

# Automated CI Project for Maven Using Jenkins Server on AWS EC2 instance and Github Web-hook

**Author:** Shashwat Kumar

**Medium Link:**

<https://shashwat9kumar.medium.com/automated-ci-project-for-maven-using-a-remote-jenkins-server-on-aws-ec2-instance-and-github-509220e191f3>

---

A CI project that automatically triggers a remote Jenkins server when changes are pushed to github for a maven project.

The project triggers a pipeline on a remote Jenkins server to run the clean, compile, test and install stage for the maven project.

## **Aim**

- Create an Ubuntu EC2 instance on AWS
- Connect to the virtual cloud machine via SSH, and install required dependencies
- Install and run jenkins server on a virtual cloud machine
- Activate github webhooks for automated build triggering
- Creating the automated pipeline for Continuous Integration
- Testing the pipeline via a test push/commit

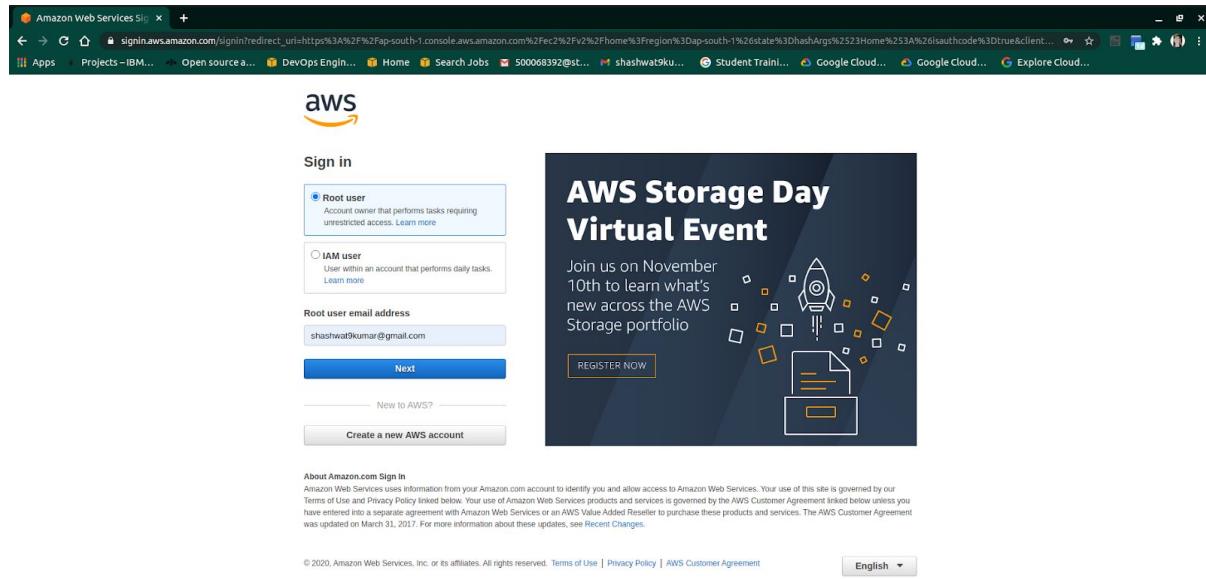
## **Step 1**

### **Create an Ubuntu EC2 instance on AWS**

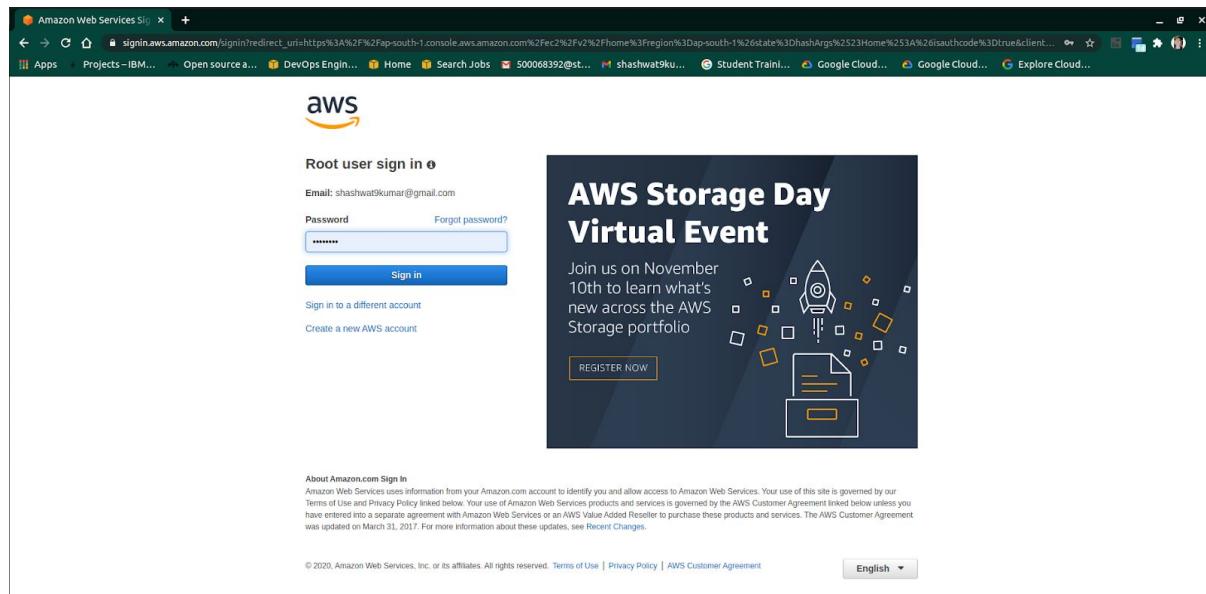
Visit the AWS sign in to console at: <https://signin.aws.amazon.com/>

And sign in using your root user account

Enter your root user username and click next



Enter your password and click [Sign In](#).



Upon sign in you'll land on the Amazon EC2 dashboard.  
On the resources section, click on instances

The screenshot shows the Amazon EC2 Dashboard. On the left, a navigation pane includes sections for Instances, Images, Elastic Block Store, and Network & Security. The 'Instances' section is expanded, showing options like Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, and Capacity Reservations. The main content area is titled 'Resources' and displays statistics for Instances, Elastic IPs, Dedicated Hosts, Snapshots, Volumes, Key pairs, Security groups, Placement groups, Load balancers, and Running instances. A prominent 'Launch instance' button is located in the 'Launch instance' section. To the right, there's an 'Account attributes' sidebar with options for Supported platforms (VPC), Default VPC (vpc-c65ddbad), Settings, EBS encryption, Zones, Default credit specification, and Console experiments. An 'Explore AWS' sidebar offers links to Run Apache Spark on EMR for Less, Launch Custom AMIs with Fast Snapshot Restore (FSR), and other services.

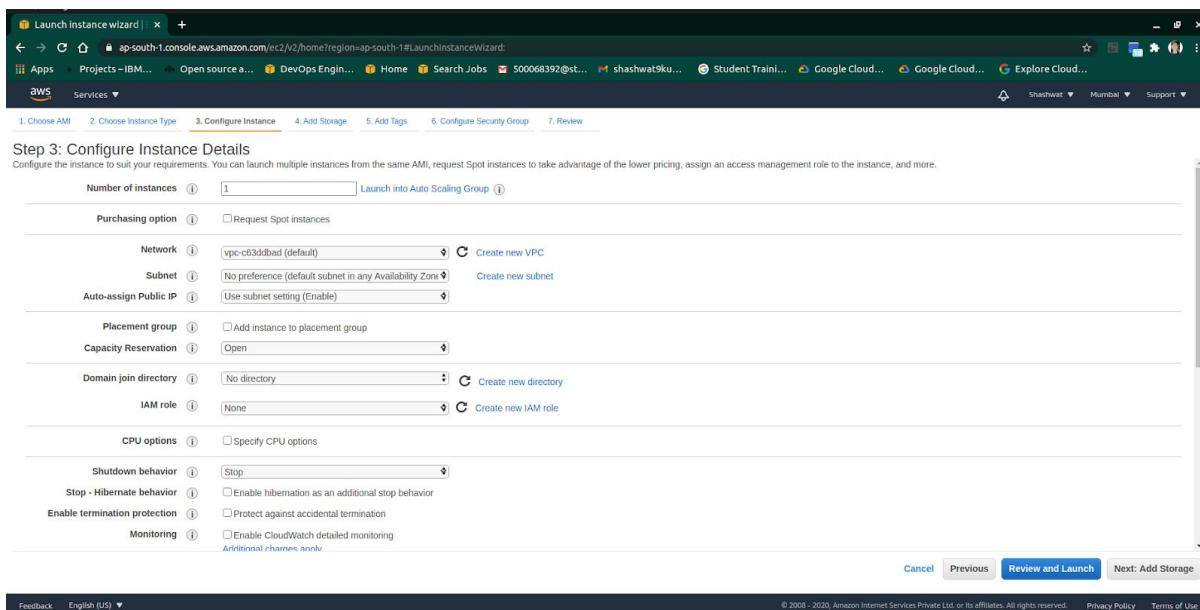
You'll be directed to a page similar to this.  
To create a new instance, click on the Launch Instance tab on the top right

The screenshot shows the 'Instances' page. The navigation pane is identical to the previous dashboard. The main area is titled 'Instances' and features a 'Launch instances' tab at the top right. Below it, there's a search bar and a table header with columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm Status, Availability zone, Public IPv4 DNS, Public IPv4 ... (truncated), and Elastic IP. A message states 'You do not have any instances in this region'. At the bottom, a note says 'Select an instance above' and provides icons for creating a new instance or launching an existing one. The footer includes standard copyright and legal links.

You'll now scroll down to find the Ubuntu Server 20.04 LTS and click on select.

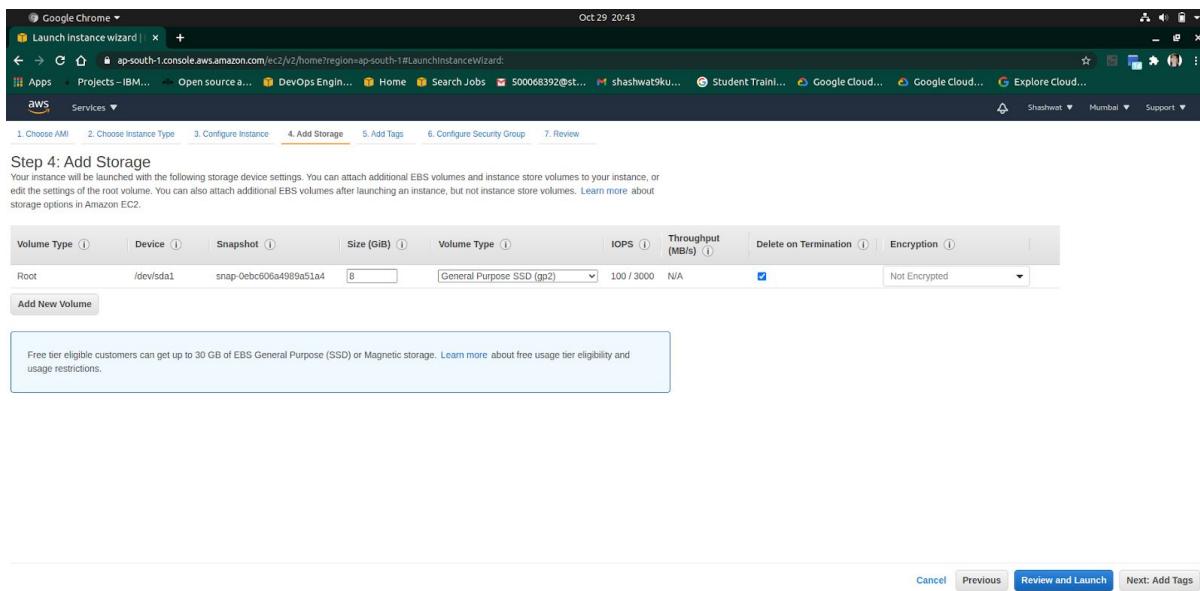
By default, you'll have a t2 micro machine selected. No need to change it.  
Click the Next: Configure Instance Details tab on the bottom right

Leave this page as default.(Though you may tweak around with some settings if you wish)  
Click on the Next: Add Storage tab on the bottom right



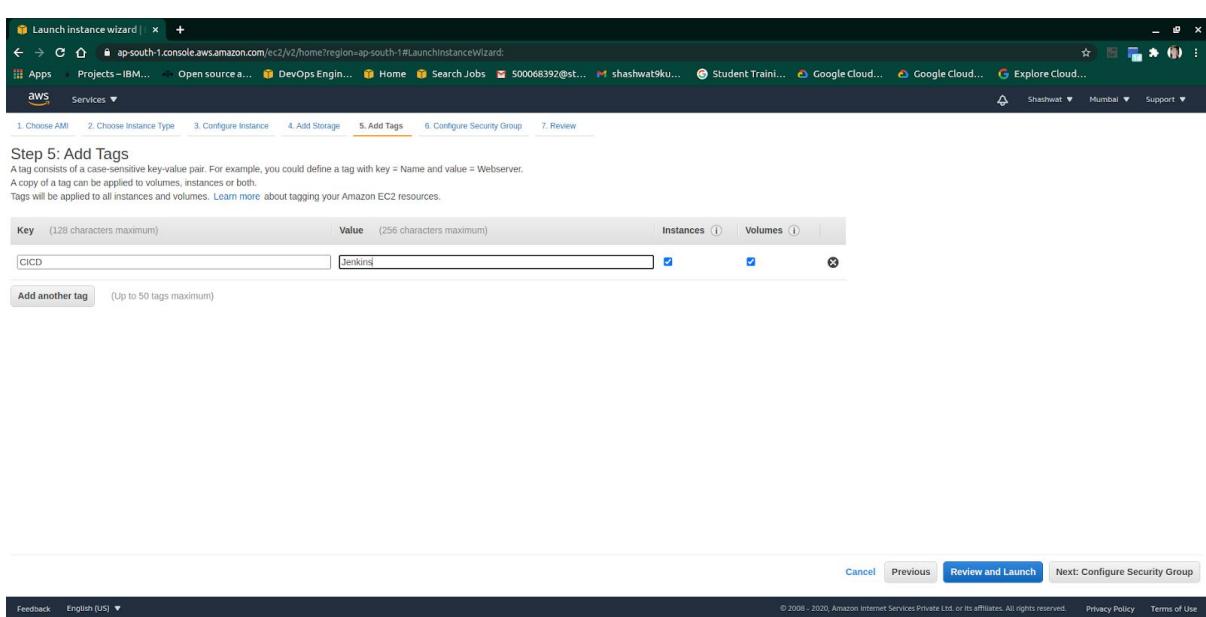
Because we are just going to use a jenkins server so we need not add any extra storage. You can leave everything as default again.

Click on Add Tags tab on the bottom right



You can add a tag as a key-value pair.

Once added, click on the Next: Configure Security Groups tab



You can give your security group a name and add a description to it as well.

Also, importantly, Jenkins runs on port 8080, so we have to open that port.

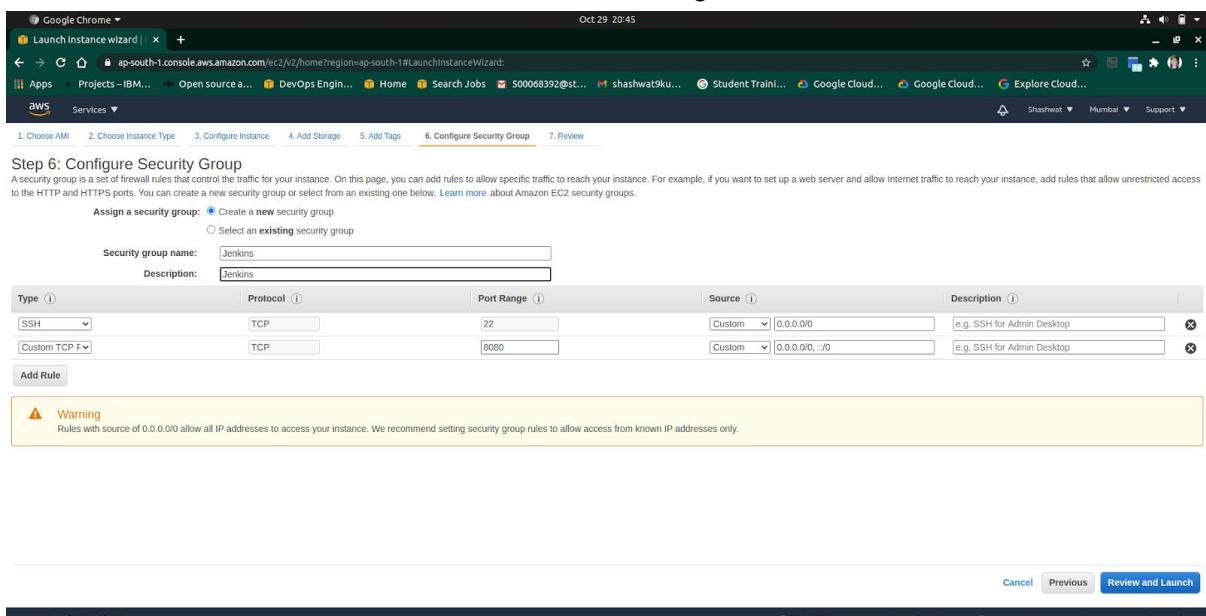
Click on the Add Rule tab

For type: select the custom TCP option

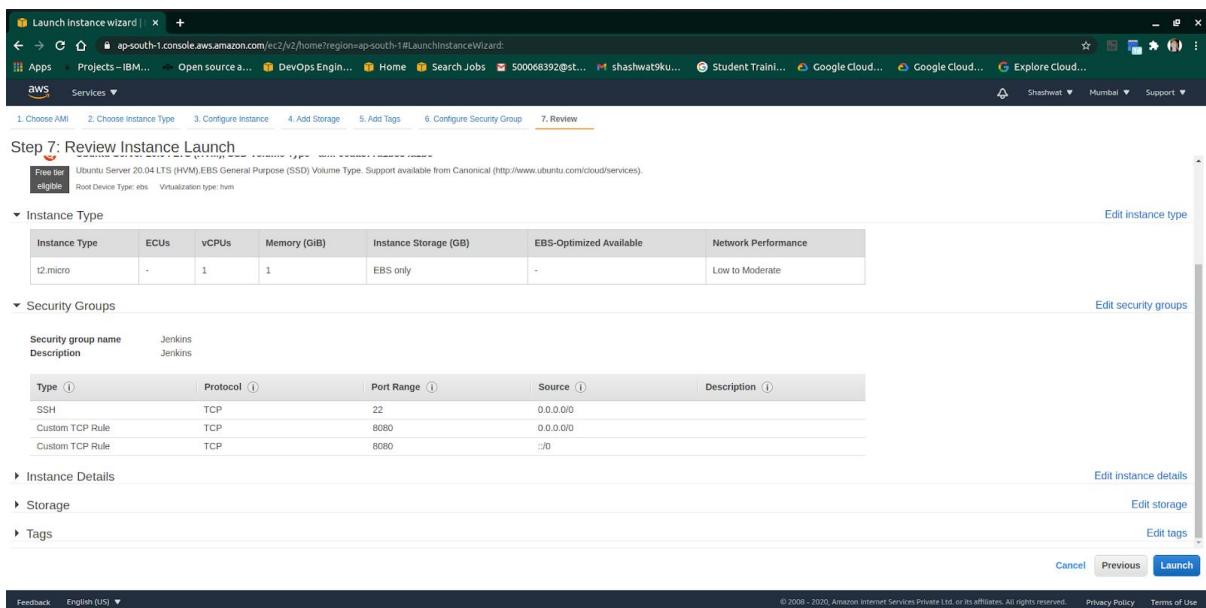
For Protocol, select TCP

For Port Range, type 8080

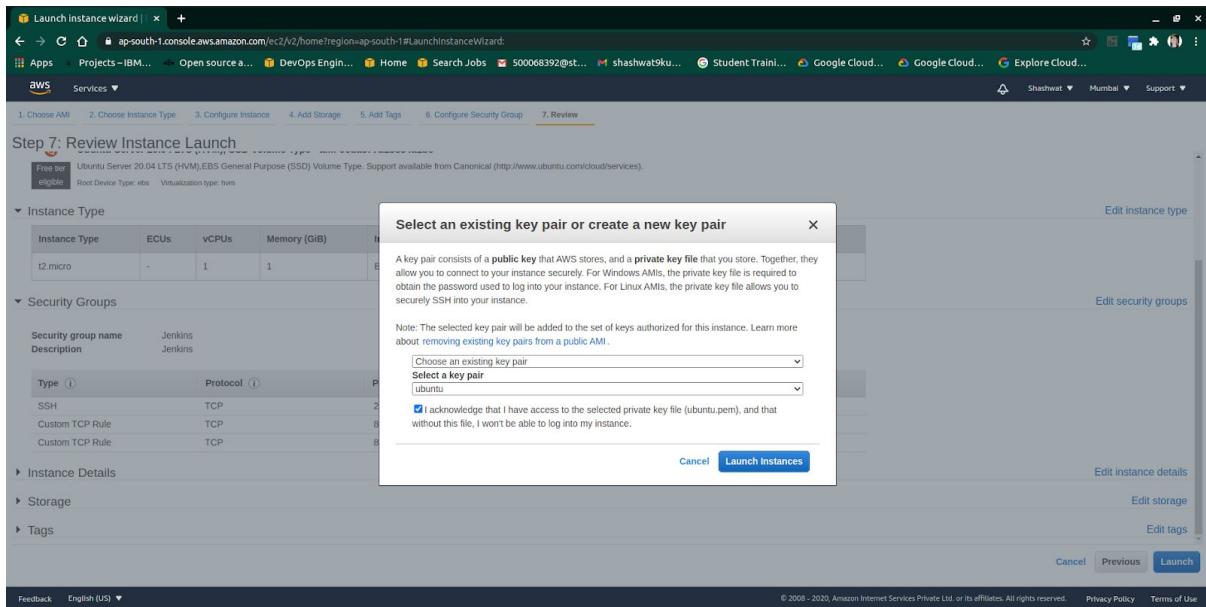
Now click on Review and Launch tab on the bottom right



Once satisfied with all the settings, click on the Launch tab on the bottom right



On the dialog box that appears, you can select an existing key pair (like done in this example), or create a new one. In either case, make sure the key pair file is downloaded and remember the download location. Click Launch Instance



The instance will take some time to launch.  
You can click the View Instance tab to see the instance

The screenshot shows the AWS Launch Status page. At the top, there's a green box with a checkmark stating "Your instances are now launching" and a link to "View launch log". Below it is a blue box with an info icon and the text "Get notified of estimated charges" followed by a description. A section titled "How to connect to your instances" follows, with a note about instances launching and a link to "View Instances". There are also helpful links for connecting to Linux instances and learning about the Free Usage Tier. A sidebar on the left lists resources like status check alarms, EBS volumes, and security groups. At the bottom right is a "View Instances" button.

If you see a green tick next to your instance state with its state as “Running”, your instance is up and running.

Congratulations, you've now launched an EC2 Ubuntu instance on AWS.

The screenshot shows the AWS EC2 Instances page. On the left is a navigation sidebar with sections like New EC2 Experience, EC2 Dashboard, Events, Tags, Limits, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, and Network & Security. The main area displays a table titled "Instances (1) Info" with one row. The row contains: Name (empty), Instance ID (i-07cbf71b7dcf956a3), Instance state (Running with a green checkmark), Instance type (t2.micro), Status check (Initializing), Alarm Status (No alarms), Availability zone (ap-south-1a), Public IPv4 DNS (ec2-52-66-25-200.ap...), and Public IPv4 IP (52.66.25.200). A "Launch Instances" button is at the top right of the table. At the bottom of the page is a footer with links to Feedback, English (US), Privacy Policy, and Terms of Use.

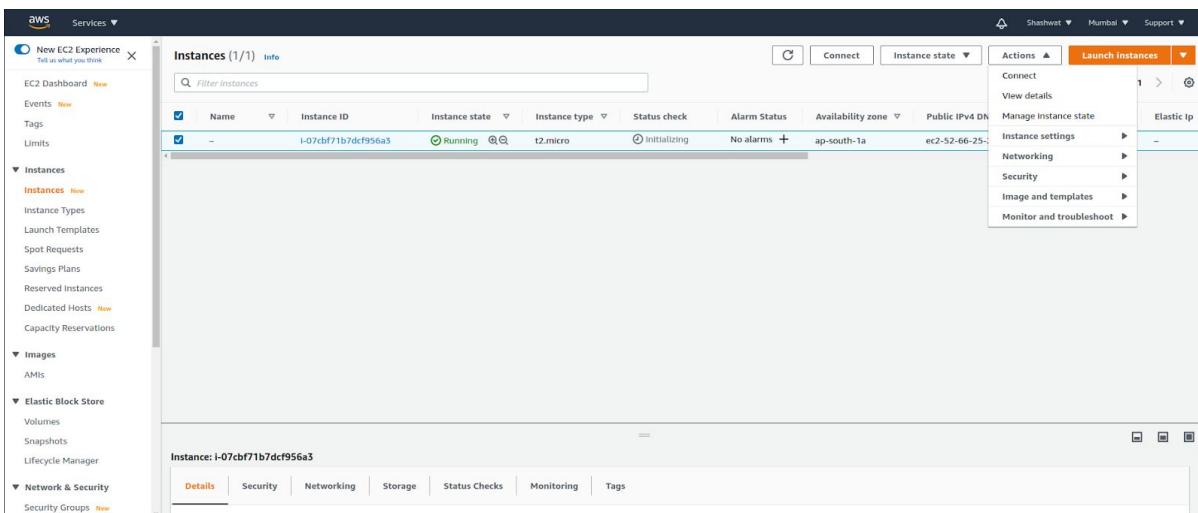
## Step 2

### Connect to the virtual cloud machine via SSH, and install required dependencies

We'll now connect to the virtual machine using the SSH manager. It would be appropriate to use a Linux machine or Machine on a virtual box. (If not, you may also install a SSH manager for Windows)

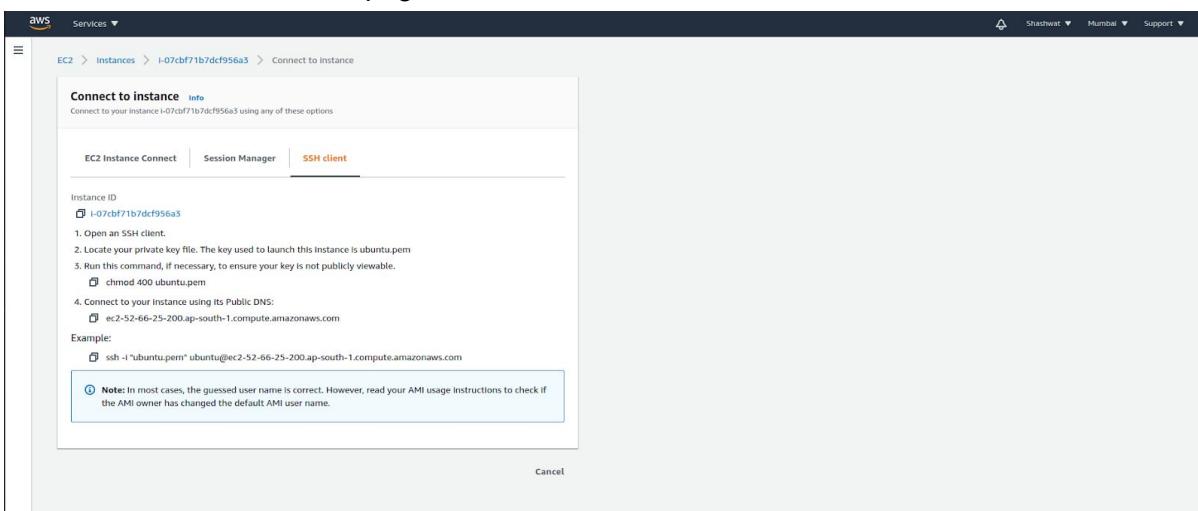
Select the running instance by checking the checkbox next to it.

Click on the Actions tab on the top right, and select Connect



The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various EC2-related options like Instances, Images, and Network & Security. The main area displays a table of instances with one row selected. The selected instance is 'i-07cbf71b7dcf956a3', which is 'Running'. The 'Actions' menu is open on the right, and the 'Connect' option is highlighted.

On the Connect to Instance page, select the SSH Client Tab



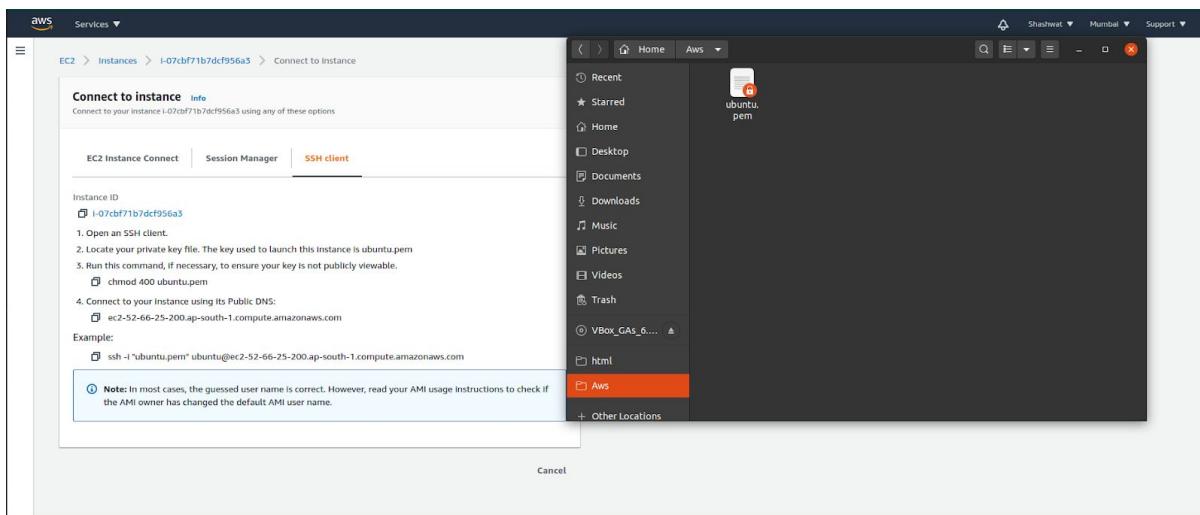
The screenshot shows the 'Connect to instance' page for instance 'i-07cbf71b7dcf956a3'. The 'SSH client' tab is selected. It provides instructions for connecting via SSH, including the instance ID and a command example:

```
ssh -i 'ubuntu.pem' ubuntu@ec2-52-66-25-200.ap-south-1.compute.amazonaws.com
```

A note at the bottom states:

Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

Open the location where you downloaded/saved the private key. Open a terminal at that file location.



Copy the command in the point 3 and run it in the terminal:

The command should be something like “chmod 400 <filename.pem>”

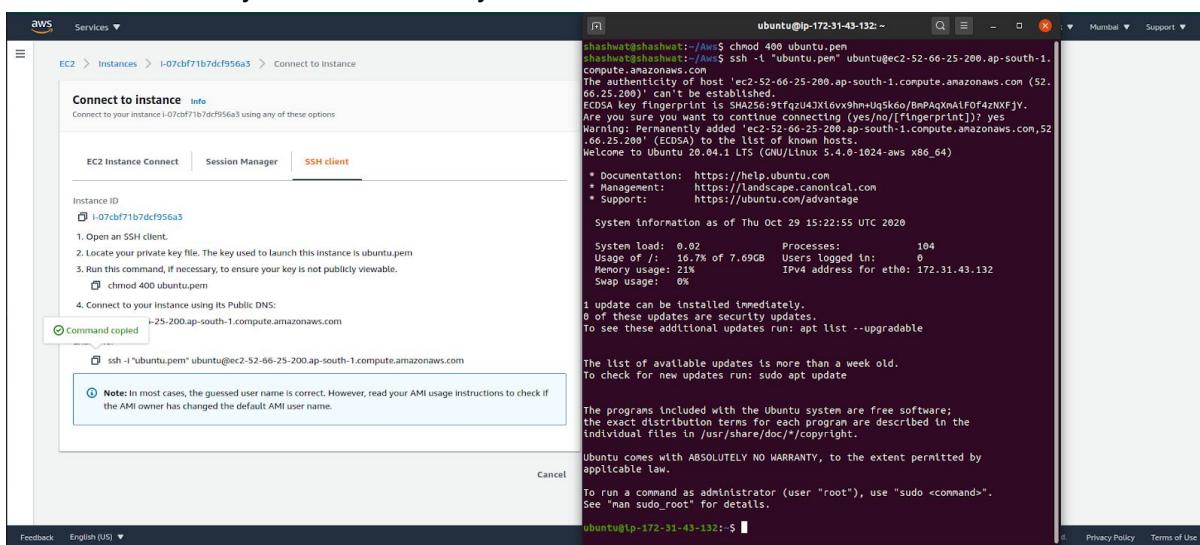
This will make the private key file accessible only to admin.

Now, copy the command under the example and run it in the terminal. Enter “Y” in the terminal when prompted.

This will help you connect remotely to the created instance.

Once done, you'll be able to see that your terminal shows `ubuntu@<ip-address>:~$`.

This confirms that you've successfully connected to the instance.



Now that you have connected to the instance, we will install the required dependencies for Jenkins.

Run the following commands in the terminal:

- `sudo apt update -y` ## to update the existing dependencies, if any

The screenshot shows the AWS EC2 Instance Connect interface. On the left, there's a navigation bar with 'Services' and a search bar. The main area shows an instance named 'I-07cbf71b7dcf956a3'. The 'SSH client' tab is selected. A terminal window is open, showing the command 'sudo apt update -y' being run. The output of the command is displayed, listing various packages and their versions from the Ubuntu focal repository.

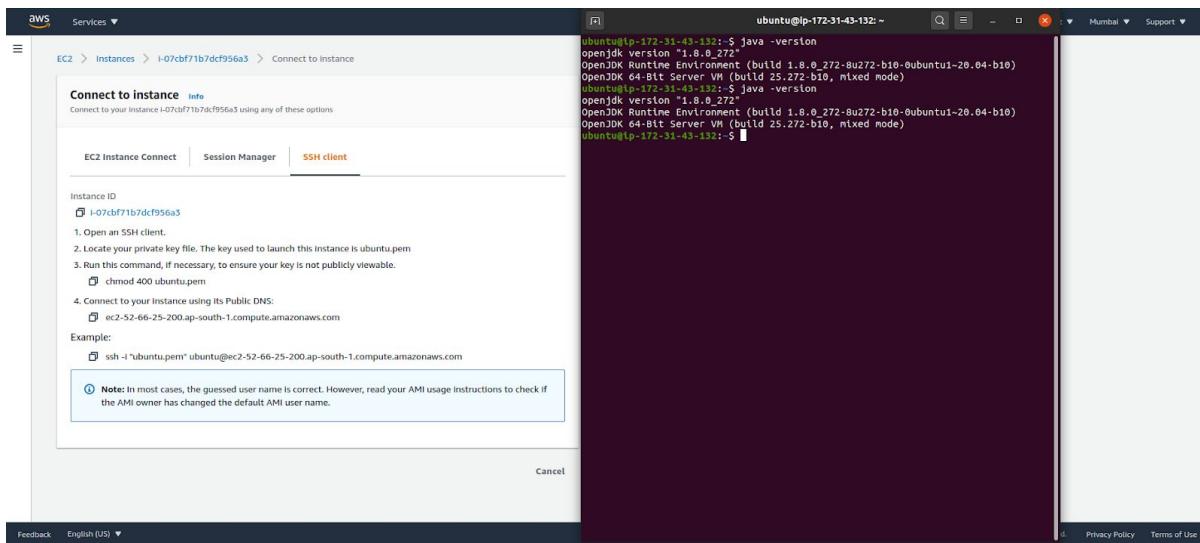
```
ubuntu@ip-172-31-43-132:~$ sudo apt update -y
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:3 https://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [1
1 kB]
Get:4 https://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
[98,3 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 Packa
ges [8028 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [356
kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [80
.0 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata
[5176 B]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages
[68 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-e
n [10,9 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages
[516 kB]
Get:12 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe Translatio
n-en [5124 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/universe Translation-en
[66,1 kB]
Get:14 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Met
adata [9204 B]
Get:15 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Package
s [516 B]
Get:16 http://security.ubuntu.com/ubuntu focal-security/multiverse Translation-e
n [540 kB]
Get:17 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 c-n-f M
etadata [116 B]
0% [5 Packages store 0 B] [12 Translation-en 1405 kB/5124 kB 27%]
```

- sudo apt install openjdk-8-jdk -y

## to install JDK 8. Jenkins uses Java to run

Confirm the java installation using the following commands:

- `java -version`
  - `javac -version`



We'll now set the JAVA\_HOME and export it into the system PATH variable.

Run the following commands:

- `export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64` ##your java installation path may be different. Check before you set the JAVA\_HOME variable

```
ubuntu@ip-172-31-43-132:~$ ls /usr/lib/jvm/
java-1.8.0-openjdk-amd64  java-8-openjdk-amd64
ubuntu@ip-172-31-43-132:~$ echo $JAVA_HOME

ubuntu@ip-172-31-43-132:~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
ubuntu@ip-172-31-43-132:~$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64
ubuntu@ip-172-31-43-132:~$
```

Now, we'll install maven

Run:

- `sudo apt install maven`

Enter "Y" when prompted to.

```

ubuntu@ip-172-31-43-132:~$ sudo apt install maven
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
libaopalliance-java libapache-pom-java libatinject-jsr330-api-java
libcdi-api-java libcommons-cli-java libcommons-io-java libcommons-lang3-java
libcommons-parent-java libgeronimo-annotation-1.3-spec-java
libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
libhawtjni-runtime-java libjansi-java libjansi-native-java libjsr305-java
libmaven-parent-java libmaven-resolver-java libmaven-shared-utils-java
libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java
libplexus-component-annotations-java libplexus-interpolation-java
libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java
libsisu-plexus-java libslf4j-java libwagon-file-java
libwagon-http-shaded-java libwagon-provider-api-java
Suggested packages:
libaopalliance-java-doc libatinject-jsr330-api-java-doc libservlet3.1-java
libcommons-io-java-doc libcommons-lang3-java-doc libasm-java libcglib-java
libjsr305-java-doc libmaven-shared-utils-java-doc liblogback-java
libplexus-cipher-java-doc libplexus-classworlds-java-doc
libplexus-sec-dispatcher-java-doc libplexus-utils2-java-doc junit4 testng
libcommons-logging-java liblog4j1.2-java
The following NEW packages will be installed:
libaopalliance-java libapache-pom-java libatinject-jsr330-api-java
libcdi-api-java libcommons-cli-java libcommons-io-java libcommons-lang3-java
libcommons-parent-java libgeronimo-annotation-1.3-spec-java
libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
libhawtjni-runtime-java libjansi-java libjansi-native-java libjsr305-java
libmaven-parent-java libmaven-resolver-java libmaven-shared-utils-java
libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java
libplexus-component-annotations-java libplexus-interpolation-java
libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java
libsisu-plexus-java libslf4j-java libwagon-file-java
libwagon-http-shaded-java libwagon-provider-api-java maven
0 upgraded, 33 newly installed, 0 to remove and 82 not upgraded.
Need to get 9209 kB of archives.
After this operation, 12.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 libapache-pom-java all 18.1 [4720 B]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 libatinject-jsr330-api-java all 1.0+ds1-5 [5348 B]

```

We set the MAVEN\_HOME variable similar to the JAVA\_HOME variable.

Run the following commands:

- MAVEN\_HOME=/usr/share/maven ## your maven installation may exist at a different location. Check before you set this variable
- PATH=\$PATH:\$MAVEN\_HOME/bin
- export MAVEN\_HOME
- export PATH

```

ubuntu@ip-172-31-43-132:~$ echo $MAVEN_HOME
ubuntu@ip-172-31-43-132:~$ echo $M2_HOME
ubuntu@ip-172-31-43-132:~$ MAVEN_HOME=/usr/share/maven
ubuntu@ip-172-31-43-132:~$ PATH=$PATH:$MAVEN_HOME/bin
ubuntu@ip-172-31-43-132:~$ export MAVEN_HOME
ubuntu@ip-172-31-43-132:~$ export PATH
ubuntu@ip-172-31-43-132:~$
```

Congratulations. You've set all the dependencies now for running JEnkis.

## Step 3

### Install and run jenkins server on a virtual cloud machine

For installation of jenkins visit the website: <https://pkg.jenkins.io/debian-stable/>  
Here you'll find the list of all the commands to install Jenkins.

The screenshot shows a web browser window with the title "Connect to instance | EC2" and the URL "Debian Jenkins Packages". The page content is as follows:

Jenkins Debian Packages

This is the Debian package repository of Jenkins to automate installation and upgrade. To use this repository, first add the key to your system:

WARNING: The gpg key use to sign our packages has been updated on 16th of April 2020, therefore you need to reimport it if you imported before this date.

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

Then add the following entry in your /etc/apt/sources.list:

```
deb https://pkg.jenkins.io/debian-stable binary/
```

Update your local package index, then finally install Jenkins:

```
sudo apt-get update
sudo apt-get install jenkins
```

The apt packages were signed using this key:

```
pub rsa4096 2020-03-30 [Sc] [expires: 2023-03-30]
62A9756BF7D80C37CFC24B86CFE32E745F2C3D5
uid Jenkins Project
sub rsa4096 2020-03-30 [E] [expires: 2023-03-30]
```

You will need to explicitly install a Java runtime environment, because Oracle's Java RPMs are incorrect and fail to register as providing a java dependency.

While still connected to the virtual instance, run the first command in the terminal.

The command:     “`wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -`”

```
ubuntu@ip-172-31-43-132:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
OK
ubuntu@ip-172-31-43-132:~$ sudo nano /etc/apt/sources.list
ubuntu@ip-172-31-43-132:~$ sudo apt-get update -y
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:2 https://pkg.jenkins.io/debian-stable binary/ Release [2044 B]
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Get:4 https://pkg.jenkins.io/debian-stable binary/ Packages [18.5 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 21.4 kB in 1s (34.8 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-43-132:~$ ls
ubuntu@ip-172-31-43-132:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon net-tools
The following NEW packages will be installed:
  daemon jenkins net-tools
0 upgraded, 3 newly installed, 0 to remove and 82 not upgraded.
Need to get 67.1 MB of archives.
After this operation, 68.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] █
```

Now run: “sudo nano /etc/apt/sources.list”

This will open an editor to edit the sources.list file

Add the following line to the end of the file:

“deb https://pkg.jenkins.io/debian-stable binary/”

Once done click CTRL+O and Enter. Click CTRL+X to exit the editor.

```

GNU nano 4.8           /etc/apt/sources.list
## review or updates from the Ubuntu security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal universe
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal universe
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates universe
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates univer>
## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal multiverse
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal multiverse
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates multiverse
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates multiv>
## N.B. software from this repository may not have been tested as
## extensively as that contained in the main release, although it includes
## newer versions of some applications which may provide useful features.
## Also, please note that software in backports WILL NOT receive any review
## or updates from the Ubuntu security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-backports main restr>
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-backports main>
## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu focal partner
# deb-src http://archive.canonical.com/ubuntu focal partner

deb http://security.ubuntu.com/ubuntu focal-security main restricted
# deb-src http://security.ubuntu.com/ubuntu focal-security main restricted
deb http://security.ubuntu.com/ubuntu focal-security universe
# deb-src http://security.ubuntu.com/ubuntu focal-security universe
deb http://security.ubuntu.com/ubuntu focal-security multiverse
# deb-src http://security.ubuntu.com/ubuntu focal-security multiverse
deb https://pkg.jenkins.io/debian-stable binary/

```

Run the following command to install jenkins:

- sudo apt-get update
- sudo apt-get install jenkins

Enter “Y” when prompted to.

```
ubuntu@ip-172-31-43-132:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
OK
ubuntu@ip-172-31-43-132:~$ sudo nano /etc/apt/sources.list
ubuntu@ip-172-31-43-132:~$ sudo apt-get update -y
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:2 https://pkg.jenkins.io/debian-stable binary/ Release [2044 B]
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Get:4 https://pkg.jenkins.io/debian-stable binary/ Packages [18.5 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 21.4 kB in 1s (34.8 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-43-132:~$ ls
ubuntu@ip-172-31-43-132:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon net-tools
The following NEW packages will be installed:
  daemon jenkins net-tools
0 upgraded, 3 newly installed, 0 to remove and 82 not upgraded.
Need to get 67.1 MB of archives.
After this operation, 68.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] █
```

Now that Jenkins is installed. We must run it.

Run the following commands:

- sudo systemctl start jenkins ## to manually start jenkins
- sudo systemctl enable jenkins ## to enable jenkins to start at system restart
- sudo systemctl status jenkins ## to check the status of jenkins- if it is running or not

If you see, an output screen similar to this, then congratulations, JEnkins server is up and running.

```

ubuntu@ip-172-31-43-132:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-43-132:~$ sudo systemctl enable jenkins
jenkins.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-43-132:~$ sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
  Loaded: loaded (/etc/init.d/jenkins; generated)
  Active: active (exited) since Thu 2020-10-29 15:48:34 UTC; 1min 8s ago
    Docs: man:systemd-sysv-generator(8)
   Tasks: 0 (limit: 1164)
  Memory: 0B
  CGroup: /system.slice/jenkins.service

Oct 29 15:48:32 ip-172-31-43-132 systemd[1]: Starting LSB: Start Jenkins at boo>
Oct 29 15:48:33 ip-172-31-43-132 jenkins[7481]: Correct java version found
Oct 29 15:48:33 ip-172-31-43-132 jenkins[7481]: * Starting Jenkins Automation >
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: (to jenkins) root on none
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: pam_unix(su-l:session): session open>
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: pam_unix(su-l:session): session clos>
Oct 29 15:48:34 ip-172-31-43-132 jenkins[7481]:     ...done.
Oct 29 15:48:34 ip-172-31-43-132 systemd[1]: Started LSB: Start Jenkins at boot>
lines 1-16/16 (END)

```

Now, it's time to visit the Jenkins dashboard.

On the AWS instance page, you'll find the public IP for your instance. Copy it.

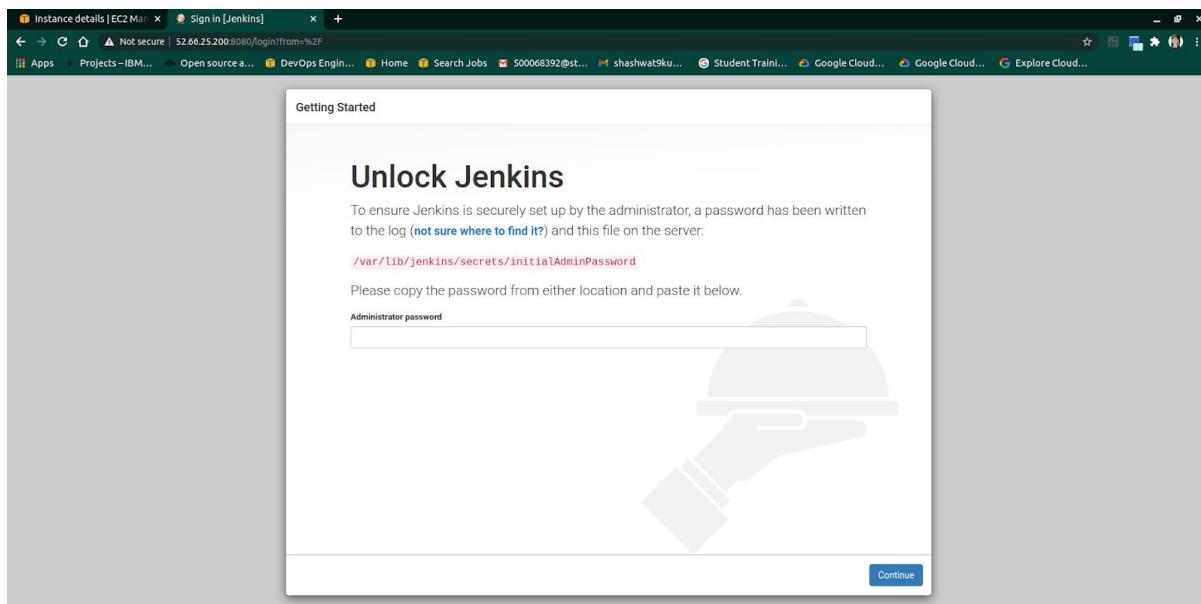
Instance summary for i-07cbf71b7dcf956a3	
Updated less than a minute ago	
Instance ID	Public IPv4 address
i-07cbf71b7dcf956a3	<a href="#">52.66.25.200</a>   open address
Instance state	Public IPv4 DNS
<span>Running</span>	<a href="#">ec2-52-66-25-200.ap-south-1.compute.amazonaws.com</a>   open address
Instance type	Elastic IP addresses
t2.micro	-
IAM Role	Subnet ID
-	<a href="#">subnet-19181771</a>

Jenkins runs on port 8080. Hence your jenkins dashboard will run on:

<http://<public-ip-of instance>:8080>

Example: <http://35.88.70.01:8080>

Once visited, you'll find Jenkins asking for the administrator password. You'll find The dashboard telling you the location of the administrator password file. Copy that location.



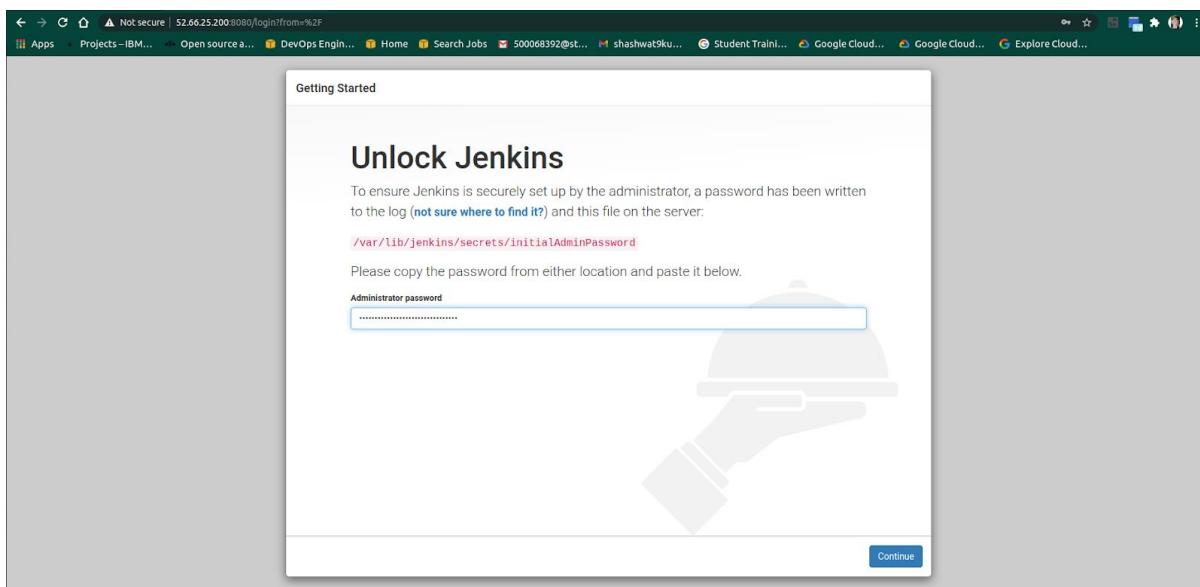
Run the following command to see the admin password:

- sudo cat /var/lib/jenkins/secrets/initialAdminPassword

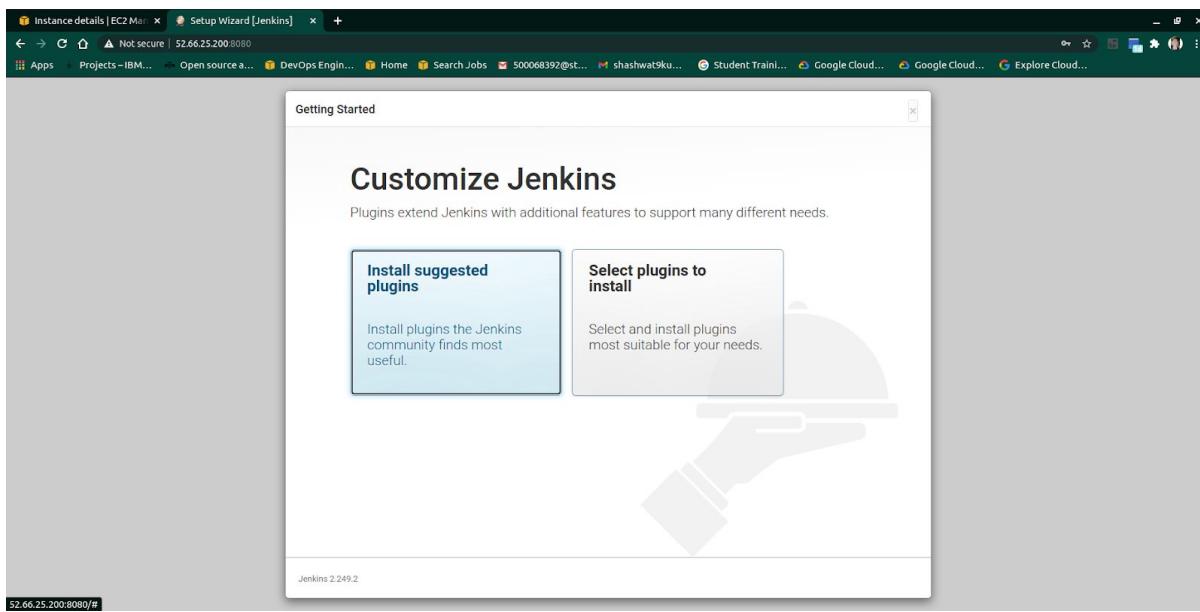
```
ubuntu@ip-172-31-43-132:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword  
b8d8cd2ad3fd4561af45b8af37596395  
ubuntu@ip-172-31-43-132:~$ █
```

Copy the output and paste it in the space given on the Jenkins dashboard.

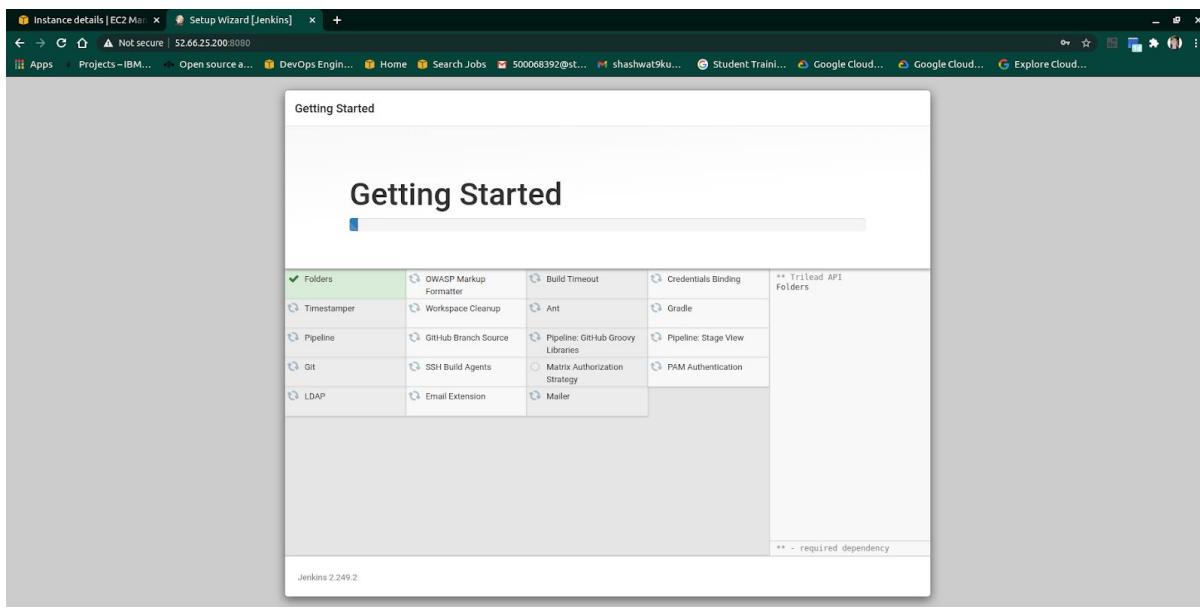
Click Continue tab



On Customize Jenkins page, select the “Install Suggested Plugins” option



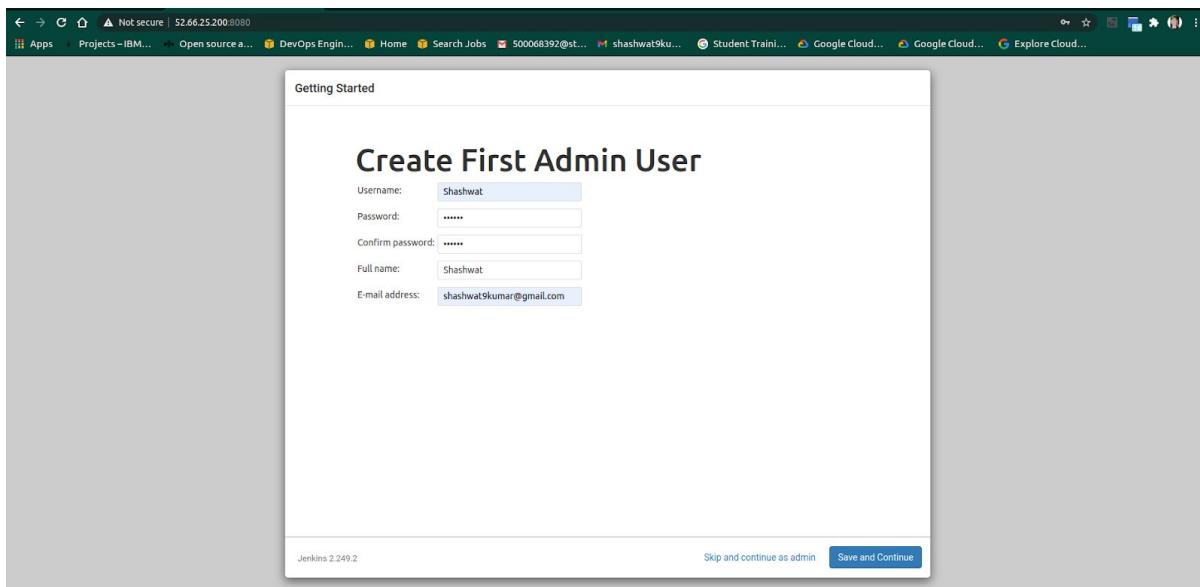
The suggested plugins take some time to get installed.



After installation, Jenkins prompts you to create an Admin user.

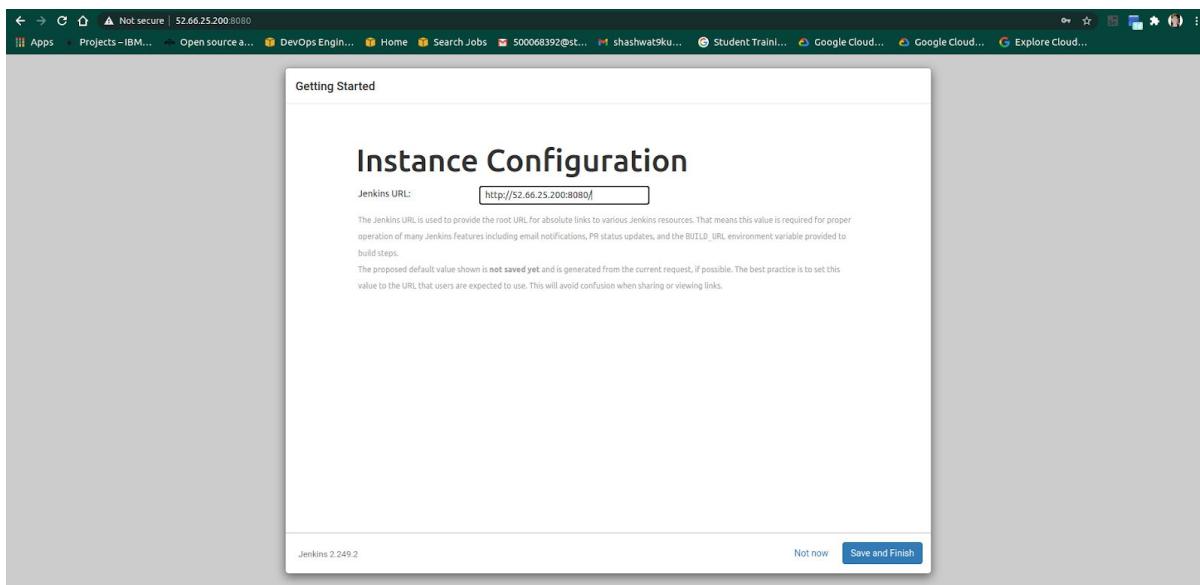
Fill in the necessary details and remember the username and password you enter.

Click on Save and Continue

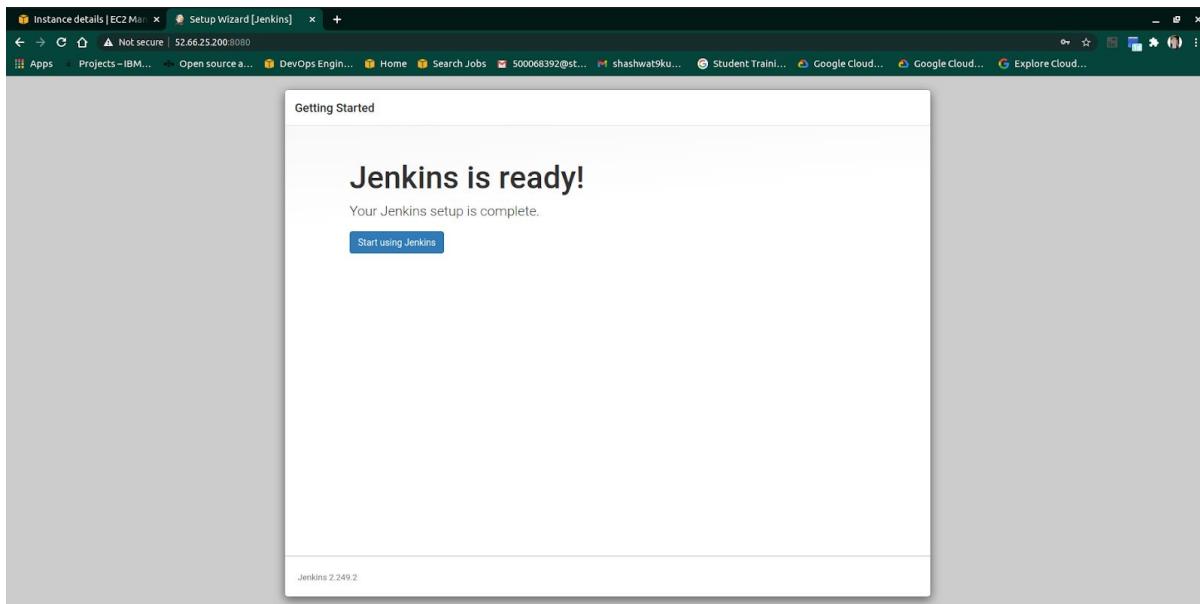


Jenkins now shows you the URL of your Jenkins server.

Click Save and Finish tab



You are now ready to use Jenkins.  
Click on Start Using Jenkins tab



You'll now see the Jenkins dashboard.

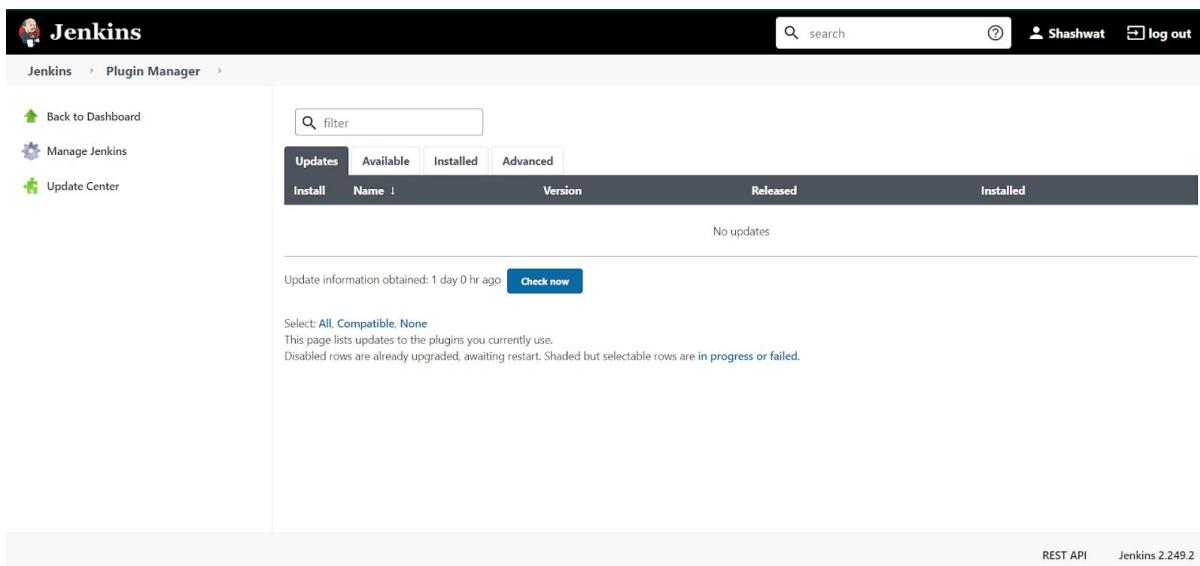
The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Lockable Resources', and 'New View'. Below these are sections for 'Build Queue' (empty) and 'Build Executor Status' (2 idle). The main area has a 'Welcome to Jenkins!' message with links to 'Create an agent' or 'configure a cloud' and 'Create a job'. At the bottom right, it says 'REST API' and 'Jenkins 2.249'.

Congratulations. You've now run your very own Jenkins Server. You can now access the Jenkins from any browser and any computer system using URL and the login details.

While still on the Jenkins Dashboard page, click on Manage Jenkins Tab on the left  
From this Page click on the Manage Plugins option

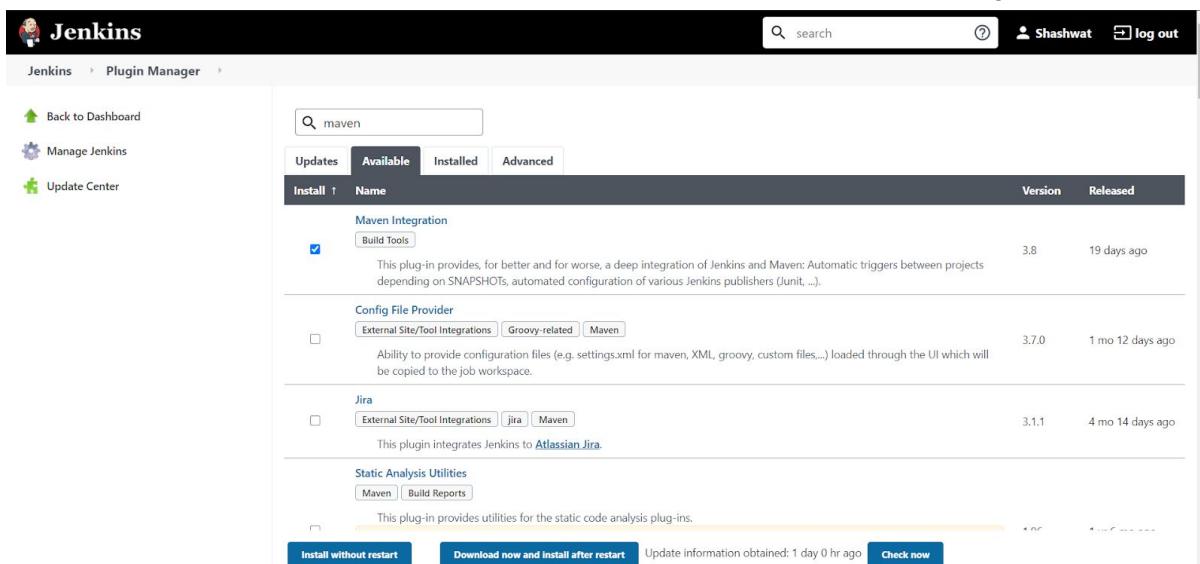
The screenshot shows the 'Manage Jenkins' page under the 'System Configuration' tab. It includes sections for 'Configure System', 'Global Tool Configuration', 'Manage Nodes and Clouds', and 'Manage Plugins'. Under 'Security', there are links for 'Configure Global Security', 'Manage Credentials', and 'Configure Credential Providers'. At the bottom, there's a 'Status Information' section with links for 'System Information', 'System Log', and 'Load Statistics'.

You'll reach the install plugins page. Click on the Available tab



The screenshot shows the Jenkins Plugin Manager interface. At the top, there's a navigation bar with links to 'Jenkins', 'Plugin Manager', 'Manage Jenkins', and 'Update Center'. On the right side of the header are 'Shashwat' and 'log out' buttons. Below the header is a search bar with a magnifying glass icon and a placeholder 'search'. The main content area has a title 'Available' with tabs for 'Updates', 'Available', 'Installed', and 'Advanced'. Underneath are filters for 'Install', 'Name', 'Version', 'Released', and 'Installed'. A message 'No updates' is displayed. Below this, a note says 'Update information obtained: 1 day 0 hr ago' with a 'Check now' button. A help note below the search bar reads: 'Select: All, Compatible, None. This page lists updates to the plugins you currently use. Disabled rows are already upgraded, awaiting restart. Shaded but selectable rows are in progress or failed.' At the bottom right, there are links for 'REST API' and 'Jenkins 2.249.2'.

Search for the Maven Integration plugin, Git plugin and Blue Ocean plugin(optional).  
Click on the Download and Install after restart option on the bottom of the page



The screenshot shows the Jenkins Plugin Manager interface with a search term 'maven' entered in the search bar. The 'Available' tab is selected. The results list four plugins:

- Maven Integration** (Build Tools): This plugin 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, ...). Version 3.8, released 19 days ago.
- Config File Provider** (External Site/Tool Integrations, Groovy-related, Maven): Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files...) loaded through the UI which will be copied to the job workspace. Version 3.7.0, released 1 mo 12 days ago.
- Jira** (External Site/Tool Integrations, jira, Maven): This plugin integrates Jenkins to Atlassian Jira. Version 3.1.1, released 4 mo 14 days ago.
- Static Analysis Utilities** (Maven, Build Reports): This plugin provides utilities for the static code analysis plug-ins. Version 1.1.1, released 1 mo 12 days ago.

At the bottom of the page, there are buttons for 'Install without restart', 'Download now and install after restart', and 'Check now'. A note says 'Update information obtained: 1 day 0 hr ago'.

Your plugins will get installed. It will take some time to do so. You can also check the Restart jenkins when installation is complete option

Jenkins > Update Center

Plugin	Status
Pipeline: GitHub Groovy Libraries	Success
Pipeline: Stage View	Success
Git	Success
SSH Build Agents	Success
Matrix Authorization Strategy	Success
PAM Authentication	Success
LDAP	Success
Email Extension	Success
Mailer	Success
Loading plugin extensions	Success
Javadoc	Pending
Maven Integration	Pending
H2 API	Pending
Config File Provider	Pending
Pipeline Maven Integration	Pending
Loading plugin extensions	Pending

 [Go back to the top page](#)  
 (you can start using the installed plugins right away)

  Restart Jenkins when installation is complete and no jobs are running

REST API Jenkins 2.249.2

---

When your plugins get installed, Jenkins will restart itself.



Please wait while Jenkins is getting ready to work ...

Your browser will reload automatically when Jenkins is ready.

Waiting for 52.66.25.200...

After Jenkins starts again, Click Manage Jenkins Option and select the Global Tool Configuration option

In the JDK section, name the JDK with a name you wish, and either check the “install automatically” or give the path to the installed jdk on the machine. The JAVA\_HOME path you stored in the previous step should be provided here.

In the MAVEN section, name the MAVEN with a name you wish, and either check the “install automatically” or give the path to the installed maven on the machine. The MAVEN\_HOME path you stored in the previous step should be provided here.

Once done, Click Save option

## Step 4

### Activate github webhooks for automated build triggering

Go to the Maven project on Github for which you want to set up the automated CI pipeline. (If you do not have an existing maven project you may push a demo maven project a github repository to follow along)

Click on the Setting tab

The screenshot shows a GitHub repository page for 'shashwat9kumar / Automated-CI-Maven-project-with\_Jenkins-and-Github'. The main area displays a commit history with several commits from 'shashwat9kumar' pushing cleaned project files. Below the commits is a README.md file containing the text 'Automated CI Project for Maven built Using Jenkins Server'. To the right, there is a sidebar with sections for 'About', 'Releases', 'Packages', and 'Languages'. The 'Languages' section shows Java at 100.0%. The 'Settings' tab is highlighted in red at the top of the sidebar.

Now, click on the Webhooks option on the left panel.

Click the Add webhook button

The screenshot shows the 'Settings' page for the same GitHub repository. The left sidebar has a 'Webhooks' option selected, indicated by a red border. The main content area is titled 'Webhooks' and contains a brief description of what webhooks are and how they work. At the top right of this section is a 'Add webhook' button.

In the payload url option, paste the jenkins server url and append "/github-webhook/" to it.

Example: <http://23.44.55.66:8080/github-webhook/>

In the Content Type select the application/JSON option  
 Check the “Just the push event”  
 Also check the “Active”  
 Now click on Add webhook button

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

**Payload URL \***

http://52.66.25.200:8080/github-webhook/

**Content type**

application/json

**Secret**

**Which events would you like to trigger this webhook?**

Just the push event.  
 Send me everything.  
 Let me select individual events.

Active

We will deliver event details when this hook is triggered.

**Add webhook**

Congratulations, you've enabled a webhook. Now github will inform Jenkins server whenever a push even occurs.

Okay, that hook was successfully created. We sent a ping payload to test it out! Read more about it at <https://docs.github.com/webhooks/#ping-event>.

shashwat9kumar / Automated-CI-Maven-project-with\_Jenkins-and-Github

**Code Issues Pull requests Actions Projects Wiki Security Insights Settings**

**Webhooks**

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

http://52.66.25.200:8080/github-... (push)

**Add webhook**

**Edit Delete**

Additionally, for more automation, we define the “Jenkinsfile” in the github project.  
 Create a new Jenkinsfile in the github project if you don't have one already.

(The Jenkinsfile used in this example is as follows:

```
pipeline{
    agent any
    stages{
        stage('Start'){
            steps{
                echo 'Start the Pipeline'
            }
        }

        stage('Clean phase Starts'){
            steps{
                echo 'Start the Clean phase'
            }
        }
        stage('Clean'){
            steps{
                sh 'mvn clean'
            }
        }

        stage('Compile phase Starts'){
            steps{
                echo 'Start the Compile phase'
            }
        }
        stage('Compile'){
            steps{
                sh 'mvn compile'
            }
        }

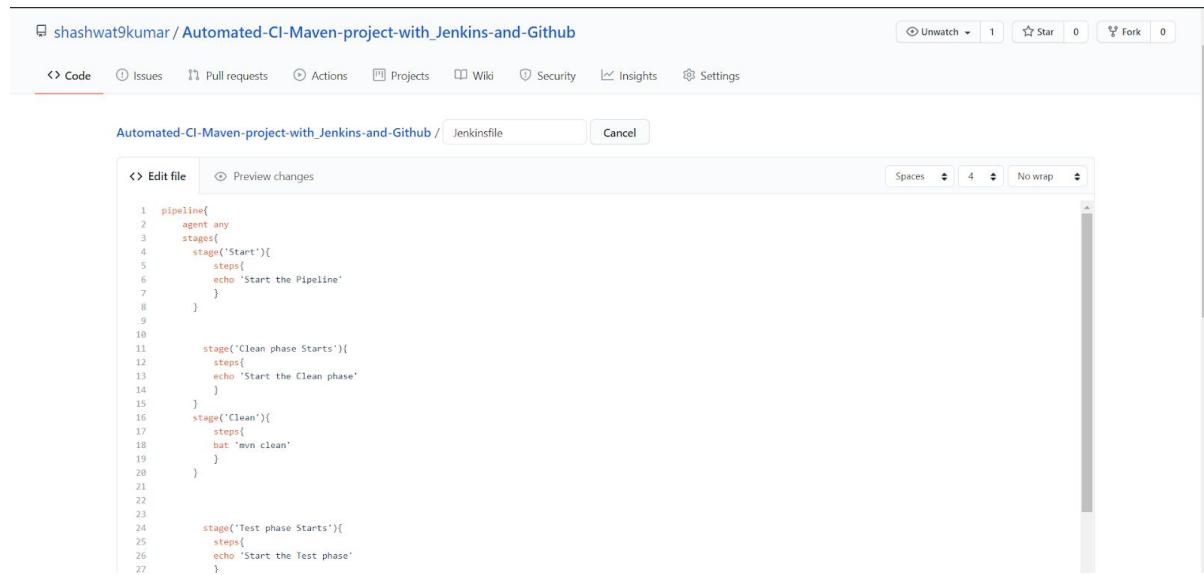
        stage('Test phase Starts'){
            steps{
                echo 'Start the Test phase'
            }
        }
        stage('Test'){
            steps{
                sh 'mvn test'
            }
        }

        stage('Install phase Starts'){
            steps{
                echo 'Start the Install phase'
            }
        }
        stage('Install'){
            steps{
                sh 'mvn install'
            }
        }

        stage('End'){
            steps{
                echo 'End the pipeline'
            }
        }
    }
}
```

Automated CI Pipeline Configuration

)



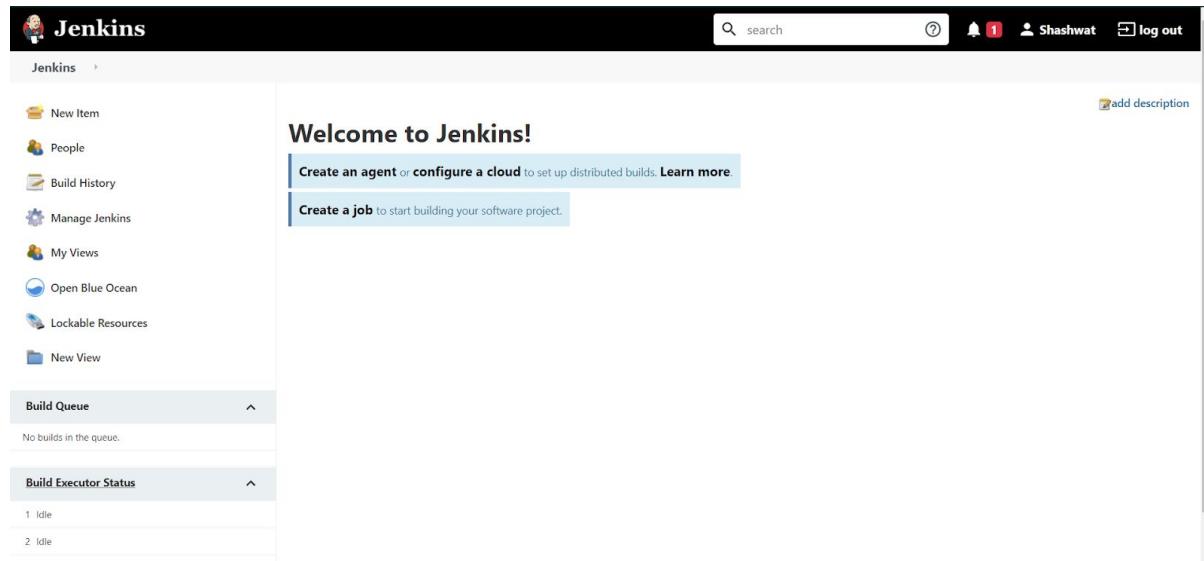
```
1 pipeline{
2   agent any
3   stages{
4     stage('Start'){
5       steps{
6         echo 'Start the Pipeline'
7       }
8     }
9   }
10  stage('Clean phase Starts'){
11    steps{
12      echo 'Start the Clean phase'
13    }
14  }
15  stage('Clean'){
16    steps{
17      bat 'mvn clean'
18    }
19  }
20}
21
22
23
24 stage('Test phase Starts'){
25   steps{
26     echo 'Start the Test phase'
27   }
28 }
```

## Step 5

### Creating the automated pipeline for Continuous Integration

Return to the Jenkins Dashboard.

Click on New Item option to create a new Jenkins pipeline



Welcome to Jenkins!

Create an agent or configure a cloud to set up distributed builds. [Learn more](#).

[Create a job](#) to start building your software project.

New Item

People

Build History

Manage Jenkins

My Views

Open Blue Ocean

Lockable Resources

New View

Build Queue

No builds in the queue.

Build Executor Status

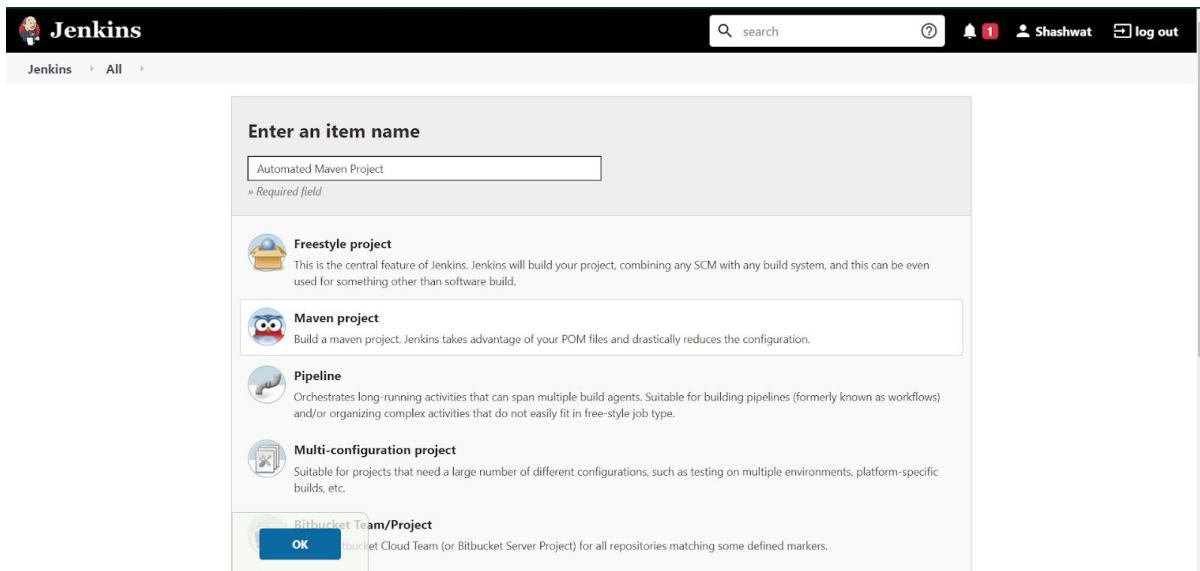
1 Idle

2 Idle

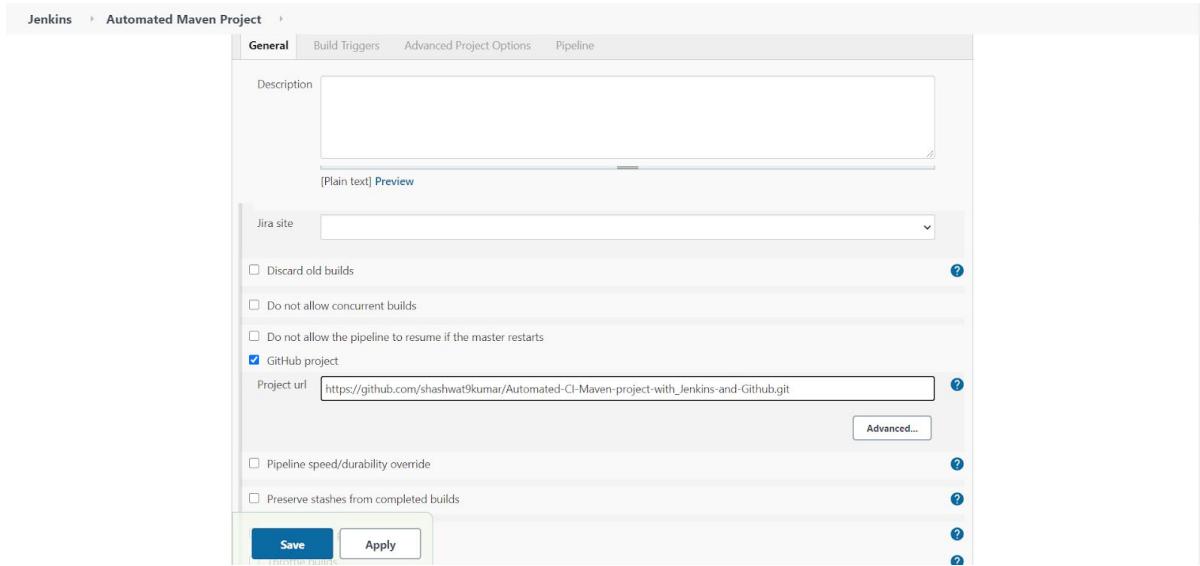
Here on this page, name the project with a name you find suitable.

Select the project type as “Pipeline”

Click OK tab



In the General Section, check the Github Project option and give the location to the project repository



In the Build Triggers section, select the GitHub hook trigger for GITScm polling option

Jenkins > Automated Maven Project >

General Build Triggers Advanced Project Options Pipeline

Throttle builds

Build Triggers

- Build after other projects are built
- Build periodically
- Build whenever a SNAPSHOT dependency is built
- GitHub Branches
- GitHub Pull Requests
- GitHub hook trigger for GITScm polling
- Maven Dependency Update Trigger
- Poll SCM
- Disable this project
- Quiet period
- Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Save Apply Advanced...

In the Pipeline section, select Pipeline Script from SCM option in the Definition box

Add SCM as Git

And give the repository link to the github repository

Click the **Save** button at the bottom

Jenkins > Automated Maven Project >

General Build Triggers Advanced Project Options Pipeline

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL: https://github.com/shashwat9kumar/

Credentials - none - Add

Advanced... Add Repository

Branches to build

Branch Specifier (blank for 'any') \*/master

Add Branch

Repository browser (Auto)

Additional Behaviours Add

Save Apply

Once done, you'll reach a page similar to this.

The screenshot shows the Jenkins web interface for a 'Maven project Automated Maven Project'. The left sidebar contains a list of actions: Back to Dashboard, Status, Changes, Workspace, Build Now, Configure, Delete Maven project, Modules, Favorite, GitHub Hook Log, Open Blue Ocean, and Rename. The main content area displays the project's workspace and recent changes. At the top right, there are links for adding a description, disabling the project, and logging out. The user 'Shashwat' is logged in.

Congratulations. You've now created an automated CI pipeline that triggers the Jenkins server whenever you push on github.

## Step 6

### Testing the pipeline via a test push/commit

And now that an automated pipeline is built, the only thing that's left to do is test it.

Open a terminal or Command line (depending upon your OS)

Move to the project repository on your local machine

Run the following commands:

- git pull origin master ## best practice is to pull before you push
- <Make some changes to the project, however small. Eg: change the ReadMe.md file>
- git add .
- git commit -m "Commit message"
- git push -u origin master

You'll see your changes pushed to the remote repository on Github.

```
D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 843 bytes | 6.00 KiB/s, done.
From https://github.com/shashwat9kumar/Automated-CI-Maven-project-with_Jenkins-and-Github
 * branch            master      -> FETCH_HEAD
   a31e302..959e68b  master      -> origin/master
Updating a31e302..959e68b
Fast-forward
 Jenkinsfile | 55 ++++++
 1 file changed, 55 insertions(+)
 create mode 100644 Jenkinsfile

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git add .

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git commit -m "Test Commit"
[master 12e314c] Test Commit
 1 file changed, 1 insertion(+), 1 deletion(-)

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/shashwat9kumar/Automated-CI-Maven-project-with_Jenkins-and-Github.git
 959e68b..12e314c  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>
```

Additionally see your pipeline on Jenkins server. It starts building too.  
Click on your project on the Jenkins Dashboard.

The screenshot shows the Jenkins dashboard with the 'Automated Maven Project' job selected. The left sidebar contains links like 'New Item', 'People', 'Build History', etc. The main area displays the project's status: 'Last Success' (3 min 2 sec - #4), 'Last Failure' (N/A), 'Last Duration' (23 sec), and a 'Fav' button. Below this is a legend and three Atom feed options. The 'Recent Changes' section is empty. The 'Stage View' section shows the pipeline stages: Declarative: Checkout SCM, Start, Clean phase Starts, Clean, Compile phase Starts, Compile, Test phase Starts, Test, Install phase Starts, Install, and End. The 'Recent Changes' section shows a single commit from Oct 31 at 15:29.

On clicking, you'll find your project has started building on its own. This was because github notifies Jenkins of the push event that occurred on it.

This screenshot shows the 'Pipeline Automated Maven Project' view. The left sidebar includes links for 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'GitHub', 'Open Blue Ocean', 'Rename', 'Pipeline Syntax', 'GitHub Hook Log', 'Build History', and 'trend'. The main area features a 'Stage View' chart with three stages: #1, #2, and #3. Stage #1 has an average stage time of 18s. Stage #2 has an average stage time of 1min 1s. Stage #3 has an average stage time of 45ms. The chart details the duration of each stage component: Start, Clean phase Starts, Clean, Compile phase Starts, Compile, Test phase Starts, Test, Install phase Starts, Install, and End.

The status of the build also mentions that



[Started by GitHub push by shashwat9kumar](#)



**Revision:** 157275df34c757aae44c87bdff876b22b76a22a8

- refs/remotes/origin/master

If the project has no errors all the stages complete successfully. And the pipeline runs successfully.

The screenshot shows the Jenkins Pipeline Automated Maven Project stage view. The left sidebar includes links for Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, GitHub, Open Blue Ocean, Rename, Pipeline Syntax, and GitHub Hook Log. The main area displays a table of stages with their average times. The stages listed are Declarative: Checkout SCM, Start, Clean phase Starts, Clean, Compile phase Starts, Compile, Test phase Starts, Test, Install phase Starts, Install, and End. The table shows three successful builds with the following approximate average times:

Stage	Average Time
Declarative: Checkout SCM	1s
Start	216ms
Clean phase Starts	105ms
Clean	5s
Compile phase Starts	79ms
Compile	7s
Test phase Starts	81ms
Test	6s
Install phase Starts	62ms
Install	5s
End	60ms

Below the table, there are three rows of build details:

- Build #3: Oct 31 14:59, 1 commit
- Build #2: Oct 31 12:51, 1 commit
- Build #1: Oct 31 12:50, No Changes

At the bottom, there is a "Permalinks" section.

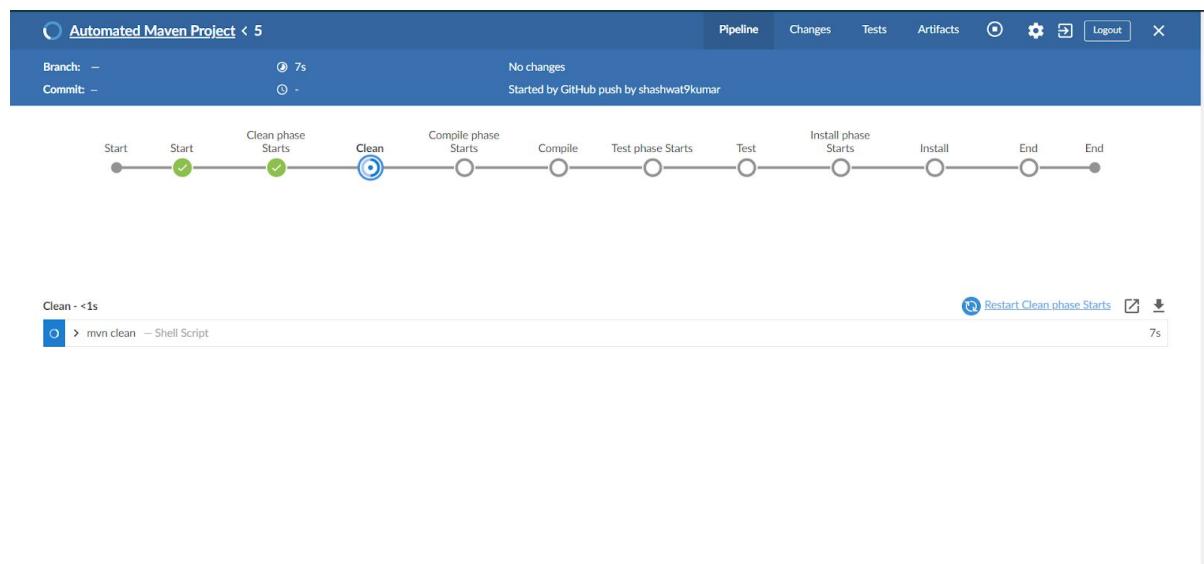
Also the console output shows BUILD SUCCESS

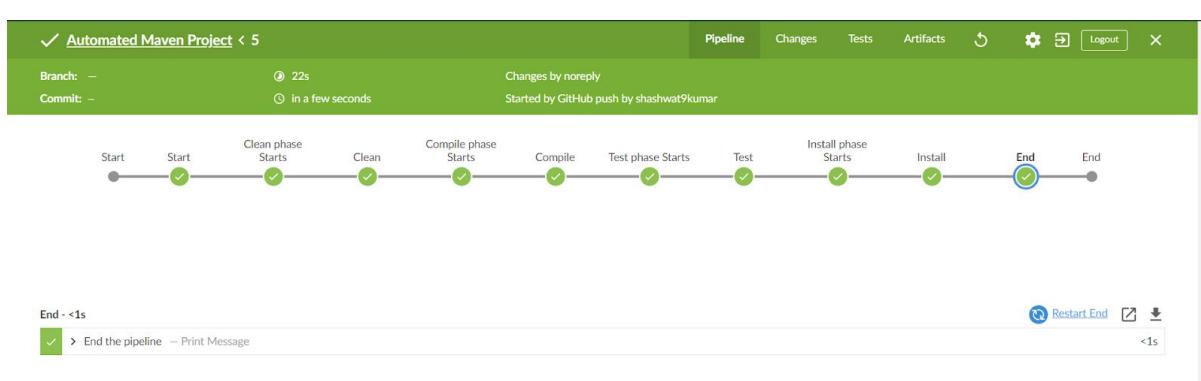
Jenkins > Automated Maven Project > #4

```
[0;1;34mINFO[0m] [1m-- 0[0;32mmaven-compiler-plugin:3.1:testCompile[m @ [1m(default-testCompile)[m @ [36mCI_with_Jenkins@0;1m --[m
[0;1;34mINFO[0m] No sources to compile
[0;1;34mINFO[0m]
[0;1;34mINFO[0m] [1m-- 0[0;32mmaven-surefire-plugin:2.12.4:test[m @ [1m(default-test)[m @ [36mCI_with_Jenkins@0;1m --[m
[0;1;34mINFO[0m] No tests to run.
[0;1;34mINFO[0m]
[0;1;34mINFO[0m] [1m-- 0[0;32mmaven-jar-plugin:2.4:jar[m @ [1m(default-jar)[m @ [36mCI_with_Jenkins@0;1m --[m
[0;1;34mINFO[0m] Building jar: /var/lib/jenkins/workspace/Automated Maven Project/target/CI_with_Jenkins-0.0.1-SNAPSHOT.jar
[0;1;34mINFO[0m]
[0;1;34mINFO[0m] [1m-- 0[0;32mmaven-install-plugin:2.4:install[m @ [1m(default-install)[m @ [36mCI_with_Jenkins@0;1m --[m
[0;1;34mINFO[0m] Installing /var/lib/jenkins/workspace/Automated Maven Project/target/CI_with_Jenkins-0.0.1-SNAPSHOT.jar to
/var/lib/jenkins/.m2/repository/CI-CD/CI_with_Jenkins/0.0.1-SNAPSHOT/CI_with_Jenkins-0.0.1-SNAPSHOT.jar
[0;1;34mINFO[0m] Installing /var/lib/jenkins/workspace/Automated Maven Project/pom.xml to /var/lib/jenkins/.m2/repository/CI-CD/CI_with_Jenkins/0.0.1-
SNAPSHOT/CI_with_Jenkins-0.0.1-SNAPSHOT.pom
[0;1;34mINFO[0m] @[1m-----[m
[0;1;34mINFO[0m] @[1;32mBUILD SUCCESS[m
[0;1;34mINFO[0m] @[1m-----[m
[0;1;34mINFO[0m] Total time: 2.317 s
[0;1;34mINFO[0m] Finished at: 2020-10-31T09:59:37Z
[0;1;34mINFO[0m] @[1m-----[m
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
[Pipeline] { (End)
[Pipeline] echo
[Pipeline] End the pipeline
[Pipeline] }
[Pipeline] // stage
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

For a more visual pipeline and better representation, you may additionally click on the Blue Ocean option on the left panel (if you installed the plugin before)

Blue Ocean provides a better visual of the rather mundane CI pipeline.





Congratulations. Now you've created, enabled and tested an automated CI pipeline.