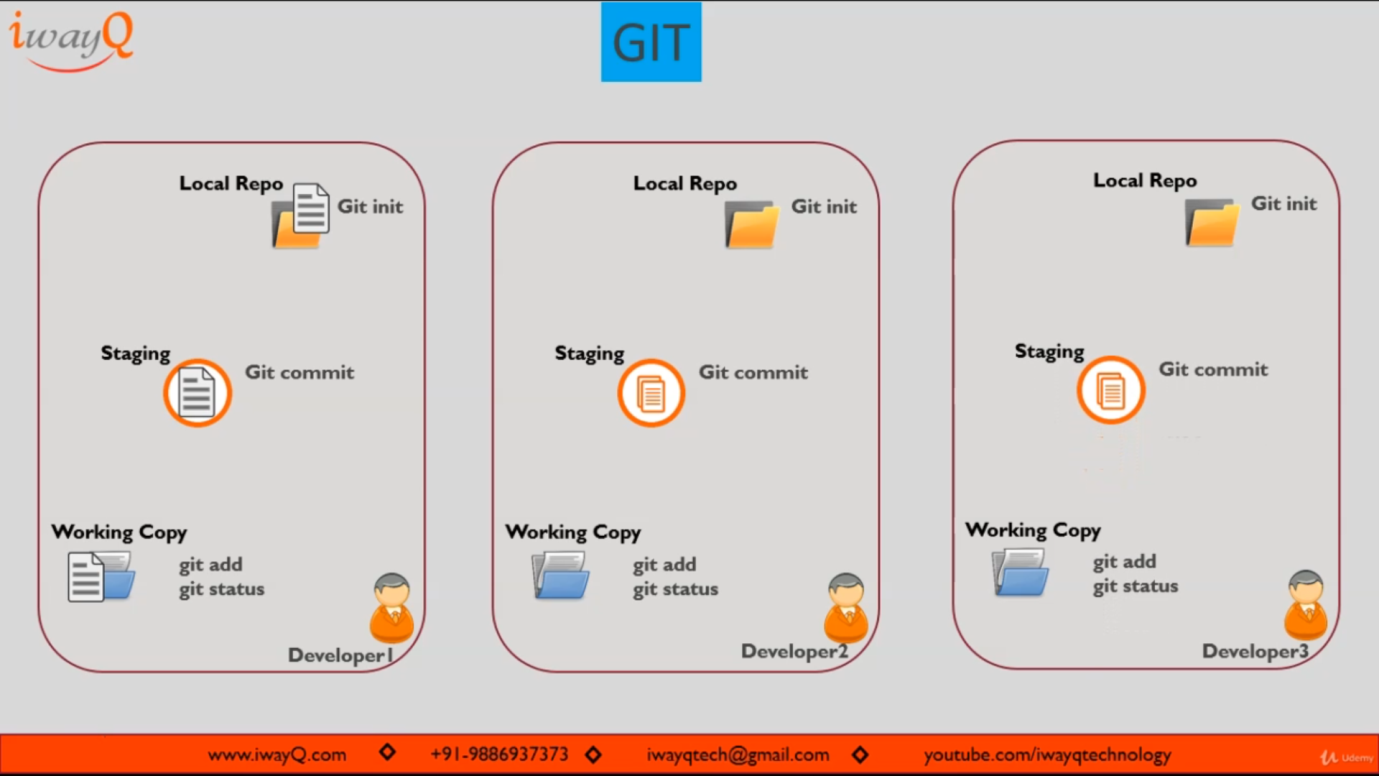
**File Management -**How to create a file,move,copy,Delete files.

**Linux file permissions-** How to set file permissions .

Linux package management – how to install/remove packages

Linux file system- how to create file system, mount, unmount File system.

**GIT**

****

* Git is a Local version control system , Developer has to follow some best practises to ensure the integrity of the source code .
* In certain times the current version of the code is not working , then has to revert back to the previous state .
* Also it is best practise to keep track of changes happening to the project, And How many times the code has modified and when it got modified..
* We are using git as a local version control ro keep track the project changes
* Example:
* Lets say Multiple developers are working on the same product, each developer has their own envi. Where they can develop the project.
* Each developer is working on their own development system
* Developer 1 , will be creating a directory where they can start writing the code that directory is called **WORKING DIRECTORY.**
* And the developer1 is keep on adding multiple core files to the same project dir.
* Same way dev2 and dev3 created working dir and started writing the code .
* To keep track of the changes happening to the project code in the working dir we use local versioning tool **GIT.**
* That means developer has to create the local repo where it can keep track of project changes.
* Whatever the developer making changes to the working copy those changes will be tracked and those changes will be stored into the local repo .
* Since loal repo is holding all the changes with respective to the timestamps ,it is possible to restore to the previous version ,it required.
* To get started creating a local repo using git we need to use **git init** cmd.
* **git init cmd-** which initialize the directory to be used as a local repo.
* Then whatever changes made to the working copy the developer has to run **git add cmd**
* **git add cmd:**  To add the changes to the staging area .
* then git commit cmd to commit those changes to the local repo.
* **What does that mean?**
* Git is a vcs to keep track of the changes that are happened to the working dir,however git not track changes automatically
* Its developer responsibility to ensure that whenever there are some changes happen to the working copy those changes must be reported to the git to store changes to the git local repo by using git add and git commit commands.
* We can revert back the changes to previous state from local repo if required .
* Git is a local vcs, the local repo which is created by the devloper1 will hold only the changes made by the devlpr1 on their working copy .

**Git Installation/lab in Linux ec2 instance**

yum install git -y

git - -version

for the above scenario ..to initialize git , first we need to create working copy using

mkdir devops-project

cd devops-project

now devops-project directory is called as an working copy .

to make this as git local repo we use

git init

ls –a now git created a .git folder (which is the local repo) where it will keep all the project related changes in .git folder.

Cmds to configure local git repo

Git config – is the command is used to configure the developer name and email id to be used for every git commit that is happening to the git local repo.

git config - - globa user.name “murali90”

git config - - global user.email [muralidevops90@gmail.com](mailto:muralidevops90@gmail.com)

Project folder:

git clone https://bitbucket.org/iwayqtech/devops-pipeline-project.git

and edit some files

git status cmd – provides the comparison between the local repo and working copy

git treats the changed files as untracked files (means changes never been updated to the local repo).

Hence we need to commit the changes to the local repo

git add . makes untracked files (shows in red colour) to tracked files(git will show tracked files in green colour)

git status

now the changes are added to staged area by using git add cmd ..and now changes are waiting to be committed to local repo

git commit -m “commit message”

while committing the changes its best practise to create a commit message why we are committing these changes to the repo

git status – now its says that nothing to commit ,working tree is clean(it says that the working copy status and local repo status are in sync).

git log cmd shows the commit id (shows the developer who committed this change with name and at what time the commit is made and the commit message)and how many times changes are committed to this repo.

To revert:

Make some changes to a file

git status

git add .

git status

git commit –m “commit message”

so now I want to revert back to previous state, now take the commit id to which you want to go and use **git reset cmd.**

git reset commit id

git log now it shows that only two commit ids

**GITHUB/BITBUCKET AS DISTRIBUTED VERSION CONTROL SYSTEM**

As above example 2 developers working individually on same project but in individual machines, now changes made by the devloper1 are not visible to the dev2 vice versa.

At certain time they need to collaborate ,to know what each other doing ,without collaboration it may lead to code integrity issues, when we merge the code base of dev1 and dev2 .

To avoid this what dev1 changing must be known by dev2 vice versa, to solve this we can use a Distributed Repository.

**Distributed Repository:** is a repo to keep our code base/source code .

Github and Bit bucket are one of the distributed repo web hosters.

The developers who are on the same project, who are pushing the code to the same repo are now treated as contributors.

Dev2 wants to know what are the changes made by dev1 , dev2 will just pull the changes from the distributed repo by using git pull cmd

He will review the changes made by dev1 and pushes the changes made by him to remote repo by using git push .

Now dev1 can pull the changes made by the dev2 .

It is best practise to pull the code before pushing , in order to avoid the merge conflicts.

git remote -v cmd shows the remote repo urL to pull/fetch source code

**Pull request**

dev2 made some changes to the code and pushed to remote repo, this changes should not cause any integrity issues ,so a prior review should be needed .

so whenever developers push the changes to the remote repo , those changes should not be merged automatically , and that requires some peer review or the approvals.

in order to avoid code integrity ,dev2 creates a new branch and starts working in created new branch

git checkout -b feature

Now performs changes and adds it to stage and later to local repo

git add .

git commit –m “changes made”

git status

Working tree is clean: means working dir and local repo are in sync

git push origin feature -- changes are pushed to feature branch of remote repo.

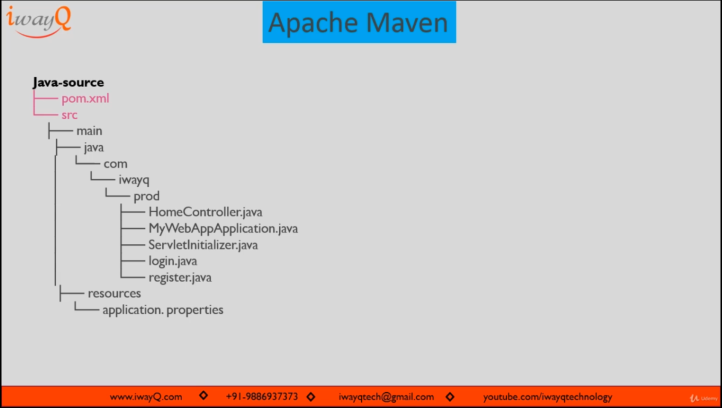
Now to merge the feature branch with master branch, dev2 must raise a pull request for approval and add reviewers in the pull request section

Now reviewer see’s what are all changes made to the code and once he is ok with the changes he will approve the pull request. Once the approvals are granted then we merge feature with master branch .(so now feature branch changes will be merged with master branch ,so now master branch will have latest healthy code written by multiple developers)

**APACHE-MAVEN(Build-Tool)**

Developers Pushed the code to remote repo, now we need to Compile ,Build,Test&Depoly it to test env.

So we are using maven as a build tool for java application, so maven expects to be in a specified folder structure as fallows



So maven know how to build this java source code and resolve the dependencies ,pom.xml is a project definition file that delivers all the required details to the apache maven, how the project source code will be build in terms of artifact names, the version compatibility ,and the dependencies that are required to download from various repository sources.

Now we submit the java code to maven for build process, and will process the code and creates the output as artefacts’ (jar,war,ear).

Those artefacts’ are deployed in further stages.

**APACHE-MAVEN-LAB**

amazon-linux-extras install java-openjdk11 –y

yum install git –y

java –version

download apache maven from <https://maven.apache.org/download.cgi>

Binary tar.gz archive [apache-maven-3.6.3-bin.tar.gz](https://mirrors.estointernet.in/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz)

Download it to /opt directory

wget <https://mirrors.estointernet.in/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz>

tar -xvzf apache-maven-3.6.3-bin.tar.gz

**SONAR-QUBE**

Now code is built, but code may have some repetitive lines or vulnerabilities, code smells, we can avoid duplicates,

So we use sonarqube as static code analysis reporter

Once the build is successful , maven will push the code to the sonar scanner for static code analysis .

Now sonarqube scans the code and the analysis reports will be stored in a database.

Sonarqube will integrated with postgres sql/oracle db for backend database to store this analysis reports

**Sonarqube-lab**

Install and configure postgres sql

Install and configure sonarqube scanner

Sonarqube expects 2 core processer and 4gb ram, so choose t2.medium

In security groups select postgres sql –port no5432

Custoum tcp ,sonarqube default port no 9000

http port no 80 and anywhere for all .

sonarqube installation prerequisites—java-openjdk 11

amazon-linux-extras list

amazon-linux-extras install postressql11 –y

now install postresql11 server package

yum install postgresl-server postgresl-devel postfressql -y

now initialize the postgresql db by this cmd

/usr/bin/postgresql-setup --initdb

Now start postgresql

systemctl start postgresql

postgres automatically creates a user in db called postgres, we can check it in tail -1 /etc/passwd

how to connect to postgresql ?

su – postgres

and now run psql ,now we are successfully connected to db

now create a user and granting the privilege for access the db, sonarqube will connect to the db and publish the analysis reports .

CREATE USER sonar WITH PASSWORD ‘sonar’ ;

ALTER USER sonar WITH SUPERUSER –cmd to to give superuser privilege

To check weather the user is created is not use cmd \du

\? For help

\q to quit

exit

clear

vi /var/lib/pgsql/data/pg\_hba.conf –edit this file to provide access for posting the reports

inside this file go to METHOD for first peer and ident and change it to trust

:wq!

Now create a db to store the anlaysis reports for sonarqube scanner

su postgres

psql

CREATE DATABASE sonar;

TO CHECK \l

Now grant the access for sonar user ON THIS SOANR DB

GRANT ALL PREVILAGES ON DATABASE sonar TO sonar;

\l to list databases

Now download sonarqube community edition

cd /opt

wget <https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.6.1.40680.zip>

tar –xvzf sonaqube

after done with all configuration login to sonarqube

un:admin

password:admin

create a new project 🡪project name-iway🡪Generate a Token🡪copy the token (this token is required to integrate sonarqube with maven)🡪 specify the programming language which your project is using to do the code analysis 🡪choose buid technology(maven)🡪 sonarqube auto generated the command which we need to run at maven level to publish the source code to sonarqube for code analysis‑‑

1 . Installing Jenkins in amazon linux in ec2

2. converting pem to ppk by load in putty gen and then save privatekey , open putty and add public ip of ec2 instance ..to get the public ip go to actions and sleect connect

3. after logging in create user Jenkins and give sudo permissions

4. sudo yum install docker

Docker service start

systemctl enable docker.service

or sudo chkconfig docker on

sudo yum install -y git

sudo reboot

11/12/2020

**Step 1 – Install Java on Amazon Linux**

The OpenJDK 8 is available under default yum repositories and OpenJDK 11 is available under Amazon Linux 2 extras repositories. You can simply install Java 11 or Java 8 on the Amazon Linux system using the following commands.

* Run below commands to **install Java 11 on Amazon Linux**:
* sudo amazon-linux-extras install java-openjdk11
* Run below commands to **install Java 8 on Amazon Linux**:
* sudo yum install java-1.8.0-openjdk

**Step 2 – Check Active Java Version**

After successfully installing Java on Amazon Linux using the above steps, Let’s verify the installed version using the following command.

**java -version**

openjdk version "1.8.0\_222"

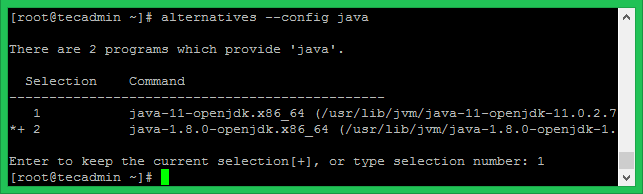
OpenJDK Runtime Environment (build 1.8.0\_222-8u222-b10-1ubuntu1~18.04.1-b10)

OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)

**Step 3 – Switch Java Version**

Use alternatives command-line utility to switch active Java version on your Amazon Linux system. Run below command from the command line and select the appropriate Java version to make it default.

alternatives --config java

[](https://tecadmin.net/wp-content/uploads/2019/10/java-amazon-linux.png)

After switching let’s check again active Java version:

**java -version**

openjdk version "11.0.7" 2020-04-14 LTS

OpenJDK Runtime Environment 18.9 (build 11.0.7+10-LTS)

OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10-LTS, mixed mode, sharing)

[**java8-centos-amazon-linux.md**](https://gist.github.com/diegopacheco/6d69e0cfaf13d4351cfa700bb4af8172#file-java8-centos-amazon-linux-md)

# Remove java 7

sudo yum remove -y java

# Install basic packages

sudo yum install -y git

# Download and install java 8

wget --no-cookies --no-check-certificate --header "Cookie: gpw\_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.tar.gz"

tar -xzvf jdk-8u131-linux-x64.tar.gz

rm -rf jdk-8u131-linux-x64.tar.gz

# Configure JAVA\_HOME

sudo vim ~/.bashrc

alias cls='clear'

export JAVA\_HOME=~/jdk1.8.0\_131

export JRE\_HOME=~/jdk1.8.0\_131/jre

export PATH=$PATH:~/jdk1.8.0\_131/bin:/~/jdk1.8.0\_131/jre/bin

source ~/.bashrc

java -version

**installing java and setting env variables**

**worked for centos vm oct5 2020-10-04**

How to Install Java 8 in CentOS / Amazon Linux?

[Raw](https://gist.github.com/diegopacheco/6d69e0cfaf13d4351cfa700bb4af8172/raw/c7c06cdb57b52a7b45c8a97d984bc9f3f83a1c8c/java8-centos-amazon-linux.md)

# Set JAVA\_HOME on CentOS / RHEL / Fedora

I’ve seen many questions on how to set JAVA\_HOME on CentOS / Fedora / RHEL Linux distributions. JAVA\_HOME is used to set the path of Java installation on a Linux or Windows system. JAVA\_HOME is just a convention and it is usually used by Java EE and Tomcat servers and build tools such as Gradle, Ant and Maven to find where Java is installed.

In this guide I’ll show you an easy and recommended way of setting JAVA\_HOME on CentOS / Fedora / RHEL Linux system. We assume you already have Java installed before you can set JAVA\_HOME.

Install Java on [CentOS 7](https://computingforgeeks.com/how-to-install-java-11-on-centos-7-fedora-29-fedora-28/" \t "_blank), [Fedora](https://computingforgeeks.com/how-to-install-java-11-on-centos-7-fedora-29-fedora-28/), [RHEL/ CentOS 8](https://computingforgeeks.com/how-to-install-java-11-openjdk-11-on-rhel-8/).

## Set JAVA\_HOME on CentOS / Fedora / RHEL

If you have more than one version of Java installed, you may want to set default version before you configure JAVA\_HOME on CentOS / Fedora / RHEL system. For this, use the command below.

sudo alternatives --config java

This will give you a prompt to confirm the default Java version you want to set.

There are 2 programs which provide 'java'.

Selection Command

-----------------------------------------------

\* 1 java-1.8.0-openjdk.x86\_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-2.el7\_6.x86\_64/jre/bin/java)

+ 2 java-1.7.0-openjdk.x86\_64 (/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.211-2.6.17.1.el7\_6.x86\_64/jre/bin/java)

Enter to keep the current selection[+], or type selection number: 1

You can set JAVA\_HOME in .bash\_profile, .bashrc file or for all Global users in /etc/profile or as bash function inside /etc/profile.d/ directory.

Add below line to any of bash dotfiles mentioned above.

export JAVA\_HOME=$(dirname $(dirname $(readlink $(readlink $(which javac)))))

Then source the file. Suppose you added this to ~/.bashrc, you’ll run:

source ~/.bashrc

Confirm Environment variable value.

$ **echo $JAVA\_HOME**  
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-2.el7\_6.x86\_64

You also need to add Java /bin directory to your PATH

export PATH=$PATH:$JAVA\_HOME/bin

Java CLASSPATH can be set using:

export CLASSPATH=.:$JAVA\_HOME/jre/lib:$JAVA\_HOME/lib:$JAVA\_HOME/lib/tools.jar

So your complete setting will have the lines:

export JAVA\_HOME=$(dirname $(dirname $(readlink $(readlink $(which javac)))))  
export PATH=$PATH:$JAVA\_HOME/bin  
export CLASSPATH=.:$JAVA\_HOME/jre/lib:$JAVA\_HOME/lib:$JAVA\_HOME/lib/tools.jar

Here is my screenshot.

Don’t forget to source the file or logout and back in.

**# Examples**$ source ~/.bashrc  
$ source ~/.bash\_profile  
$ source /etc/profile  
$ source /etc/profile.d/java.sh

Then confirm:

$ echo $JAVA\_HOME  
$ echo $PATH  
$ echo $CLASSPATH

And that’s all. You application should locate the Java installation directory.

**method1**

**SIMPLE STEPS.**

**First, check the version of JDK**

**Step 1**:**Ensure you have right version of JAVA installed or upgrade if necessary** (Optional)

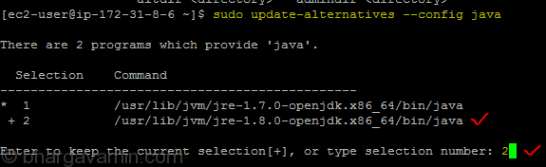
The latest version of JAVA is 1.8, by default Amazon Linux has JAVA version 1.7 so you would directly want to upgrade it use below command :

sudo yum install java-1.8.0-openjdk.x86\_64

If you already have JAVA installed you can change/check the JAVA version using below command.

$ sudo update-alternatives --config java

Select an option as shown in the image below:



Note: It is advisable to remove the previous version so that it doesn’t switch back.

Command to check package is installed or not

$ rpm -qa | grep nano nanao is the package name

Also, you can use Yum command like below.

$ yum list installed|grep 'nano'

**Step 2:** **Find out where JAVA is!**

For Linux systems, you can recursively run the commandfile followed by whichcommand to find the JAVA installation location as shown in the image below.

$ file $(which java)

/usr/bin/java: symbolic link to `/etc/alternatives/java'

The above output shows that java is pointing to a /etc/alternatives/java file but that not the actual location of JAVA hence you will need to dig in more to fetch its actual path.

**Step 3** : **Follow the lead!**

In the previous step, we located /etc/alternatives/java file this file will get us to the actual location where JAVA config files are.

Run the file command on that location /etc/alternatives/java.

$ file /etc/alternatives/java

/etc/alternatives/java: symbolic link to `/usr/lib/jvm/java-8-openjdk.x86\_64/bin/java'

There you go… You’ve now located JAVA config file location which we will use in below steps to set JAVA environment variable

You can re-affirm the location running  file command on the symbolic path:

$ file /usr/lib/jvm/java-8-openjdk.x86\_64/bin/java

/usr/lib/jvm/java-8-openjdk.x86\_64/bin/java: ELF 64-bit LSB executable...

This means that the JAVA is installed perfectly, Now go ahead and copy the path of above output

/usr/lib/jvm/jdk-1.8.0-openjdk.x86\_64/bin

**Step 4:  Set JAVA environment variable**

To set the JAVA\_HOME environment variables on Linux/Unix go to .baschrc file.

**Note:** .bashrc file is different for each user in Linux, hence you will need to update the same file for every user you want to set environment variable for.

Copy paste below two lines in the .bashrc file found in home the directory of ec2-user and root user:

 export JAVA\_HOME="/usr/lib/jvm/jdk-1.8.0-openjdk.x86\_64"

 PATH=$JAVA\_HOME/bin:$PATH

Save the file.

**Method2**

**OpenJDK Installation**

sudo yum install wget

yum install java-1.8.0-openjdk

**Procedure**

1. Find the JRE installation home directory.

Example: /usr/lib/jvm/jre1.8.0\_65

1. Export it in the JAVA\_HOME environment variable.

Example:

export JAVA\_HOME=/usr/lib/jvm/jre1.8.0\_65

export PATH=$JAVA\_HOME/bin:$PATH

1. Add these lines at the end of the user profiles in the ~/.bash\_profile file or, as a superuser, at the end of the global profiles in the /etc/profile file.
2. Restart system and login Log on again.

**MAVNE INSTALLATION WORKED**

# How to Install Apache Maven on CentOS 7

[**Ravi Saive**](https://www.tecmint.com/author/admin/)**August 10, 2018 Categories**[**CentOS**](https://www.tecmint.com/category/linux-distros/centos/)[**6 Comments**](https://www.tecmint.com/install-apache-maven-on-centos-7/#comments)

**Apache Maven** is a open source software project management and build automation tool, that is based on the conception of a project object model (**POM**), which is primarily used for deploying Java-based applications, but can also be used on projects written in **C#**, **Ruby** and other programming languages.

In this article, I will explain how to install and configure latest version of **Apache Maven** on a **CentOS 7** system (the given instructions also works on **RHEL** and **Fedora** distribution).

#### Prerequisites

* A newly deployed or existing CentOS 7 server instance.
* **Java Development Kit (JDK)** – Maven 3.3+ require JDK 1.7 or above to execute.

### Install OpenJDK 8 in CentOS 7

**Java Development Kit (JDK)** is a primary requirement to install **Apache Maven**, so first install Java on CentOS 7 system from the default repository and verify the version using following commands.

# yum install -y java-1.8.0-openjdk-devel

# java -version

If installation went well, you see the following output.

openjdk version "1.8.0\_141"

OpenJDK Runtime Environment (build 1.8.0\_141-b16)

OpenJDK 64-Bit Server VM (build 25.141-b16, mixed mode)

### Install Apache Maven in CentOS 7

Next, go to the [official Apache Maven download](https://maven.apache.org/download.cgi) page and grab the latest version or use the following wget command to download it under the maven home directory **‘/usr/local/src’**.

# cd /usr/local/src

# wget http://www-us.apache.org/dist/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz

Extract the downloaded archive file, and rename it using following commands.

# tar -xf apache-maven-3.5.4-bin.tar.gz

# mv apache-maven-3.5.4/ apache-maven/

### Configure Apache Maven Environment

Now we need to configure the environments variables to pre-compiled Apache Maven files on our system by creating a configuration file **‘maven.sh’** in the **‘/etc/profile.d’** directory.

# cd /etc/profile.d/

# vim maven.sh

Add the following configuration in **‘maven.sh’** configuration file.

# Apache Maven Environment Variables

# MAVEN\_HOME for Maven 1 - M2\_HOME for Maven 2

export M2\_HOME=/usr/local/src/apache-maven

export PATH=${M2\_HOME}/bin:${PATH}

Now make the **‘maven.sh’** configuration file executable and then load the configuration by running the **‘source’** command.

# chmod +x maven.sh

# source /etc/profile.d/maven.sh

### Check Apache Maven Version

To verify Apache Maven installation, run the following **maven** command.

# mvn --version

And you should get a output similar to the following:

Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T19:33:14+01:00)

Maven home: /usr/local/src/apache-maven

Java version: 9.0.4, vendor: Oracle Corporation, runtime: /opt/java/jdk-9.0.4

Default locale: en\_US, platform encoding: UTF-8

OS name: "linux", version: "4.17.6-1.el7.elrepo.x86\_64", arch: "amd64", family: "unix"

That’s It! You have successfully installed **Apache Maven 3.5.4** on your **CentOS 7** system. If you have any problems related to installation, do share with us in the comment section.

## ****Installing Maven on Linux****

To install Maven on the Linux operating system, download the latest version from the [Apache Maven site](https://maven.apache.org/), select the Maven binary tar.gz file, for example: apache-maven-3.3.9-bin.tar.gz.

Extract the archive to your desired location.

### ****3.1. Adding Maven to the Environment Path****

Open the command terminal and run the following commands to set the environment variables:

|  |  |
| --- | --- |
|  | $ export M2\_HOME=/usr/local/apache-maven/apache-maven-3.3.9 |
|  |  |
|  | $ export M2=**$M2\_HOME**/bin |
|  |  |
|  | $ export MAVEN\_OPTS=-Xms256m -Xmx512m |

with M2\_Home path corresponding with the location of your extracted Maven files.

Now append the M2 variable to the system path:

|  |  |
| --- | --- |
|  | $ export PATH=**$M2**:**$PATH** |

Finally, verify if Maven has been added by running:

|  |  |
| --- | --- |
|  | $ mvn -version |

The output should be as follows:

|  |  |
| --- | --- |
|  | Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2016-12-03T17:27:37+05:30) |
|  | Maven home: /usr/local/apache-maven/apache-maven-3.3.9 |
|  |  |
|  | Java version: 1.8.0\_75, vendor: Oracle Corporation |
|  |  |
|  | Java home: /usr/local/java-current/jdk1.8.0\_75/jre |

You have successfully installed Maven on your Linux system.

**OFFICIAL WEBSITE FOR MAVEN**

<https://maven.apache.org/download.cgi>

Create directory in /opt a s maven (u can create direcrory anywhere)

Download the tar for linux by using

wget https://www-us.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz

Extract the gzip file

Tar –zxvf

And go to /opt/maven/apachemaven/

Now set the maven path

export M2\_HOME=/opt/maven/

export PATH=/opt/maven/bin:$PATH

**TO MAKE PATH PERMANENT GO TO USER HOME DIRECTORY AND EDIT ls –a**

**.BASH\_PROFILE AND PALCE THIS STUFF**

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-1.el7\_6.x86\_64/

export M2\_HOME=/opt/maven/apache-maven-3.6.0/

export PATH=$PATH:$JAVA\_HOME/bin:$M2\_HOME/bin:$HOME/bin

**INATALLING JENKINS USING Repos in linux or centos**

Previously during software development, developers would submit their code to a code repository like [GitHub or Git Lab](https://www.tecmint.com/github-alternatives-to-host-open-source-projects/) usually, the source code would be fraught with bugs and errors. To make it even worse, developers would have to wait until the entire source code was built & tested to check for errors. This was tedious, time-consuming and frustrating. There was no iterative improvement of code, and overall, the software delivery process was slow. Then came **Jenkins**.

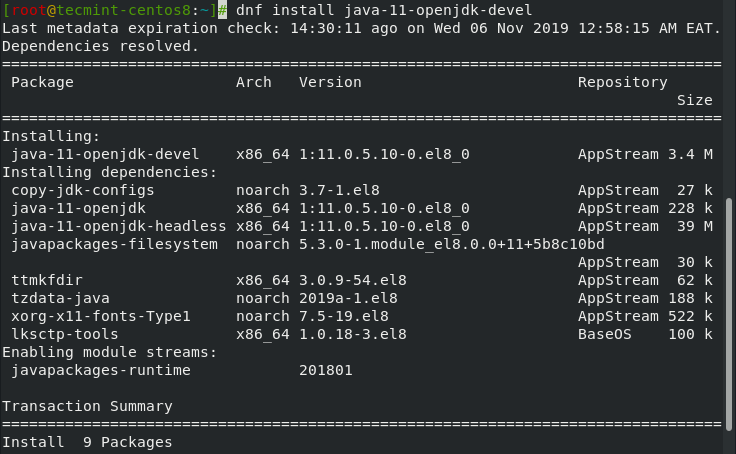
**Jenkins** is a free and opensource continuous integration tool written in **Java** that allows developers to continuously develop, test and deploy code in a simple and effective way. It automates tasks thereby saving time and takes away the stressful part of the software development process.

In this article, we demonstrate how you can install **Jenkins** on **CentOS 8** Linux.

### Step 1: Install Java on CentOS 8

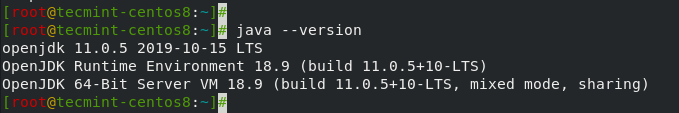
For **Jenkins** to function, you need to install either **Java JRE 8** or **Java 11**. In the example below, we decided to go with the installation of **Java 11**. Therefore, to install **Java 11**, run the command.

# dnf install java-11-openjdk-devel

[](https://www.tecmint.com/wp-content/uploads/2019/11/Install-Java-on-CentOS-8.png)Install Java on CentOS 8

To verify the installation of **Java 11**, run the command.

# java --version

[](https://www.tecmint.com/wp-content/uploads/2019/11/Check-Java-Version.png)Check Java Version

The output confirms that **Java 11** has been successfully installed.

### Step 2: Add Jenkins Repository on CentOS 8

Since **Jenkins** is not available in **CentOS 8** repositories, therefore we are going to add **Jenkins Repository** manually to the system.

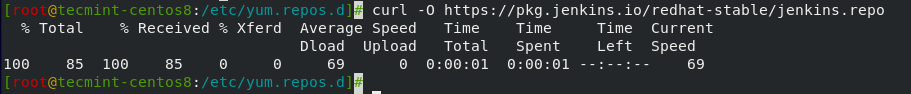
Begin by adding **Jenkins Key** as shown.

# rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

Now append Jenkin’s repository to **CentOS 8**.

# cd /etc/yum/repos.d/

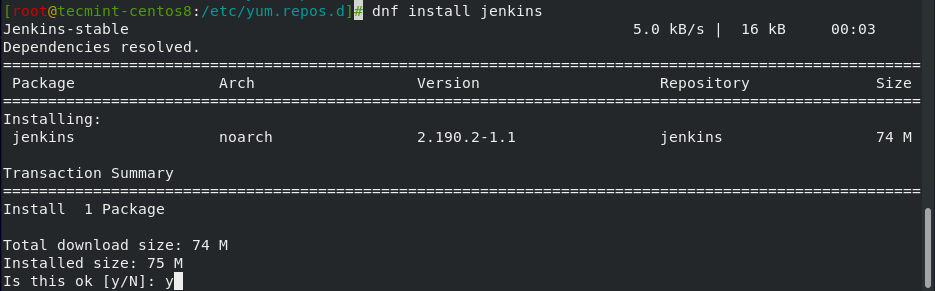
# curl -O https://pkg.jenkins.io/redhat-stable/jenkins.repo

[](https://www.tecmint.com/wp-content/uploads/2019/11/Add-Jenkins-Repository-on-CentOS-8.png)Add Jenkins Repository on CentOS 8

### Step 3: Install Jenkins on CentOS 8

Having successfully added **Jenkins** repository, you can proceed to install **Jenkins** by running.

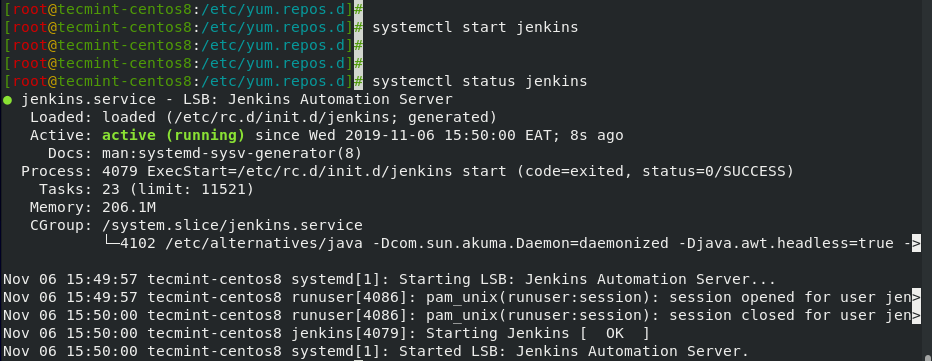
# yum install jenkins

[](https://www.tecmint.com/wp-content/uploads/2019/11/Install-Jenkins-on-CentOS-8.png)Install Jenkins on CentOS 8

Once installed, start and verify the status of **Jenkins** by executing the commands.

# systemctl start jenkins

# systemctl status jenkins

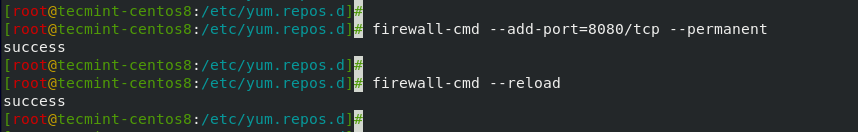
[](https://www.tecmint.com/wp-content/uploads/2019/11/Start-and-Verify-Jenkins-Status.png)Start and Verify Jenkins Status

The output above shows that Jenkins is up and running.

Next, you need to configure the firewall to allow access to port **8080** which is used by **Jenkins**. To open the port on the firewall, run the commands.

# firewall-cmd --add-port=8080/tcp --permanent

# firewall-cmd --reload

[](https://www.tecmint.com/wp-content/uploads/2019/11/Open-Jenkins-Port-on-Firewall.png)Open Jenkins Port on Firewall

### Step 4: Setting up Jenkins on CentOS 8

With the initial configurations done, the only remaining part is setting up **Jenkins** on a web browser. To achieve this, browse your server’s IP address as shown:

http://server-IP:8080

The first section requires you to unlock **Jenkins** using a password. This password is placed in the file **/var/lib/Jenkins/secrets/initialAdminPassword** file.

To read the password, simply use the [cat command](https://www.tecmint.com/13-basic-cat-command-examples-in-linux/) as shown.

# cat /var/lib/Jenkins/secrets/initialAdminPassword

[](https://www.tecmint.com/wp-content/uploads/2019/11/View-Jenkins-Admin-Password.png)View Jenkins Admin Password

Copy & paste the password in the Administrator password text field & click ‘**Continue**‘.

Unlock Jenkins

In the second stage, you will be presented with 2 options: ‘**Install using suggested plugins**‘ or ‘**Select plugins to install**‘.

For now, click on ‘**Install using suggested plugins**‘ to install essential plugins for our setup.

Install Suggested Plugins

Shortly, the installation of the plugins will get underway.

Jenkins Plugin Installation

In the next section, fill out the fields in order to create the **First Admin** user. After you are done, click on ‘**Save and continue**‘.

Create Jenkins Admin User

The ‘**Instance Configuration**‘ section will provide you with the default Jenkins URL. For simplicity, it’s recommended to leave it as it is and click ‘**Save and Finish**‘.

Jenkins Instance Configuration

At this point, **Jenkins** setup is now complete. To access the Jenkins dashboard, simply click on ‘**Start using Jenkins**‘.

Jenkins Installation Complete

Jenkins’s dashboard is displayed below.

Jenkins Dashboard

Next time you log into **Jenkins**, simply provide the **Admin** username and the password you specified when creating the Admin user.

Jenkins Admin Login

##### Conclusion

That was a step-by-step procedure of how to install **Jenkins Continuous Integration** tool on **CentOS 8**. To learn more about **Jenkins**. Read [Jenkins Documentation](https://jenkins.io/doc/book/getting-started/). Your feedback on this guide is most welcome.

Tags

**INATALLING JENKINS USING DOCKER**

docker pull jenkins:2.60.3

docker images

**to run a container of Jenkins**

**installing sonar-qube**

[From a Docker image](https://docs.sonarqube.org/latest/setup/get-started-2-minutes/)

Find the Community Edition Docker image on [Docker Hub](https://hub.docker.com/_/sonarqube/).

1. Start the server by running:

$ docker run -d --name sonarqube -p 9000:9000 <image\_name>

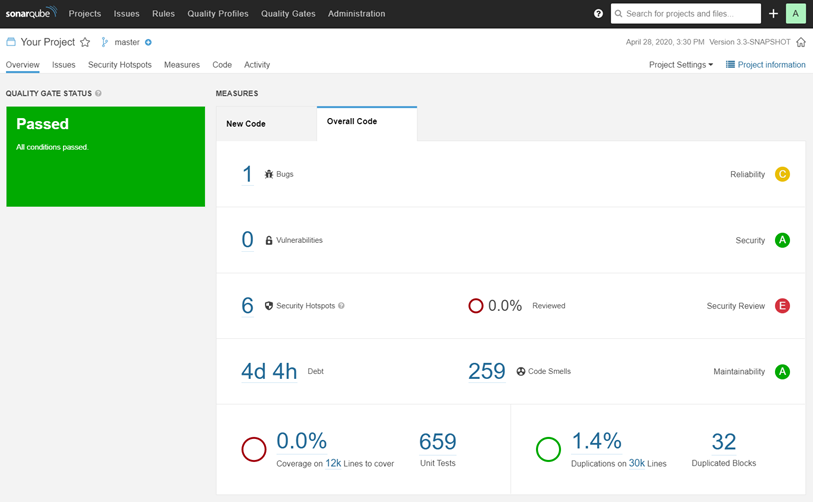
1. Log in to [http://localhost:9000](http://localhost:9000/) with System Administrator credentials (login=admin, password=admin).

## Analyzing a Project

Now that you're logged in to your local SonarQube instance, let's analyze a project:

1. Click the **Create new project** button.
2. When asked **How do you want to create your project**, select **Manually**.
3. Give your project a **Project key** and a **Display name** and click the **Set Up** button.
4. Under **Provide a token**, select **Generate a token**. Give your token a name, click the **Generate** button, and click **Continue**.
5. Select your project's main language under **Run analysis on your project**, and follow the instructions to analyze your project. Here you'll download and execute a Scanner on your code (if you're using Maven or Gradle, the Scanner is automatically downloaded).

After successfully analyzing your code, you'll see your first analysis on SonarQube:



**Installing sonatype nexus repo**

**docker pull sonatype/nexus3**

### Persistent Data

There are two general approaches to handling persistent storage requirements with Docker. See [Managing Data in Containers](https://docs.docker.com/engine/tutorials/dockervolumes/) for additional information.

1. Use a docker volume. Since docker volumes are persistent, a volume can be created specifically for this purpose. This is the recommended approach.
2. $ docker volume create --name nexus-data

$ docker run -d -p 8081:8081 --name nexus -v nexus-data:/nexus-data sonatype/nexus3

1. Mount a host directory as the volume. This is not portable, as it relies on the directory existing with correct permissions on the host. However it can be useful in certain situations where this volume needs to be assigned to certain specific underlying storage.
2. $ mkdir /some/dir/nexus-data && chown -R 200 /some/dir/nexus-data

$ docker run -d -p 8081:8081 --name nexus -v /some/dir/nexus-data:/nexus-data sonatype/nexus3

* Default user is admin and the uniquely generated password can be found in the admin.password file inside the volume. See [Persistent Data](https://hub.docker.com/r/sonatype/nexus3#user-content-persistent-data) for information about the volume.
* It can take some time (2-3 minutes) for the service to launch in a new container. You can tail the log to determine once Nexus is ready:

$ docker logs -f nexus

Important note :When stopping, be sure to allow sufficient time for the databases to fully shut down.

docker stop --time=120 <CONTAINER\_NAME>

**For sonarqube anlaysis**

* Genereate a token in myaccount
* And in git repo or branch open pom.xml
* under sonar section give sonarqube server url and under sonar token section give sonar token which u have generated

**Pushing the artifacts to nexus repository u can do it in two ways**

**Method 1**

In the same pom.xml under nexus section paste the nexus server url and give credentials of nexus server

**Giving the credentials of nexus server in maven settings.xml**

1. in maven installation path vi /opt/maven/conf/settings.xml this is path for maven installed path in linux server
2. if you have installed maven automatically in manage Jenkins 🡪gobal tool configurations ,you can find the maven path under Jenkins home path /var/lib/Jenkins/tools/Hudson.taks.maven/maven-3.5./conf/settings.xml
3. In settings.xml under server section copy the server section and paste and edit the credentials nexus server and save the file

Eg: <server>

<id>deploymentRepo</id>

<username>repouser</username>

<password>repopwd</password>

</server>

-->

<!-- Another sample, using keys to authenticate.

<server>

<id>siteServer</id>

<privateKey>/path/to/private/key</privateKey>

<passphrase>optional; leave empty if not used.</passphrase>

</server>

-->

<server>

<id>nexus</id>

<username>admin</username>

<password>admin\_123</password>

</server>

</servers>

# Hosting Maven Repositories

A hosted Maven repository can be used to deploy your own as well as third-party components. A default installation of Nexus Repository Manager includes a two hosted Maven repositories. The maven-releases repository uses a release version policy and the maven-snapshots repository uses a snapshot version policy.

**now create a repository for maven build maven repository**

1. in nexus under settings icon repository –create repo for maven project take m2 group and create
2. now copy the create repo url of releases and paste it in the same pom.xml under distribution management section
3. and copy the created snapshot repo url of snapshots and paste it in the same pom.xml under snapshot repository section
4. and commit
5. and run the job in Jenkins

Method 2

install the nexus artefact plugin and configure it in Jenkins

in manage Jenkins section –configuration 🡪 you will get a nexus link and now

and now create a nexus user in Jenkins and give the credentials in th nexus section

and now copy the url of nexus server and paste here and save

now go to Jenkins job under bulid section select nexus artifacts and configure the details and now run the Jenkins job

Flow-1 Deploying the artifact to tomcat

You need to download apache tomcat 8 and unzip by using tar –xvzf

To start tomcat go to bin dir inside tomcat and ./startup.sh

To check tomcat is up or ps –ef | grep tomcat and check port number 8080

By default tomcat uses 8080 but Jenkins uses 8080 so u nedd to change the port number for tomcat ,u can change it by editing the settings.xml in connector section (line number 69 in file)and change the port number and stop tomcat and start the tomcat

To shutdown ./shutdown.sh

Now give permissions for startup.sh so that any user start up the tomcat ..dont give full permisiions in organization

chmod +x startup.sh

chmod +x shutdown.sh

Creating soft link for startup .sh and shutdown.sh

ln -s /opt/tomcat/bin/startup.sh /usr/local/bin/tomcatup ------for starting tomcat

ln -s /opt/tomcat-8.5.49/bin/shutdown.sh /usr/local/bin/tomcatdown ---for stopping tomcat

for tomcat logs /opt/tomcat/logs/catalina.out

tomcatup but page not loading for my centos 7

iptables -I OUTPUT -o -enp0s3 -d 0.0.0.0/0 -j ACCEPT

iptables -I INPUT -i -enp0s3 -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -I INPUT -j ACCEPT

and try to login now 403 acces denied because we didn’t give permmison for tomcat users to login so give permission by doing following things

**cmd :find / -name context.xml**

output :/opt/tomcat-8.5.49/conf/context.xml

/opt/tomcat-8.5.49/webapps/host-manager/META-INF/context.xml

/opt/tomcat-8.5.49/webapps/manager/META-INF/context.xml

vi /opt/tomcat-8.5.49/webapps/host-manager/META-INF/context.xml

comment the line number 19,20 as fallows

19 <!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"

20 allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

vi /opt/tomcat-8.5.49/webapps/manager/META-INF/context.xml

19 <!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"

20 allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

Now to add user got to /opt/tomcat/conf

And edit tomcat-users.xml

vi tomcat-users.xml

and paste following under second NOTE or from line 36

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<role rolename="admin-gui"/>

<role rolename="admin-script"/>

<user username="admin" password="admin" roles="admin-gui,manager-gui,manager-script,manager-jmx,manager-status"/>

<user username="deployer" password="deployer" roles="manager-script"/>

<user username="tomcat" password="tomcat" roles="manager-gui"/>

Now tomcat installation …configuration are all done

Now got Jenkins and manage plugins and in available section search Deplot to conatainer plugin and install it without restart

After restart go to your job configuration and in **post build Actions 🡪select🡪Deploy waror ear to container**

|  |  |  |
| --- | --- | --- |
| WAR/EAR files |  |  |
|  | |  |  |
|  |  | |  |
|  | Context path |  |  |
|  | |  |  |
|  |  | |  |
|  | Containers | Add Container  Add container tomcat 8.x Remote  Credentials: add tomcatcatuser in Jenkins database ..this user should be in the tomcat user.xml |  |
|  | |  |  |
|  | Deploy on failure |  |  |

Save the config of the job and build now the job

And check the war is deployed into tomcat or not by going into manager app

With one click jenkins will get the code form github

Build the pacakage by using maven

Source code analaysis using sonarqube

Copy the .war and snapshot to the nexus repository manager

And deploy the artefact to the dev or stage env into tomcat

And to upadate ur job u need to configure and now add to timestamp to the console output go to build env section and select **add timestamp to the console output**

In build env section select **delete workspace before build starts**

U need to maintain only last 5 build to do that got job config in general section **Discard old builds**

**Days to keep builds**

**Max# no.of builds give 5** here it will store only build information not the packages

**And to store the packages along job infm u need go to**

**advanced section**

**Max# no.of builds to keep with artifacts give 5**

The config done to this job ..sticks only to this job

**Upto now we are doing manual job triggering**

To automate Build trigger we have 3 options

**1.poll scm---- this is crontab format**

**Min Hours Dayofmonth Month Day of week**

**\***

**How Jenkins will know that difference bw previous build and current build**

**Based on the commit ids (if both commitids are same means we don’t have changes in the code )**

**Both Poll SCM and the build takes cron expression as the input. So, How do they actually differ?**

"Poll SCM" polls the SCM periodically for checking if any changes/ new commits were made and shall build the project if any new commits were pushed since the last build, whereas the "build"  shall build the project periodically irrespective to whether or not any changes were made.

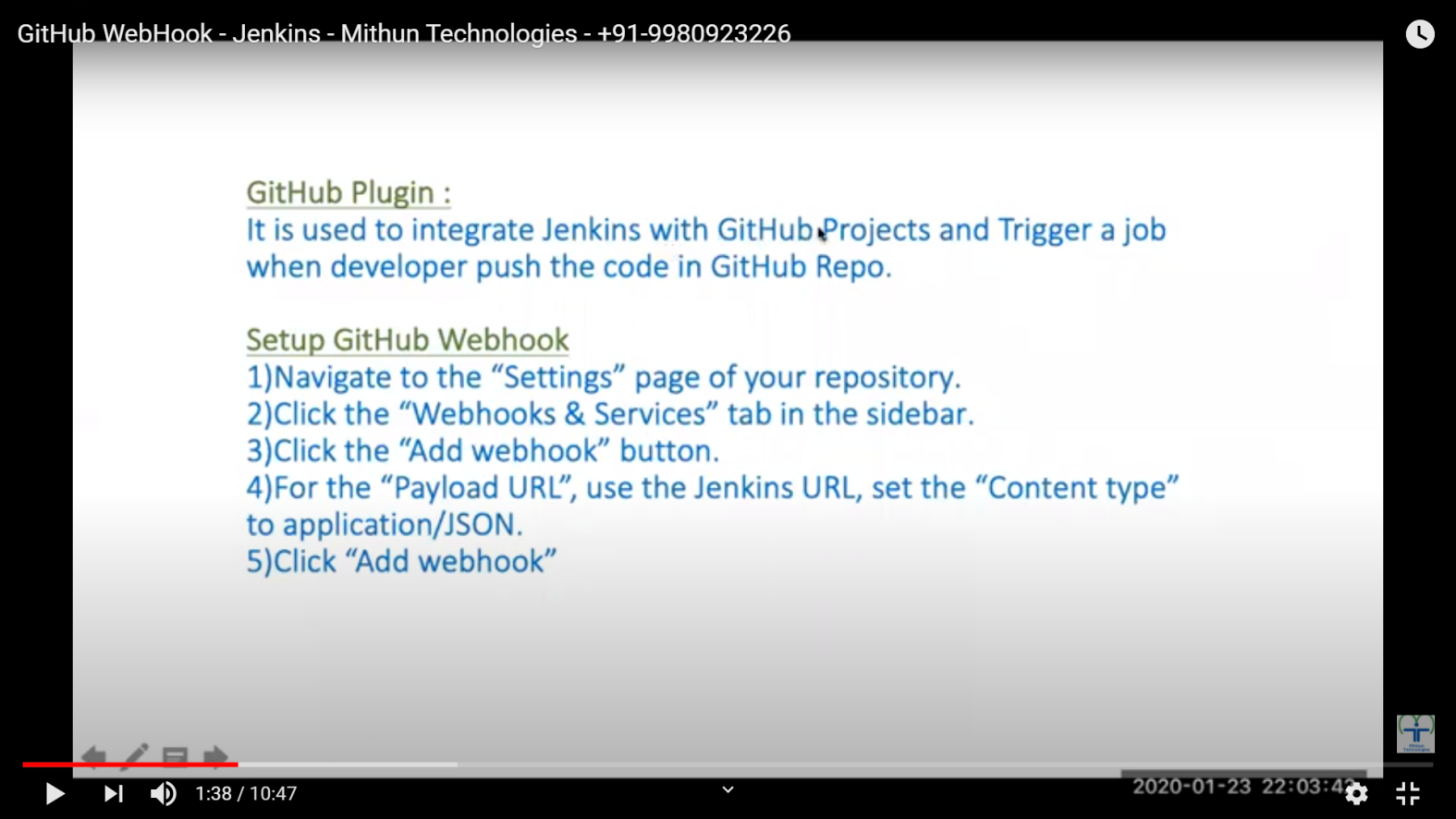
**2.Build periodically**

**this is also crontab format**

**Min Hours Dayofmonth Month Day of week**

**\***

**3.Github webhook**

****

**This is freestyle job …this type of job we are not using much in companies now**

Payload url ----- <http://192.168.43.36:8080/github-webhook>

**Payload delivery unsuccessful**

* 192.168.0.32 is a private IP on your own network. GitHub needs a public IP to hit, so you'll need to actually enter your public IP there. Depending on how your network is configured, you'll need to configure port forwarding on your router (if it's a home internet connection). You can use something like ngrok for temporary testing.

**Jenkins job configuration history plugin**

This plugin saves **a copy of the configuration file** of jobs and agents (config.xml) for every change made and of the system configuration (<config-name>.xml). You can also see what changes have been made by which user if you configured a security policy.

# Job Import Plugin

Import jobs from one Jenkins server to another Jenkins instance/server.

Useccase is migrating jobs og Jenkins sever1 to Jenkins server 2

Install the plugin in Jenkins sever 2 and go to configure system and search job import ang give the url of jenkins1 and give the credentials of Jenkins sever1

Now go to job config history plugin in sever2 and start

**Jenkins master-slave configuration**

**Are u going install Jenkins in slave linux instance**

**Ans: no**

**We are going install Jenkins slave in another linux server**

**Prerequisites for master Jenkins and slave is Java jdk**

**If you are using maven u need to install maven in the slave serves**

**For improving the performance of master Jenkins we use master-slave configuration**

### Terraform Installation - Linux Server- Mithun

**Terraform Installation**

**#Login as a root user in ec2 instance**  
sudo su -  
  
**#You will need to upgrade your system and packages**  
yum update -y  
  
**#Install wget and unzip packages**  
yum install wget unzip vim -y  
  
**#Download the terraform software.**  
**#Use https://www.terraform.io/downloads.html to download the terraform software.**  
  
wget https://releases.hashicorp.com/terraform/0.12.21/terraform\_0.12.21\_linux\_amd64.zip  
  
**#Extract the terraform software.**  
unzip terraform\_0.12.21\_linux\_amd64.zip -d /usr/local/bin/  
  
**#Check the version**  
terraform -v (OR) terraform version  
  
**#Help**  
terraform -help