**Automating Infrastructure using Terraform**

**DESCRIPTION**

Use Terraform to provision infrastructure

Description:

Nowadays, infrastructure automation is critical. We tend to put the most emphasis on software development processes, but infrastructure deployment strategy is just as important. Infrastructure automation not only aids disaster recovery, but it also facilitates testing and development.

Your organization is adopting the DevOps methodology and in order to automate provisioning of infrastructure there's a need to setup a centralized server for Jenkins.

Terraform is a tool that allows you to provision various infrastructure components. Ansible is a platform for managing configurations and deploying applications. It means you'll use Terraform to build a virtual machine, for example, and then use Ansible to install the necessary applications on that machine.

Considering the Organizational requirement you are asked to automate the infrastructure using

Terraform first and install other required automation tools in it.

**Tools required:** Terraform, AWS account with security credentials, Keypair

**Expected Deliverables:**

• Launch an EC2 instance using Terraform

• Connect to the instance

• Install Jenkins, Java and Python in the instance

**Project By:-Chaitra Boregowda**

**INDEX**

1. **Introduction**
2. **Installation of pre-requisites**
3. **Install terraform**
4. **Create IAM user on AWS environment**
5. **Terraform**
6. **Credentials file creation**
7. **Create PEM file**
8. **Create TF File**
9. **Starting the case with terraform**
10. **Configure EC2 environment**

***Java Run Time***

***Jenkins***

***Python***

**INTRODUCTION**

***This document provides a comprehensive guide to setting up and deploying infrastructure on AWS using Terraform. The process includes creating IAM users, configuring credentials, and deploying an EC2 environment with necessary software like Java, Jenkins, and Python. .For this project,I am working in an Ubuntu distribution and using a Chrome web browser.***

**Setting up Prerequisites**

Before starting, ensure the following software is installed on your system:

* Git
* AWS CLI
* Terraform
* A text editor (e.g., VS Code)

**INSTALLATION OF TERRAFORM**

Terraform is an open-source infrastructure-as-code software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON.

**STEP 1: Update the Machine**

Update your system with the below command:

**sudo apt update -y**

**STEP 2: Open Terminal and Install Terraform Package**

Ensure that your system is up to date, and you have the gnupg, software-properties-common, and curl packages installed. You will use these packages to verify HashiCorp's GPG signature and install HashiCorp's Debian package repository.

Use the below command:

**wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg**

**echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]**

**https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list**

Now install Terraform with the following command:

**sudo apt install -y terraform**

**STEP 3: Verify the Installation of Terraform**

Use the below command to verify the installation of Terraform:

**terraform --version**

**Setting Up AWS**

**STEP 1: Login to the AWS Account Using Credentials**

1. Open your web browser and go to the [AWS Management Console](https://aws.amazon.com/console/).
2. Click on **Sign in to the Console**.
   * If you do not see the option to create a new AWS account, first choose **Sign in to a different account**, and then click on **Create a new AWS account**.
3. Enter your AWS account credentials (email and password) to log in.

**STEP 2: Create an IAM User**

Once logged in, navigate to the **IAM** (Identity and Access Management) service.

In the IAM dashboard, click on **Users** in the left navigation pane.

Click on **Add user**.

Provide a username for the new user.

Under **Select AWS access type**, choose **Programmatic access** to enable access via the AWS CLI, SDKs, or APIs.

Click **Next: Permissions**.

**Terraform: Credentials File Creation**

To enable Terraform to authenticate with AWS, you need to create a credentials file in your .aws directory. This file will store your AWS access keys.

Let's now create a file called credentials in your /home/.aws and add the keys that were created on AWS Console. To do this, execute:

**Create the Credentials File Using vi**:

Open a terminal and use the following command to create and edit the credentials file:

**sudo vi ~/.aws/credentials**

**Add Your AWS Credentials to the File**:

**[default]**

**aws\_access\_key\_id = YOUR\_ACCESS\_KEY**

**aws\_secret\_access\_key = YOUR\_SECRET\_KEY**

After that press ESC, :wq

**Create PEM file**

Later we are going to associate both public and private keys with AWS EC2 instances. Let us generate the key pair using the following command:

**sudo ssh-keygen -t rsa -b 2048**

By default, the above command will generate the public as well as private key at location **/root/.ssh/id\_rsa**, but we override the destination with a custom **/home/chaitra/keys/demo/key-for-demo-3**

**sudo ssh-keygen -t rsa -b 2048 -f ~/home/chaitra/keys/demo/key-for-demo3**

This will generate two files:

~**/home/chaitra/keys/demo/key-for-demo-3** :- The private key file

~**/home/chaitra/keys/demo/key-for-demo-3.pub** :- The public key file

**Create TF File**

To create an EC2 machine using Terraform, we need to define a Terraform configuration file (main.tf). This file specifies the AWS provider, region, access keys, AMI identifier, instance type, security group, and key pair. Below are the steps to create this file and initialize the Terraform project.

**Create a Directory for Your Terraform Project**:

mkdir project1

cd project1

mkdir terraform

cd terraform

**2. Create a File Named main.tf**

touch main.tf

**3. Edit main.tf and Add the Following Content:**

**terraform {**

**required\_providers {**

**aws = {**

**source = "hashicorp/aws"**

**version = "5.49.0"**

**}**

**}**

**}**

**provider "aws" {**

**region = "us-east-1"**

**// profile = "default"**

**}**

**# Security group for EC2 instance**

**resource "aws\_security\_group" "webtraffic" {**

**name = "webtraffic"**

**description = "Allow inbound and outbound traffic"**

**dynamic "ingress" {**

**for\_each = var.ingress\_rules**

**content {**

**description = "Inbound Rule"**

**from\_port = ingress.value**

**to\_port = ingress.value**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**}**

**dynamic "egress" {**

**for\_each = var.egress\_rules**

**content {**

**description = "Outbound Rule"**

**from\_port = egress.value**

**to\_port = egress.value**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**}**

**}**

**# EC2 instance**

**resource "aws\_instance" "ec2" {**

**ami = "ami-04b70fa74e45c3917"**

**instance\_type = "t2.micro"**

**key\_name = "key-for-demo-3"**

**vpc\_security\_group\_ids = [aws\_security\_group.webtraffic.id]**

**tags = {**

**Name = "MyEc2Instance"**

**}**

**connection {**

**type = "ssh"**

**user = "chaitra"**

**private\_key = file("/home/chaitra/keys/demo/key-for-demo-3")**

**host = aws\_instance.ec2.public\_ip**

**}**

**}**

**resource "aws\_key\_pair" "deployer"{**

**key\_name = "key-for-demo-3"**

**public\_key ="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCf2cLxKwuHCcyqRdrwBhzte7bGnaIMDEmfUcTLfLB5jMmB3CeYYJ+ocIyJpjabK/5QGW5vo9rsuWbH+W5zIUIBm3qrE+4kLTkfxi9u/TWrgsvlPRFDYYM+Y+h9bFhcZkSaYYKpMrAOWfzjjJMwTlGe8ZENxpTKwRlAVf1yXSSnGj5gUgd3CjzzdU6KfoHkBArB2C6TsPlEK1O+flBRCb7QqM6CnOxPi5cqEQ6JlqyCP86enKVn+0jlBMMuC5nZPSueal+8HSM/0QBKEr4Qs7pqlPaB/QJb0CHoLLY1K2bxXKmDac61XzGLckM11UUjMLBmBNpn0Yvay09xJsyBg6q/ chaitra@chaitra-VirtualBox"** }

**Starting the case with terraform**

Now we are executing the command terraform Init, that this command download the artifacts from AWS.

**cd /home/chaitra/project1/terraform~$ terraform init**

**The result of the terraform init is following ;-**

**Initializing the backend...**

**Initializing provider plugins...**

**- Finding hashicorp/aws versions matching ">= 2.0.0"...**

**- Installing hashicorp/aws v4.5.0...**

**- Installed hashicorp/aws v4.5.0 (signed by HashiCorp)**

**Terraform has been successfully initialized!**

**You may now begin working with Terraform. Try running "terraform plan" to see**

**any changes that are required for your infrastructure. All Terraform commands**

**should now work.**

**If you ever set or change modules or backend configuration for Terraform,**

**rerun this command to reinitialize your working directory. If you forget, other**

**commands will detect it and remind you to do so if necessary.**

Now let's run the terraform plan command. This will allow us to see what Terraform will do before we decide to define the infrastructure.

**cd /home/chaitra/project1/terraform~$ terraform plan**

**The result of the terraform plan is following ;-**

**$ terraform plan**

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

# aws\_instance.ec2 will be created

+ resource "aws\_instance" "ec2" {

+ ami = "ami-04b70fa74e45c3917"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ availability\_zone = (known after apply)

+ cpu\_core\_count = (known after apply)

+ cpu\_threads\_per\_core = (known after apply)

+ id = (known after apply)

+ instance\_state = (known after apply)

+ instance\_type = "t2.micro"

+ ipv6\_address\_count = (known after apply)

+ ipv6\_addresses = (known after apply)

+ key\_name = "key-for-demo-3"

+ outpost\_arn = (known after apply)

+ placement\_group = (known after apply)

+ primary\_network\_interface\_id = (known after apply)

+ private\_dns = (known after apply)

+ private\_ip = (known after apply)

+ public\_dns = (known after apply)

+ public\_ip = (known after apply)

+ secondary\_private\_ips = (known after apply)

+ security\_groups = (known after apply)

+ source\_dest\_check = true

+ subnet\_id = (known after apply)

+ tags = {

+ "Name" = "MyEc2Instance"

}

+ tenancy = (known after apply)

+ volume\_tags = (known after apply)

+ vpc\_security\_group\_ids = (known after apply)

+ ami = "ami-04b70fa74e45c3917"

+ arn = (known after apply)

+ associate\_public\_ip\_address = (known after apply)

+ availability\_zone = (known after apply)

+ cpu\_core\_count = (known after apply)

+ cpu\_threads\_per\_core = (known after apply)

+ id = (known after apply)

+ instance\_state = (known after apply)

+ instance\_type = "t2.micro"

+ ipv6\_address\_count = (known after apply)

+ ipv6\_addresses = (known after apply)

+ key\_name = "key-for-demo-3"

+ outpost\_arn = (known after apply)

+ placement\_group = (known after apply)

+ primary\_network\_interface\_id = (known after apply)

+ private\_dns = (known after apply)

+ private\_ip = (known after apply)

+ public\_dns = (known after apply)

+ public\_ip = (known after apply)

+ secondary\_private\_ips = (known after apply)

+ security\_groups = (known after apply)

+ source\_dest\_check = true

+ subnet\_id = (known after apply)

+ tags = {

+ "Name" = "MyEc2Instance"

}

+ tenancy = (known after apply)

+ volume\_tags = (known after apply)

+ vpc\_security\_group\_ids = (known after apply)

+ root\_block\_device {

+ delete\_on\_termination = true

+ device\_name = (known after apply)

+ encrypted = (known after apply)

+ iops = (known after apply)

+ kms\_key\_id = (known after apply)

+ throughput = (known after apply)

+ volume\_id = (known after apply)

+ volume\_size = (known after apply)

+ volume\_type = (known after apply)

}

}

# aws\_key\_pair.deployer will be created

+ resource "aws\_key\_pair" "deployer" {

+ arn = (known after apply)

+ fingerprint = (known after apply)

+ id = (known after apply)

+ key\_name = "key-for-demo-3"

+ public\_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCf2cLxKwuHCcyqRdrwBhzte7bGnaIMDEmfUcTLfLB5jMmB3CeYYJ+ocIyJpjabK/5QGW5vo9rsuWbH+W5zIUIBm3qrE+4kLTkfxi9u/TWrgsvlPRFDYYM+Y+h9bFhcZkSaYYKpMrAOWfzjjJMwTlGe8ZENxpTKwRlAVf1yXSSnGj5gUgd3CjzzdU6KfoHkBArB2C6TsPlEK1O+flBRCb7QqM6CnOxPi5cqEQ6JlqyCP86enKVn+0jlBMMuC5nZPSueal+8HSM/0QBKEr4Qs7pqlPaB/QJb0CHoLLY1K2bxXKmDac61XzGLckM11UUjMLBmBNpn0Yvay09xJsyBg6q/ chaitra@chaitra-VirtualBox"

}

# aws\_security\_group.webtraffic will be created

+ resource "aws\_security\_group" "webtraffic" {

+ arn = (known after apply)

+ description = "Allow inbound and outbound traffic"

+ egress = (known after apply)

+ id = (known after apply)

+ ingress = (known after apply)

+ name = "webtraffic"

+ owner\_id = (known after apply)

+ revoke\_rules\_on\_delete = false

+ vpc\_id = (known after apply)

}

Plan: 3 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────

Note: You didn't specify an "-out" parameter to save this plan, so Terraform can't guarantee that exactly these actions will be performed if

"terraform apply" is subsequently run.

**And the apply command**

**cd /home/chaitra/project1/terraform~$ terraform apply**

**The output of the terraform apply is following ;-**

**Do you want to perform these actions?**

**Terraform will perform the actions described above.**

**Only 'yes' will be accepted to approve.**

**Enter a value: yes**

**aws\_key\_pair.deployer: Creating...**

**aws\_security\_group.webtraffic: Creating...**

**aws\_key\_pair.deployer: Creation complete after 2s [id=key-for-demo-3]**

**aws\_security\_group.webtraffic: Creation complete after 5s [id=sg-0abc1234def567890]**

**aws\_instance.ec2: Creating...**

**aws\_instance.ec2: Still creating... [10s elapsed]**

**aws\_instance.ec2: Still creating... [20s elapsed]**

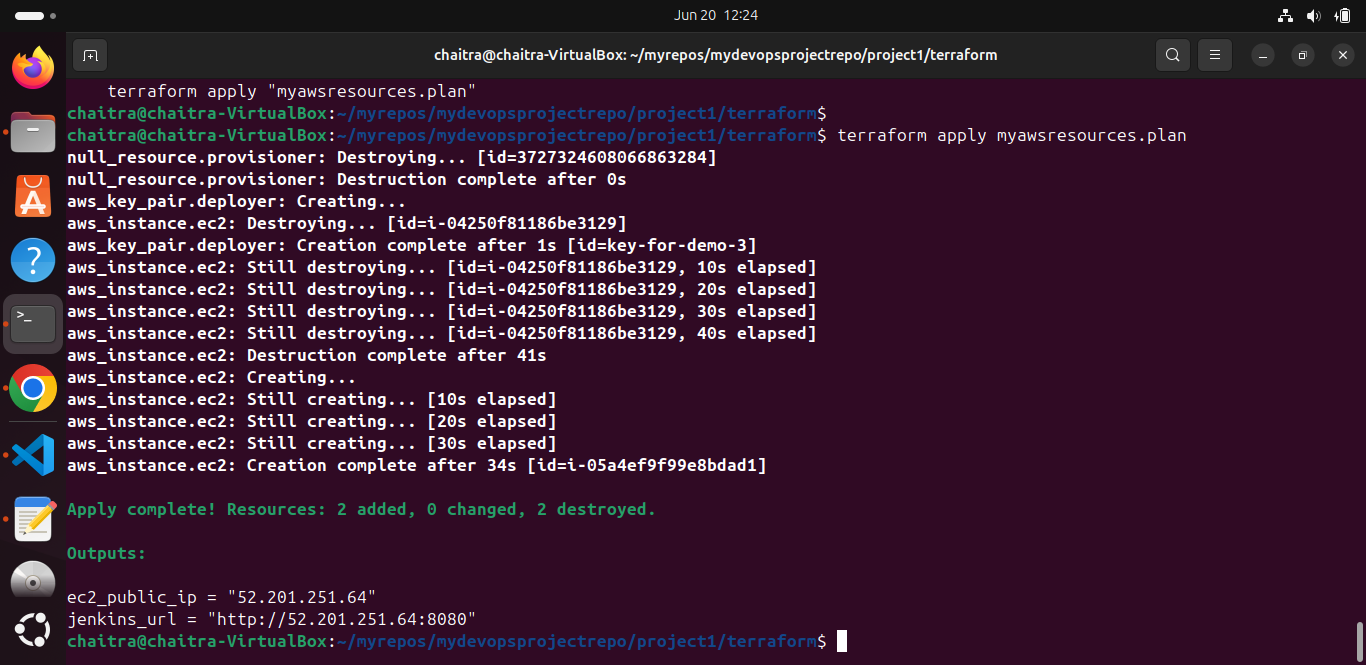
**aws\_instance.ec2: Still creating... [30s elapsed]**

**aws\_instance.ec2: Still creating... [40s elapsed]**

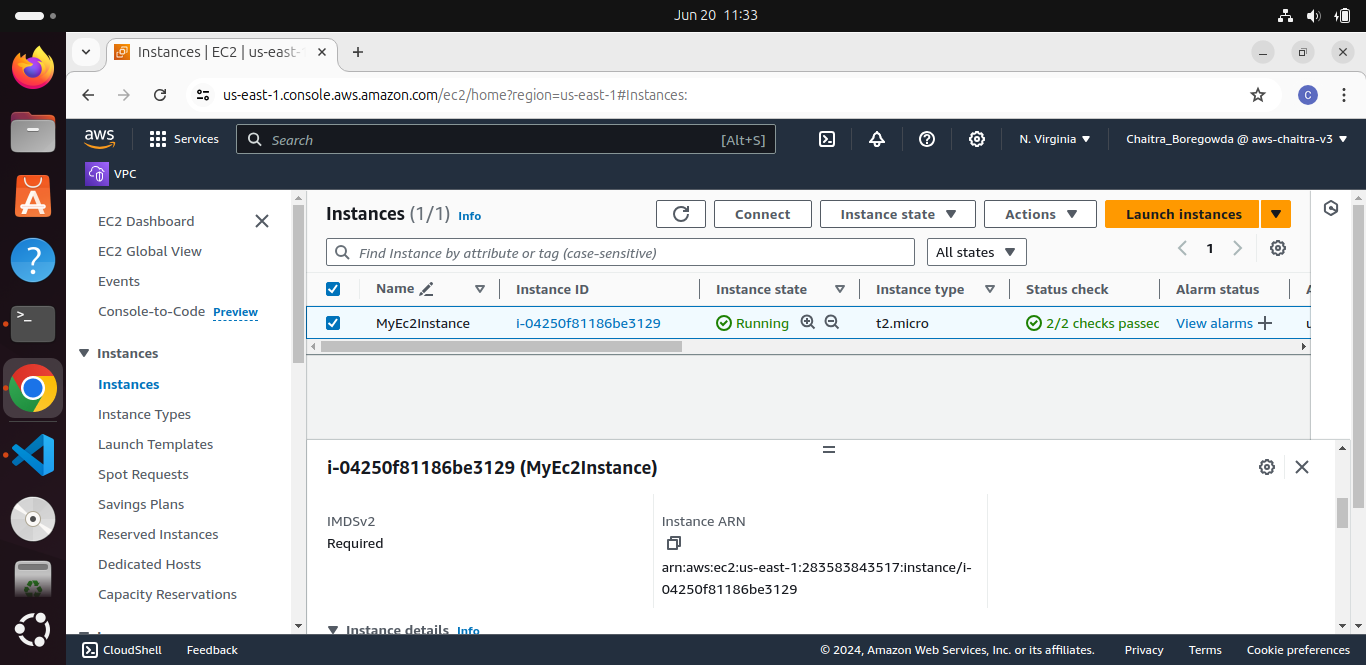
**aws\_instance.ec2: Creation complete after 45s [id=i-0abc1234def567890]**

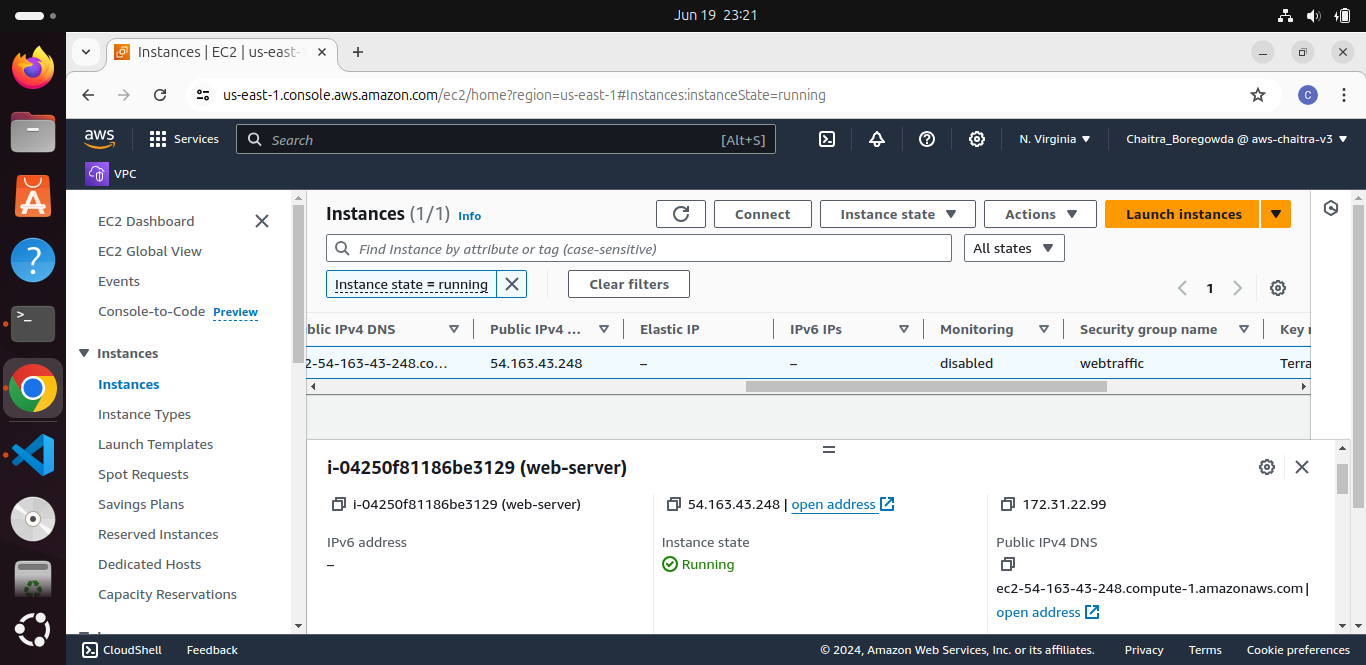
**Apply complete! Resources: 3 added, 0 changed, 0 destroyed.**

**Output:**

****

**Ok now your EC2 machine has been created, you can check the service on AWS Console**



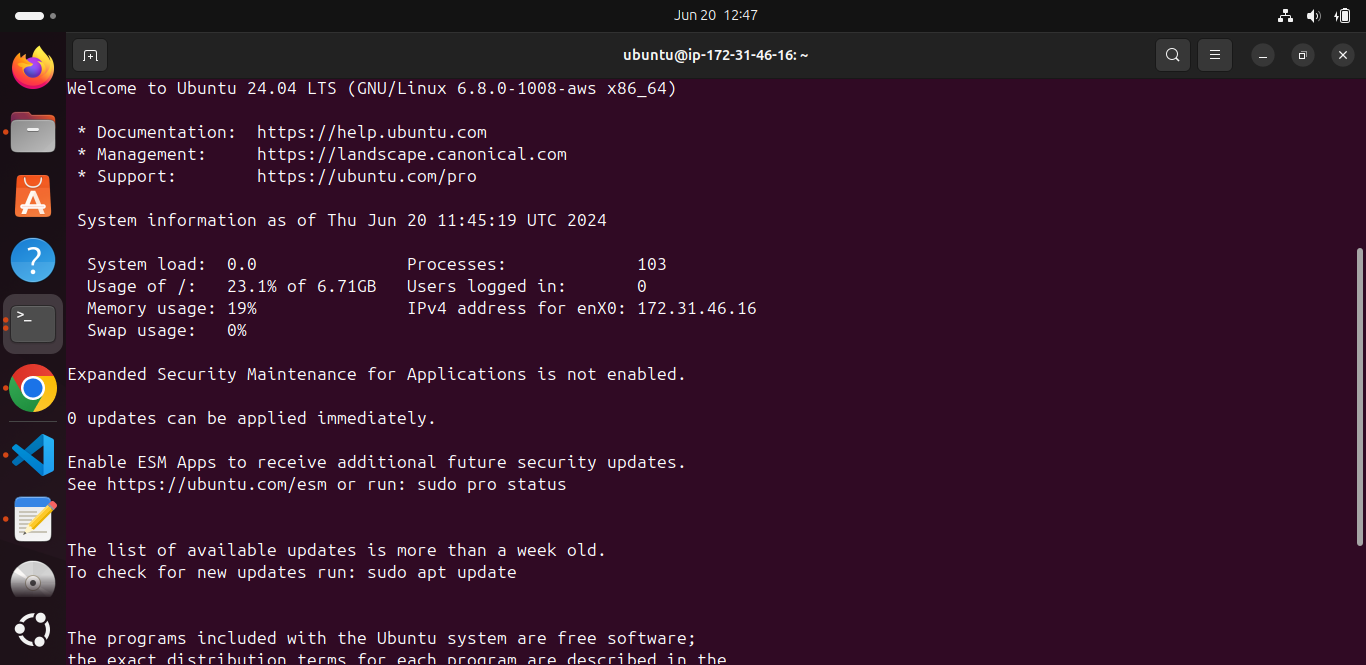


**Configure EC2 environment To install Java, Jenkins and python, you need to connect to EC2 machine from ssh, bellow the executed to connect:**

**STEP1 :- connect the instance using SSH Connect the ec2 instance using SSH connection in the local machine.**

**ssh -i "/home/chaitra/keys/demo/key-for-demo-3"**

**ubuntu@ip-172-31-46-16.eu-central-1 .compute-1.amazonaws.com**

****

**Step 2: Java Run Time**

First important item about these processes, JRE is very important for all artifacts, because of this, this is the first step.

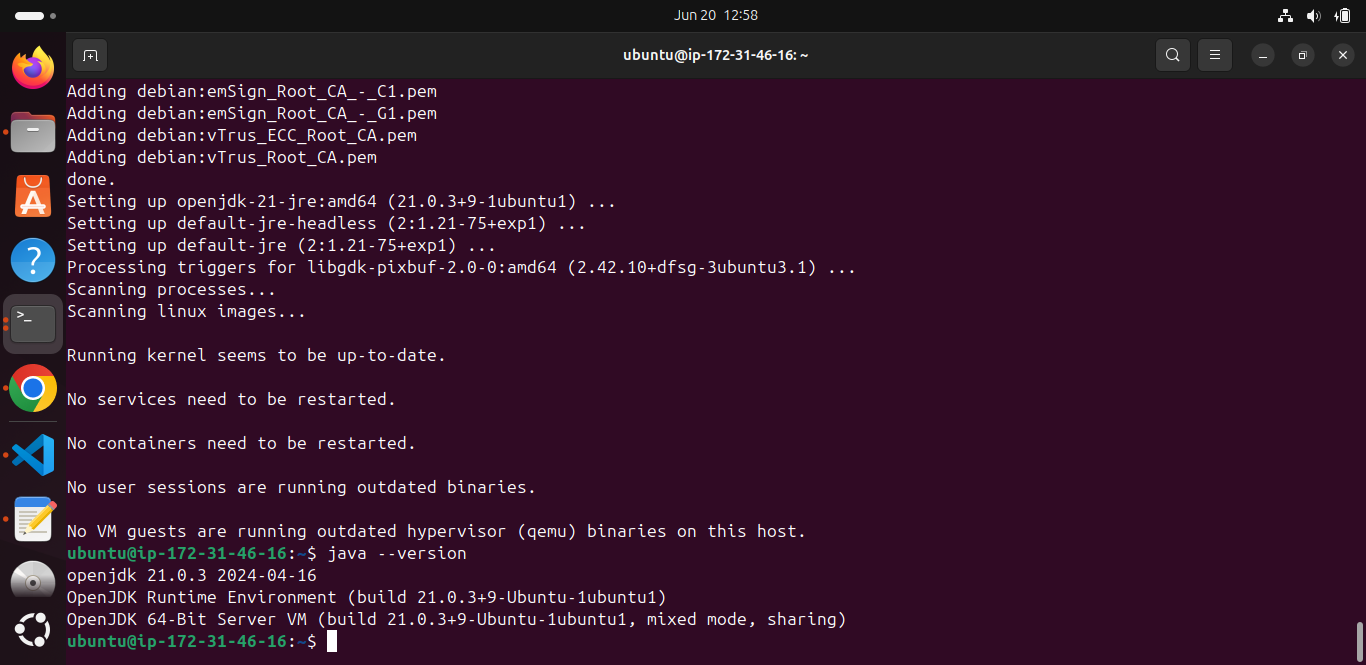
**sudo apt-get update**

**Install Default JRE**:

Install the default JRE package (which is OpenJDK) using the following command:

**sudo apt install default-jre**

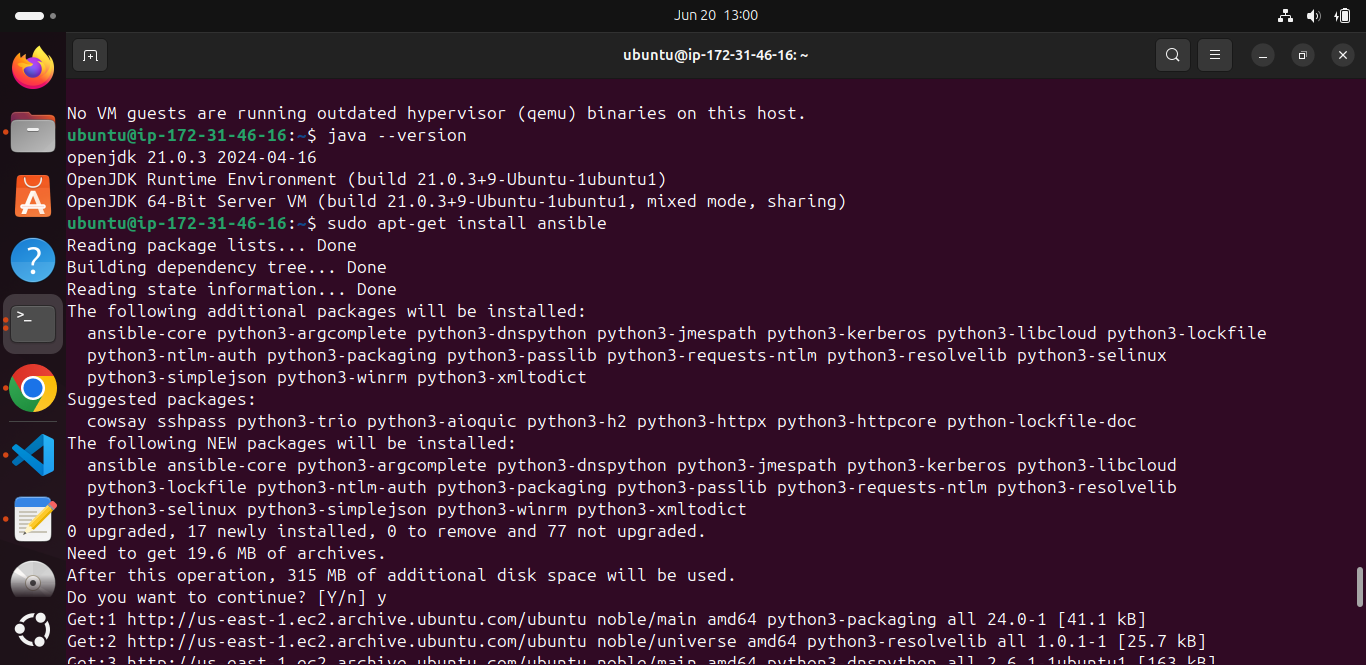
Verify the installation of java using the command “java --version”,



**Step 3: Ansible**

Use the below command to install Ansible

**sudo apt-get install ansible**

****

**Step 4 : Jenkins**

On the same terminal that executes the java installation, can be execute the Jenkins installation, with the command:

**Add Jenkins Repository**:

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

**Add Jenkins Debian Repository**

**sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'**

**Update Package List:**

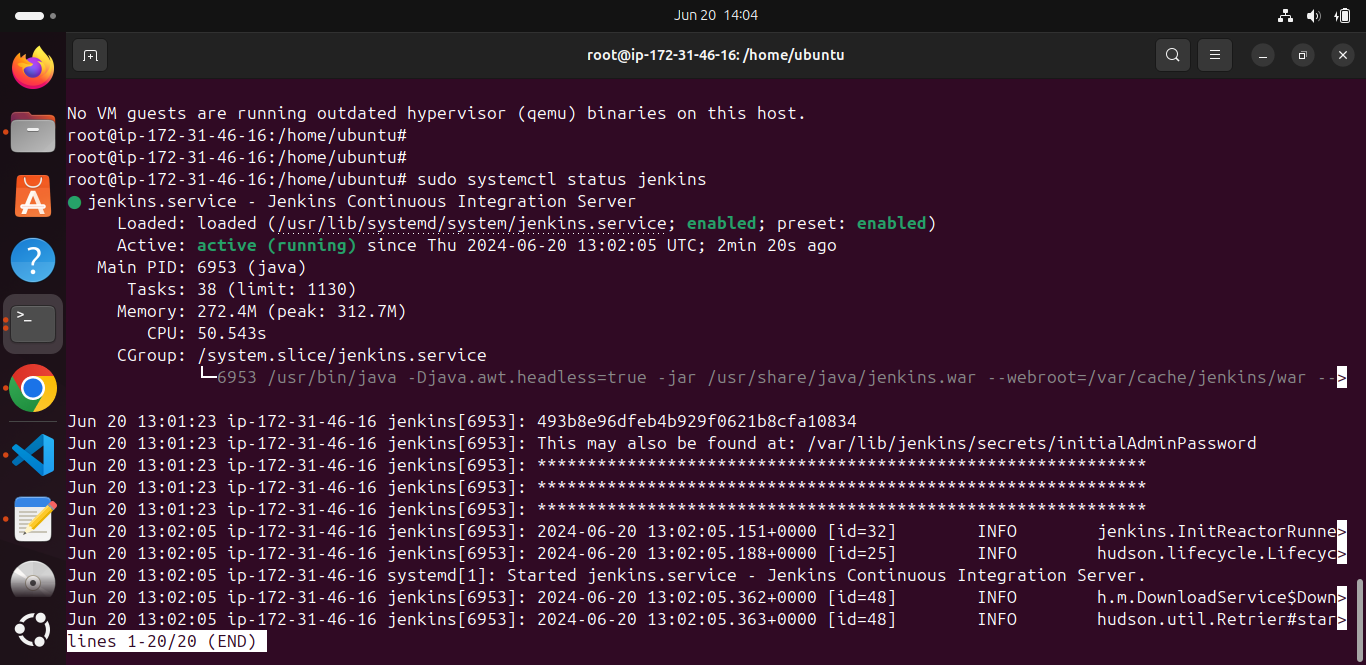
**sudo apt update**

**Install Jenkins**

**sudo apt install jenkins**

**Start Jenkins:**

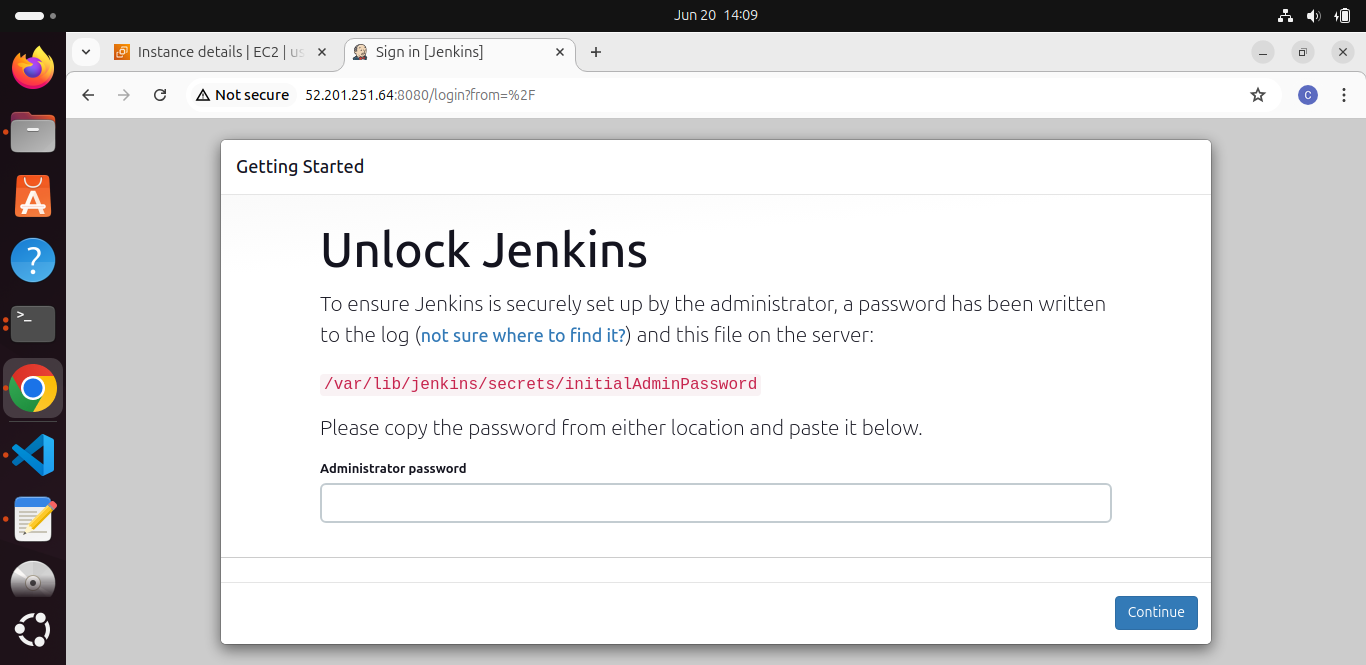
**sudo systemctl start jenkins**

****

**To access Jenkins on your server via its IPv4 address and port 8080, you would typically use the following URL format:**

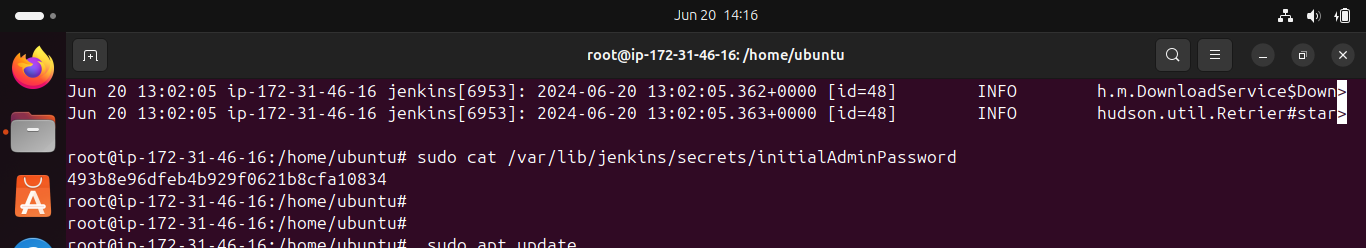
**Copy the IPV4 of this Server [public ip address of the instance running on aws console]**

**<http://54.123.456.789:8080>**

****

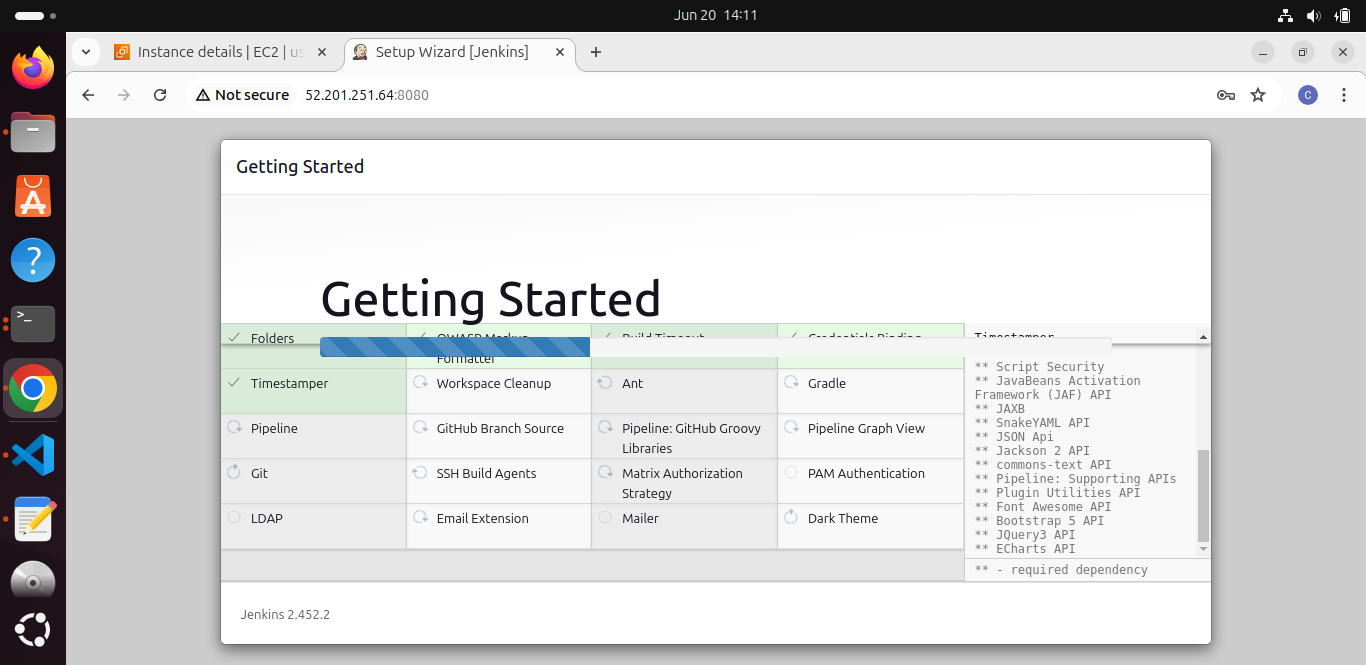
On terminal type the following command to verify Jenkins Initial admin password, is necessary to execute this command:

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

****

**Copy this password and paste on the field Administrator password on browser**

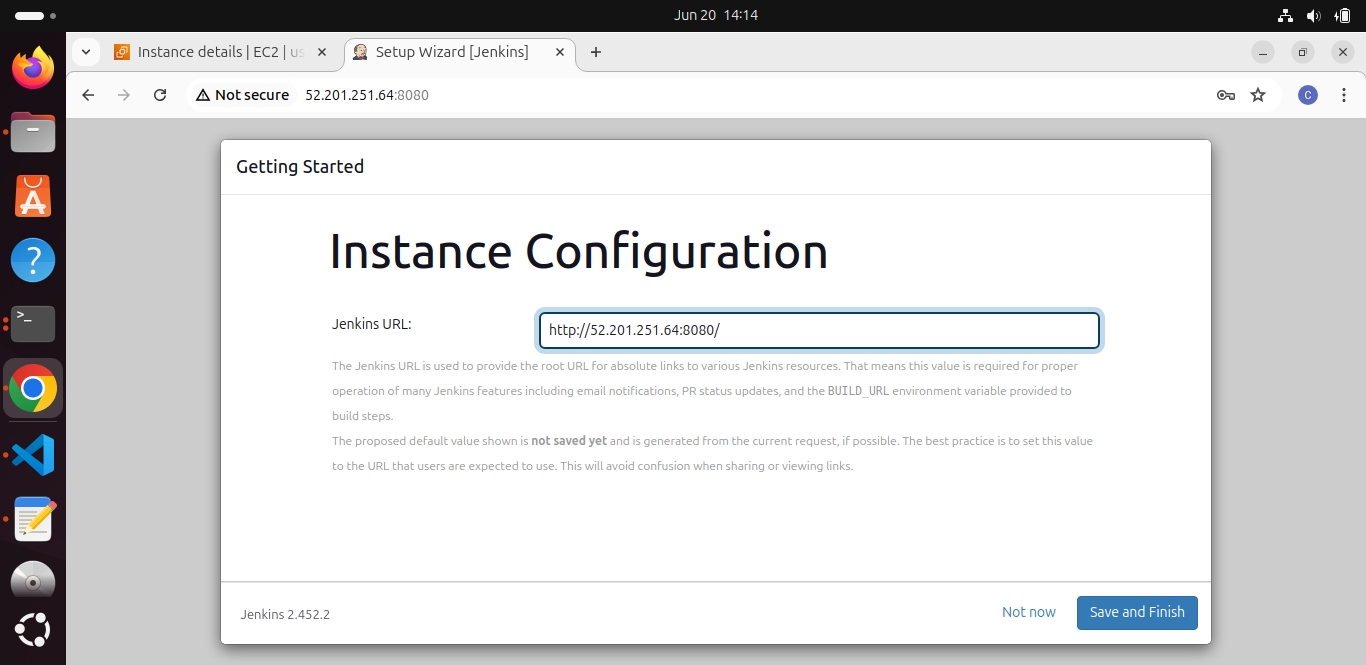
**Click on Install suggested plugins**

****

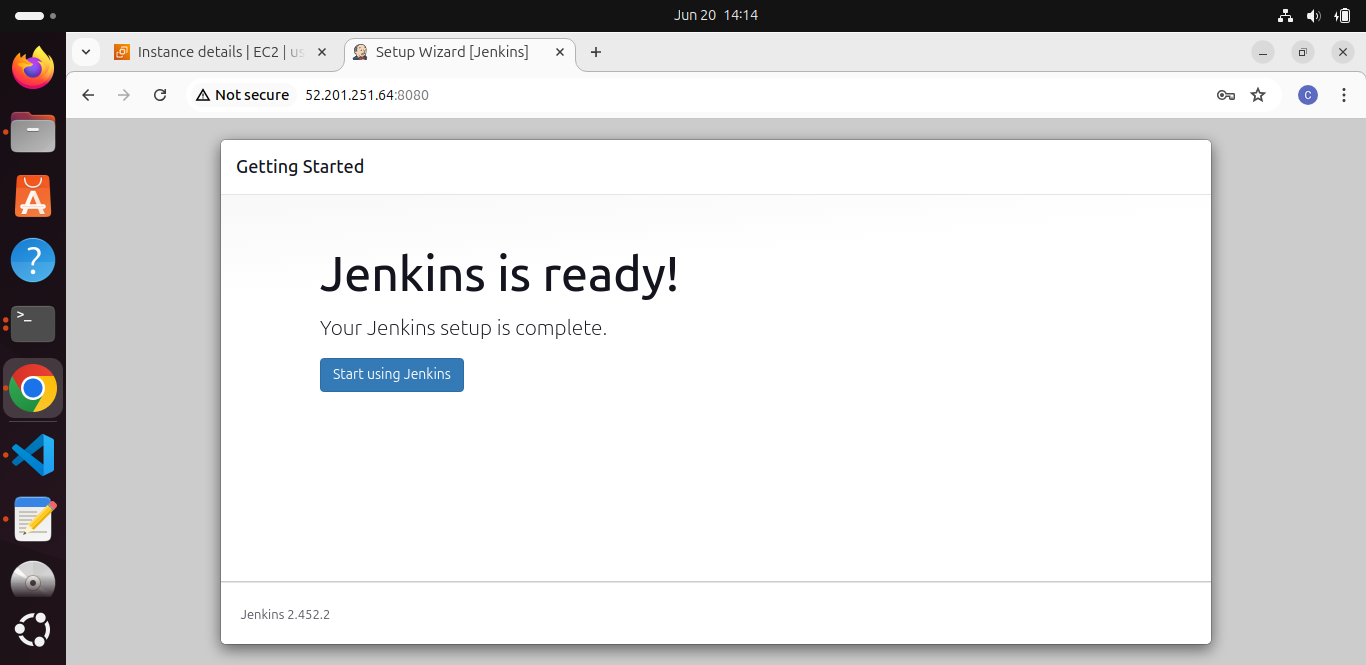
**To facility the currently process, we going to proceed with admin access and username is Chaitra Boregowda.**

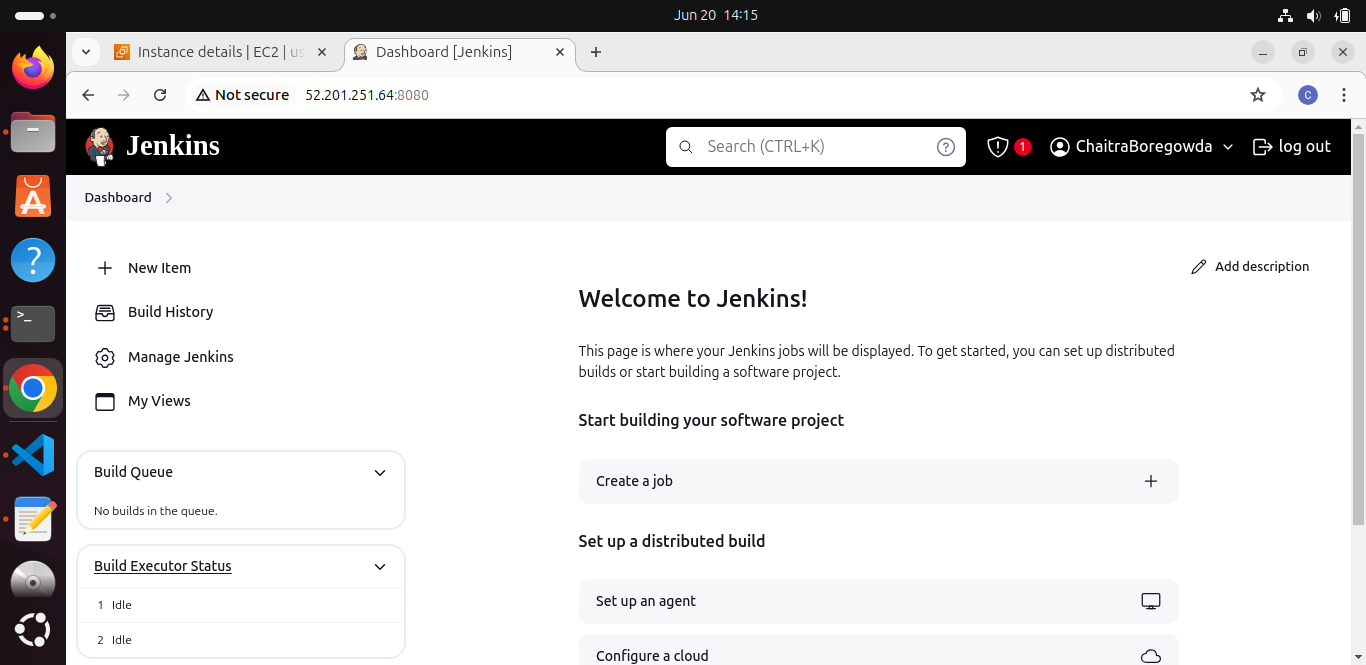
**Is possible to change the Jenkins default URL, but at this moment we proceed with this url and port**

**<http://54.123.456.789:8080>**

****

**The default installation has executed**

****

****

***Jenkins is Started on the instance.***

***Step 5 :- Python***

**Get updates with command sudo apt update**

**The software-properties-common package gives you better control over your package manager by letting you add PPA (Personal Package Archive) repositories. Install the supporting software with the command:**

**sudo apt install software-properties-common**

**Deadsnakes is a PPA with neweí releases than the default Ubuntu repositories. Add the PPA by entering the following:**

**sudo add-apt-repository ppa:deadsnakes/ppa**

**You will be prompted to press Enter to continue. This command adds the repository to your system's list of software sources and Refresh Package Lists:**

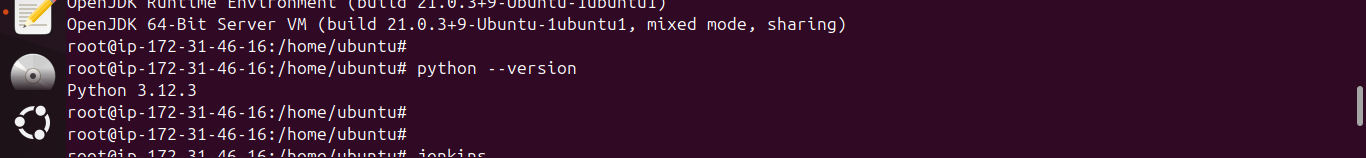
**sudo apt update**

**Now you can install python with command**

**sudo apt-get install python-is-python3**

And last execute the command verify that python is installed

**python --version**

****

**Everything is done! All set . Thankyou**

**PROJECT DONE BY** :- **CHAITRA BOREGOWDA**