

# **PG DO - DevOps Capstone Project**

**Project -3**

**Tested By Kalaivani Ravishankar**

**Vani212008@gmail.com**

**<https://github.com/vanishankar/Newrepo.git>**

# **Create a CI/CD Pipeline to convert the legacy development process to a DevOps process.**

## **Introduction:**

In this project I will apply the skills and knowledge which were developed throughout Post Graduate Program in DevOps

- **Working in AWS**
- **Docker**
- **Git, GitHub,**
- **Jenkins,**
- **Tomcat Apache and**
- **Maven**

## **Application:**

The application I used sample java application. I used ec2 instance to create Docker server and Jenkins server

## **Tools used:**

1. **EC2**
2. **Jenkins**
3. **Docker**
4. **Git**

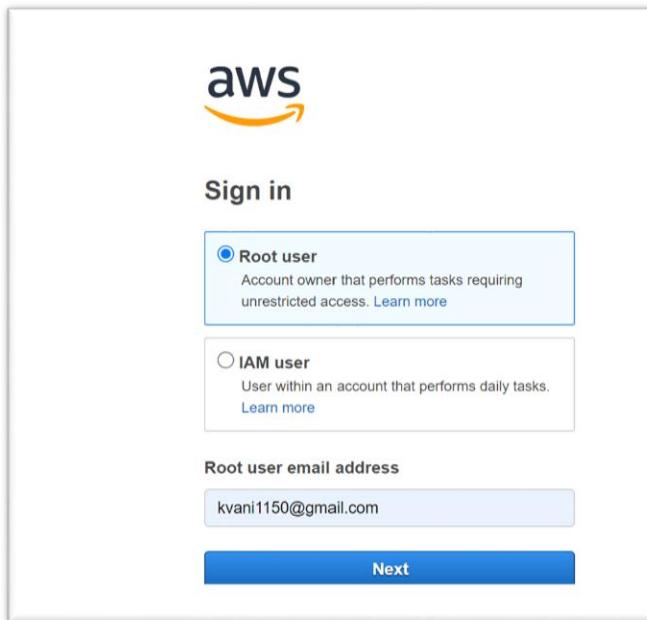
## **Steps to create ec2-instance:**

### **Step 1: Login to the aws console: -**

I use this link <https://console.aws.amazon.com/ec2/> to login to aws console account.

Select Root User and I gave email address to login into aws console

## aws sign in page:



## Step 2: Choose Launch Instance

First of all, click on 'Launch Instance' button shows in the below picture for launch/create new ec2 instance in aws:

### Launching ec2:

A screenshot of the AWS EC2 Instances page. The top navigation bar includes the AWS logo, a search bar, and a "Launch instances" button which is highlighted with a red box. The main area shows a table of existing instances. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. Two instances are selected: "Docker\_server" and "jenkins\_server".

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
jenkins	i-006493c979cecce41	Stopped	t2.micro	-	No alarms +
dockerservice	i-0b3834b0a833102f6	Stopped	t2.micro	-	No alarms +
Tomcatserver	i-0beb71a3abe067d0f	Stopped	t2.micro	-	No alarms +
Docker_server	i-0a3a1d05b413712a1	Stopped	t2.micro	-	No alarms +
jenkins_server	i-01ddc0489cf68d06	Stopped	t2.micro	-	No alarms +

It will Lauch new instance

## Step 3: choose AMI

In this step, choose AMI (I chose Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type - ami-0022f774911c1d690 (64-bit x86) / ami-0e449176cecc3e577 (64-bit Arm))

The screenshot shows the AWS Launch Instance Wizard interface. At the top, there's a message about replacing the launch experience with a new one, with an 'Opt-in to the new experience' button. Below this, a navigation bar includes tabs for Step 1 (Choose AMI), Step 2 (Choose Instance Type), Step 3 (Configure Instance), Step 4 (Add Storage), Step 5 (Add Tags), Step 6 (Configure Security Group), and Step 7 (Review). The 'Cancel and Exit' button is also visible.

**Step 1: Choose an Amazon Machine Image (AMI)**

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search for an AMI by entering a search term e.g. "Windows"

**Quick Start**

- My AMIs
- Amazon Linux** Free tier eligible Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type - ami-0022f774911c1d690 (64-bit x86) / ami-0e449176cecc3e577 (64-bit Arm) Select
- AWS Marketplace
- Community AMIs

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Feedback Looking for language selection? Find it in the new Unified Settings © 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 63°F Cloudy 9:42 AM 5/26/2022

I chose Amazon Linux 2 Ami(hvm) and click select button

## Step 4: Choose EC2 Instance Types

In this step, choose ec2 instance type shown in below picture:

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance	IPv6 Support
	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	t2	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes

I selected t2.micro and click configuration instance detail

## Step 5: Configure Instance Detail:

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower price to the instance, and more.

Number of instances  Launch into Auto Scaling Group

Purchasing option  Request Spot instances

Network

Subnet

Auto-assign Public IP

Hostname type

In this step I selected number of instances is two and leave remaining as its

## Step 6: Add Storage of Ec2 Instance

In this step, I chose storage. By default, ec2 t2-micro provide 8gb ssd. Shown in the below picture:

## Step 7: Tag Instance of Ec2 Instance

In this step, I added tag of instance with a key-value pair

Key value pair shown in fig

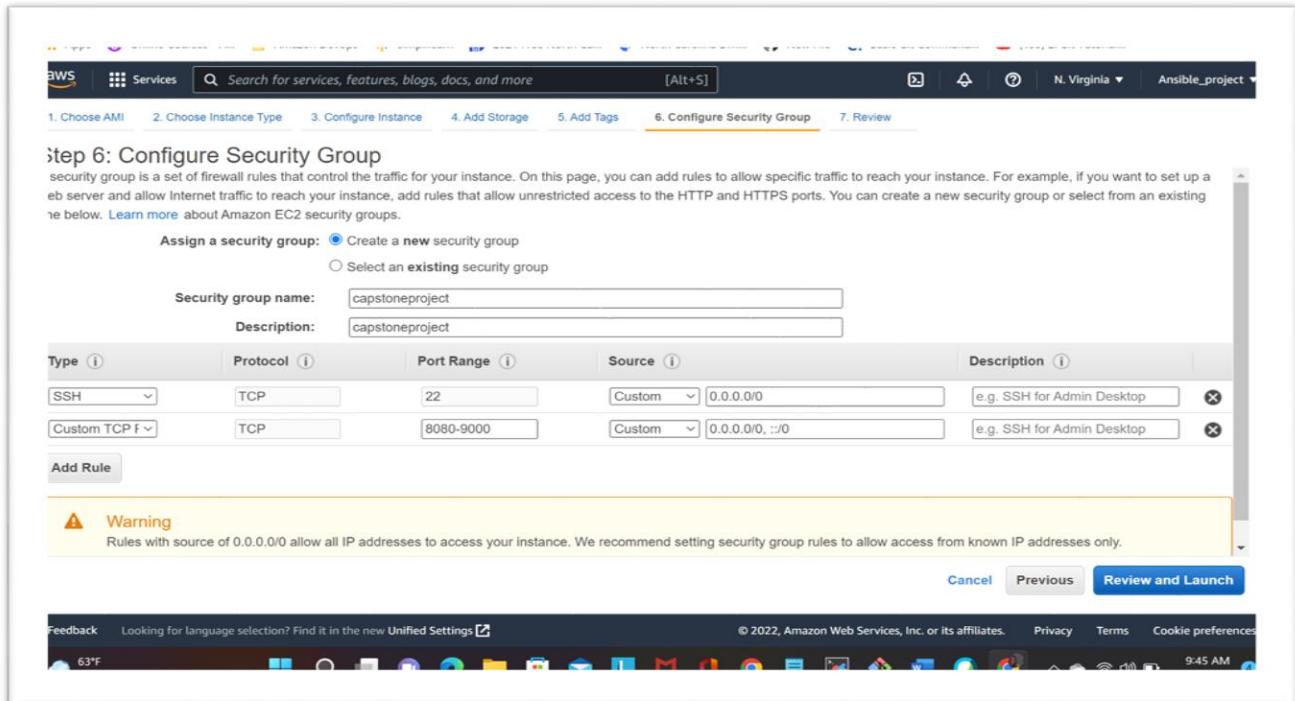
The screenshot shows the 'Add Tags' step of an AWS EC2 instance creation wizard. At the top, a navigation bar lists steps 1 through 7. Step 5, 'Add Tags', is highlighted with an orange underline. Below the navigation, a section titled 'Step 5: Add Tags' contains descriptive text about tags. A table allows users to add key-value pairs. In the first row, the 'Key' column has 'Name' and the 'Value' column has 'Tomcat\_deployserver'. To the right of the table are three checkboxes labeled 'Instances', 'Volumes', and 'Network Interfaces', all of which are checked. Below the table is a button labeled 'Add another tag'.

Key -----> name

Value --->Tomcat\_deployserver

## Step 8: Configure Security Groups

In this next step of configuring Security Groups



I created security group called capstone project and Defining protocols which I want enabled on my instance

## Step 9: Once, the firewall rules are set- Review and launch

Shown in below fig

**Step 7: Review Instance Launch**

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

**AMI Details**

**Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type - ami-0022f774911c1d690**

**Free tier eligible** Amazon Linux 2 comes with five years support. It provides Linux kernel 5.10 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is n...  
Root Device Type: ebs Virtualization type: hvm

**Instance Type**

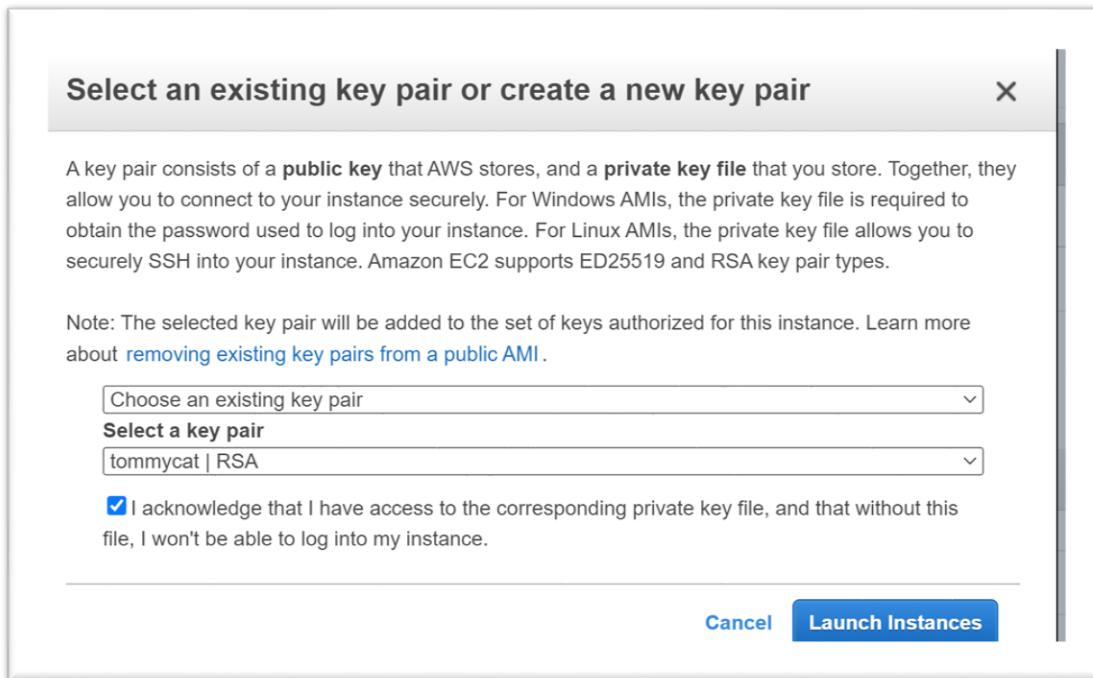
Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

**Review**

**Launch**

After clicking launch button it will take into creating keypair page

**Step 10: Create Key-Pair for Instance Access:** Creating a keypair for ec2 instance, (I used existing keypair) create new key pair and add the name of this key. Then download and save it in your secured folder. Shown in below fig.



**And finally click launch instance. It will launch 2 instances**

**Next, we are going to connect our using mobaxterm to connect ec2 instance using pem file**

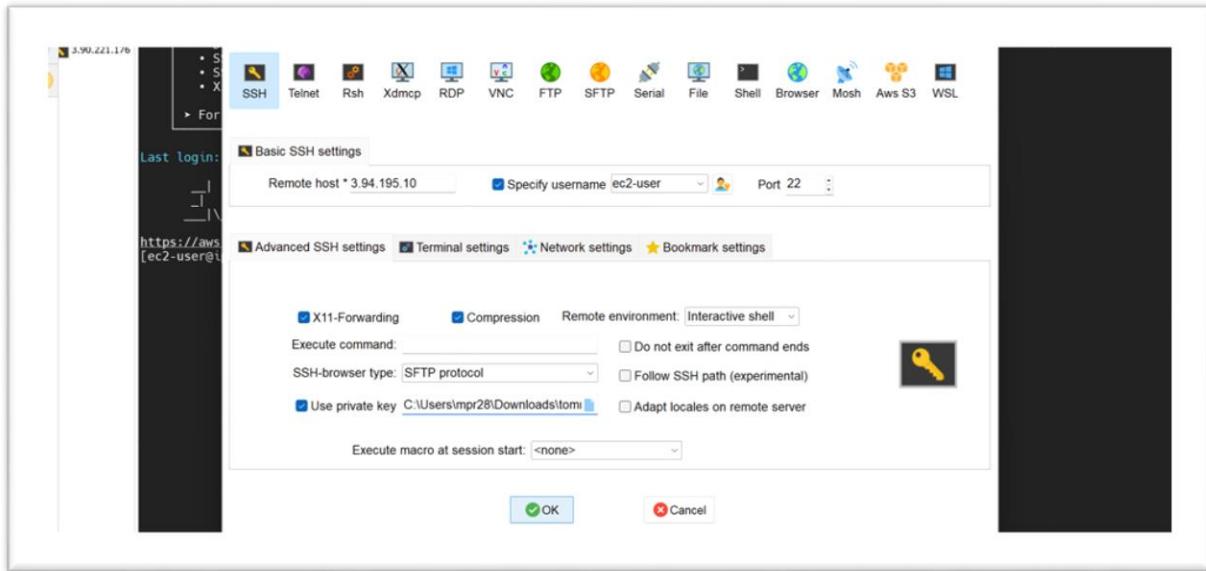
**Use pem file in MobaXterm to connect ec2 machine in AWS: -**

I used MobaXterm to connect ec2 machine. I downloaded mobaxterm .go to the session settings to create a new session, fill up the remote host and username.

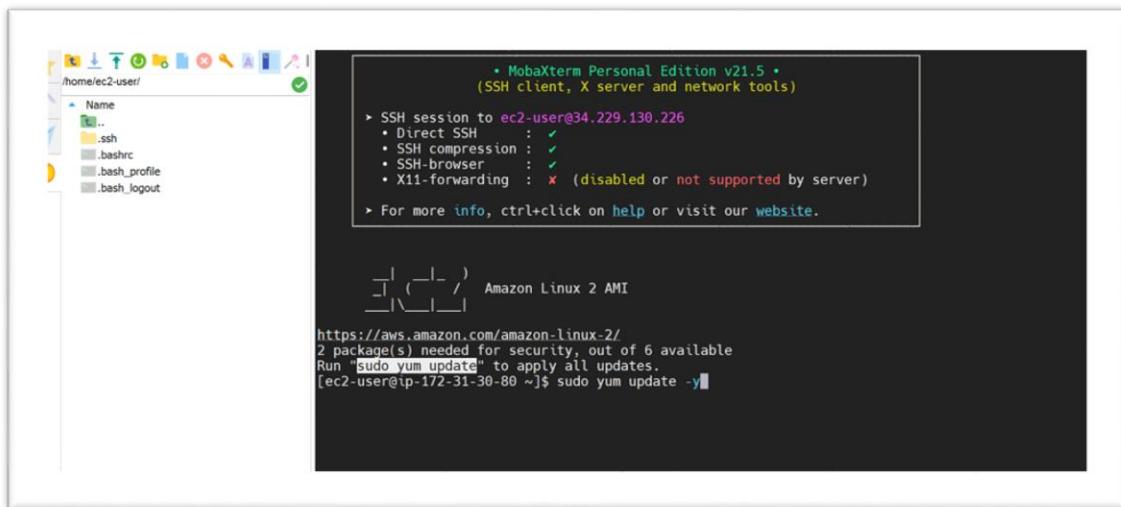
And click advance setting and select use private key, select your private key.

It will help to connect to ec2-instance. As shown in the below fig:10

## Configuring mobaxterm:-



After clicking ok it taking into page shown in below fig



I created two ec2 instance machine (Jenkins and Docker server)

## **Install Git on Docker\_ server and Jenkins server**

Using following command

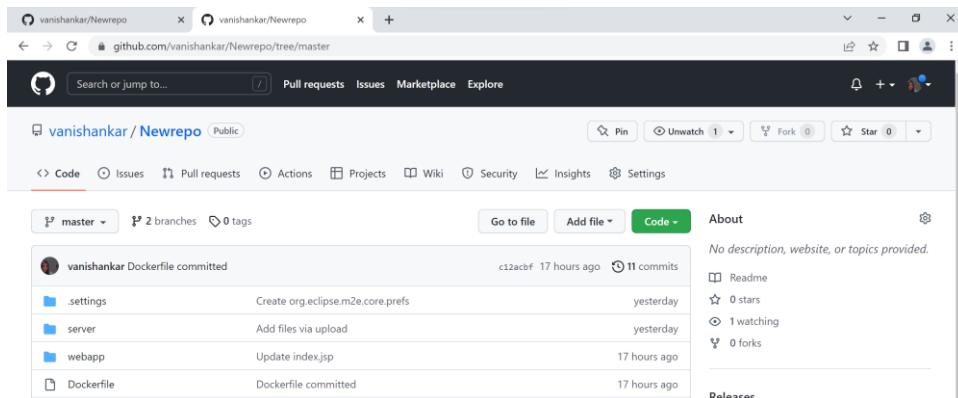
**yum install git -y**

```
complete.  
root@ip-172-31-24-109 ~]# git --version  
git version 2.32.0
```

**Git will be successfully installed in both server**

## **GitHub Repo**

- Create a GitHub project and check in the code of Aetna project. Also, we will be checking in the Docker file.
- The link to the public GitHub repo is  
<https://github.com/vanishankar/Newrepo.git>



## **Installing Docker:**

Command to install docker

**Yum install docker.**

The following pic shows docker version

```
git version 2.32.0  
[root@ip-172-31-24-109 ~]# docker --version  
Docker version 20.10.13, build a224086  
[root@ip-172-31-24-109 ~]#
```

**Command to start docker and check the status:**

```
service docker start  
service docker status
```

**Creating user:**

I created user called admin in both the server using following commands

```
useradd admin
```

```
passwd admin (Shown in below fig)
```

```
Docker version 20.10.13, build a224086  
[root@ip-172-31-24-109 ~]# useradd admin  
[root@ip-172-31-24-109 ~]# passwd admin  
Changing password for user admin.  
New password:  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password:  
passwd: all authentication tokens updated successfully.  
[root@ip-172-31-24-109 ~]#
```

```
[root@ip-172-31-19-44 ~]# useradd admin  
[root@ip-172-31-19-44 ~]# passwd admin  
Changing password for user admin.  
New password:  
BAD PASSWORD: The password is shorter than 8 characters  
Retype new password:  
passwd: all authentication tokens updated successfully.  
[root@ip-172-31-19-44 ~]#
```

## Add a user to docker group to manage docker: -

I added a user in docker group using following command

```
usermod -aG docker admin
```

## Adding Users to Sudoers File:

Next step is to add admin user into sudoers file using following commands

```
sudo visudo
```

Shown in the following fig

```
##  
## Allow root to run any commands anywhere  
root    ALL=(ALL)      ALL  
admin   ALL=(ALL)      NOPASSWD: ALL  
  
## Allows members of the 'sys' group to run networking, software,
```

The place where you see `root ALL=(ALL:ALL) ALL`, that's where we'll be amending our user name. And next use following command to configure `sshd_config` file to enable password less authentication, show in the following

## Enable Password Authentication in below File Path

`vi /etc/ssh/sshd_config`. uncomment the password authentication and change `no` to `yes` shown in following fig

```
# To disable tunneled clear text passwords, change to no here!  
PasswordAuthentication yes  
#PermitEmptyPasswords no  
PasswordAuthentication yes
```

## Reload Service with the command

**service sshd reload**

```
[root@ip-172-31-24-109 ~]# visudo  
[root@ip-172-31-24-109 ~]# vi /etc/ssh/sshd_config  
[root@ip-172-31-24-109 ~]# service sshd restart  
Redirecting to /bin/systemctl restart sshd.service  
[root@ip-172-31-24-109 ~]#
```

And I followed same steps in Jenkins server to add user and enable password less authentication.

Next steps are to create a connection between docker machine and Jenkins by downloading sshkey ,using following command .login as an admin using following command

**sudo su – admin**

## And generate sshkey using command **ssh-keygen**

**After generating I copied the public key to Jenkins server using following command **ssh-copy-id****

**admin@172-31-24-109 (ssh-copy-id user@<target-server> -  
Authorized\_keys)**

```
[admin@ip-172-31-24-109 ~]$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/admin/.ssh/id_rsa):  
Created directory '/home/admin/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/admin/.ssh/id_rsa.  
Your public key has been saved in /home/admin/.ssh/id_rsa.pub.  
The key fingerprint is:  
SHA256:QYdC4e2c97yVY+537WfzgAP2ei9c/Xo9yJJLrYpXhq0 admin@ip-172-31-24-109.ec2.internal  
The key's randomart image is:  
+---[RSA 2048]---+  
..o....|  
..o..|  
..o|  
o o|  
S * .|  
+ 0....|  
+o0B +|  
.E.+0 =X|  
..o+++=*0|  
SHA256:QYdC4e2c97yVY+537WfzgAP2ei9c/Xo9yJJLrYpXhq0 admin@ip-172-31-24-109.ec2.internal
```

**And next step is to configured Jenkins server to install Jenkins, Maven, java**

## JAVA

<http://openjdk.java.net/install/>

Install Java in Linux

`yum install java-1.8.0-openjdk-devel`

Java is installed in `usr/lib` directory

**Set Java Home Path in Bash profile**

`JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86_64`

`PATH=$PATH:$HOME/bin:$JAVA_HOME:`

**Install maven**

Download maven packages <https://maven.apache.org/download.cgi> onto Jenkins server. In this case, I am using `/opt/maven` as my installation directory.

```
Last login: Sat Jun  4 19:50:20 UTC 2022 on pts/0
[root@ip-172-31-19-44 ~]# mvn -v
Apache Maven 3.8.5 (3599d3414f046de2324203b78ddcf9b5e4388aa0)
Maven home: /opt/apache-maven-3.8.5
Java version: 1.8.0_312, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.10.112-108.499.amzn2.x86_64", arch: "amd64", family: "unix"
[root@ip-172-31-19-44 ~]#
```

**I used wget command to download maven (wget is used to download files from an HTTP/HTTPS or FTP server)**

`wget https://dlcdn.apache.org/maven/maven-3/3.8.5/binaries/apache-maven-3.8.5-bin.tar.gz`

**Next step is to extract the archive using following command**

`tar xzvf apache-maven-3.8.5-bin.tar.gz`

## Maven home path in bash profile

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-  
1.amzn2.0.2.x86_64  
  
MAVEN_HOME=/opt/apache-maven-3.8.5  
  
M2=/opt/apache-maven-3.8.5/bin  
  
PATH=$PATH:$HOME/bin:$JAVA_HOME:$MAVEN_HOME:$M2
```

The following fig shows maven and java path in .bash\_profile

```
User specific environment and startup program  
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86_64  
MAVEN_HOME=/opt/apache-maven-3.8.5  
= /opt/apache-maven-3.8.5/bin  
PATH=$PATH:$HOME/bin:$JAVA_HOME:$MAVEN_HOME:$M2
```

## Install Jenkins:-

Install Jenkins using the rpm or by setting up the Repo

I downloaded the latest Version of Jenkins form <https://pkg.jenkins.io/redhat-stable/> and install

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key  
amazon-linux-extras install epel
```

After downloading Start the Jenkins Services using following command

```
Service jenkins start
```

(Shown in following Fig.)

```
[root@ip-172-31-19-44 ~]# service jenkins status
jenkins.service - Jenkins Continuous Integration Server
  Loaded: loaded (/usr/lib/systemd/system/jenkins.service; disabled; vendor preset: disabled)
  Active: active (running) since Sun 2022-06-05 16:51:07 UTC; 54s ago
    Main PID: 9133 (java)
      CGroup: /system.slice/jenkins.service
              └─9133 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=%C/jenkins/war --httpPort=8080

Jun 05 16:51:03 ip-172-31-19-44.ec2.internal jenkins[9133]: 2022-06-05 16:51:03.903+0000 [id=28]           INFO      jenkins.InitReacto...nsion
Jun 05 16:51:05 ip-172-31-19-44.ec2.internal jenkins[9133]: 2022-06-05 16:51:05.579+0000 [id=28]           INFO      jenkins.InitReacto...loade
Jun 05 16:51:06 ip-172-31-19-44.ec2.internal jenkins[9133]: 2022-06-05 16:51:06.923+0000 [id=28]           INFO      jenkins.InitReacto...dents
```

## Accessing Jenkins

By default, jenkins runs at port 8080, You can access jenkins at

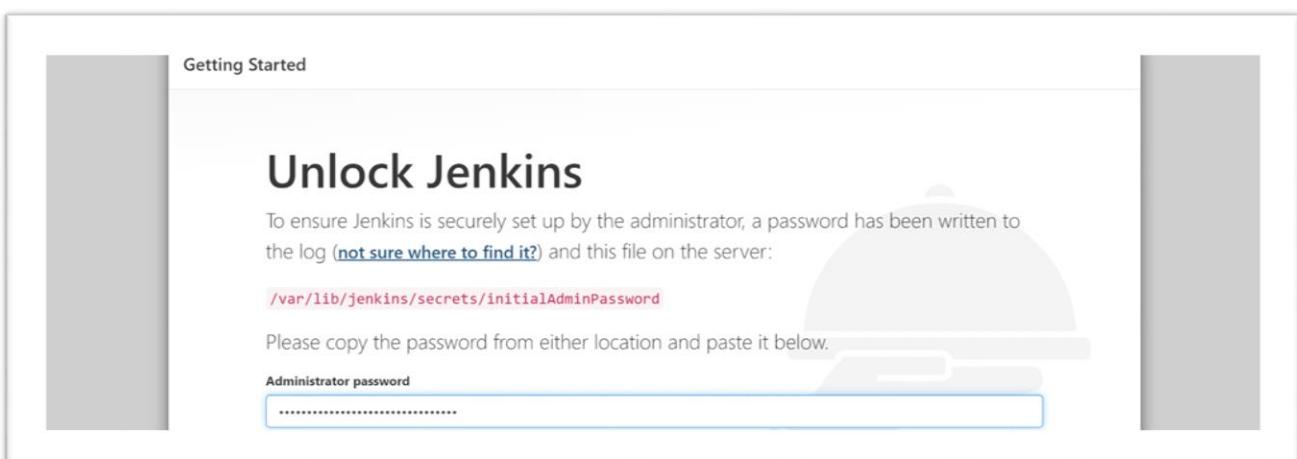
<http://YOUR-SERVER-PUBLIC-IP:8080>

## Configure Jenkins:

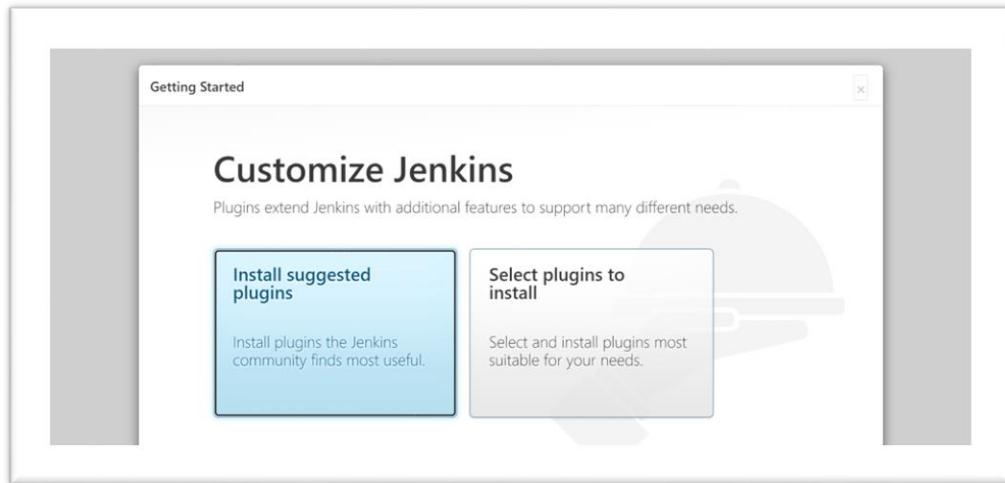
Grab the default password

Password Location:/var/lib/jenkins/secrets/initialAdminPassword, paste it

It is shown in following pic



**Choose suggested plugins. Shown in below fig:23**



**And enter username, password email address shown in below**

## Getting Started

# Create First Admin User

Username:	<input type="text" value="vani"/>
Password:	<input type="password" value="*****"/>
Confirm password:	<input type="password" value="*****"/>
Full name:	<input type="text" value="vani"/>
E-mail address:	<input type="text" value="vani123@mail.com"/>

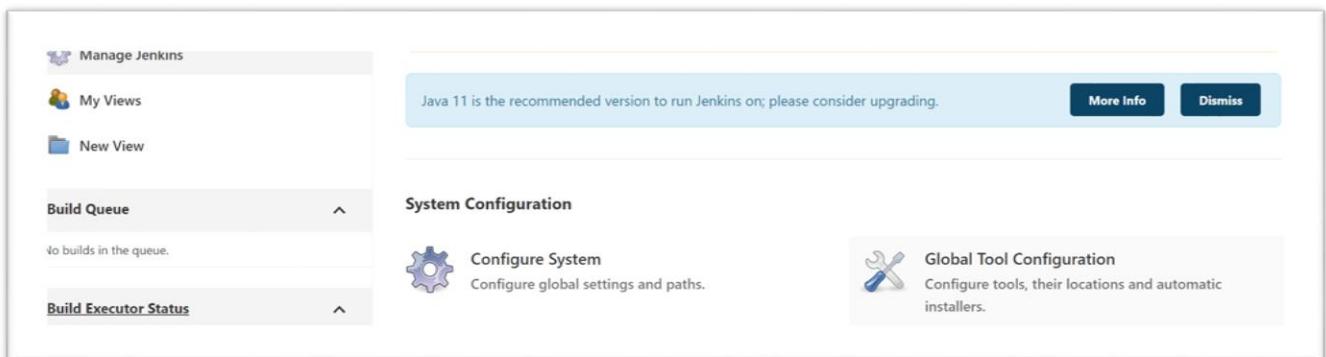
Click save and continue.it taken you into **jenkins dashboard**

## Jenkins Dashboard:

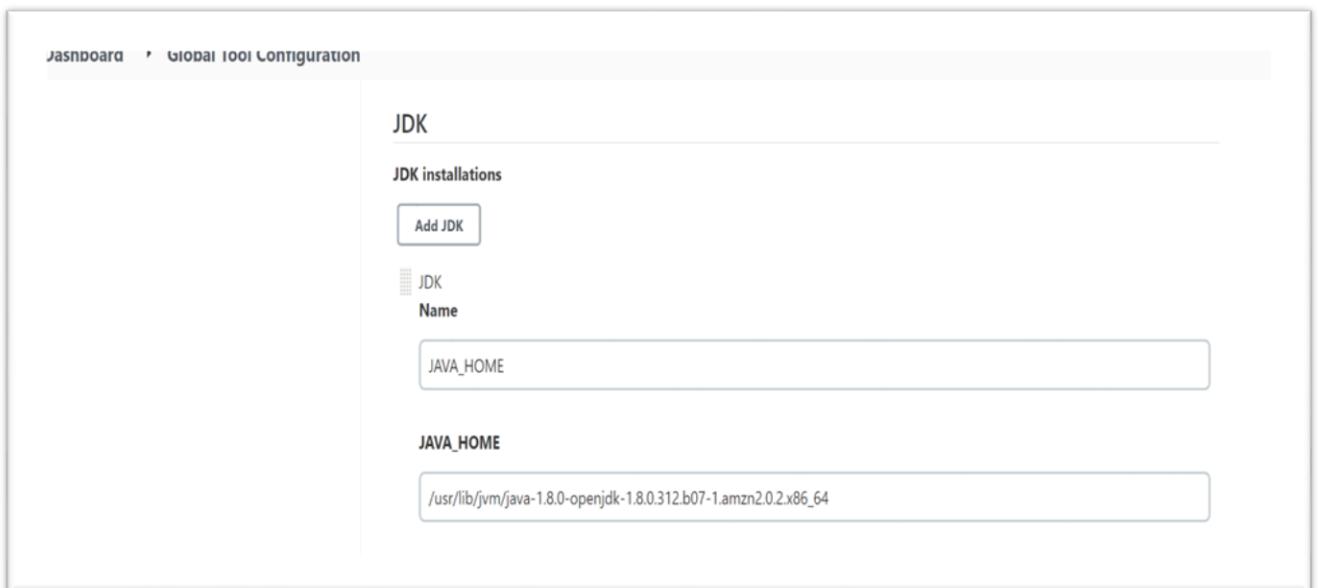
The screenshot shows the Jenkins dashboard. At the top, there is a navigation bar with the Jenkins logo, a search bar, and user information (vani). Below the navigation bar, the left sidebar contains links: Dashboard, New Item, People, Build History, Manage Jenkins, My Views, and New View. The main content area features a large "Welcome to Jenkins!" heading, a brief description of the page's purpose, and a "Start building your software project" button. A "Create a job" link is also visible at the bottom of the main content area.

## Configure java and maven path

Manage Jenkins > Global Tool Configuration > JDK



Click add JDK and give JDK name as JAVA\_HOME and path of JDK



I gave path of jdk is [/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86\\_64](/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86_64)

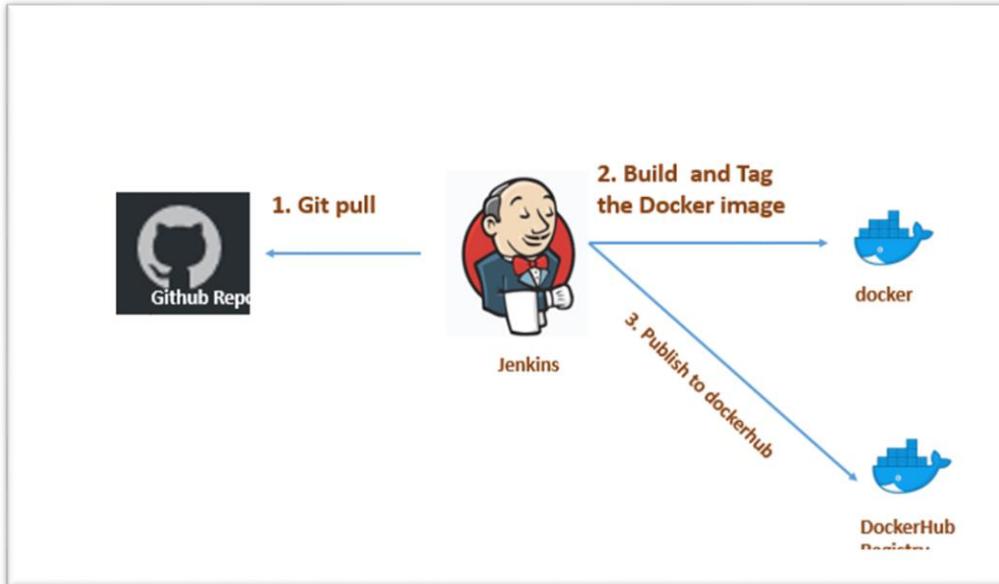
Next configure maven path, click maven give name for maven as MAVEN\_HOME And path of maven

The screenshot shows the Jenkins 'Maven' configuration screen. At the top, there's a header 'Maven'. Below it, a section titled 'Maven installations' contains a button 'Add Maven'. Under this, there's a row for 'Name' with the value 'MAVEN\_HOME' and a corresponding icon. Below the 'Name' field is another row for 'MAVEN\_HOME' containing the path '/opt/apache-maven-3.8.5'.

Maven \_home /opt/apache-maven-3.8.5

After finishing all configure setting we are going to push the docker image into docker hub using jenkins.

## Publish Docker image to Docker Hub using Jenkins: -



For creating docker image we need to write a **Dockerfile**. Go to the docker server, login as dockeradmin user using following command

**`sudo su – dockeradmin`**

### Create a Docker file using vi editor

The Dockerfile added to the project will use a prebuilt tomcat image. Over this image, it will add the war file that was built inside the container to the tomcat webapps directory so that microservice api can be accessed through any web browser.

**Dockerfile:**

**`vi Dockerfile`**

**`FROM tomcat:latest`**

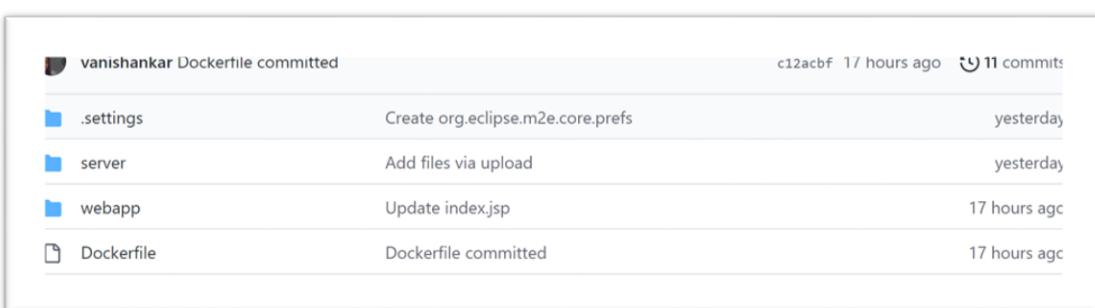
**`COPY ./webapp.war /usr/local/tomcat/webapps`**

```
RUN cp -r /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
```

Dockerfile:-

```
FROM tomcat:latest
COPY ./webapp.war /usr/local/tomcat/webapps
RUN cp -r /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
~
```

After docker file is created, I pushed into GitHub



## LOGIN TO DOCKERHUB

Next **login to docker hub** using following command

docker login,

I Gave username and password

It will allow us to login to docker hub to push image Shown in fig

```
admin@ip-172-31-24-109 ~]$ docker login
Login with your Docker ID to push and pull images from Docker Hub.
  Hint: https://hub.docker.com
Username: vanishankar
Password:
WARNING! Your password will be stored unencrypted in /home/admin/.docker/config.json.
Configure a credential helper to remove this warning. See
  https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
admin@ip-172-31-24-109 ~]$
```

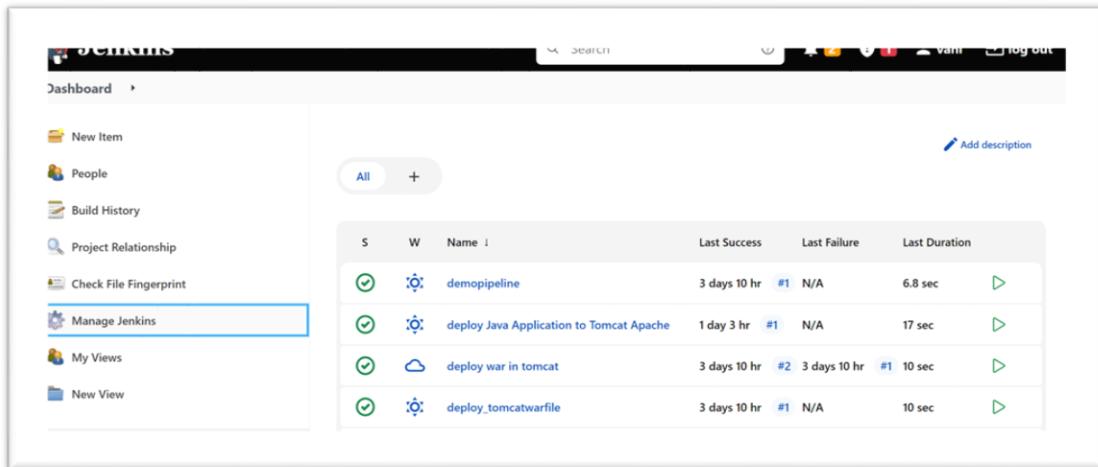
After I moved to browser (browse <http://YOUR-SERVER-PUBLIC-IP:8080>) And configure jenkins

## Setting Up Environment in jenkins

### Install Maven Integration plugin and publish over ssh

This plugin is required to used integrate maven (maven integration) and publish over ssh is used to Send build artifacts over SSH.

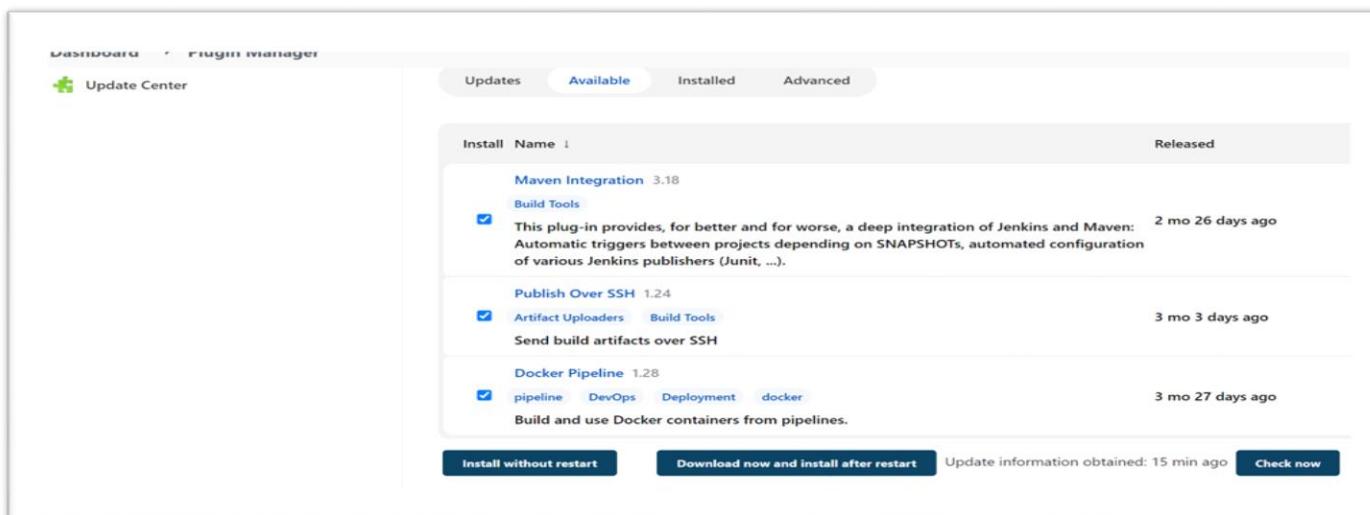
Click on Manage Jenkins -> Manage Plugins-> search for maven integration and publish over ssh -> Install it without restart. Shown in below fig



The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins' (which is selected and highlighted in blue), 'My Views', and 'New View'. The main area displays a table of pipelines:

S	W	Name	Last Success	Last Failure	Last Duration
✓	⌚	demopipeline	3 days 10 hr	#1 N/A	6.8 sec
✓	⌚	deploy Java Application to Tomcat Apache	1 day 3 hr	#1 N/A	17 sec
✓	☁️	deploy war in tomcat	3 days 10 hr	#2 3 days 10 hr	#1 10 sec
✓	⌚	deploy_tomcatwarfile	3 days 10 hr	#1 N/A	10 sec

## Plugins Manager:



The screenshot shows the Jenkins plugin manager. At the top, there are tabs for 'Updates', 'Available' (which is selected and highlighted in blue), 'Installed', and 'Advanced'. Below the tabs, there's a table of available plugins:

Install	Name	Released
<input checked="" type="checkbox"/>	Maven Integration 3.18	2 mo 26 days ago
<input checked="" type="checkbox"/>	Build Tools	
<input checked="" type="checkbox"/>	This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTs, automated configuration of various Jenkins publishers (Junit, ...).	
<input checked="" type="checkbox"/>	Publish Over SSH 1.24	3 mo 3 days ago
<input checked="" type="checkbox"/>	Artifact Uploaders Build Tools	
<input checked="" type="checkbox"/>	Send build artifacts over SSH	
<input checked="" type="checkbox"/>	Docker Pipeline 1.28	3 mo 27 days ago
<input checked="" type="checkbox"/>	pipeline DevOps Deployment docker	
<input checked="" type="checkbox"/>	Build and use Docker containers from pipelines.	

At the bottom of the page, there are buttons for 'Install without restart', 'Download now and install after restart', 'Update information obtained: 15 min ago', and 'Check now'.

Click on **Manage Jenkins** ->**system configuration-> ssh server** and configure all the details shown in foll fig.

## System configuration

The screenshot shows the Jenkins 'Manage Jenkins' page. On the left, there is a sidebar with various links: 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted), 'My Views', 'New View', 'Build Queue' (with the message 'No builds in the queue.'), and 'Build Executor Status'. The main content area is titled 'Manage Jenkins' and contains a yellow warning box: 'Building on the built-in node can be a security issue. You should see distributed builds. See [the documentation](#)'. Below that is a blue info box: 'Java 11 is the recommended version to run Jenkins on; please cons'. Under the title 'System Configuration', there is a box with a gear icon labeled 'Configure System' and the description 'Configure global settings and paths.'

## Configure publish over ssh

I gave name for server is Tomcat\_deployer, hostname is private Ip address of docker machine, then I gave username as admin which I already created and password which help to connect user

SSH Server

Name ?

Tomcat\_deployserver

Hostname ?

172.31.24.109

Username ?

admin

Remote Directory ?

Use password authentication, or use a different key ?

Passphrase / Password ?

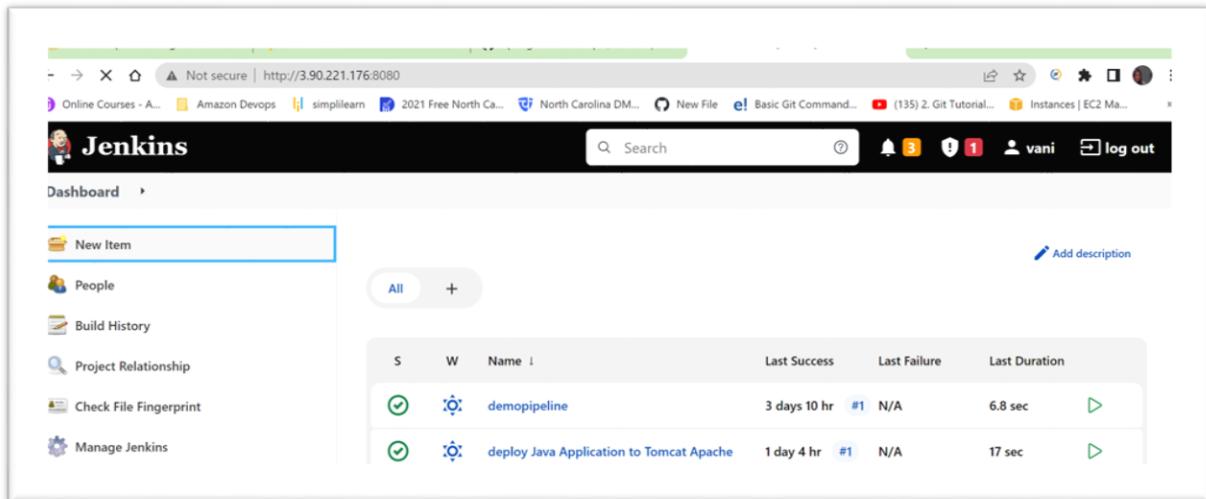
\*\*\*\*\*

Save    Apply

Once I finished all configuration. Next, I created a new job for pushing docker image using docker hub.

## CREATING A NEW JOB

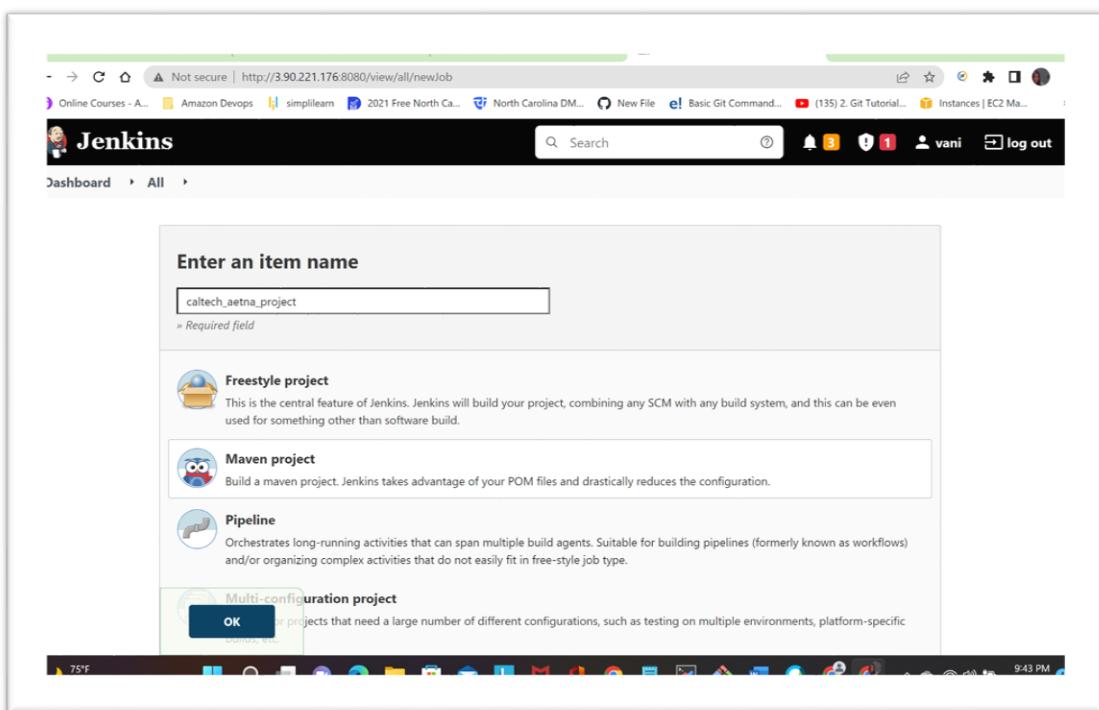
Navigate to dashboard click new item



The screenshot shows the Jenkins dashboard at <http://3.90.221.176:8080>. The top navigation bar includes links for Online Courses, Amazon Devops, simplilearn, 2021 Free North Ca..., North Carolina DM..., New File, Basic Git Command..., (135) 2. Git Tutorial..., Instances | EC2 Ma..., and a log out link. The main header says "Jenkins". On the left, there's a sidebar with links for Dashboard, New Item (which is highlighted with a blue border), People, Build History, Project Relationship, Check File Fingerprint, and Manage Jenkins. The main content area shows a table of existing jobs:

S	W	Name	Last Success	Last Failure	Last Duration
✓	⌚	demopipeline	3 days 10 hr	#1 N/A	6.8 sec
✓	⌚	deploy Java Application to Tomcat Apache	1 day 4 hr	#1 N/A	17 sec

**It will navigate you to the following screen:**



Click on Maven Project and I gave a name for project(caltech\_aetna\_project).

Once done, click on Ok button.

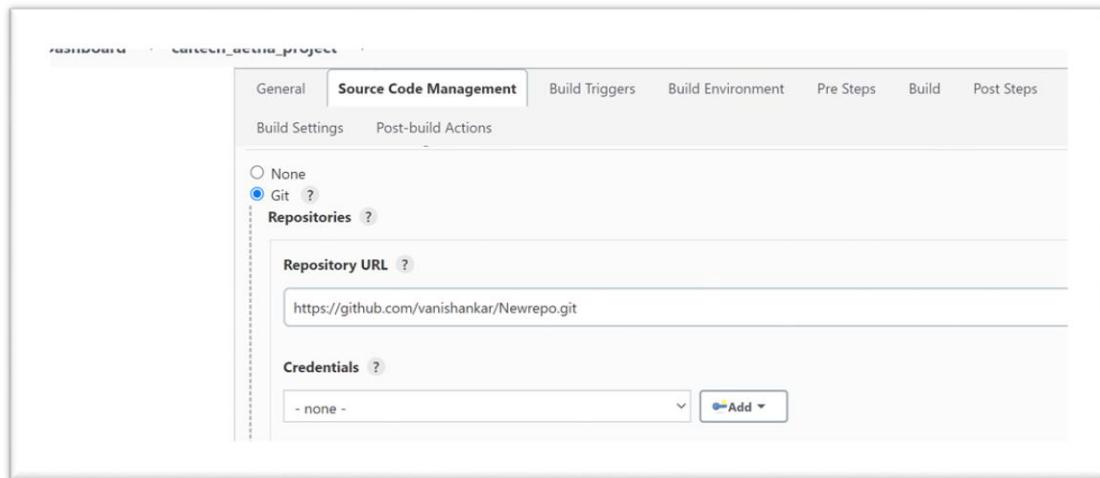
Navigate down to general Section and give description about project, next



## Source code management

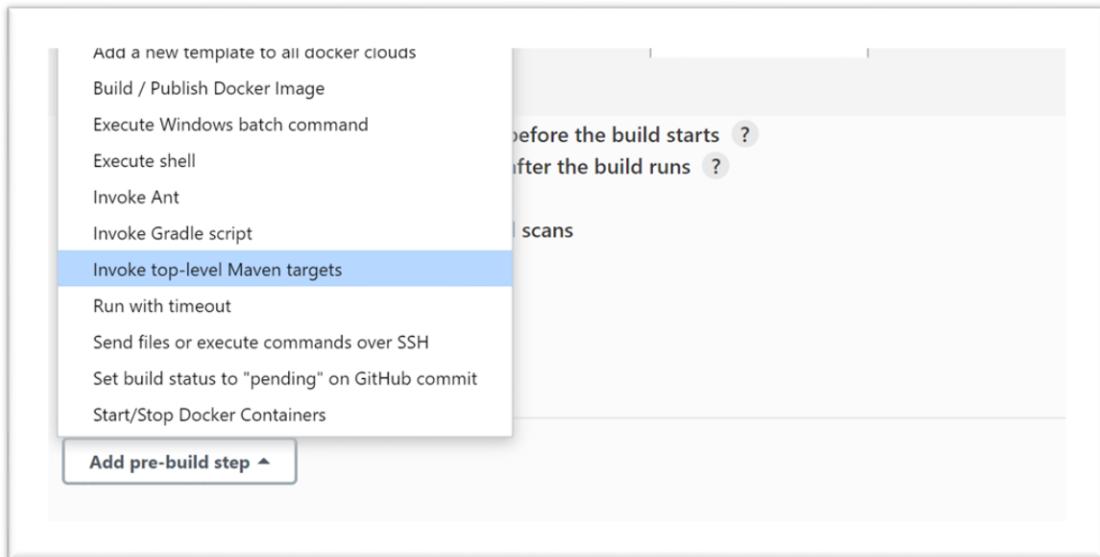
navigate to source code management give git hub repo URL

GitHub:<https://github.com/vanishankar/Newrepo.git>, shown in below fig

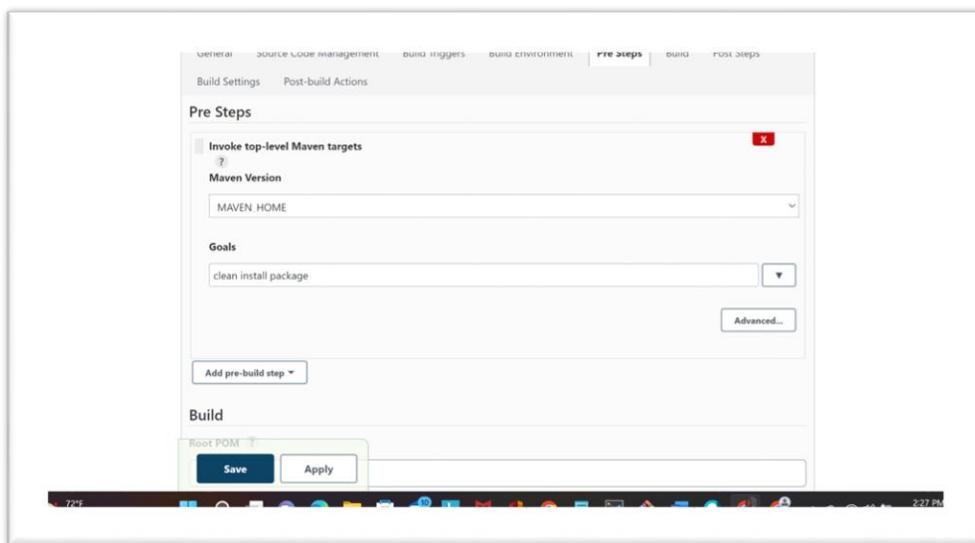


Next, I navigate to pre steps select top level maven goals shown in foll fig

## Pre build steps:

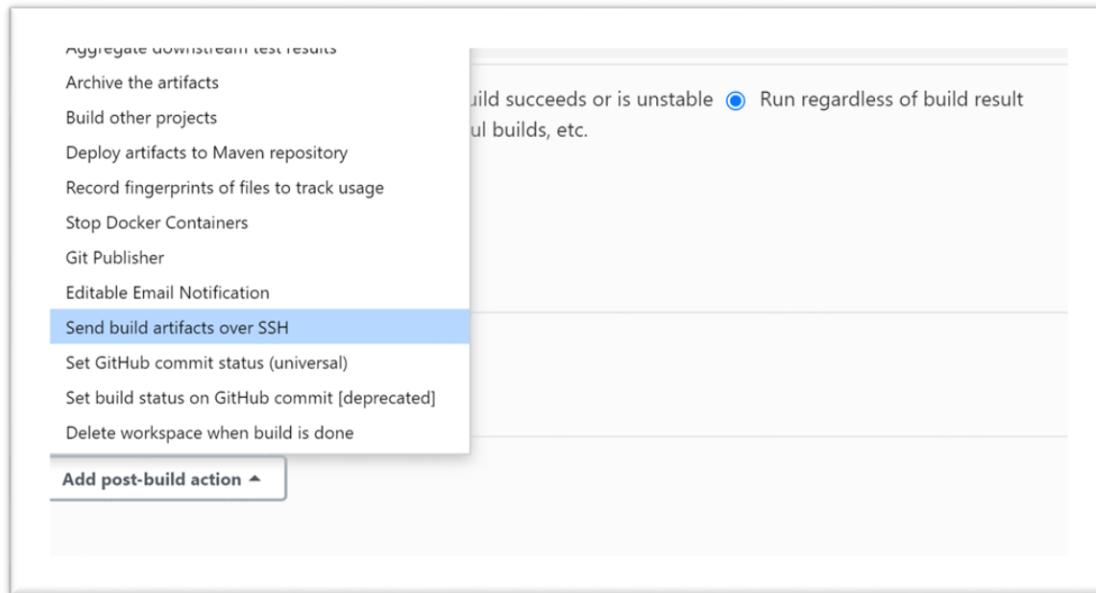


and I gave maven goals (**clean, install, package.**) Clean will delete all previously compiled Java .class files and resources (like. properties) in your project. Your build will start from a clean slate. **Install** will then compile, test & package your Java project and even *install/copy* your built .jar/.war file into your local Maven repository



**package:** Converts your .java source code into a .jar/.war file and puts it into the /target folder. **Next, I navigate to post steps I click send build artifacts over ssh( to build artifacts to docker server)**

## Post Build Action



Select **Send build artifacts over SSH** under **Post-Build Actions** section. I Selected my remote machine from the dropdown Tomcat\_deployerserver

## Under Transfer Set

- **Source Files** - add relative path to the build directory where build artifacts reside.  
(I add source file –webapp/target/webapp.war)
- **Remove Prefix** - First part of the file path that should not be created on the remote server.
- **Remote Directory** - Destination folder where all these stuffs should be placed. The directory name mentioned here will be created under your SFTP home path configured on your SSH server. (I select current directory to store webapps.war file)

## POST-BUILD-ACTIONS:

**POST-BUILD ACTIONS**

**Send build artifacts over SSH** X

SSH Publishers

SSH Server

Name ?

Tomcat\_deployserver

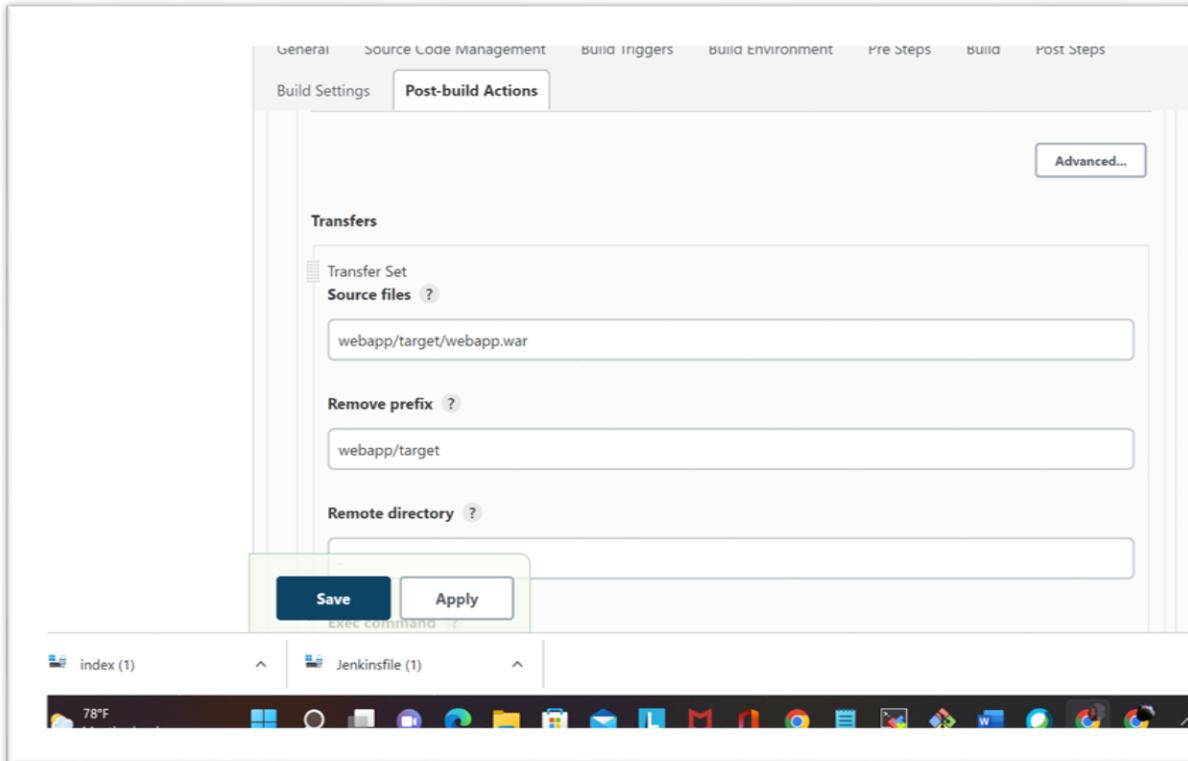
Verbose output in console ?

Credentials ?

Retry ?

Label ?

## POST-BUILD-ACTIONS



Enter **Exec command** which you want to run on a remote server.

and to automate docker file and give the following command

**docker build -t .** -----> [Build an image from a Dockerfile and tag the name](#)

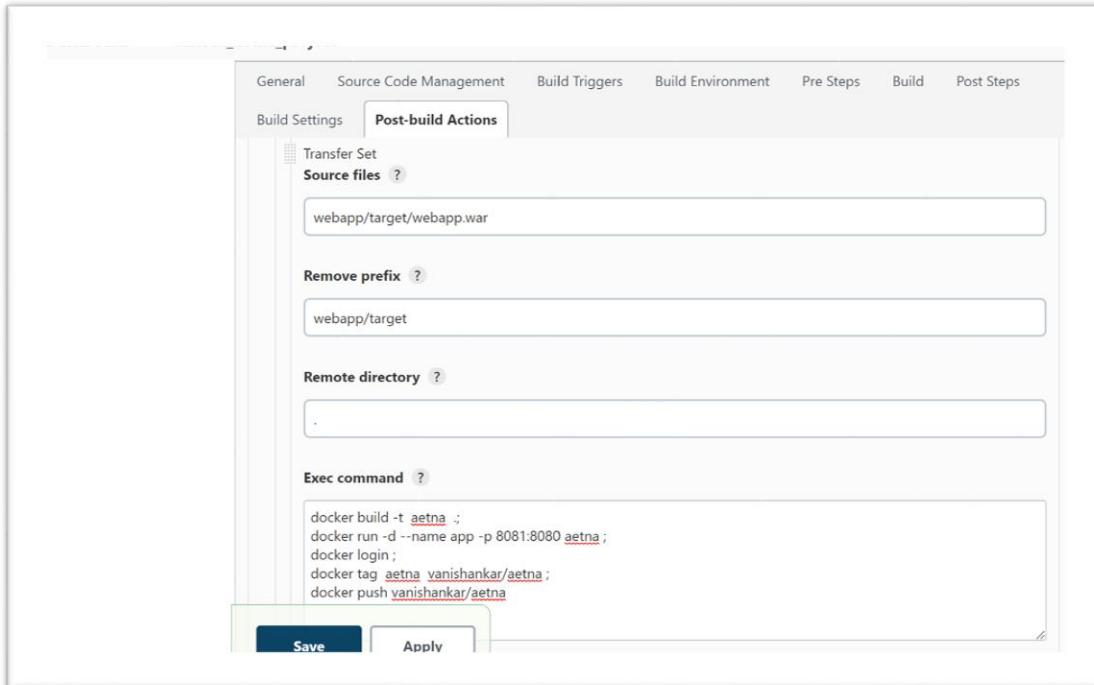
**docker run -d -- name app -p 8081:8080 aetna** -----> [Run a command in a new container in a port 8081 externally](#)

**docker login** -----> [Log in to a Docker registry](#)

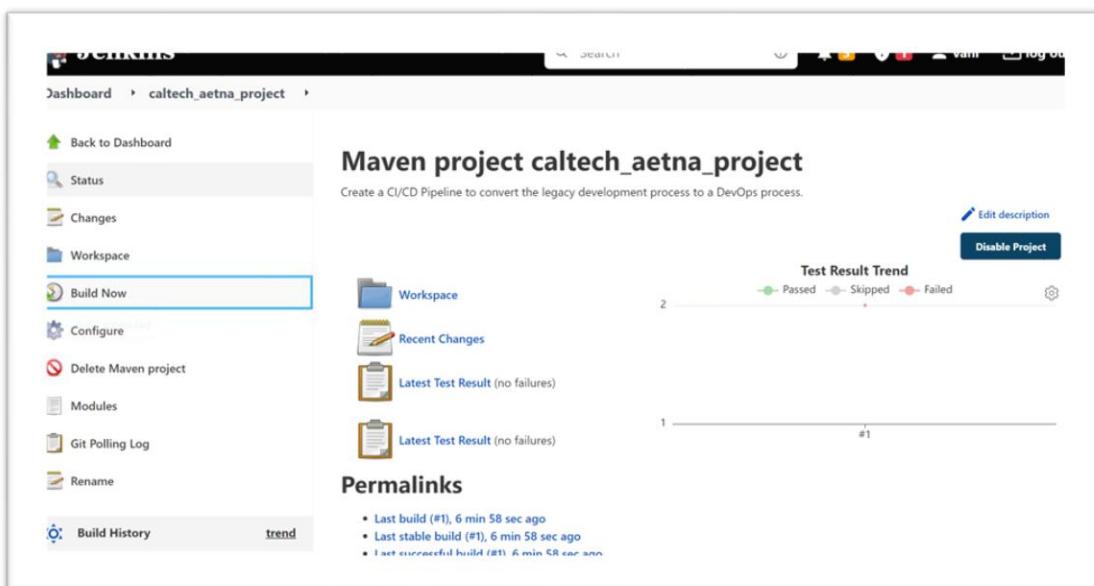
**docker tag aetna vanishankar/aetna** -----> [it Create a tag aetna that refers to SOURCE\\_IMAGE](#)

**docker push vanishankar/aetna** -----> [Push an aetna image to a repository](#)

**This docker command will help to build image, run the container, login to docker hub and pushes the image what we created**



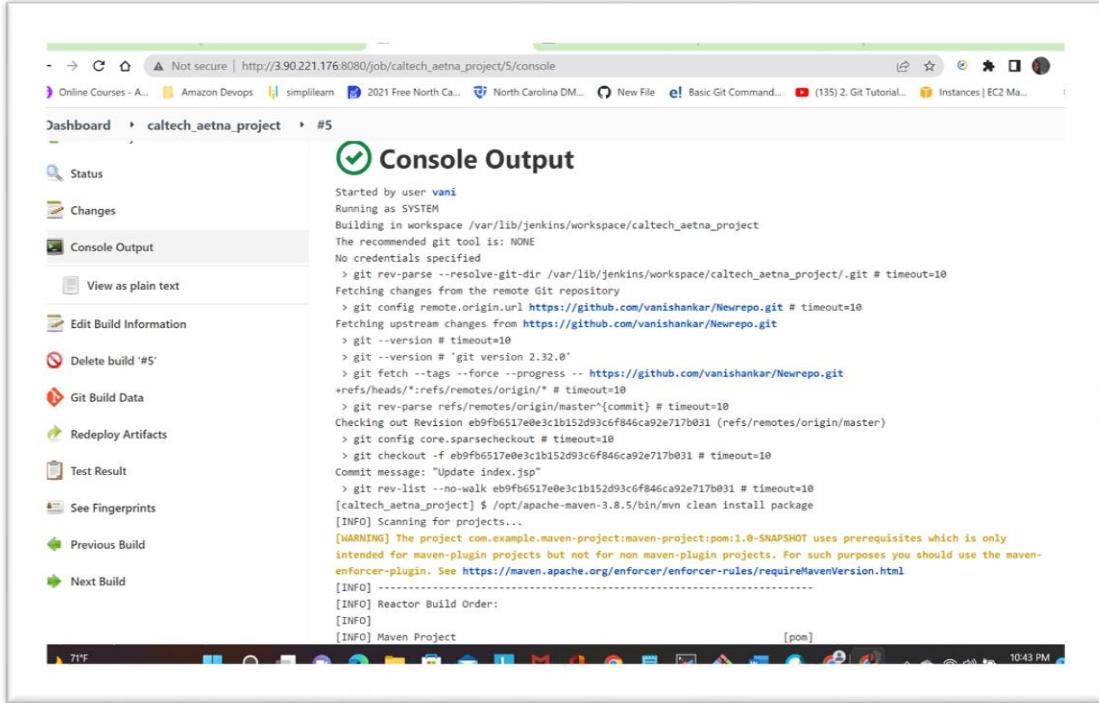
Click save and move to dashboard click Build now



**Finally, our caltech\_aetna\_project job is success. Below fig shows output of our job**

### Console output:

The foll fig shows output of our job



The screenshot shows a Jenkins console output page. The URL is [http://3.90.221.176:8080/job/caltech\\_aetna\\_project/5/console](http://3.90.221.176:8080/job/caltech_aetna_project/5/console). The page title is "Console Output". The left sidebar shows project navigation: Dashboard, Status, Changes, **Console Output** (selected), View as plain text, Edit Build Information, Delete build #5, Git Build Data, Redeploy Artifacts, Test Result, See Fingerprints, Previous Build, and Next Build. The main content area displays the Jenkins log output:

```
Started by user vani
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/caltech_aetna_project
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/caltech_aetna_project/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/vanishankar/Newrepo.git # timeout=10
Fetching upstream changes from https://github.com/vanishankar/Newrepo.git
> git --version # timeout=10
> git version 2.32.0
> git fetch --tags --force --progress -- https://github.com/vanishankar/Newrepo.git
+refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^(commit) # timeout=10
Checking out Revision eb9fb6517e0e3c1b152d93c6f846ca92e717b031 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f eb9fb6517e0e3c1b152d93c6f846ca92e717b031 # timeout=10
Commit message: "Update index.jsp"
> git rev-list --no-walk eb9fb6517e0e3c1b152d93c6f846ca92e717b031 # timeout=10
[caltech_aetna_project] $ /opt/apache-maven-3.8.5/bin/mvn clean install package
[INFO] Scanning for projects...
[WARNING] The project com.example.maven-project:maven-project:pom:1.0-SNAPSHOT uses prerequisites which is only intended for maven-plugin projects but not for non maven-plugin projects. For such purposes you should use the maven-enforcer-plugin. See https://maven.apache.org/enforcer/enforcer-rules/requireMavenVersion.html
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] Maven Project [pom]
```

### Console output

The screenshot shows a web browser window displaying the Jenkins console output for a job named 'caltech\_aetna\_project'. The URL is http://3.90.221.176.8080/job/caltech\_aetna\_project/5/console. The output shows the execution of various Maven plugins: surefire, jar, install, resources, and compiler. It also includes a test report for the 'TestGreeter' class, indicating 2 tests run, 0 failures, 0 errors, and 0 skipped. The Jenkins interface includes a navigation bar at the top and a toolbar at the bottom.

```
[INFO] --- maven-surefire-plugin:2.11:test (default-test) @ server ---
[INFO] Surefire report directory: /var/lib/jenkins/workspace/caltech_aetna_project/server/target/surefire-reports

-----
TESTS
-----
Running com.example.TestGreeter
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.115 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

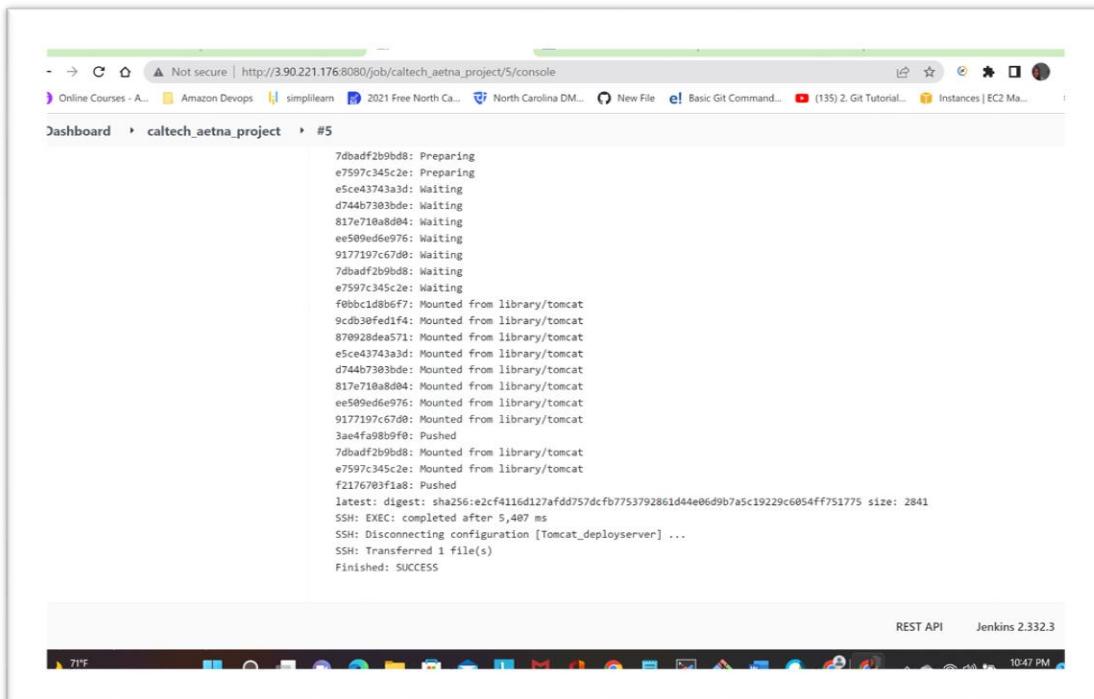
[JENKINS] Recording test results
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ server ---
[INFO] Building jar: /var/lib/jenkins/workspace/caltech_aetna_project/server/target/server.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ server ---
[INFO] Installing /var/lib/jenkins/workspace/caltech_aetna_project/server/target/server.jar to /var/lib/jenkins/.m2/repository/com/example/maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.jar
[INFO] Installing /var/lib/jenkins/workspace/caltech_aetna_project/server/pom.xml to /var/lib/jenkins/.m2/repository/com/example/maven-project/server/1.0-SNAPSHOT/server-1.0-SNAPSHOT.pom
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ server ---
[INFO] Using 'utf-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/caltech_aetna_project/server/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ server ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
```

## Console output:

The screenshot shows a web browser window displaying the Docker build logs for a job named 'caltech\_aetna\_project'. The URL is http://3.90.221.176.8080/job/caltech\_aetna\_project/5/console. The logs show the creation of a Docker container named 'apple', pulling the 'tomcat:latest' image, copying the 'webapp.war' file into the container, running the container, and finally pushing it to a repository. A warning message at the end advises against storing unencrypted passwords in config files.

```
Step 1/3 : FROM tomcat:latest
--> c795915cb678
Step 2/3 : COPY ./webapp.war /usr/local/tomcat/webapps
--> bb5d49590e08
Step 3/3 : RUN cp -r /usr/local/tomcat/webapps.dist/* /usr/local/tomcat/webapps
--> Running in 88fc7a2e437a
Removing intermediate container 88fc7a2e437a
--> 8571bd16da67
Successfully built 8571bd16da67
Successfully tagged aetna:latest
b0d82a3ac962a3eb1f18618d1abc0711a06673ad37994e147794bad0add2921d
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/admin/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

## Console output



The screenshot shows a browser window displaying the Jenkins job console output for a build. The URL is [http://3.90.221.176:8080/job/caltech\\_aetna\\_project/5/console](http://3.90.221.176:8080/job/caltech_aetna_project/5/console). The output shows a series of commands being run, including mounting from a library and pushing to a repository, followed by an SSH disconnect message and a success status.

```

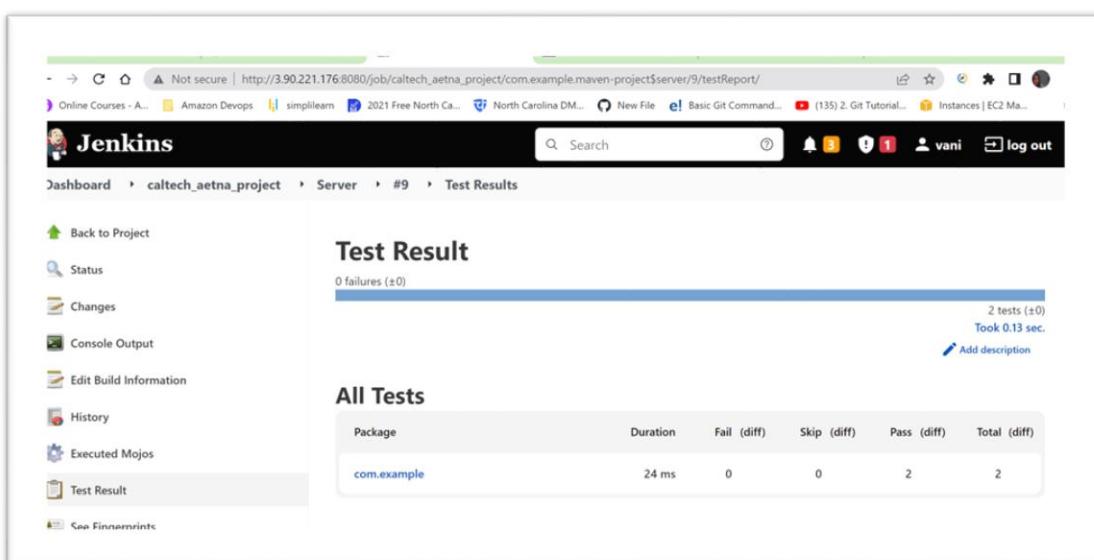
7dbadf2b9bd8: Preparing
e7597c345c2e: Preparing
e5ca43743a3d: Waiting
d744b7303bde: Waiting
817e710a8d04: Waiting
ee599d6e976: Waiting
9177197c6708: Waiting
7dbadf2b9bd8: Waiting
e7597c345c2e: Waiting
f0b0c1d8b6f7: Mounted from library/tomcat
9cdb30fed1f4: Mounted from library/tomcat
870928de571: Mounted from library/tomcat
e5ca43743a3d: Mounted from library/tomcat
d744b7303bde: Mounted from library/tomcat
817e710a8d04: Mounted from library/tomcat
ee599d6e976: Mounted from library/tomcat
9177197c6708: Mounted from library/tomcat
3aeaf98b9f0: Pushed
7dbadf2b9bd8: Mounted from library/tomcat
e7597c345c2e: Mounted from library/tomcat
f2176703f1a8: Pushed
latest: digest: sha256:e2cf4116d127afdd757dcfb7753792861d44e06d9b7a5c19229c6054ff751775 size: 2841
SSH: EXEC: completed after 5,407 ms
SSH: Disconnecting configuration [Tomcat_deployserver] ...
SSH: Transferred 1 file(s)
Finished: SUCCESS

```

REST API Jenkins 2.332.3

## Test result:

The below fig shows test results



The screenshot shows the Jenkins Test Result page for a specific build. The URL is [http://3.90.221.176:8080/job/caltech\\_aetna\\_project/com.example.maven-project\\$server/9/testReport/](http://3.90.221.176:8080/job/caltech_aetna_project/com.example.maven-project$server/9/testReport/). The page displays a summary with 0 failures and 2 tests passed, which took 0.13 seconds. Below this, a table titled "All Tests" shows the details for the single package "com.example".

Package	Duration	Fail (diff)	Skip (diff)	Pass (diff)	Total (diff)
com.example	24 ms	0	0	2	2

## Test Result:

The screenshot shows the Jenkins Test Result page for the com.example.TestGreeter job. The main title is "Test Result : TestGreeter". It indicates "0 failures (±0)" and "2 tests (±0) Took 24 ms.". Below this, there is a table titled "All Tests" showing two test cases: "greetShouldIncludeGreetingPhrase" (Duration: 14 ms, Status: Passed) and "greetShouldIncludeTheOneBeingGreeted" (Duration: 10 ms, Status: Passed). The left sidebar contains links for Back to Project, Status, Changes, Console Output, Edit Build Information, History, Executed Mojos, Test Result, and See Fingerprints.

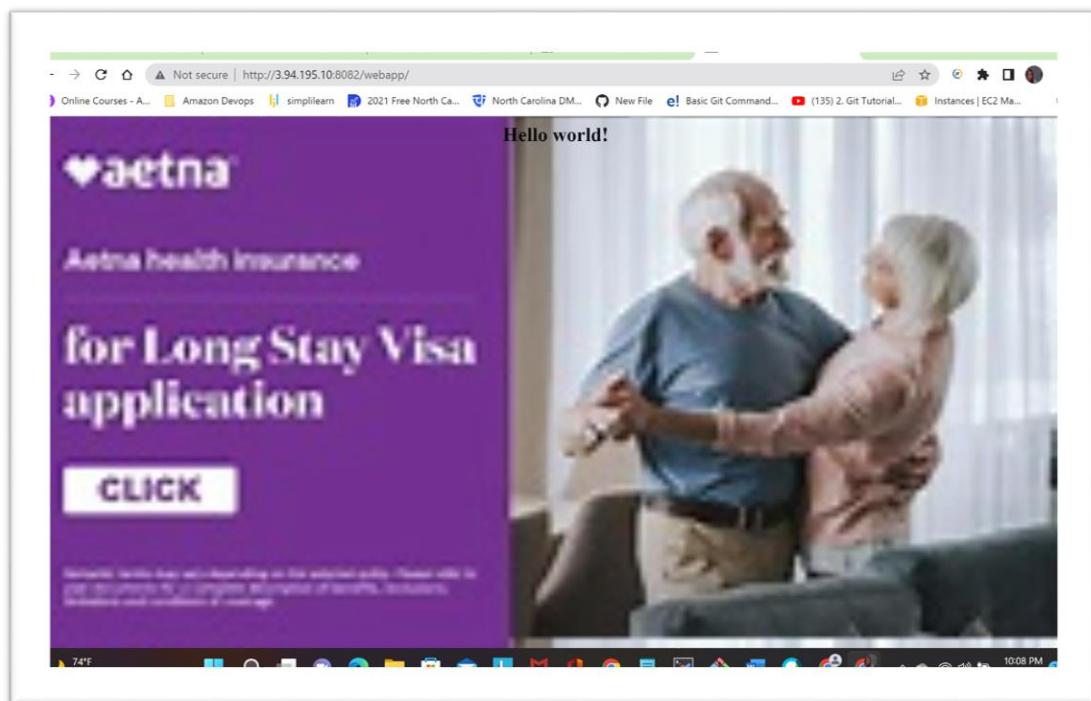
Next, we can access from browser with docker\_server Ip address with port (we gave 8081) ACCESSING FROM BROWSER

The screenshot shows a browser window displaying the Apache Tomcat 10.0.21 welcome page. The URL in the address bar is "http://3.94.195.10:8082/webapp". The page header says "Apache Tomcat/10.0.21" and features the Apache logo. A green banner at the top says "If you're seeing this, you've successfully installed Tomcat. Congratulations!". Below the banner, there is a cartoon cat icon and a list of recommended reading: "Security Considerations How-To", "Manager Application How-To", and "Clustering/Session Replication How-To". On the right side, there are buttons for "Server Status", "Manager App", and "Host Manager". At the bottom, there is a "Developer Quick Start" section with links for "Tomcat Setup", "First Web Application", "Realms & AAA", "JDBC DataSources", "Examples", "Servlet Specifications", and "Tomcat Versions".

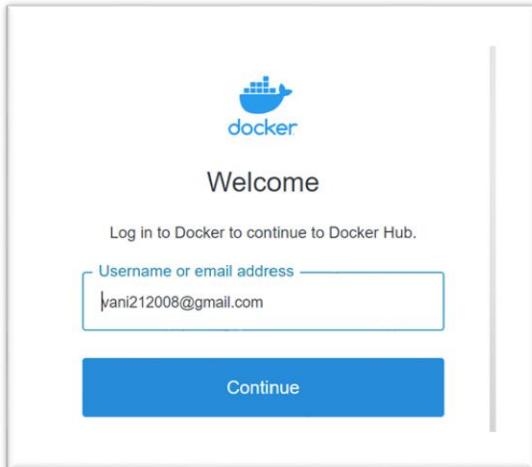
Our tomcat is running

## FINAL OUTPUT:

finally, we accessed from browser



Now login to docker hub with id our image aetna is pushed to **Docker hub**



Now login to docker hub with id and click continue button.it will navigate into Docker hub repo home page

## DOCKERHUB REPOSITORY

A screenshot of a web browser window showing a user's Docker Hub profile. The profile page for "vanishankar" is displayed, featuring a profile picture, the name "vanishankar", a "Edit profile" link, and a status message "Community User Joined December 28, 2021". Below this, there are tabs for "Repositories", "Starred", and "Contributed", with "Repositories" being the active tab. It shows 13 repositories, with two visible: "vanishankar/aetna" (Container image) and "vanishankar/aetnaprojects". The browser's address bar shows "hub.docker.com/u/vanishankar". The taskbar at the bottom of the screen includes icons for weather, system, and date/time information.

image aetna is pushed to **Docker hub**. Successfully pushed our docker image into docker hub using jenkins.

**Concepts used in the project is containerization**(docker) it helps Faster delivery of your applications, Deploy and scale more easily Get higher density and run more workloads

## Docker Troubleshooting

If you see the below error, it means docker service is not running on the machine. Start the docker service and try the build again

docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock.  
Is the docker daemon running?

**If you see the below error “Permission Denied”**

```
+ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon
socket at unix:///var/run/docker.sock: Post
http://%2Fvar%2Frun%2Fdocker.sock/v1.40/containers/create: dial unix
/var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
```

Then add jenkins user to the docker group and restart the jenkins service as shown below:

**sudo usermod -aG docker admin**

**sudo service jenkins stop**

**sudo service jenkins start**

**Finally, I deleted the all images and container running in the Docker server using following commands**

**To check or list out what are the images are present in our server I used following command**

**docker images**

**To list out all container**

**docker ps –a or docker container ls**

**To list out only running container:**

`docker ps`

## Delete docker images

```
docker rmi -f image id
```

## Delete docker container

`docker rm -f container id`

## Stop docker service

`docker stop container id`

## Delete unused image

## docker image prune

## Delete unused container

## docker container prune

## Check the status

## Service docker status

**Following fig shows deleted, pruned images and container**

## Image and container prune:

```
admin@ip-172-31-24-109 ~]$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
admin@ip-172-31-24-109 ~]$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
admin@ip-172-31-24-109 ~]$ docker system prune
WARNING! This will remove:
 - all stopped containers
 - all networks not used by at least one container
 - all dangling images
 - all dangling build cache

Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
admin@ip-172-31-24-109 ~]$ █
```

## Conclusion

Advantages and USPs: Docker The main reason is that Jenkins work really well with Docker. Without Docker you need to install additional tools and add different agents to Jenkins. With Docker, there is no need to install additional tools, you just use images of these tools. Jenkins will download them from internet for you (Docker Hub). Jenkins is a popular continuous integration and delivery (CI/CD) tool. Jenkins is open source and has a great developer community that maintains its existing software and builds plugins to increase its functionality. Though with excellent community support, while using Jenkins, you may suffer from issues like configuration files clashes which you can resolve by running Jenkins in a container technology like Docker.

Kalaivani

GitHub: <https://github.com/vanishankar/Newrepo.git>

Ravishankar