**Project: Capstone II**

You are hired as a DevOps Engineer for Analytics Pvt Ltd. This company is a product based organization which uses Docker for their containerization needs within the company. The final product received a lot of traction in the first few weeks of launch. Now with the increasing demand, the organization needs to have a platform for automating deployment, scaling and operations of application containers across clusters of hosts. As a DevOps Engineer, you need to implement a DevOps lifecycle such that all the requirements are implemented without any change in the Docker containers in the testing environment.

Up until now, this organization used to follow a monolithic architecture with just 2 developers. The product is present on: <https://github.com/hshar/website.git>

**Following are the specifications of the lifecycle:**

1. Git workflow should be implemented. Since the company follows a monolithic architecture of development, you need to take care of version control. The release should happen only on the 25th of every month.

2. CodeBuild should be triggered once the commits are made in the master branch.

3. The code should be containerized with the help of the Dockerfile. The Dockerfile should be built every time if there is a push to GitHub. Create a custom Docker image using a Dockerfile.

4. As per the requirement in the production server, you need to use the Kubernetes cluster and the containerized code from Docker Hub should be deployed with 2 replicas. Create a Node-Port service and configure the same for port 30008.

5. Create a Jenkins Pipeline script to accomplish the above task.

6. For configuration management of the infrastructure, you need to deploy the configuration on the servers to install necessary software and configurations.

7. Using Terraform, accomplish the task of infrastructure creation in the AWS cloud provider.

**Architectural Advice:**

Software’s to be installed on the respective machines using configuration management.

**Worker1:** Jenkins,

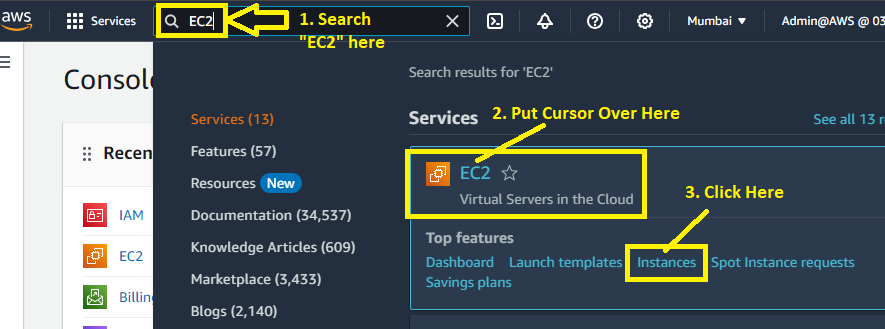
**Worker2:** Docker, Kubernetes

**Worker3:** Java, Docker, Kubernetes

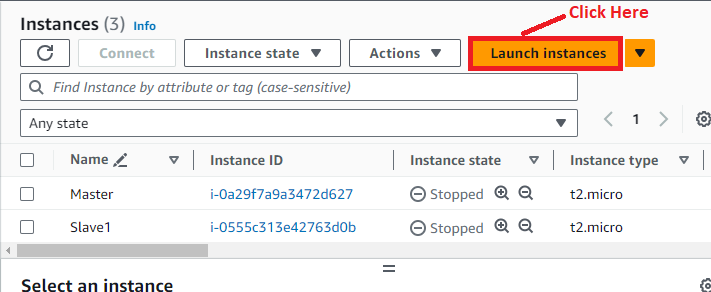
**Worker4:** Docker, Kubernetes

**A. Create an Instance Manually on AWS**

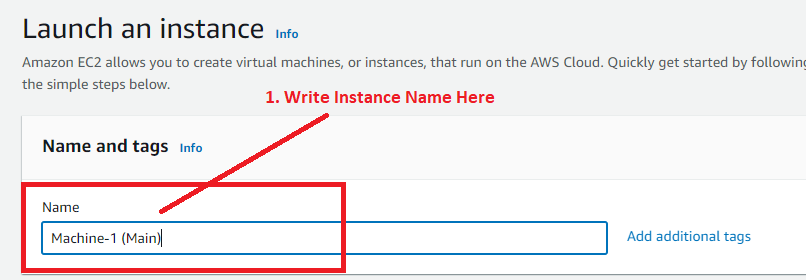
**Step 1: Go** to **“Services” section** & **search “EC2”** here. **Put cursor** over **“EC2”** & **click** on **“Instances”** here.



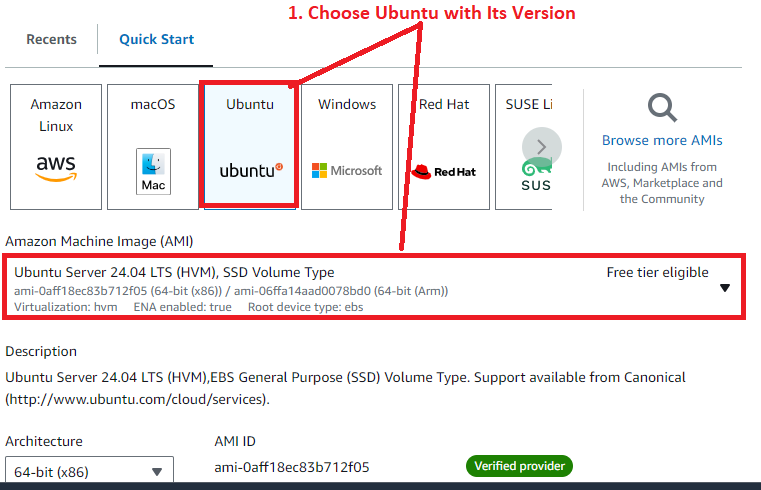
**Step 2: Click** on **“Launch Instances”.**

****

**Step 3: Choose “Name”** as **“Machine-1 (Main)”** in **“Name and tags” section.**

****

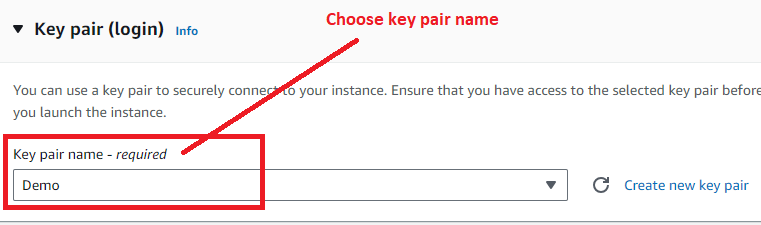
**Step 4: Choose “AMI”** as **“ubuntu”.**

****

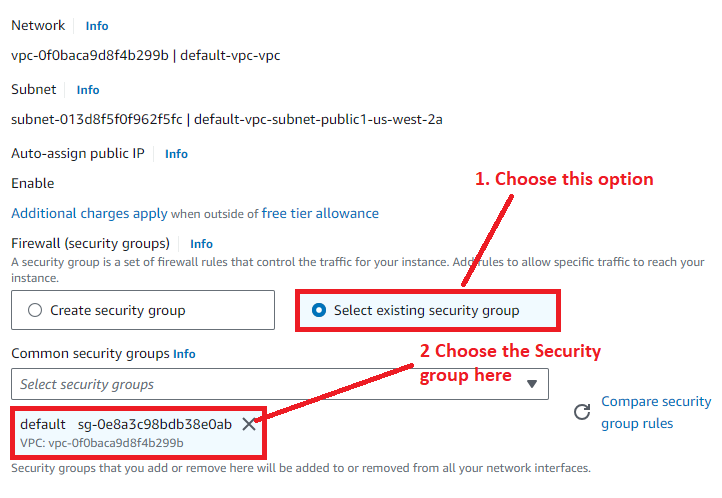
**Step 5: Choose “Instance type”** as **“t2.medium”.**

****

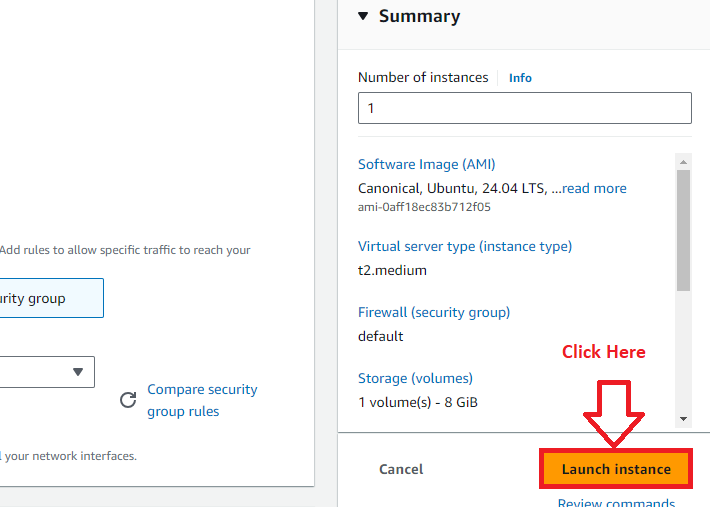
**Step 6: Choose “key pair (name) – required” as “Demo”.**

****

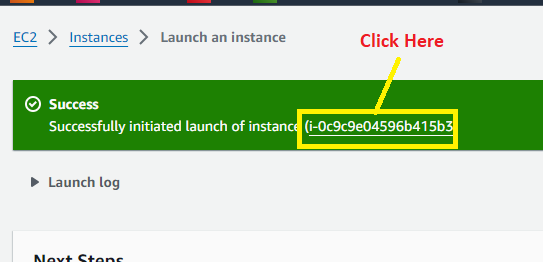
**Step 7: In “Network Settings”, choose “Select existing security group”** in **“Firewall (security groups)”** & **While choose “Common security groups”** as **“default”.**

****

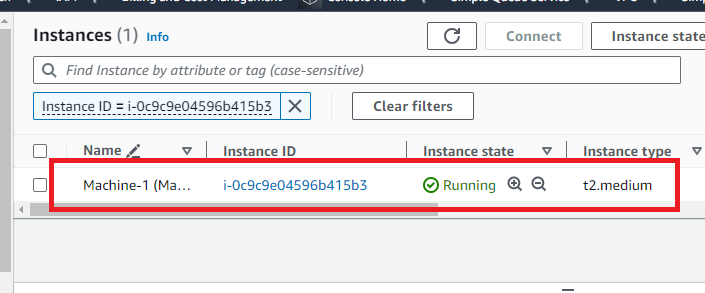
**Step 8: Click** on **“Launch Instance”** in **“Summary”.**

****

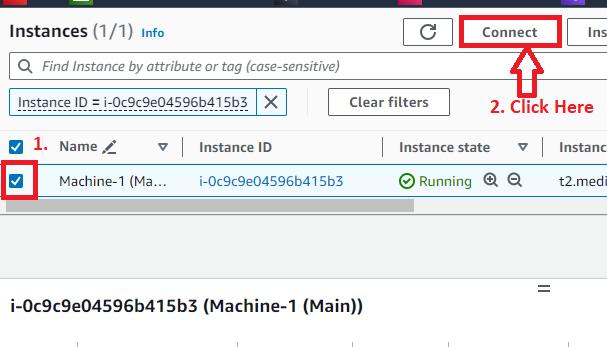
**Step 9: “Instance”** will be **successfully launched, click** on **“hyperlink (i-0c9c9e04596b415b3).**

****

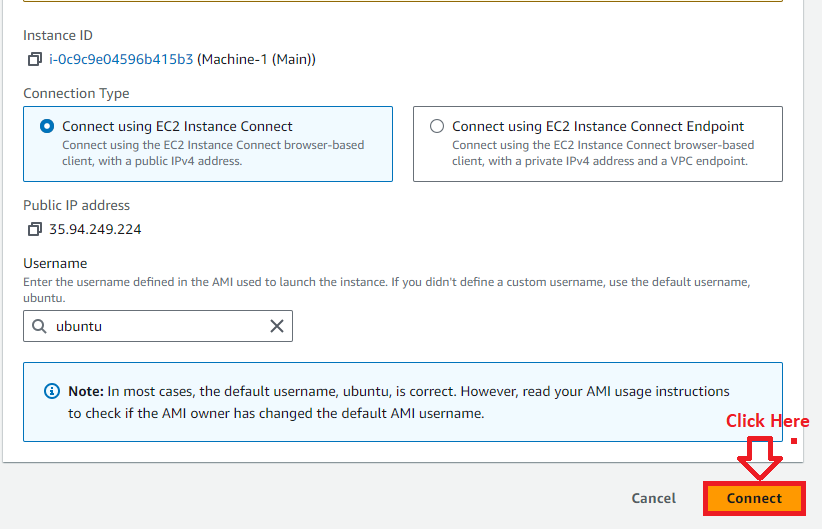
**Step 10: The Instance “[Machine-1 (Main)]”** will be **in “Running” State.**

****

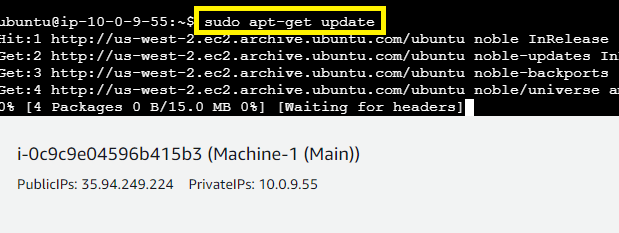
**Step 11: Select** the **Instance** & **Click** on **“Connect”.**

****

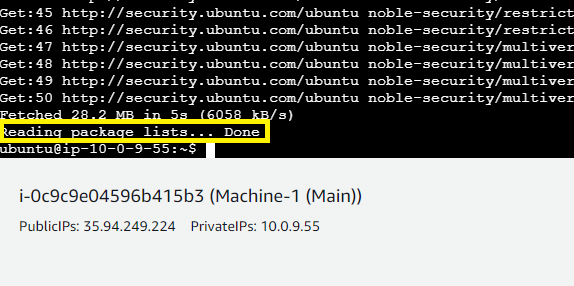
**Step 12: Again, click** on **“Connect”.**

****

**Step 13: First, update** the **machine using** the **command: sudo apt-get update.**

****

**Step 14: The machine [Machine-1 (Main)]** will be **successfully updated.**

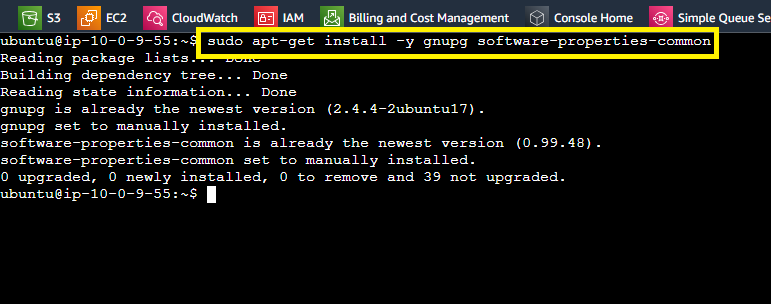
****

**B. Install Terraform on Instance [Machine-1 (Main)]**

**Step 1: First, we** will **install** the **“gnupg”, “software-properties-common” & “curl” packages** to **verify** the **“Hashicorp GPG signature”** & **install “Hashicorp’s Debian package repository”.**

***Paste this command & press “enter” from the keyboard.***

**sudo apt-get install -y gnupg software-properties-common**

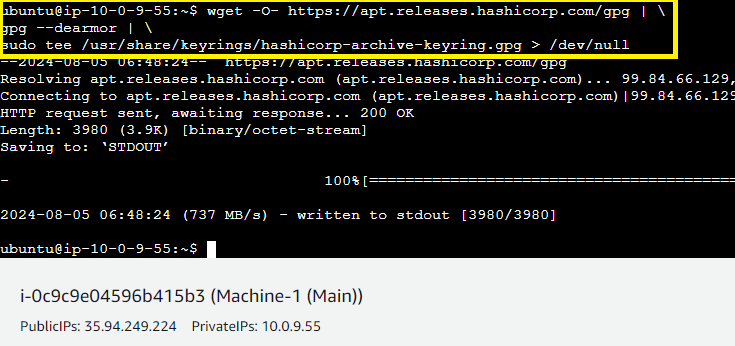
****

**Step 2: Install** the **“Hashicorp GPG Key”. Paste** the **below script** & **press “enter” from** the **keyboard.**

**wget -O- https://apt.releases.hashicorp.com/gpg | \**

**gpg --dearmor | \**

**sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null**

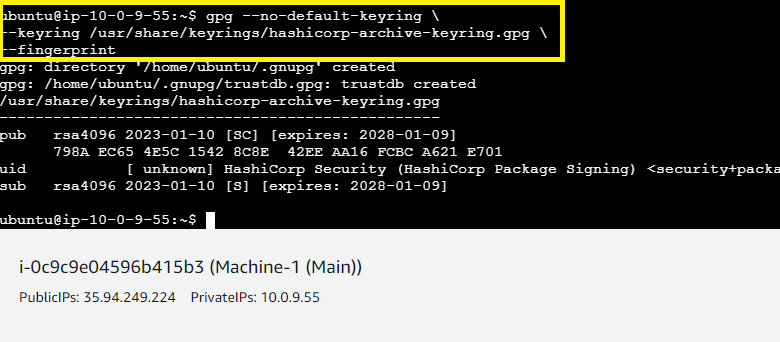
****

**Step 3: Verify the Key’s Fingerprint using this command:**

**gpg --no-default-keyring \**

**--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \**

**--fingerprint**

****

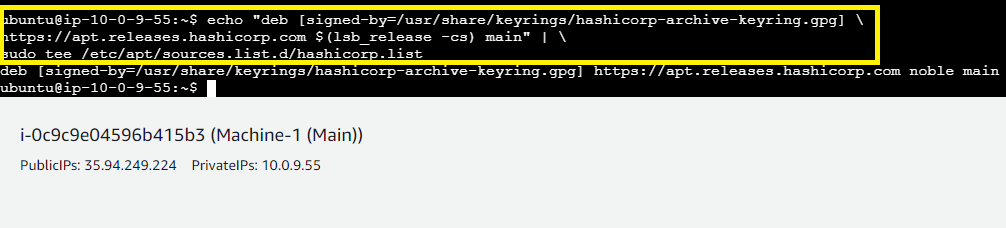
**Step 4: Add** the **official “Hashicorp repository”** to **your system. The lsb\_release –cs command finds** the **distribution release codename** for **your current system** such as **“buster”, “groovy” or “sid”.**

**Paste this command & press “enter” from the keyboard.**

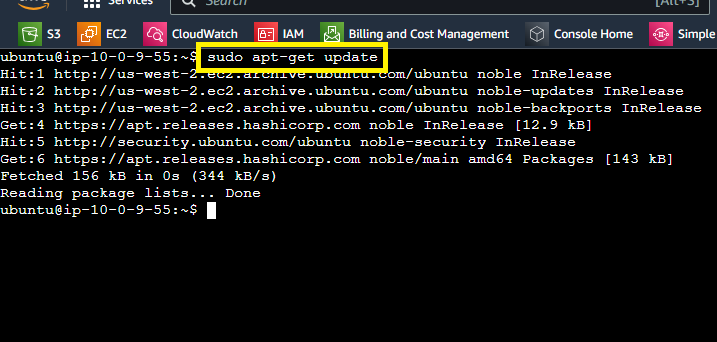
**echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \**

**https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | \**

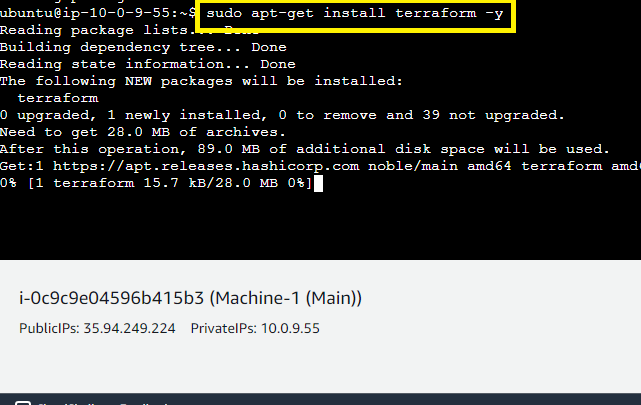
**sudo tee /etc/apt/sources.list.d/hashicorp.list**

****

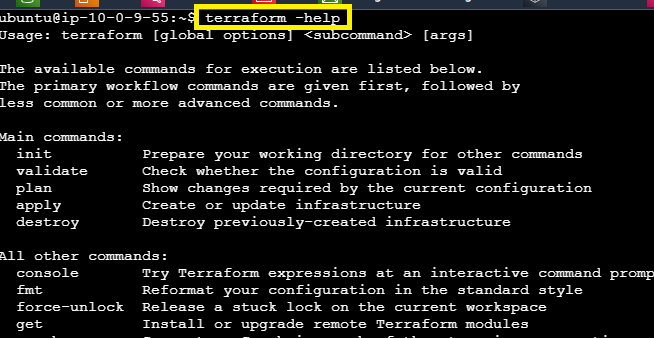
**Step 5: Again, update** the **machine using** the **command: sudo apt-get update.**

****

**Step 6: Now, install “Terraform” using** this **command: sudo apt-get install terraform –y**

****

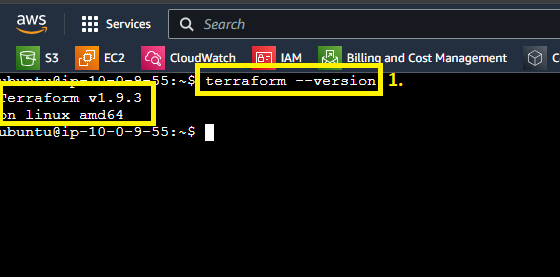
**Step 7: To check that “Terraform”** is **successfully installed** or **not. Use** this **command “terraform –help” to check that “Terraform”** is **installed** or **Not.**

****

**It** will **display** the **“Terraform Commands”.**

**Step 8: To check the “Terraform Version”, type this command: terraform --version**

**It** will **show** the **“Terraform Version”.**

****

**C. Run “Terraform Script” to Create Other Three Instances**

**Step 1: Create** a **“main.tf” file using** the **command: nano main.tf**

****

**Step 2: Paste** this **terraform script** in the **file:**

**provider "aws" {**

**region = "us-west-2"**

**}**

**resource "aws\_instance" "Kubernetes\_Master" {**

**ami = "ami-0aff18ec83b712f05"**

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

**subnet\_id = "subnet-013d8f5f0f962f5fc"**

**key\_name = "Demo"**

**tags = {**

**Name = "Machine-3"**

**}**

**}**

**resource "aws\_instance" "Kubernetes\_Slave1" {**

**ami = "ami-0aff18ec83b712f05"**

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

**subnet\_id = "subnet-013d8f5f0f962f5fc"**

**key\_name = "Demo"**

**tags = {**

**Name = "Machine-2"**

**}**

**}**

**resource "aws\_instance" "Kubernetes\_Slave2" {**

**ami = "ami-0aff18ec83b712f05"**

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

**subnet\_id = "subnet-013d8f5f0f962f5fc"**

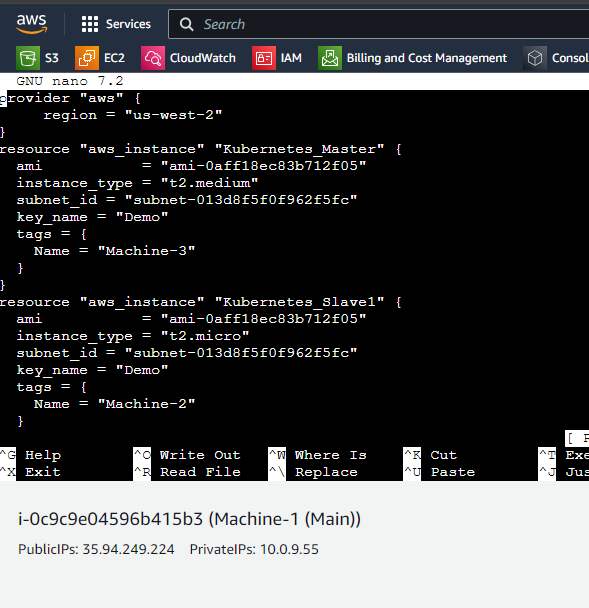
**key\_name = "Demo"**

**tags = {**

**Name = "Machine-4"**

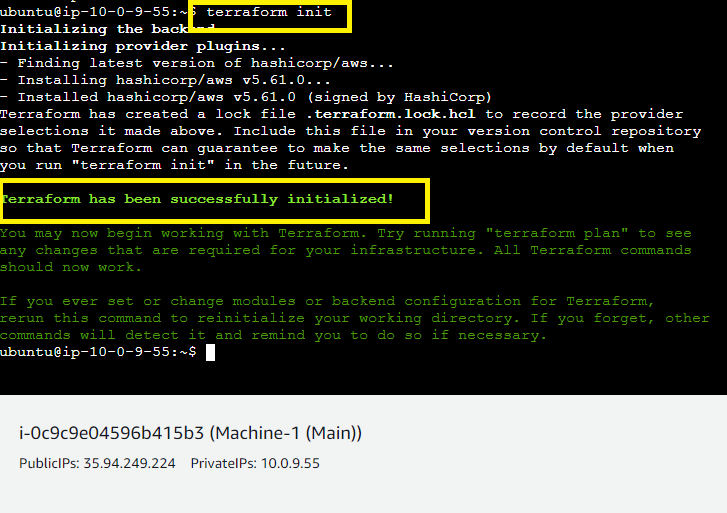
**}**

**}**

****

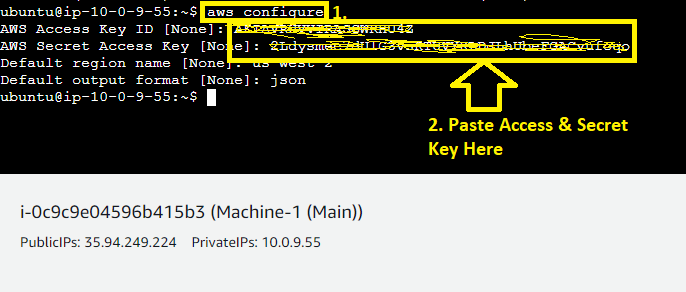
**Do “CTRL+X”** to **“Exit” & Press “Yes”** to **“Save”. Press”enter”** from the **keyboard** to **completely exit.**

**Step 3: Run** the **“terraform init” command** for **initialization.**

****

**It** will **initialize** the **“Terraform” & installed** the **needed plugins.**

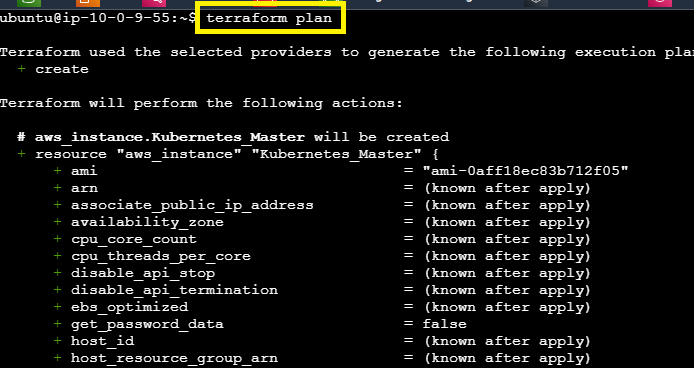
**Step 4: Now, run** the **“aws configure” command** to **execute** the **“Terraform Script”.**

****

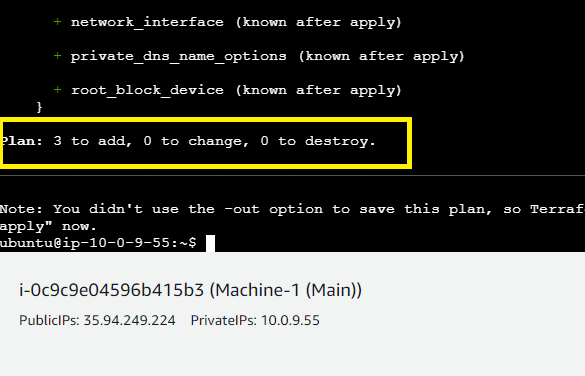
**Put “AWS Access Key”** & **“AWS Secret Key”** with **region name** here.

**Default Output Format** will be **“json” here.**

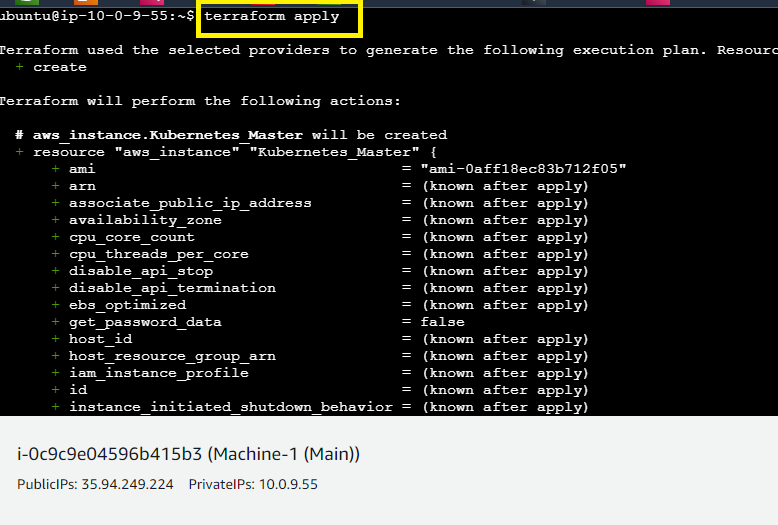
**Step 5: Now, use** the **“terraform plan” command** to **add** the **“plan”.**

****

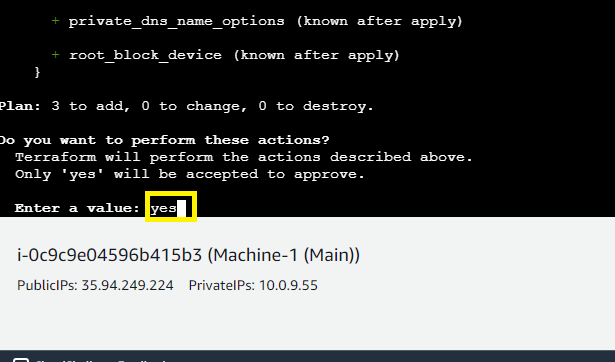
**Plan will be successfully added**

****

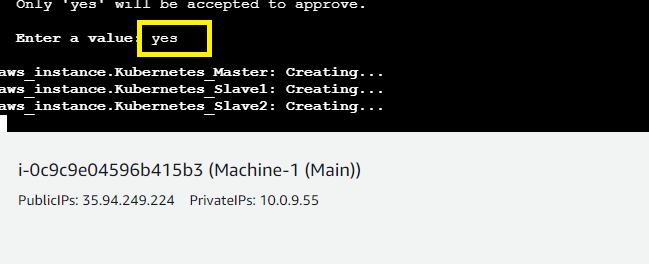
**Step 6: Now, use** the **“terraform apply” command** for **applying** the **changes.**

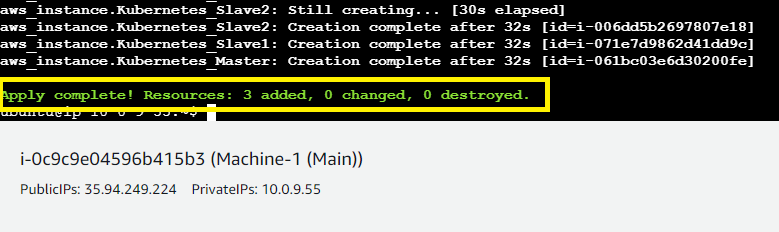
****

**Type “Yes” to continue.**

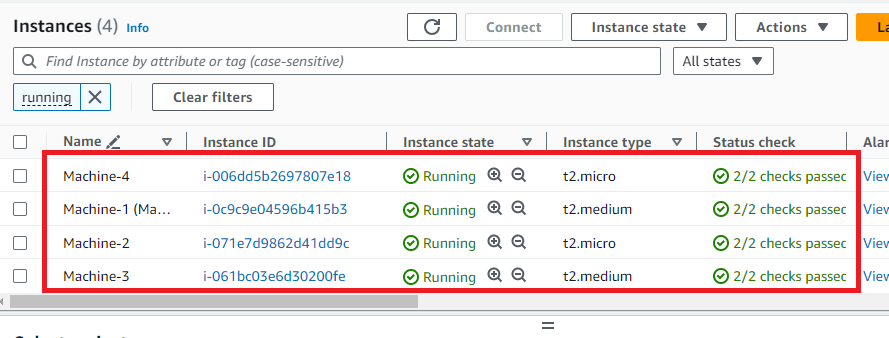
****

**The instance** will be **successfully created.**

****

****

**Step 7: All the machines** will be **successfully created.**

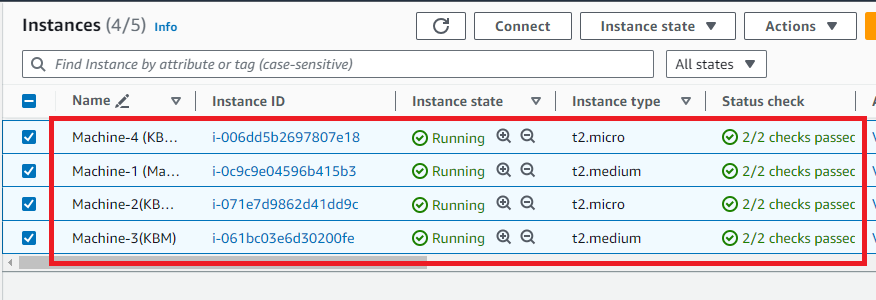
****

**Step 8: Now, we will rename these created machines:**

**Machine-2** as **Machine-2(KBSV1) – KBSV1** means **Kubernetes Slave1**

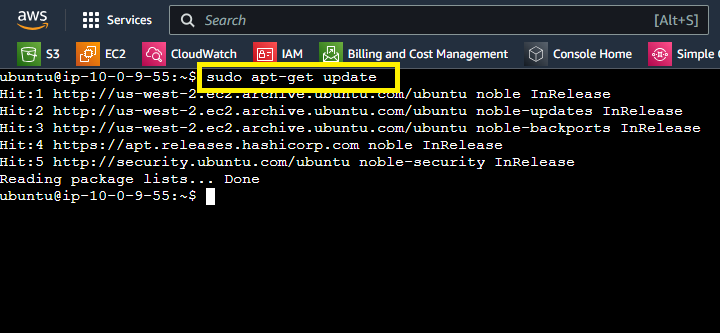
**Machine-3** as **Machine-3(KBM) – KBM** means **Kubernetes Master**

**Machine-4** as **Machine-4 (KBSV2) – KBSV2** means **Kubernetes Slave2**

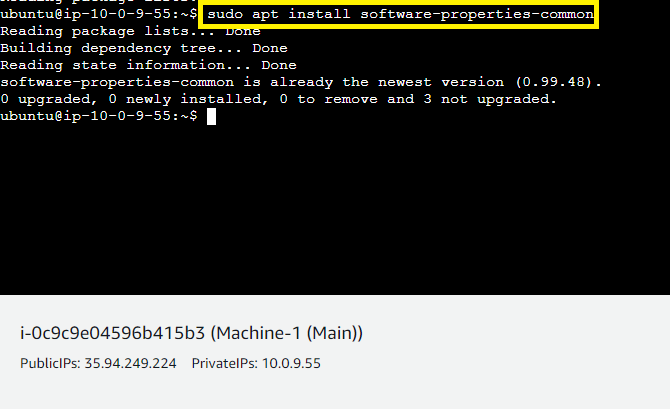
****

**D. Install “Ansible” on Machine 1 (main)**

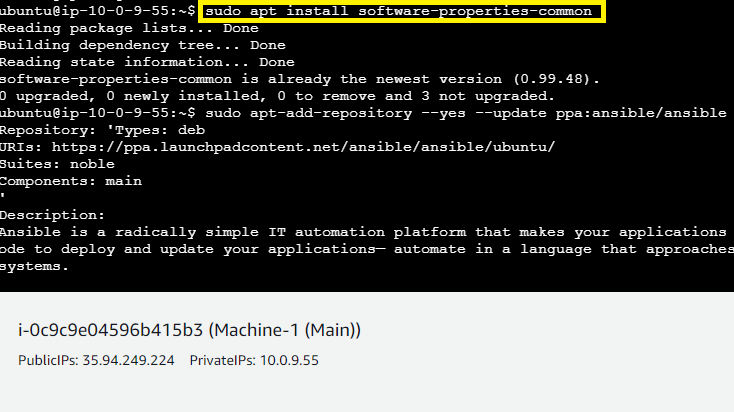
**Step 1: Go** to **“Machine-1”** & **first update** the **machine using** the **command: sudo apt-get update.**

****

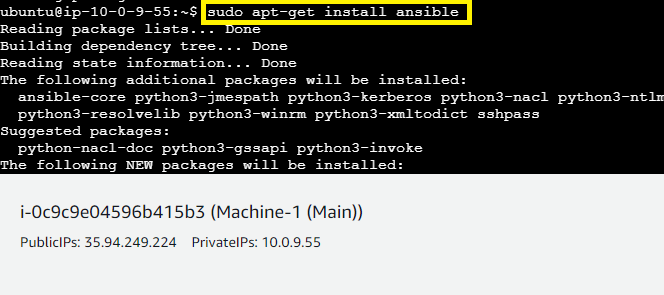
**Step 2: Run** this **command: sudo apt install software-properties-common. All packages** will be **already installed here.**

****

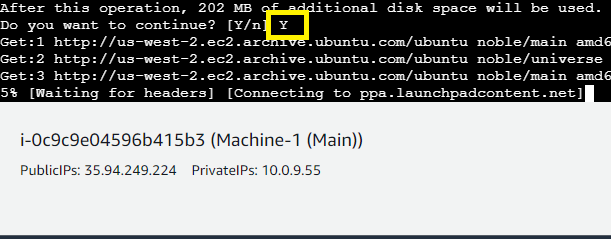
**Step 3: Now, add** the **repository** to **install** the **“Ansible” machine using this command: sudo apt-add-repository --yes --update ppa:ansible/ansible**

****

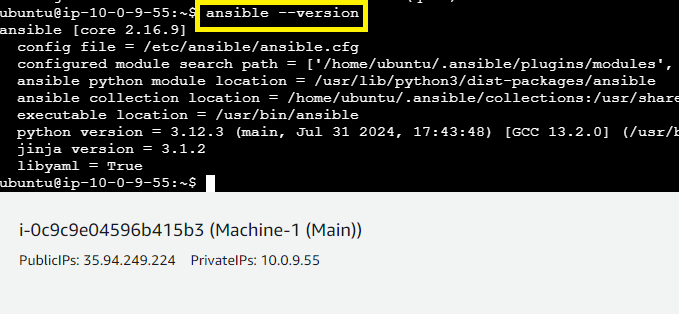
**Step 4: Install Ansible using the command: sudo apt-get install ansible**

****

**Type “Y”** to **continue** the **“Ansible Installation”.**

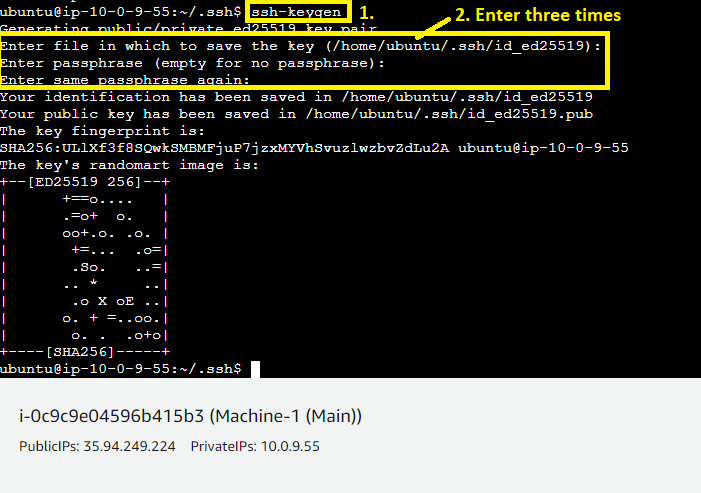
****

**Step 5: Type “ansible --version”** to **check** the **version** & **if version** is **shown, it means “Ansible”** has been **successfully installed.**

**  
Step 6: Go** to the **.ssh directory using** the **command: cd .ssh/**

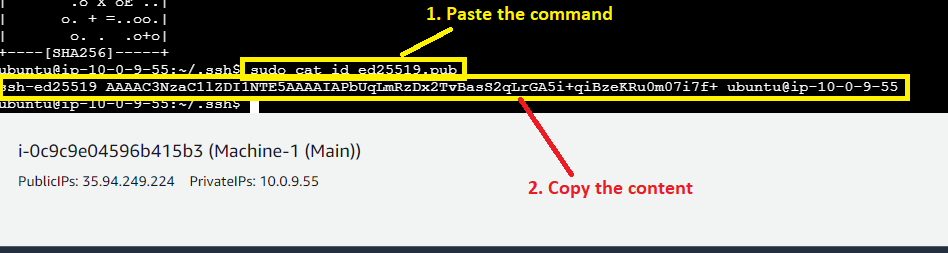
**Type “ssh-keygen”** to **create** the **public & private keys.**

**Press “enter” from** the **keyboard three times.**

****

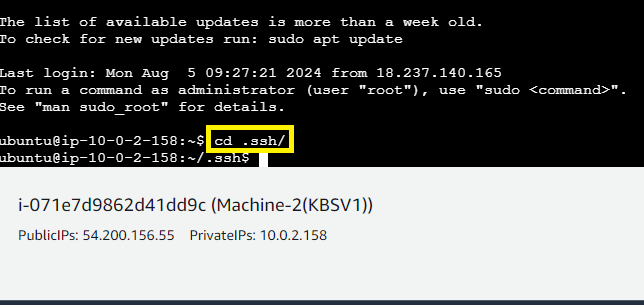
**The “Public” & “Private” Keys** will be **successfully generated.**

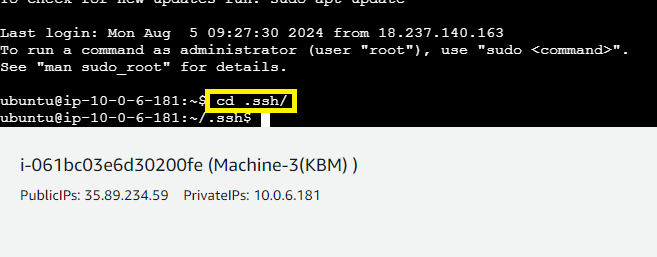
**Step 7: Type** this **command** to **copy** the **“id\_ed25519.pub”** content: **sudo cat id\_ed25519.pub**

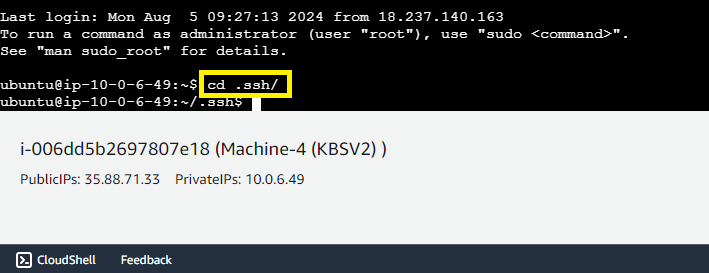
****

**Copy** this **content using** the **right click.**

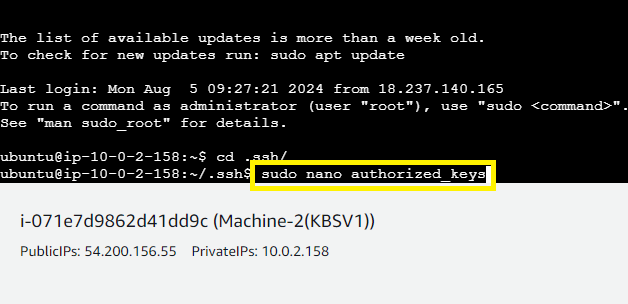
**Step 8: Now, we** will **paste** this **key content** to **all three machines one by one. First, you** have to **go to the “.ssh” directory** in **all machines using** the **command: cd .ssh/**

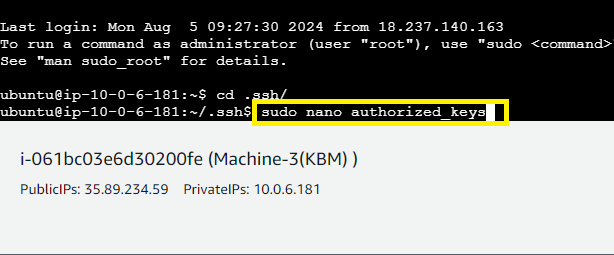
****

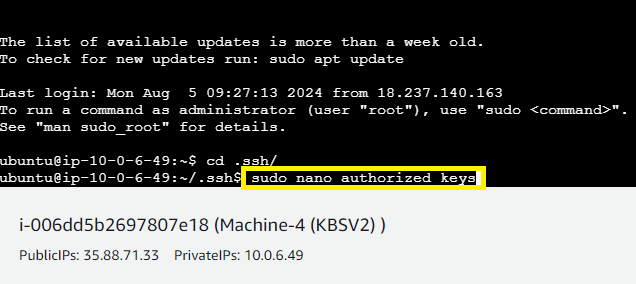
****

****

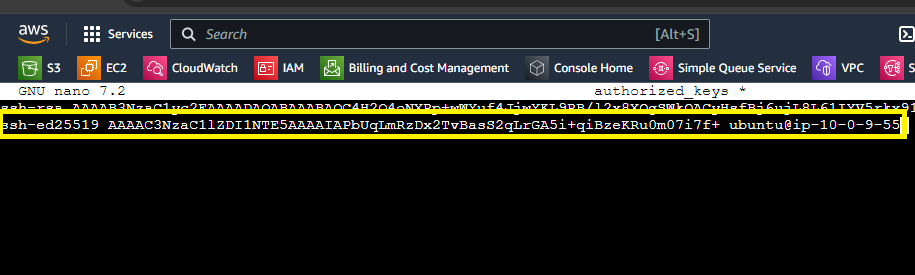
**Step 9: Open** the **“authorized\_keys”** in **all three machines using** the **command: sudo nano authorized\_keys**

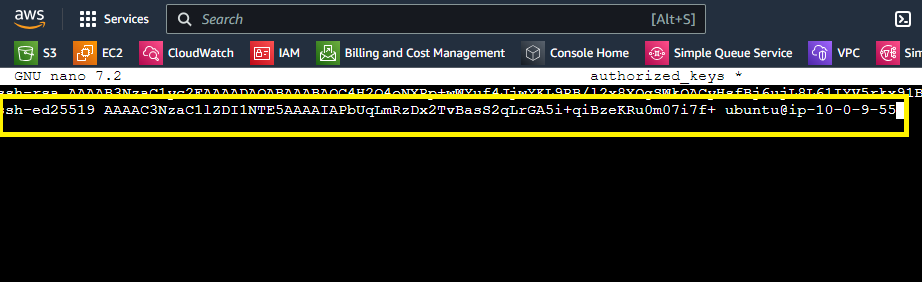
****

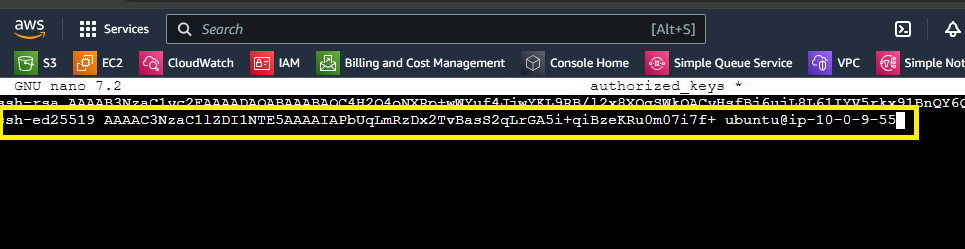
****

****

**Step 11: Paste** the **“id\_ed25519.pub” content** in **all** the **three machines.**

****

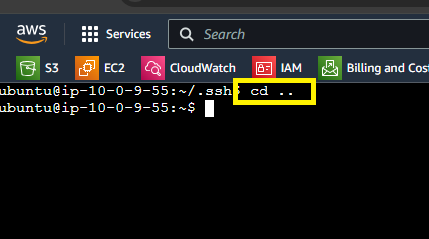
****

****

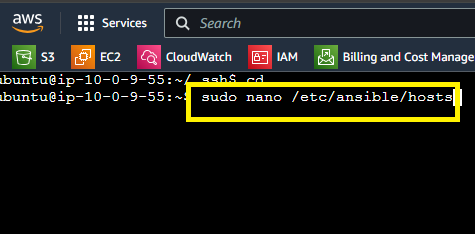
**Do “CTRL+X”** to **exit** & **press “Y”** to **save** the **file. Press “enter”** from the **keyboard** to **completely exit out from** all **three authorized\_keys files.**

**E. Paste the Private IP Addresses of Slaves in Hosts File**

**Step 1: Exit** from **.ssh/ directory.**

****

**Step 2: Go** to **the “hosts” file** in **“Ansible” using the command: sudo nano /etc/ansible/hosts.**

****

**Step 3: Paste the slaves & master private IP Addresses here:**

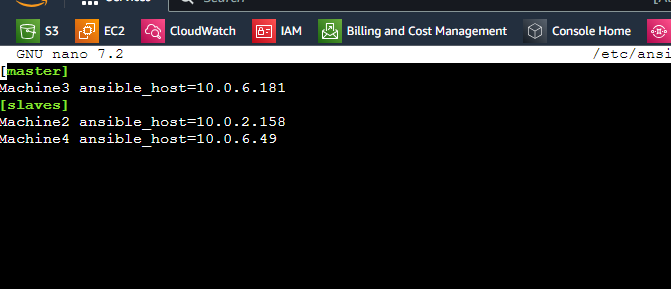
**[master]**

**Machine3 ansible\_host=10.0.6.181**

**[slaves]**

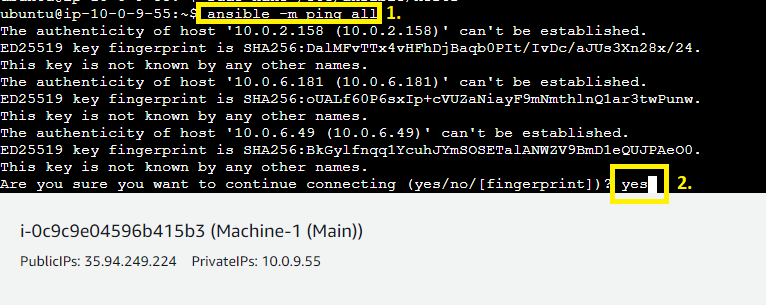
**Machine2 ansible\_host=10.0.2.158**

**Machine4 ansible\_host=10.0.6.49**

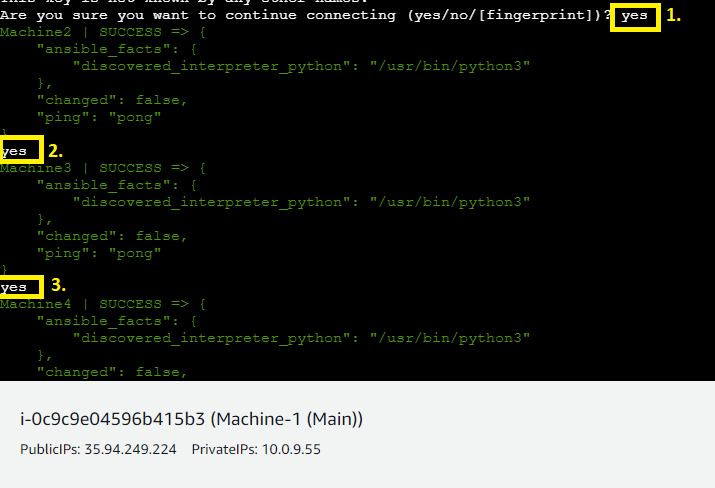
****

**Do “CTRL+X” to exit & Press “Y” for saving the file. Press “enter” from the keyboard to completely exit.**

**Step 4: Now, we** will **ping all machines using** the **command: ansible -m ping all**

****

**Type “yes” three times, all the machines will be successfully connected.**

****

**F. Create Three Scripts** for **Installing Required Tools** on **Machines**

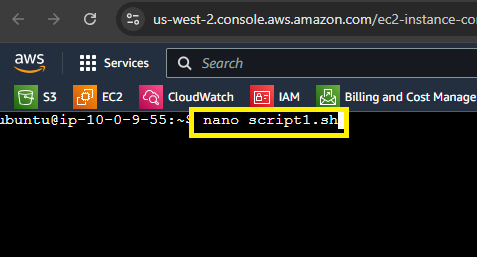
**Worker 1 -** Jenkins & Java Must Be Installed.

**Worker 2:** Docker & Kubernetes

**Worker 3:** Java, Docker & Kubernetes

**Worker 4:** Docker & Kubernetes

**Step 1: First, create** the **script file** to **install “Java”** & **“Jenkins”** Over **“Machine-1”. Use** this **command** to **create** the **“script1.sh” file: nano script1.sh**

****

**Step 2: Paste** the **following scripts here** to **Install “Java” & “Jenkins”.**

**sudo apt-get update**

**sudo apt-get install openjdk-17-jre-headless -y**

**sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \**

**https://pkg.jenkins.io/debian/jenkins.io-2023.key**

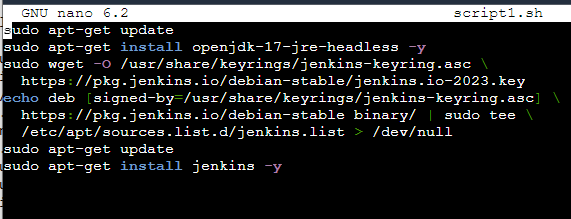
**echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \**

**https://pkg.jenkins.io/debian binary/ | sudo tee \**

**/etc/apt/sources.list.d/jenkins.list > /dev/null**

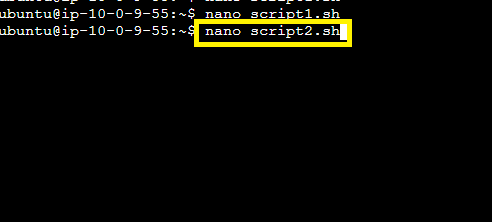
**sudo apt-get update**

**sudo apt-get install jenkins -y**

****

**Do “CTRL+X”** to **exit from** the **file** & **Save** the **file using “CTRL+S”. Exit** from **the file** by **pressing** the **“enter”** from the **keyboard.**

**Step 3: Now, create** the **script file** to **install “Java”, “Docker”, & “Kubernetes”** Over **“Machine-3”. Use** this **command** to **create** the **“script2.sh” file: nano script2.sh**

****

**Step 4: Paste** the **following scripts here** to **Install “Java” & “Docker” & “Kubernetes”.**

**sudo apt-get update**

**sudo apt-get install openjdk-17-jre-headless -y**

**sudo apt-get install docker.io -y**

**sudo systemctl enable --now docker**

**sudo swapoff -a**

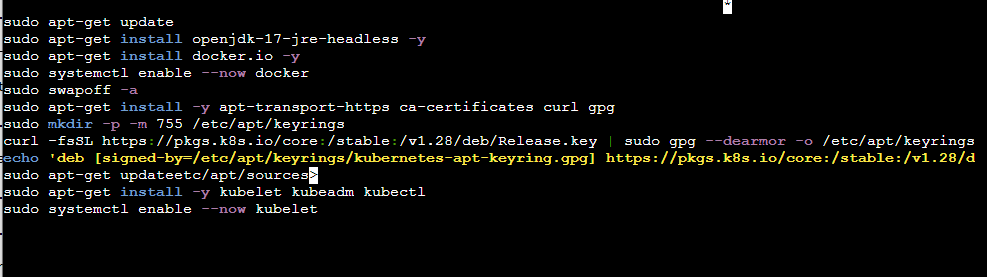
**sudo apt-get install -y apt-transport-https ca-certificates curl gpg**

**sudo mkdir -p -m 755 /etc/apt/keyrings**

**curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyringsecho 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/dsudo apt-get updateetc/apt/sources>**

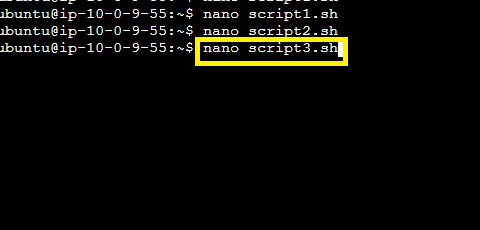
**sudo apt-get install -y kubelet kubeadm kubectl**

**sudo systemctl enable --now kubelet**

****

**Do “CTRL+X”** to **exit from** the **file** & **Save** the **file using “CTRL+S”. Exit** from **the file** by **pressing** the **“enter”** from the **keyboard.**

**Step 5: Again, create** the **script file** to **install “Docker”** & **“Kubernetes”** over **“Machine-2” & “Machine-4”. Use** this **command** to **create** the **“script3.sh” file: nano script3.sh**

****

**Step 6: Paste** the **following scripts here** to **Install “Docker” & “Kubernetes”.**

**sudo apt-get update**

**sudo apt-get install docker.io -y**

**sudo systemctl enable --now docker**

**sudo swapoff -a**

**sudo apt-get install -y apt-transport-https ca-certificates curl gpg**

**sudo mkdir -p -m 755 /etc/apt/keyrings**

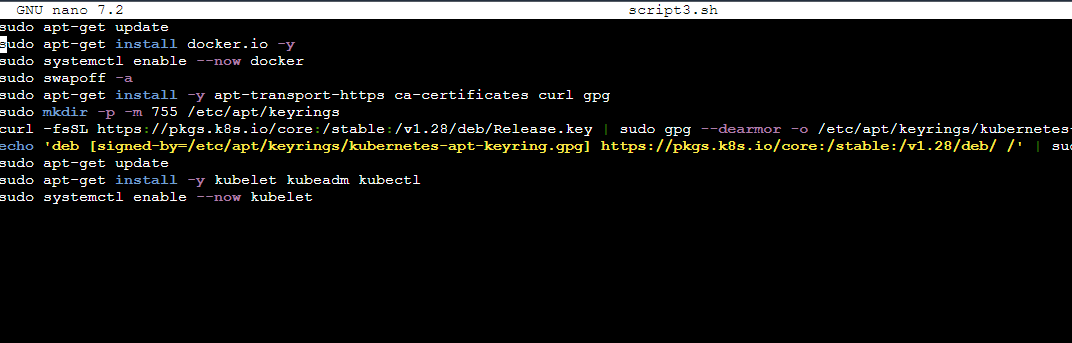
**curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg**

**echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list**

**sudo apt-get update**

**sudo apt-get install -y kubelet kubeadm kubectl**

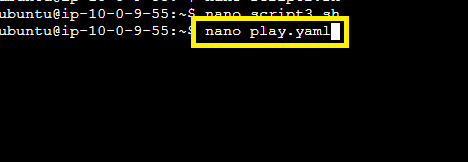
**sudo systemctl enable --now kubelet**

****

**Do “CTRL+X”** to **exit from** the **file** & **Save** the **file using “CTRL+S”. Exit** from **the file** by **pressing** the **“enter”** from the **keyboard.**

**G. Create the Playbooks to Run these Scripts to Install the Much Needed Tools**

**Step 1: Create a playbook using the command: nano play.yaml**

****

**Step 2: Paste this script to run all the script files here:**

**---**

**- name: install Jenkins & Java on Machine 1**

**become: true**

**hosts: localhost**

**tasks:**

**- name: running script1**

**script: script1.sh**

**- name: install Java, Docker & Kubernetes on Machine-1**

**become: true**

**hosts: master**

**tasks:**

**- name: running script2**

**script: script2.sh**

**- name: install Docker & Kubernetes on Machine-2&4**

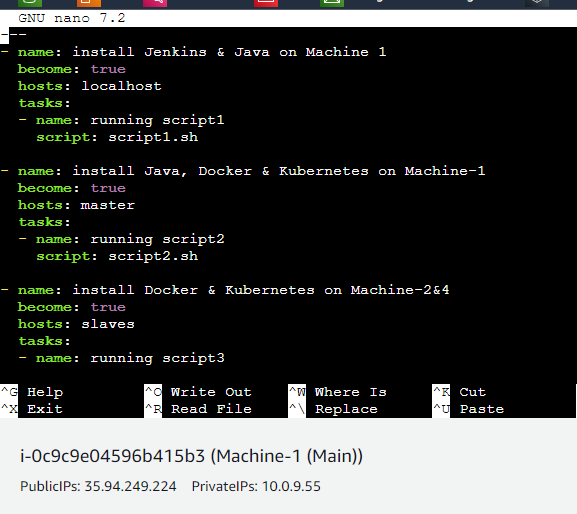
**become: true**

**hosts: slaves**

**tasks:**

**- name: running script3**

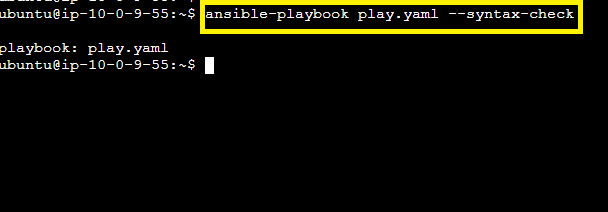
**script: script3.sh**

****

**Do “CTRL+X”** to **exit from** the **file** & **Save** the **file using “CTRL+S”. Exit** from **the file** by **pressing** the **“enter”** from the **keyboard.**

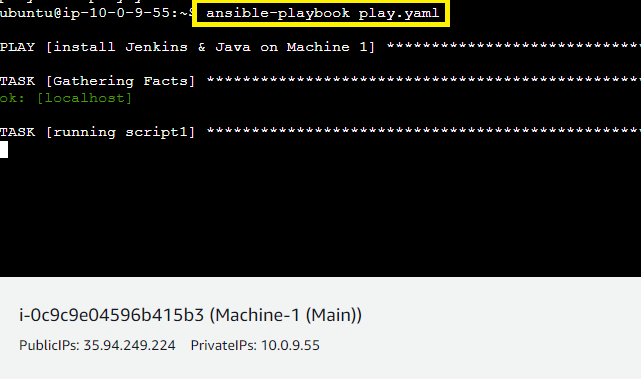
**Step 3: Check** the **syntax using** the **command: ansible-playbook play.yaml --syntax-check**

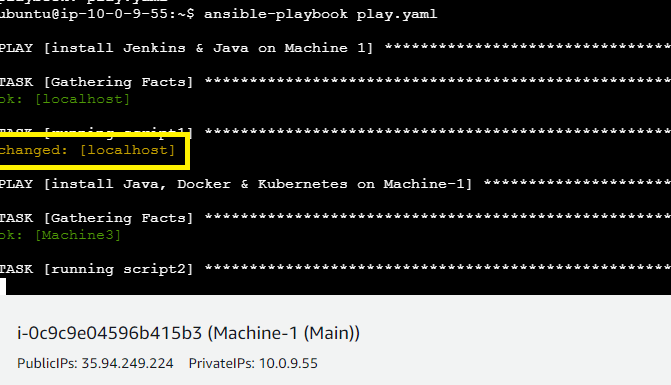
**Syntax** is **“OK” here. No Problem.**

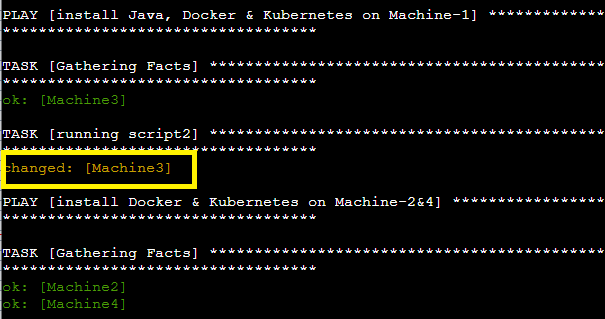
****

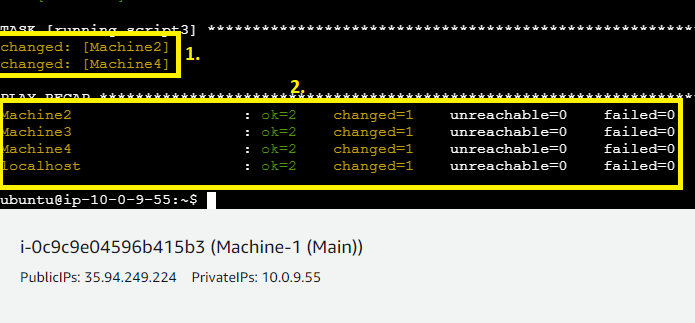
**Step 4: Now, we will run** the **“play.yaml” using** the **command: ansible-playbook play.yaml**

**Execution** of **“Scripts”** will **getting started.**

****

****

****

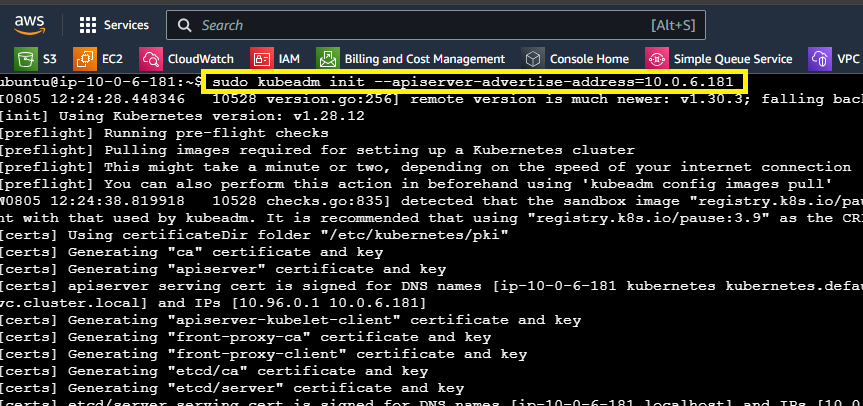
****

**G. Configure Kubernetes Slaves Properly on Machine-3**

**Step 1: Go** to **“Machine-3 (KBM)”** & **paste** this **command** to **initialize** the **“kubeadm”.**

**sudo kubeadm init --apiserver-advertise-address=** **10.0.6.181**

**Press “enter”** from the **keyboard.**

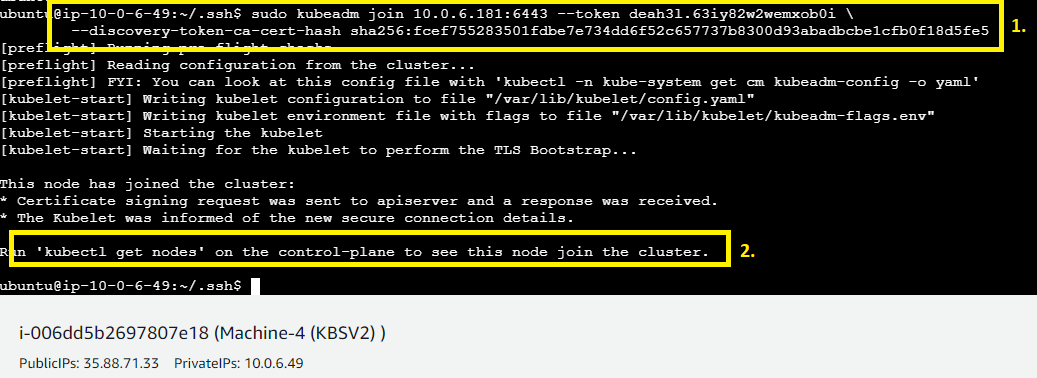
****

**Step 2: Copy** this **token** & **Paste** it into **Machine-2 & Machine-4** one by one **using “sudo” command.**

**sudo kubeadm join 10.0.6.181:6443 --token deah3l.63iy82w2wemxob0i \**

**--discovery-token-ca-cert-hash sha256:fcef755283501fdbe7e734dd6f52c657737b8300d93abadbcbe1cfb0f18d5fe5**

****

****

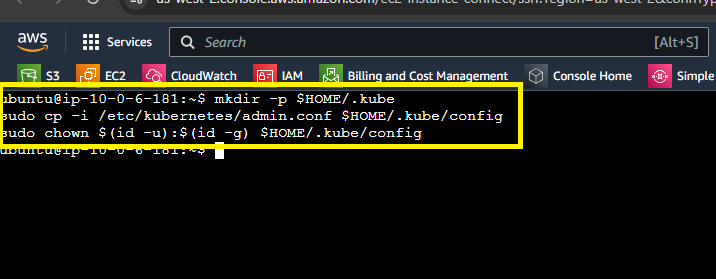
**Both** the **Machine 2 & 4 successfully joined** the **nodes.**

**Step 3: Paste these commands in “Master” Nodes:**

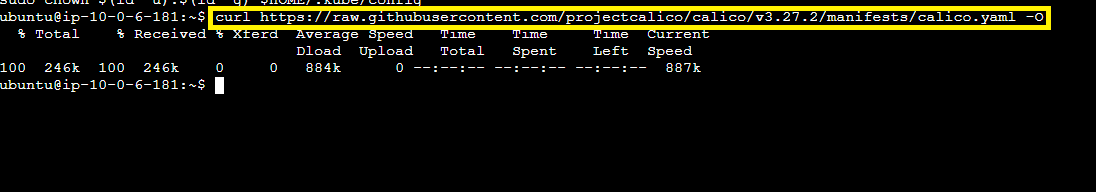
**mkdir -p $HOME/.kube**

**sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

**sudo chown $(id -u):$(id -g) $HOME/.kube/config**

****

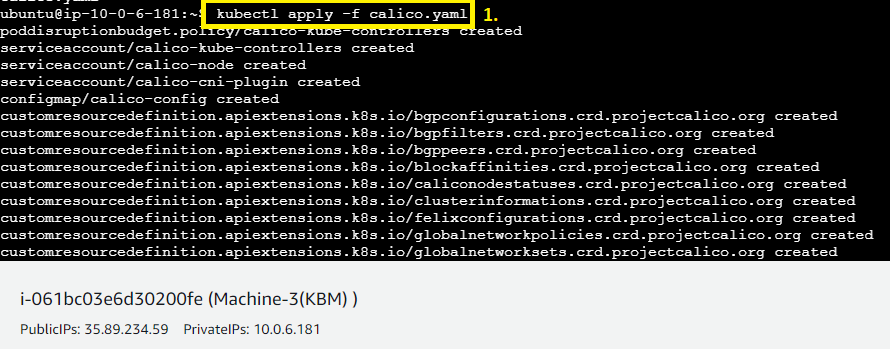
**Step 4: Install the “Calico Network” to run the cluster using this command: curl https://raw.githubusercontent.com/projectcalico/calico/v3.27.2/manifests/calico.yaml -O**

****

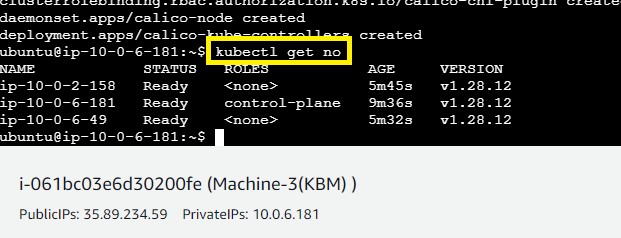
**Step 5: Run this command: ls, your ‘calico.yaml” file will be shown.**

****

**Step 6: Run this command: kubectl apply -f calico.yaml**

****

**Step 7: All** the **deployment** has been **successfully created. Now, run** the **command: kubectl get no** to **get** all the **nodes.**

****

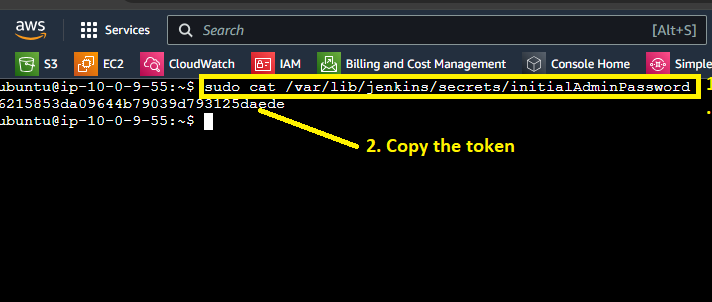
**H. Configure Jenkins Setup Properly Here on Machine-1**

**Step 1: Copy** the **“Machine-1 (Main) Public IP Address”** with **8080** in **browser address bar. A command** will be **given, copy** the **command from here.**

****

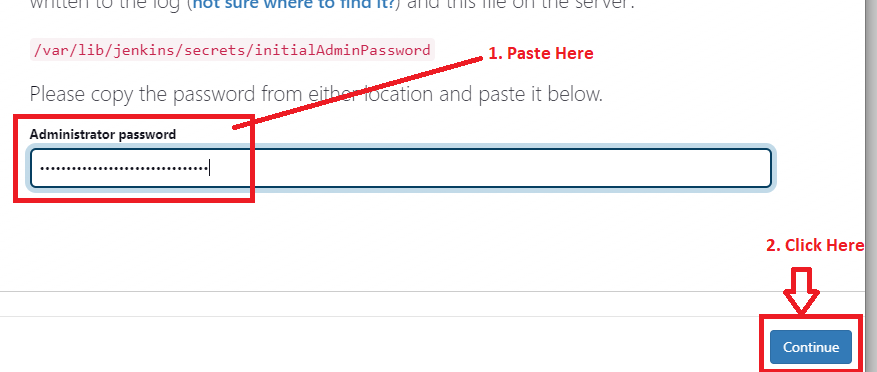
**Step 2: Go to “Machine-1 (Main)” & paste the command with “sudo cat”.**

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

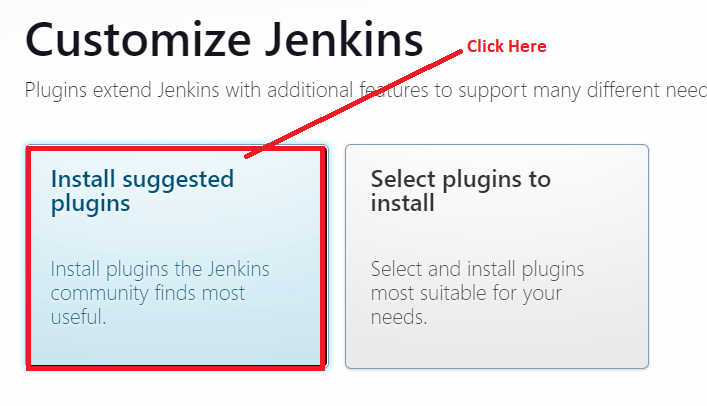
****

**A token will be given. Copy this token from here.**

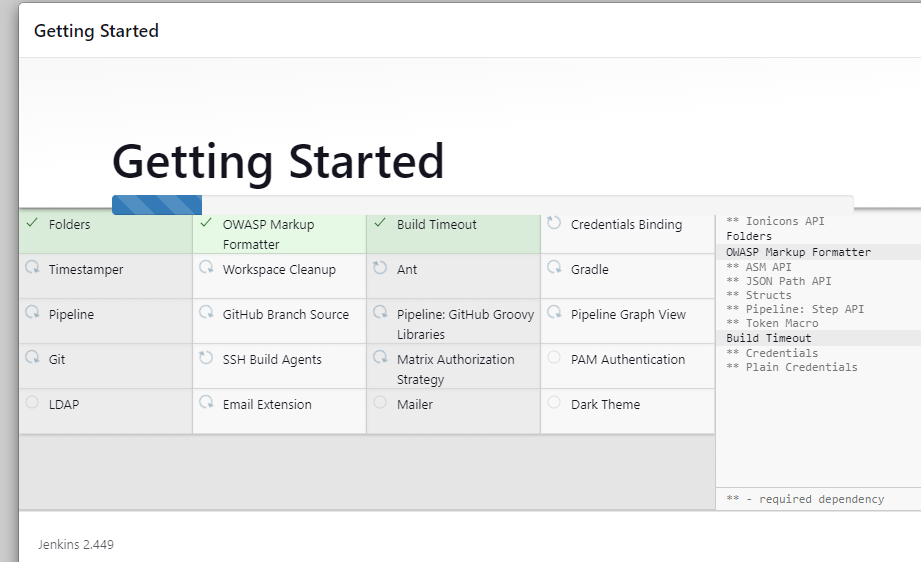
**Step 3: Paste** the **token** in **“Administrator Password” section** & **click** on **“Continue”.**

****

**Step 4: Click** on **“Install Suggested Plugins”.**

****

**Step 5: Plugins installation** will be **automatically started.**

****

**Step 6: Create** the **user** by **filling these given details:**

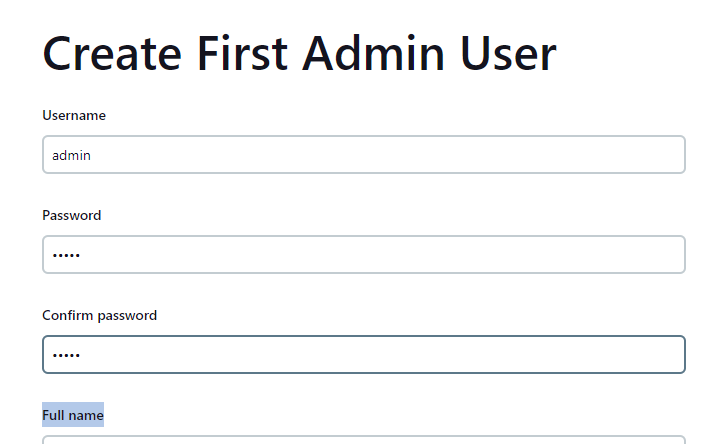
**Username: - admin**

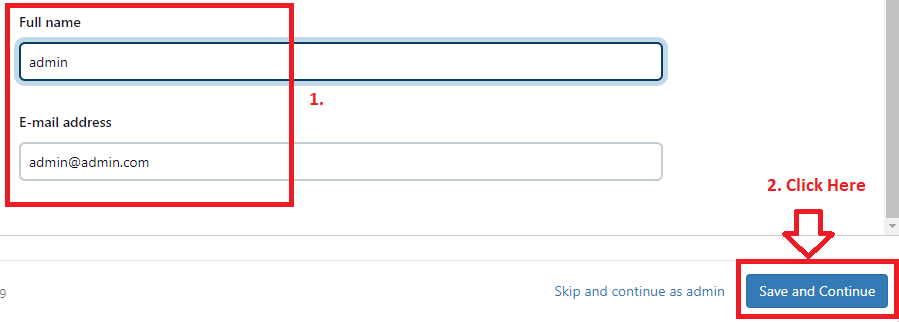
**Password: - admin**

**Confirm Password: - admin**

**Full name: - admin**

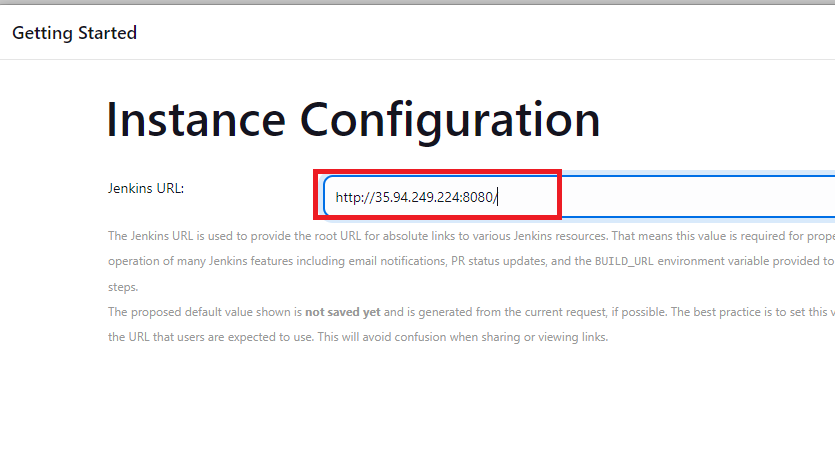
**E-mail address:** [**admin@admin.com**](mailto:admin@admin.com)

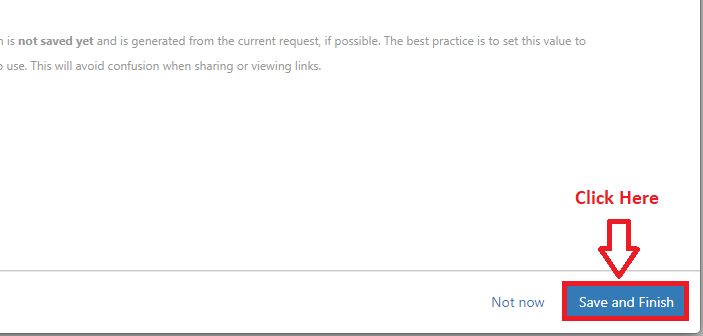
****

****

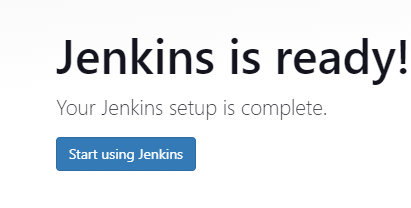
**Click** on **“Save and Continue”.**

**Step 7: Click** on **“Save and Finish”.**

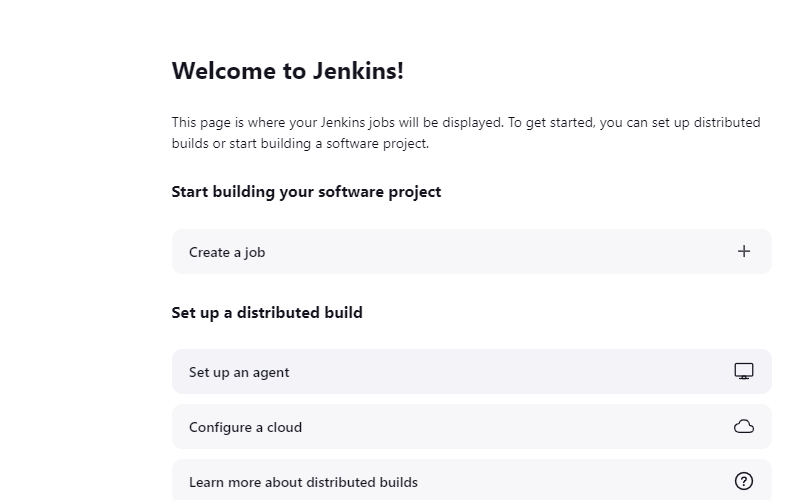
****

****

**Step 8: Click** on **“Start using Jenkins”.**

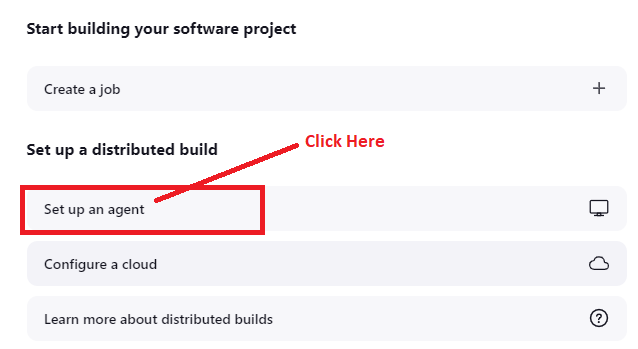
****

**Step 9: “Jenkins Dashboard”** will be **set up successfully.**

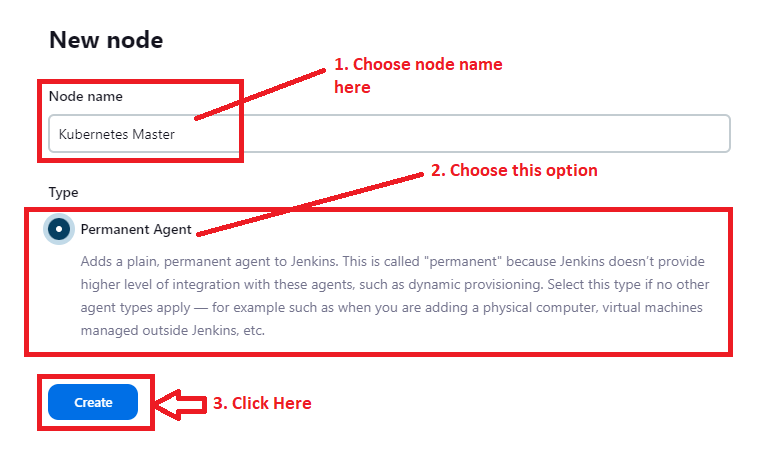
****

**I. Add “Kubernetes Master (Machine-3)” as a Node Here**

**Step 1: Click** on **“Set up an agent”.**

****

**Step 2: Choose “Node name”** as **“Kubernetes Master”** & **“Type”** as **“Permanent Agent”.**

****

**Step 3: Choose the following options here:**

**Description: - Kubernetes Master**

**Remote root directory: - /home/ubuntu/jenkins**

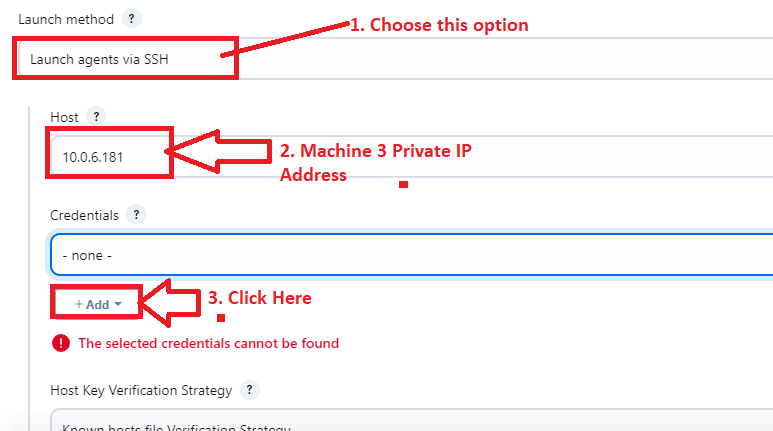
**Label:- Kubernetes-Master**

****

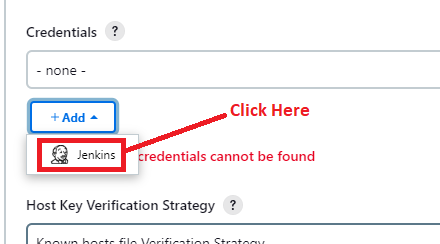
**Launch Method: - Launch agents via SSH**

**Host: - 10.0.6.181 (Master Private IP Address)**

**Click** on **“Add”** in **“Credentials”.**

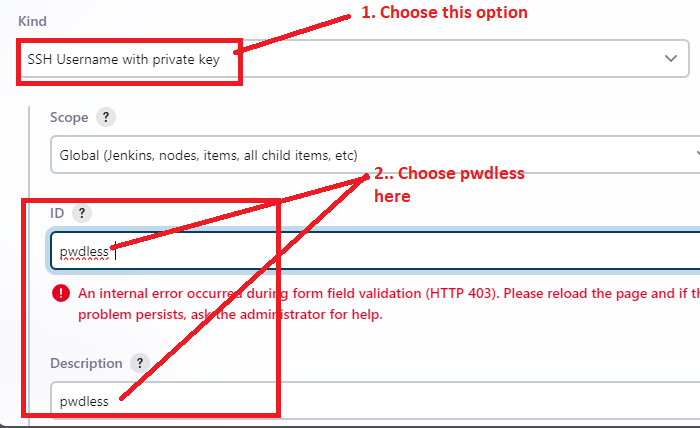
****

**Click on “Jenkins”.**

****

**Step 4: Choose “kind”** as **“SSH Username** with **private key”** with following fields:

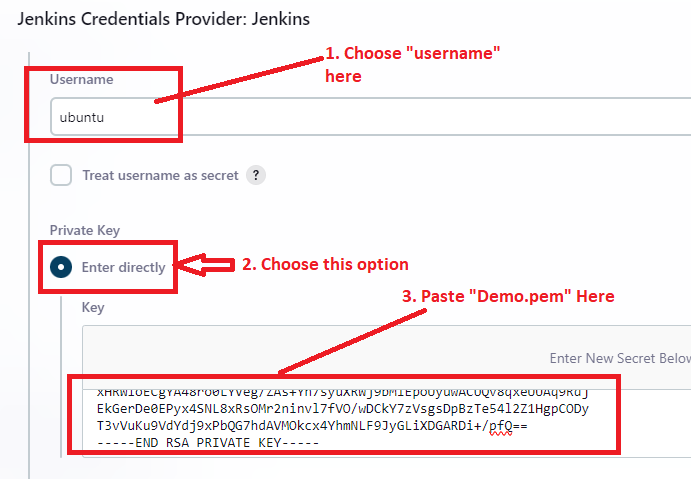
**ID & Description: -** pwdless

****

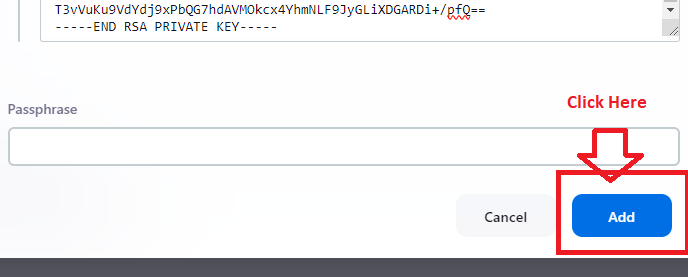
**Step 5: Choose “Username”** as **“Ubuntu”.**

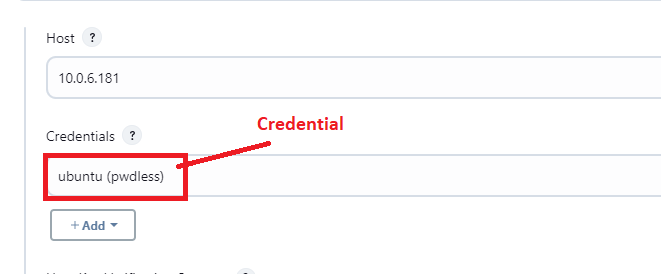
**Choose “Enter directly”** & **click** on **“Add”.**

**Paste the “Demo.pem” key content here.**

****

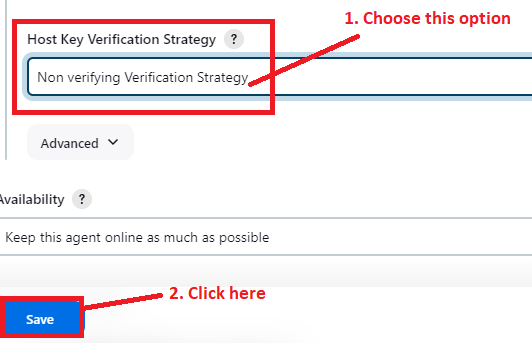
**Step 6: Click on “Add”.**

****

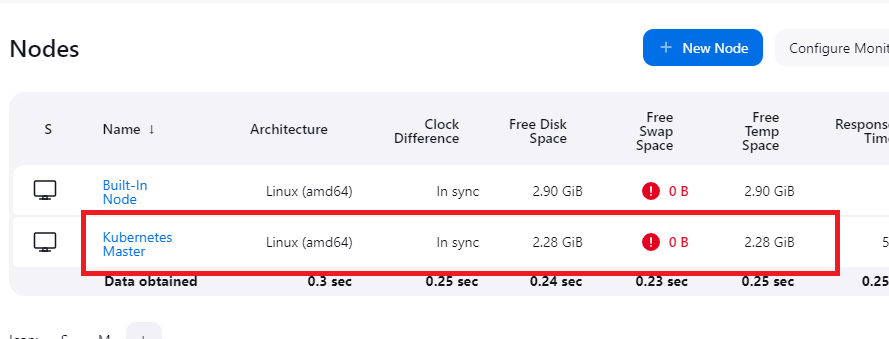
****

**Step 7: Choose “Host Key Verification Strategy”** as **“Non verifying Verification Strategy”.**

**Click** on **“Save”.**

****

**Step 8: Your “Kubernetes Master” has been successfully added as a “Node”.**

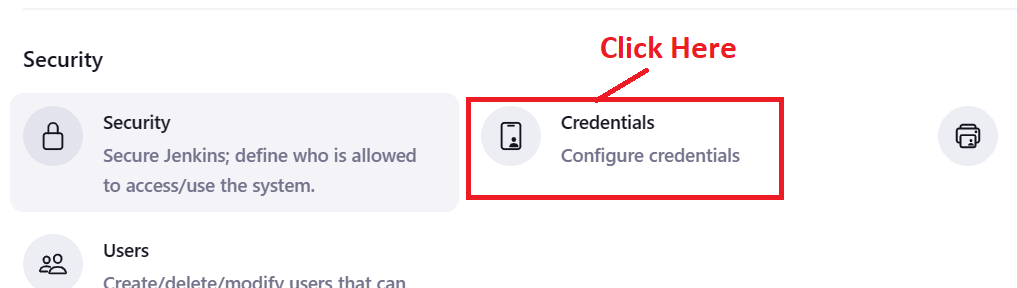
****

**J. Create Jenkins Credentials for Pipeline Creation**

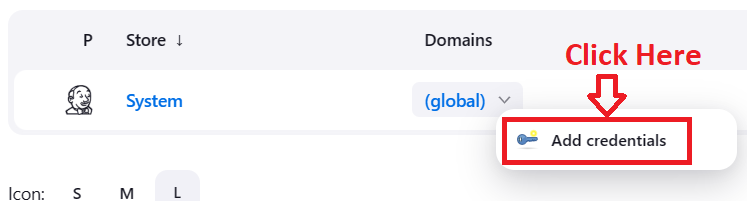
**Step 1: Go to “Manage Jenkins”.**

****

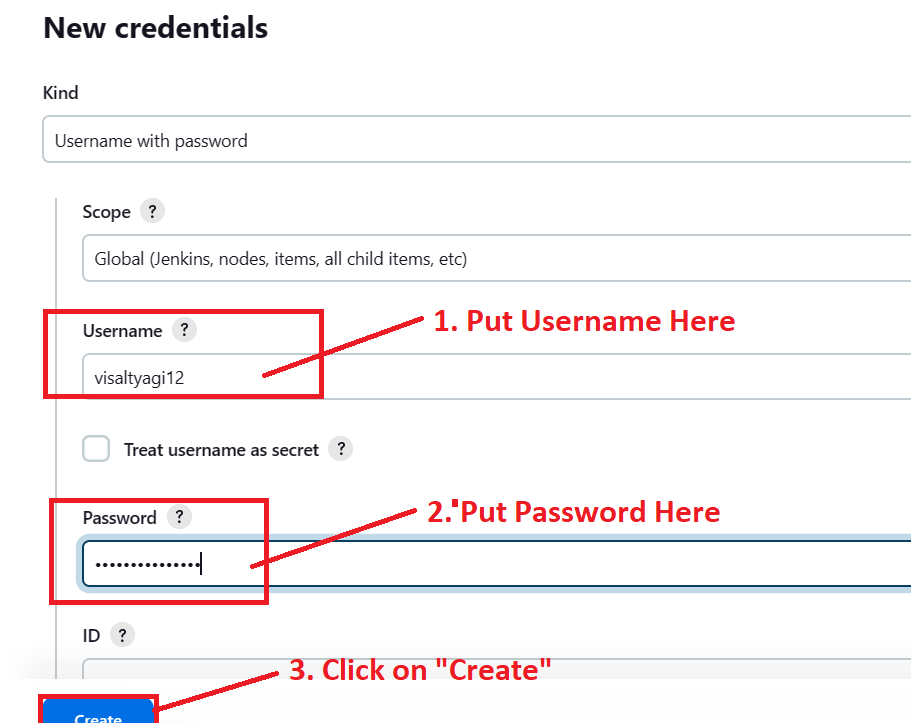
**Step 2: Click** on **“Credentials”.**

****

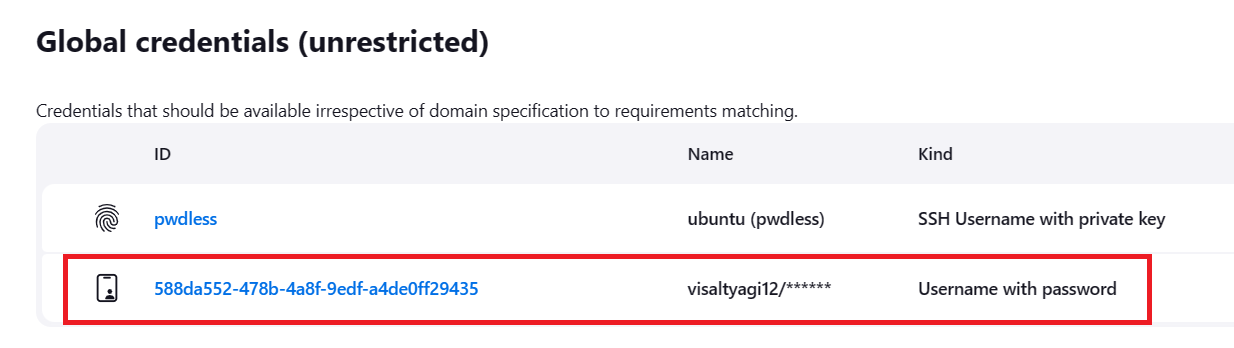
**Step 3: Click on “Global>Add Credentials”.**

****

**Step 4: Put DockerHub username & password here.**

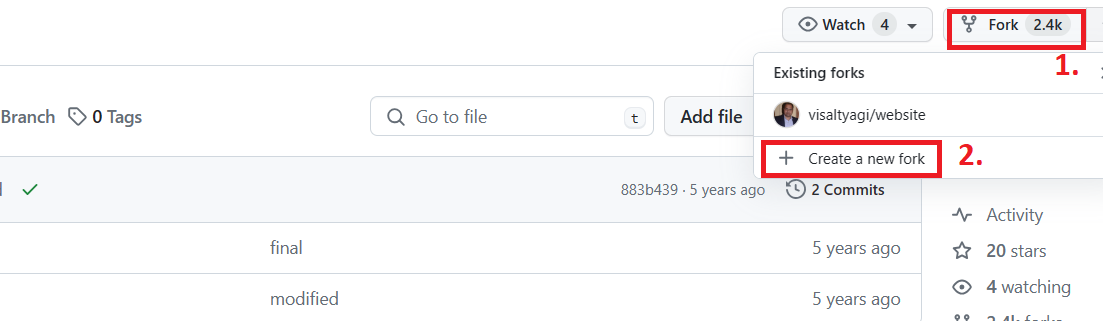
****

**Step 5: “DockerHub” Credentials has been successfully created.**

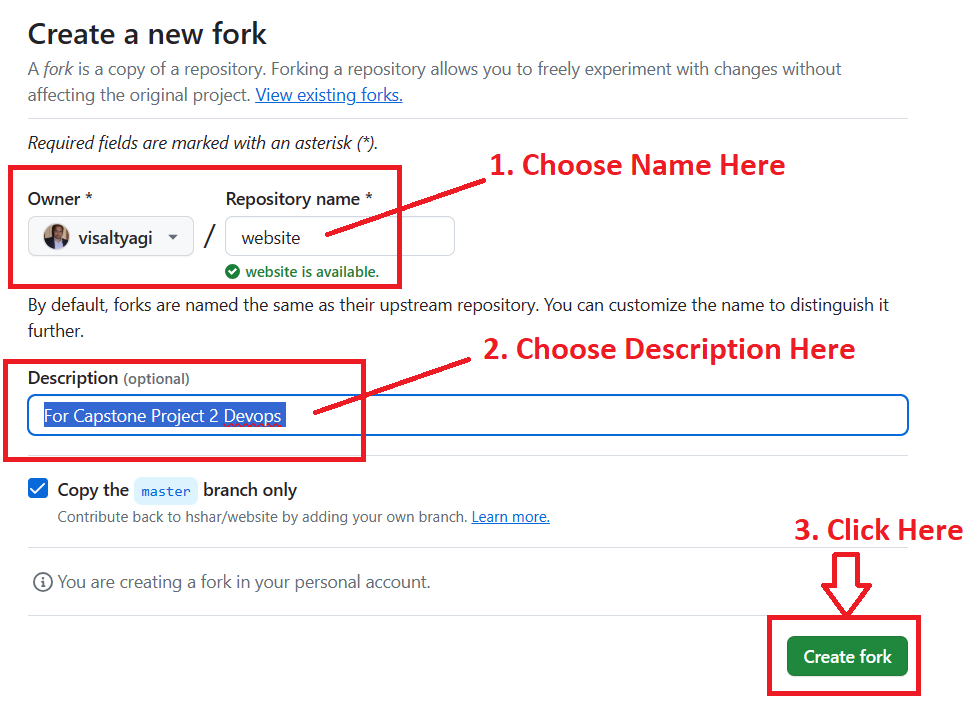
****

**K. Fork the Repository in the GitHub Account**

**Step 1: Click on “Fork> Create a new fork”.**

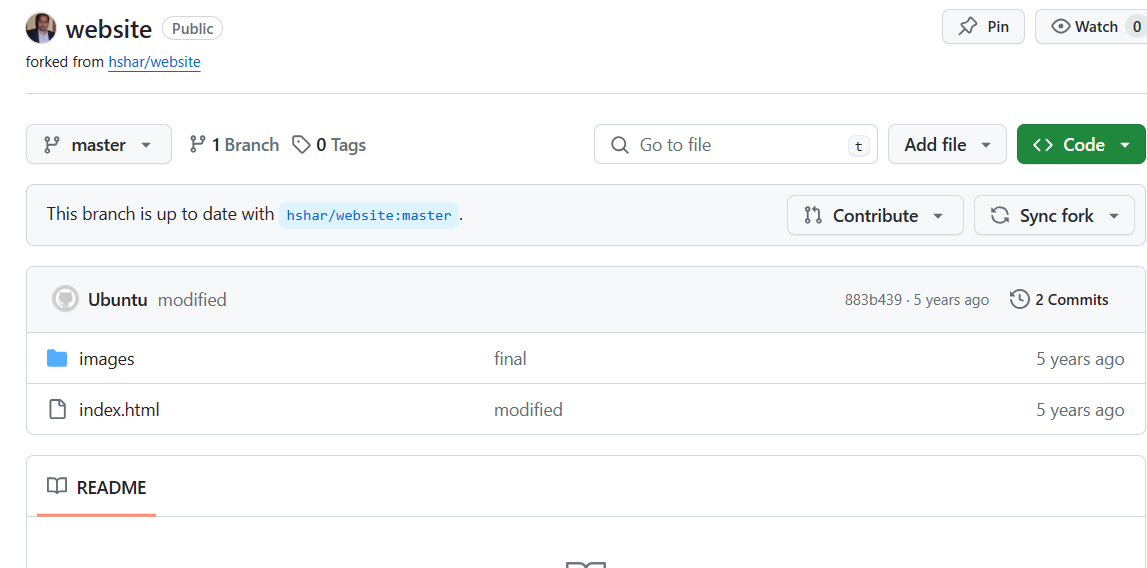
****

**Step 2: Choose “Repository Name” as “website”, while “Description” as “For Capstone Project 2 Devops”.**

****

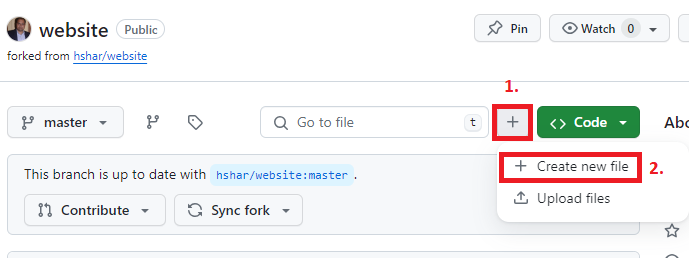
**Click on “Create fork”.**

**Step 3: Repository has been successfully forked.**

****

**L. Create a Docker file in Given GitHub Repository**

**Step 1: Click** on **“+”>” Create new file”.**

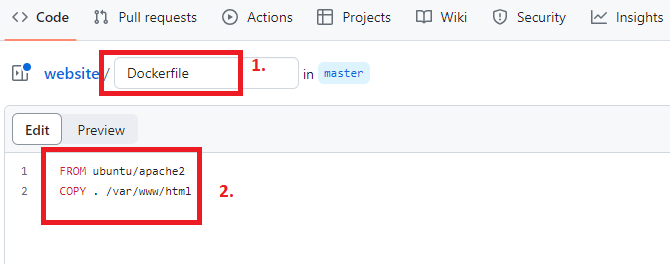
****

**Step 2: Paste this content here:**

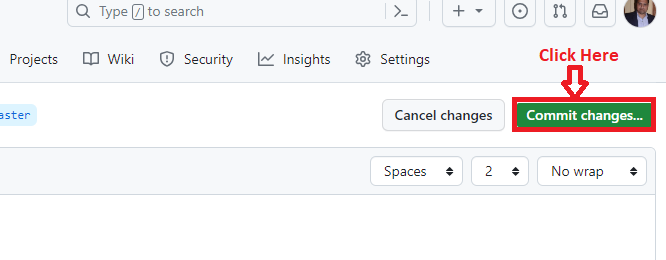
**FROM ubuntu/apache2**

**COPY . /var/www/html**

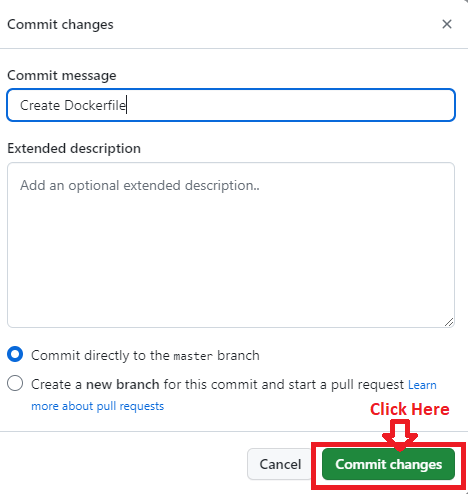
**Put file name as “Dockerfile”.**

****

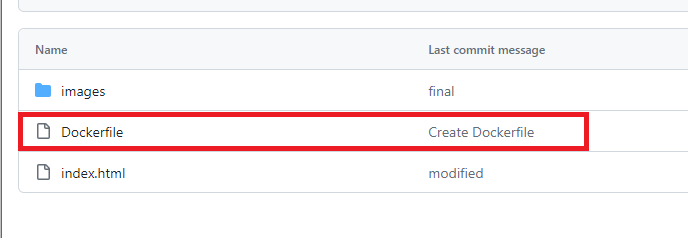
**Step 3: Click on “Commit Changes”.**

****

**Step 4: Again, click on “Commit Changes”.**

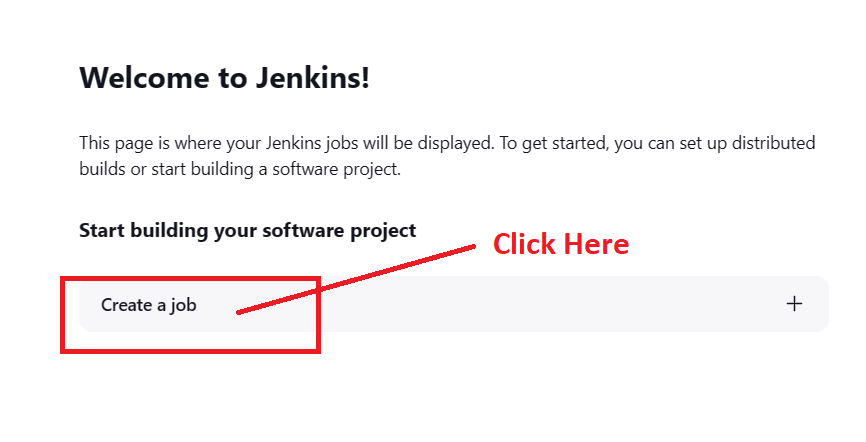
****

**Step 5: “Dockerfile” will be successfully created.**

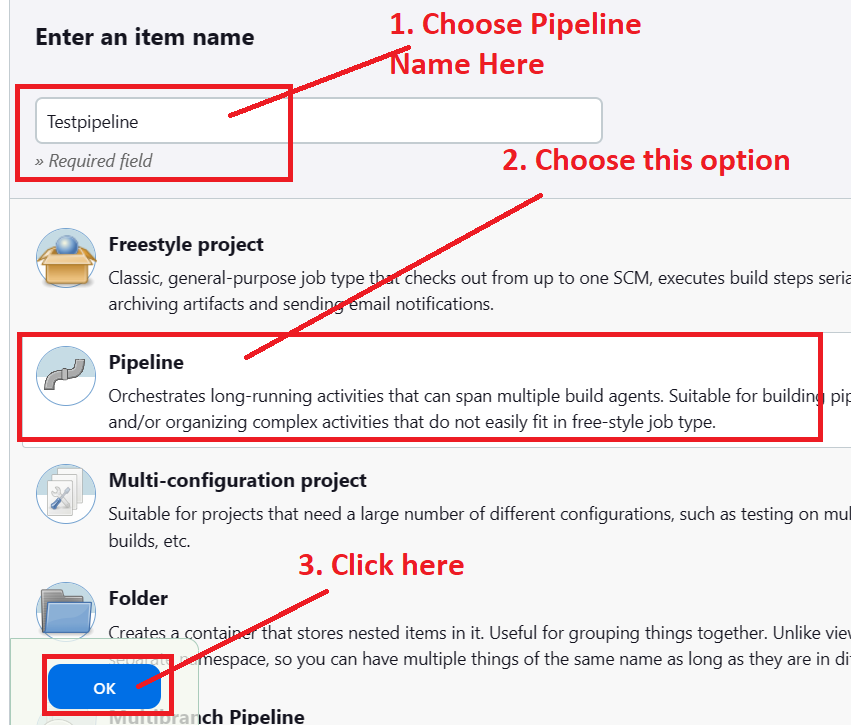
****

**M. Create a Pipeline to Automate the Tasks**

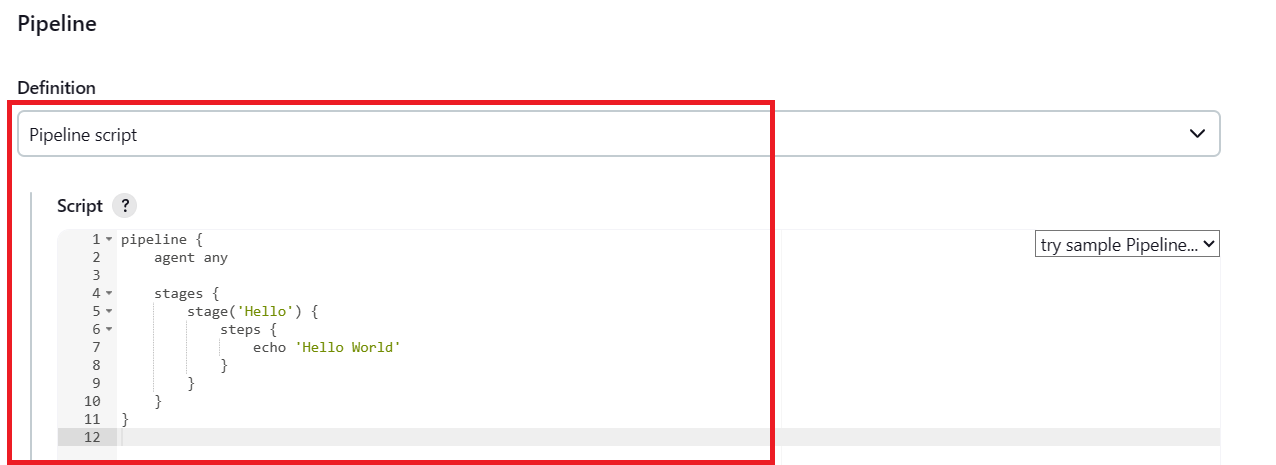
**Step 1: Click** on **“Create a job”.**

****

**Step 2: Choose “Item Name” as “Testpipeline” with “Pipeline” option.**

****

**Step 3: Go to “Pipeline” section & choose “Hello World” script here.**

****

**Step 4: Now, we will use this script to check that pipeline script is working properly or not.**

**pipeline {**

**agent none**

**environment {**

**DOCKERHUB\_CREDENTIALS=credentials("588da552-478b-4a8f-9edf-a4de0ff29435")**

**}**

**stages {**

**stage('Hello') {**

**steps {**

**echo 'Hello World'**

**}**

**}**

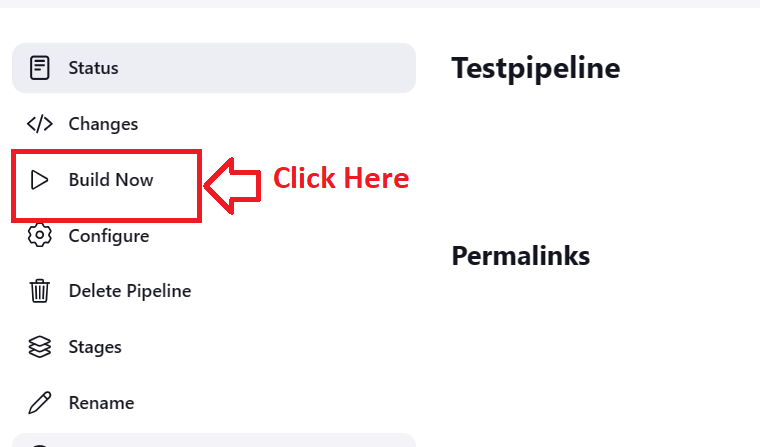
**}**

**}**

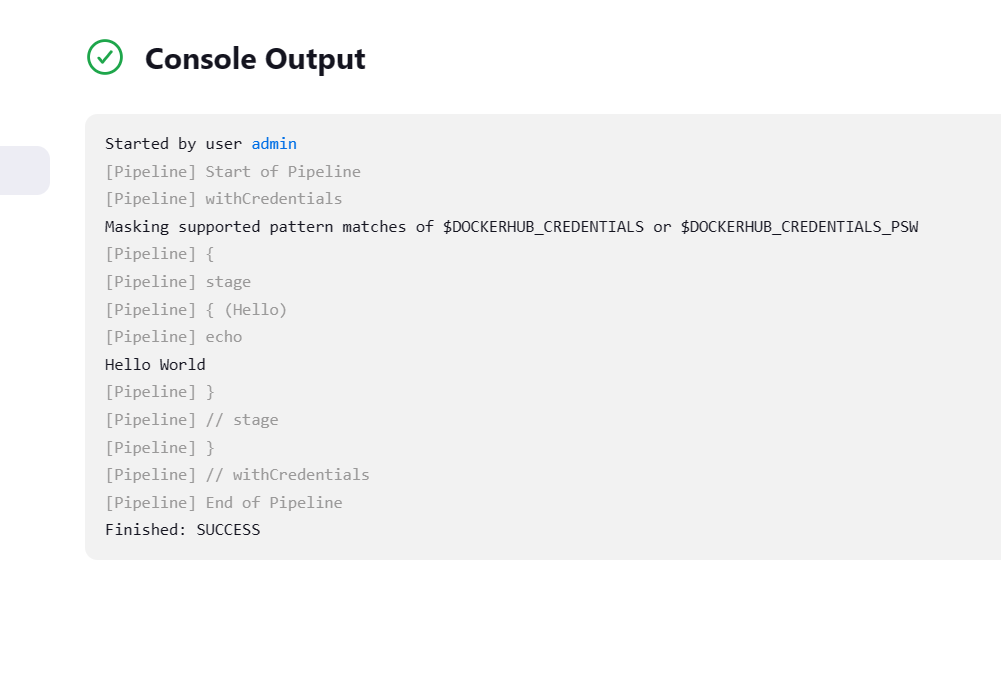
**Click on “Save”.**

****

**Step 5: Click** on **“Build Now”.**

****

**Step 6: The Sample “Hello World” will be successfully created.**

****

**Step 7: Again, paste this script in “Pipeline” section. Click on “Save”.**

**pipeline {**

**agent none**

**environment {**

**DOCKERHUB\_CREDENTIALS=credentials('588da552-478b-4a8f-9edf-a4de0ff29435')**

**}**

**stages {**

**stage('Hello') {**

**steps {**

**echo 'Hello World'**

**}**

**}**

**stage('Git') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

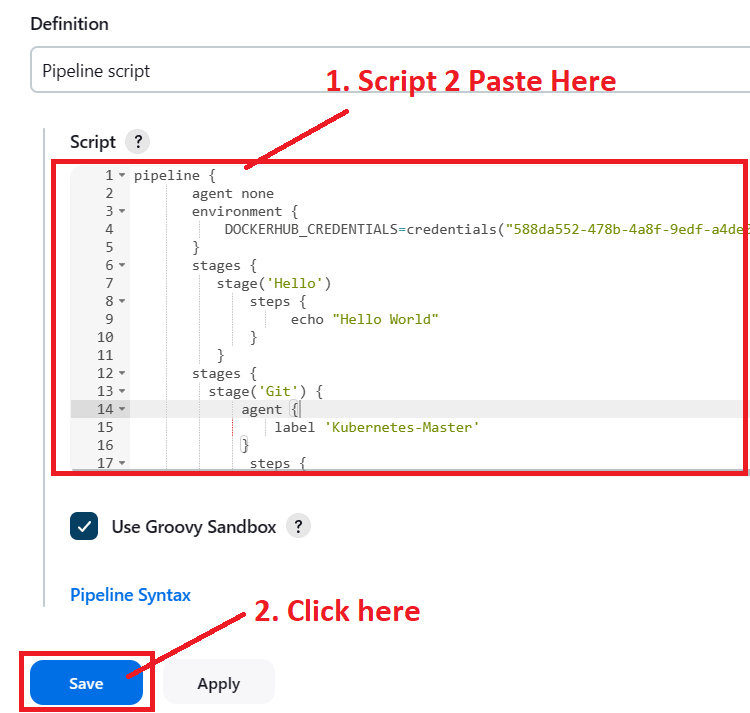
**steps {**

**git 'https://github.com/visaltyagi/website.git'**

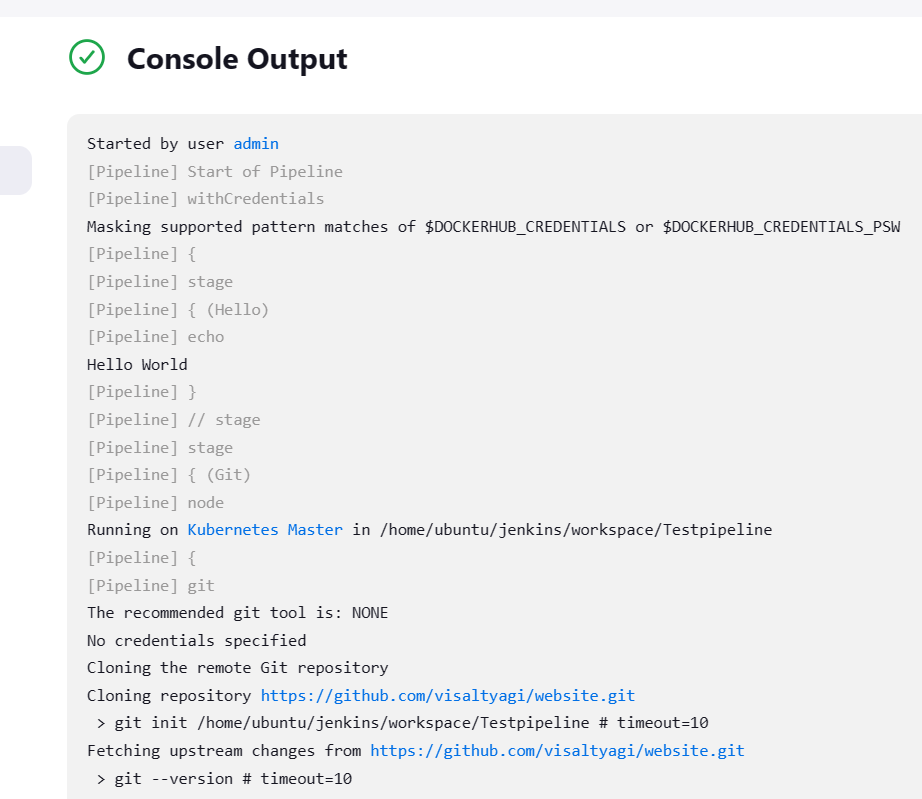
**}**

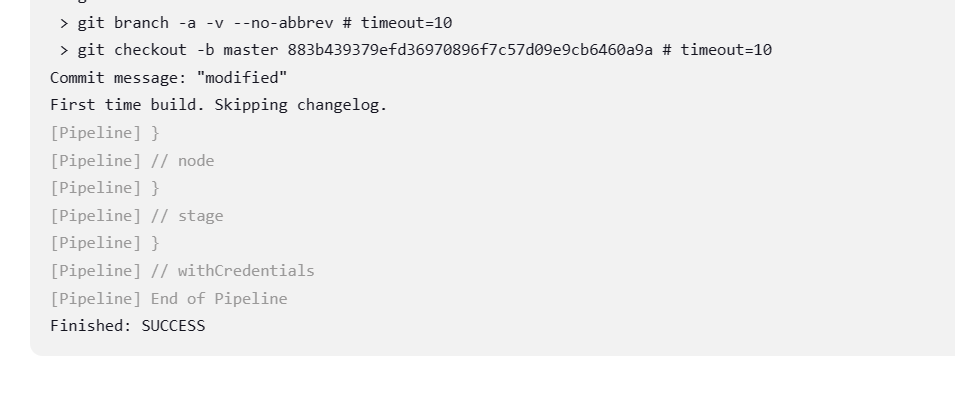
**}**

**}**

**}**

**Step 8: Click on “Build Now”. Build will be successfully created.**

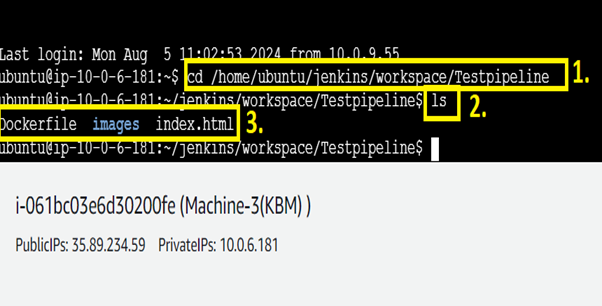
****

****

**Step 9: The “GitHub Repository Content” will automatically come in “Kubernetes Master (Machine-3)”.**

**Run this test using the command: cd /home/ubuntu/Jenkins/workspace/Testpipeline**

**Do “ls” here. You will find the content here.**



**Step 10: Now, we will push the “Docker Hub Image” using the pipeline code.**

**pipeline {**

**agent none**

**environment {**

**DOCKERHUB\_CREDENTIALS=credentials('588da552-478b-4a8f-9edf-a4de0ff29435')**

**}**

**stages {**

**stage('Hello') {**

**steps {**

**echo 'Hello World'**

**}**

**}**

**stage('Git') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

**steps {**

**git 'https://github.com/visaltyagi/website.git'**

**}**

**}**

**stage('Docker') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

**steps {**

**sh 'sudo docker build /home/ubuntu/jenkins/workspace/Testpipeline -t visaltyagi12/project2'**

**sh 'sudo echo $DOCKERHUB\_CREDENTIALS\_PSW | sudo docker login -u $DOCKERHUB\_CREDENTIALS\_USR --password-stdin'**

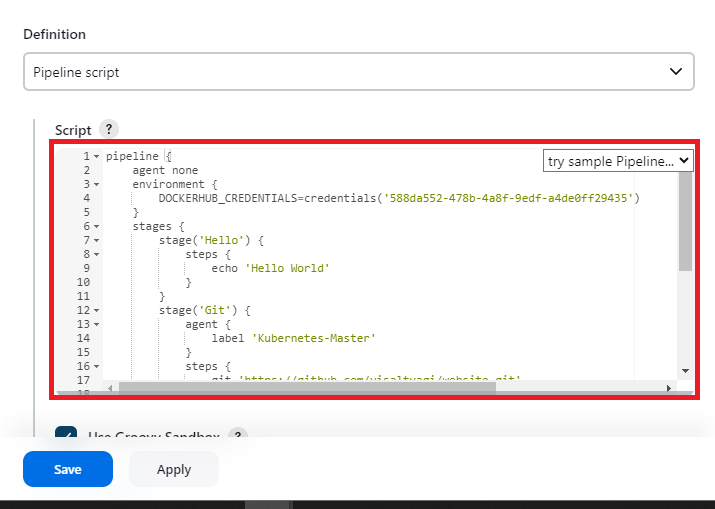
**sh 'sudo docker push visaltyagi12/project2'**

**}**

**}**

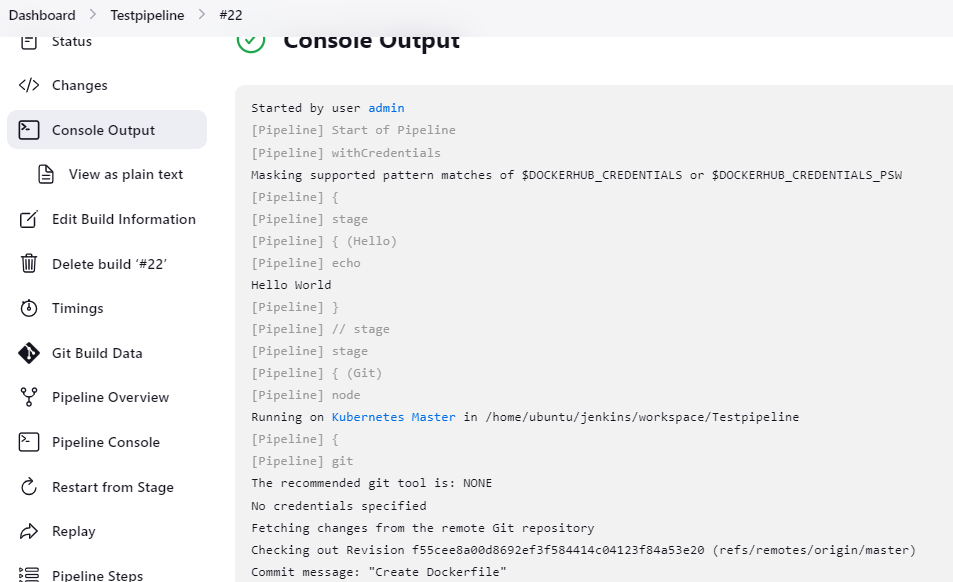
**}**

**}**

****

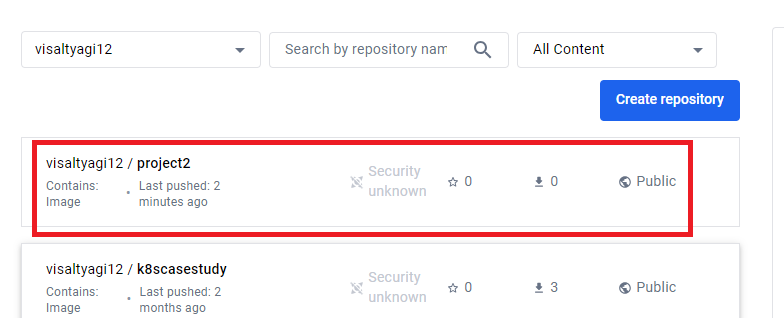
**Click** on **“Save”.**

**Step 11: Click on “Build Now”. Build will be successfully created & “Dockerfile” will be successfully pushed to “Docker Hub” Account.**

****

****

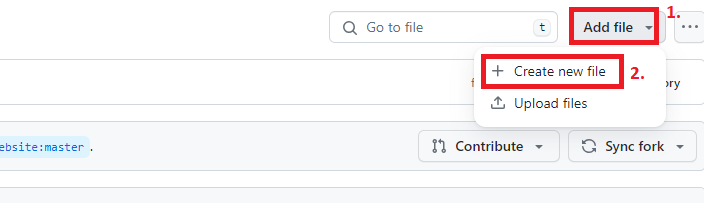
**Step 12: Login into your DockerHub Account & you will notice that “visaltyagi12/project2” will be successfully pushed to Dockerhub.**

****

**Step 13: Now, we will create the “deployment.yaml” & “service.yaml” file to deploy the website using the “Kubernetes” tool.**

**Go to “GitHub” account & click on “Add file”.**

**Click on “Create new file”.**

****

**Step 14: Paste this content here:**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**labels:**

**app: nginx**

**spec:**

**replicas: 2**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

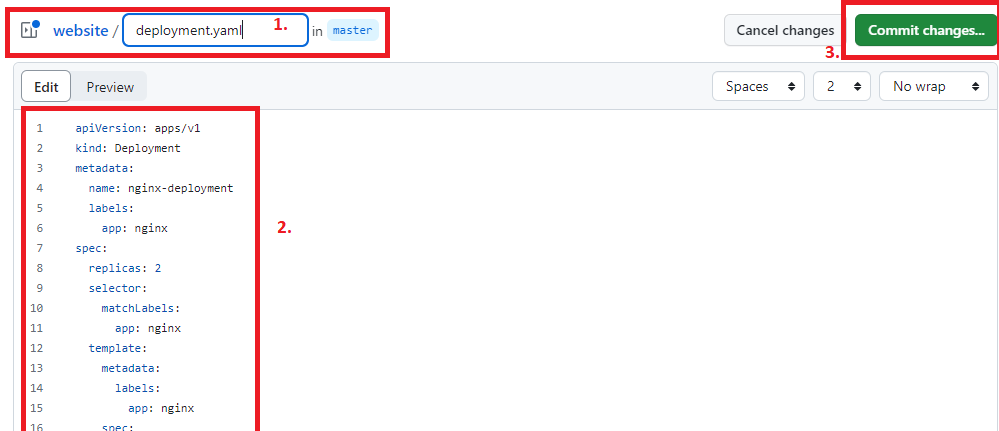
**- name: nginx**

**image: visaltyagi12/project2:latest**

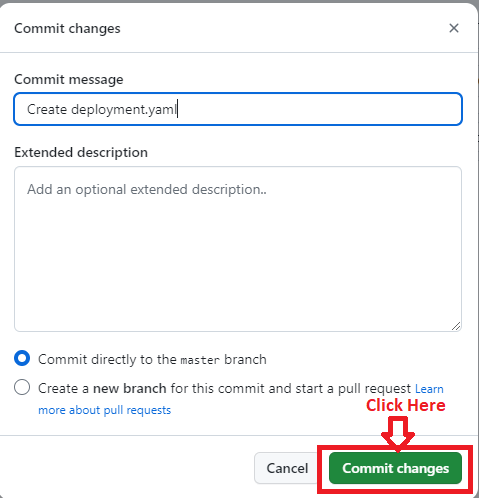
**ports:**

**- containerPort: 80**

**Put name** as **“deployment.yaml”** & **click** on **“Commit Changes”.**

****

**Step 15: Click** on **“Commit Changes”.**

****

**Step 16: Create a “service.yaml” file for deploying website over node port 30008.**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: my-service**

**spec:**

**type: NodePort**

**selector:**

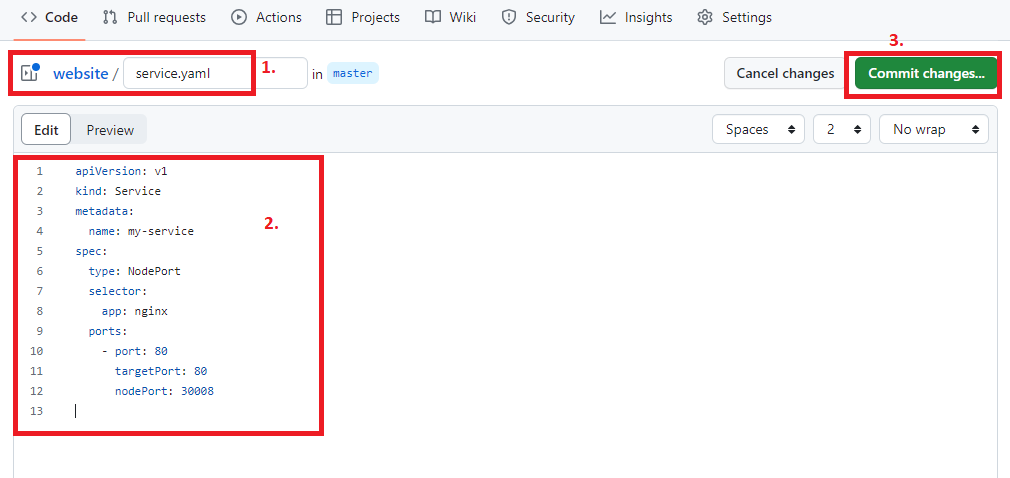
**app: nginx**

**ports:**

**- port: 80**

**targetPort: 80**

**nodePort: 30008**

****

**Step 17: Do “Commit Changes” & your both yaml files will be shown in “Git Hub Repository”.**

****

**Step 18: Go to “Configure” in “Jenkins” & paste the following content here:**

**pipeline {**

**agent none**

**environment {**

**DOCKERHUB\_CREDENTIALS=credentials('588da552-478b-4a8f-9edf-a4de0ff29435')**

**}**

**stages {**

**stage('Hello') {**

**steps {**

**echo 'Hello World'**

**}**

**}**

**stage('Git') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

**steps {**

**git 'https://github.com/visaltyagi/website.git'**

**}**

**}**

**stage('Docker') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

**steps {**

**sh 'sudo docker build /home/ubuntu/jenkins/workspace/Testpipeline -t visaltyagi12/project2'**

**sh 'sudo echo $DOCKERHUB\_CREDENTIALS\_PSW | sudo docker login -u $DOCKERHUB\_CREDENTIALS\_USR --password-stdin'**

**sh 'sudo docker push visaltyagi12/project2'**

**}**

**}**

**stage('K8s') {**

**agent {**

**label 'Kubernetes-Master'**

**}**

**steps {**

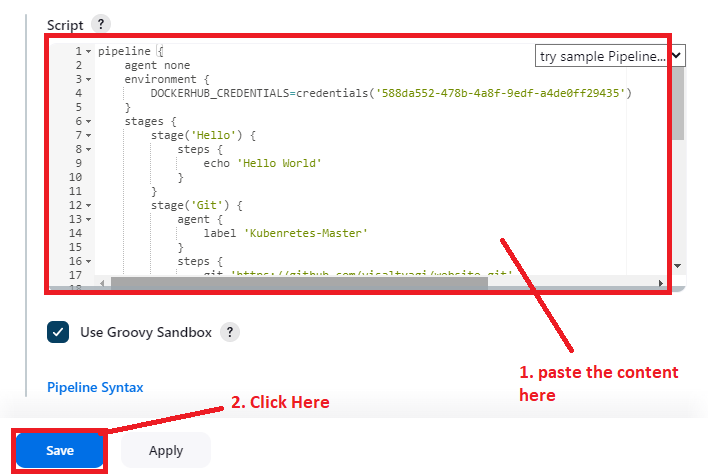
**sh 'kubectl apply -f deployment.yaml'**

**sh 'kubectl apply -f service.yaml'**

**}**

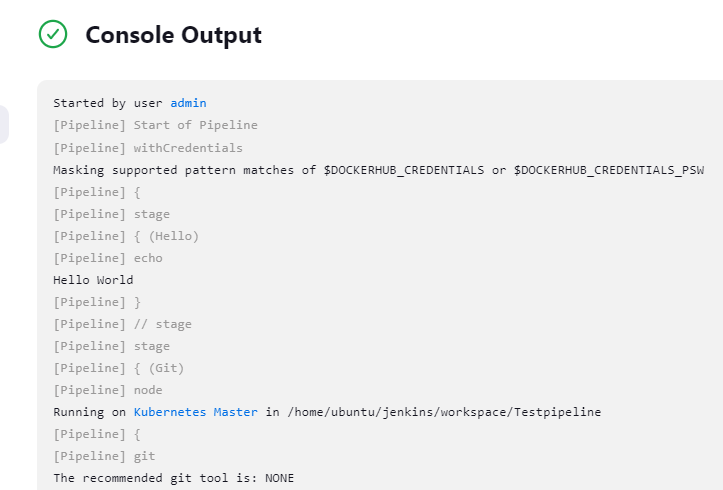
**}**

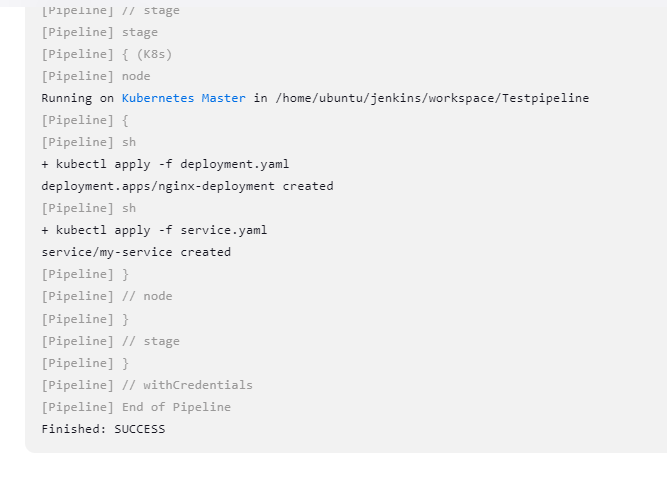
**}**

**}**

**Click on “Save”.**

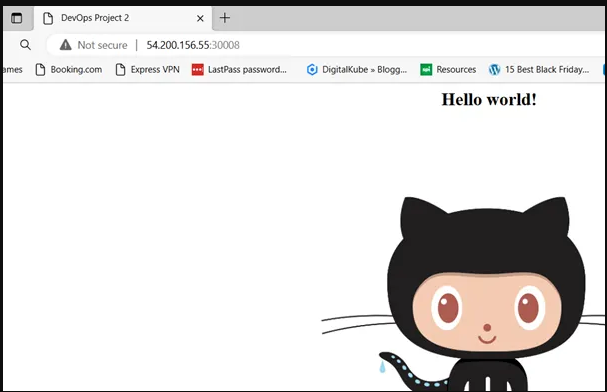
**Step 19: Again, click on “Build Now”. Now, Build will be successfully created & website has been successfully deployed over Slaves through Kubernetes Architecture.**

****

****

**Step 20: Paste Both Slaves IP one by one in the browser address bar & website will be successfully deployed through “Kubernetes”.**



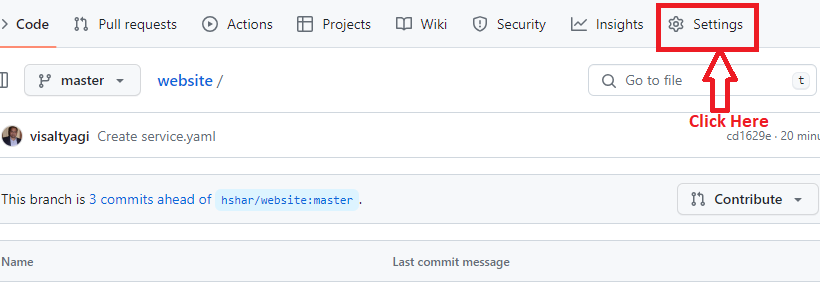






**N. Automate the Pipeline using Github Webhooks.**

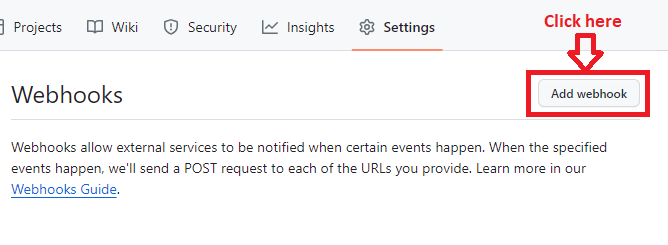
**Step 1: Go** to **“GitHub”** & **click** on **“Settings”.**

****

**Step 2: Click** on **“Webhooks”.**

****

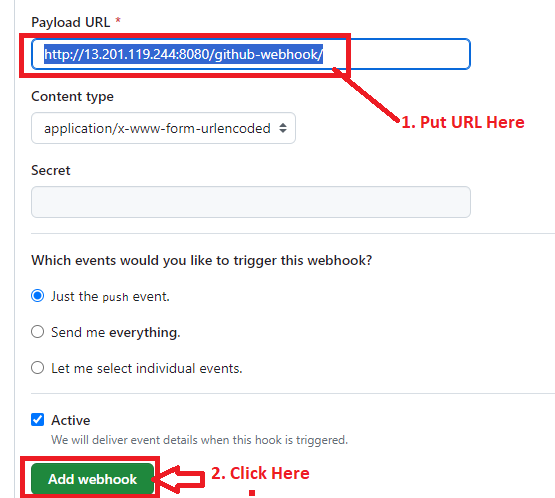
**Step 3: Click** on **“Add webhook”.**

****

**Step 4: Choose “Payload URL” as** [**http://13.201.119.244:8080/github-webhook/**](http://13.201.119.244:8080/github-webhook/)

**Click on “Add webhook”.**

**Leave other fields as it is.**

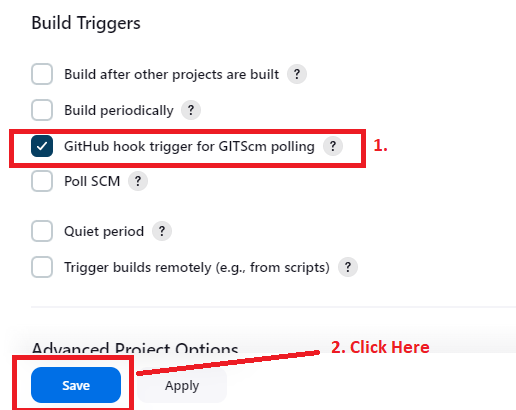
****

**Step 5: Webhook has been successfully created.**

****

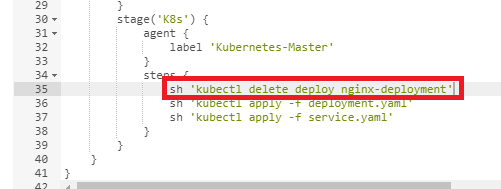
**Step 6: Go** to **“Testpipeline”** & **choose “GitHub hook trigger for GITScm polling”** in **“Build Triggers”.**

**Click on “Save”.**

****

**O. Do the Changes & Test the Pipeline**

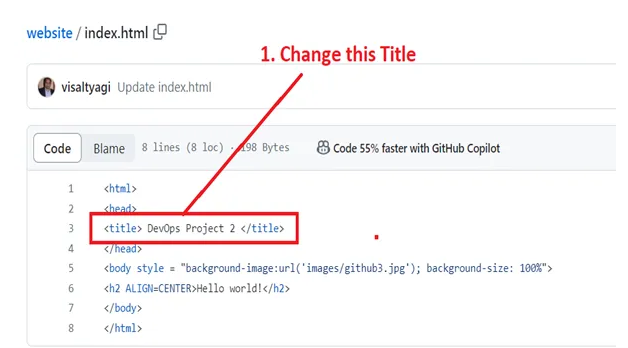
**Step 1: Go to “Configure” & add this line to “pipeline code”. Save the file.**

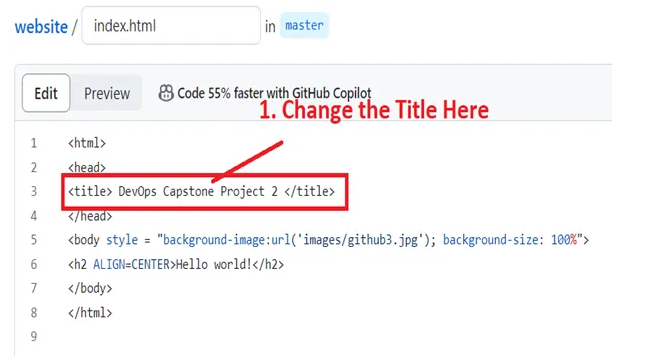
****

**Step 2: Now, go to “index.html” file in “GitHub Repository”.**

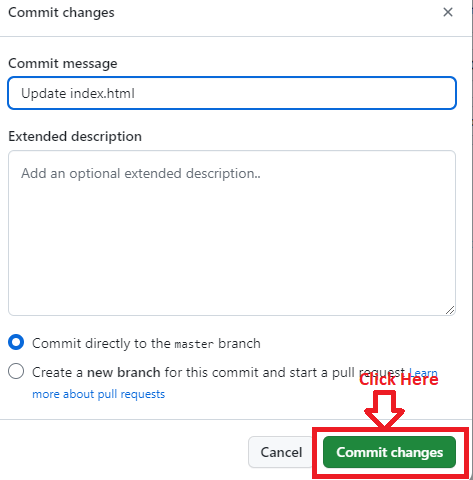
**Change “Title” from “DevOps Project 2” to “DevOps Capstone Project 2”.**

**Click on “Commit Changes”.**

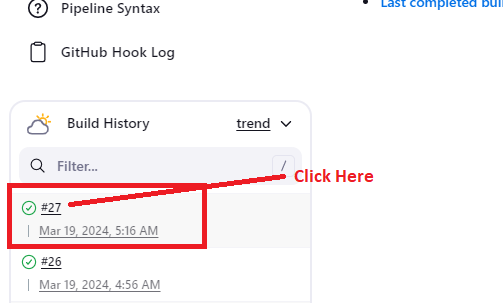




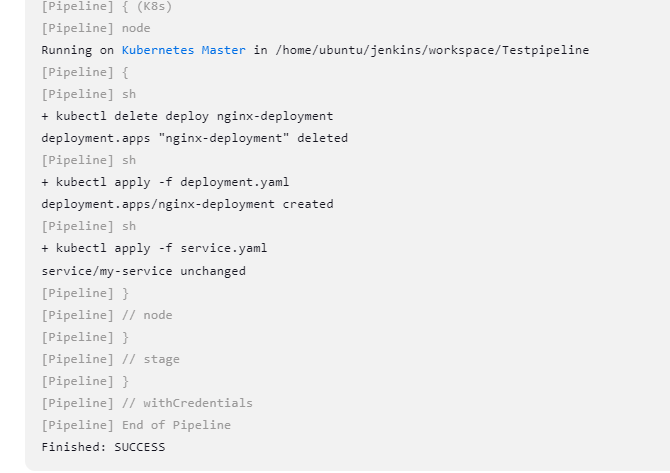
**Step 3: Again, click** on the **“Commit Changes”.**

****

**Step 4:** An automatic build will be starting created.

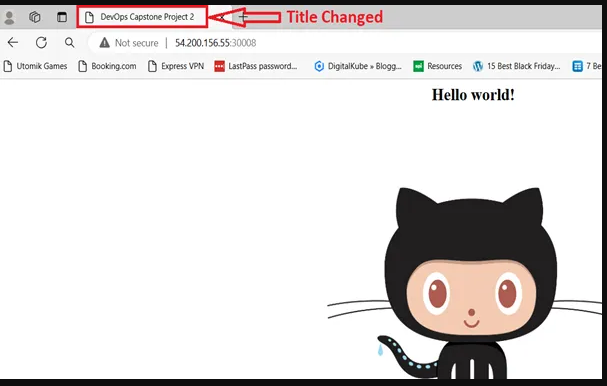
****

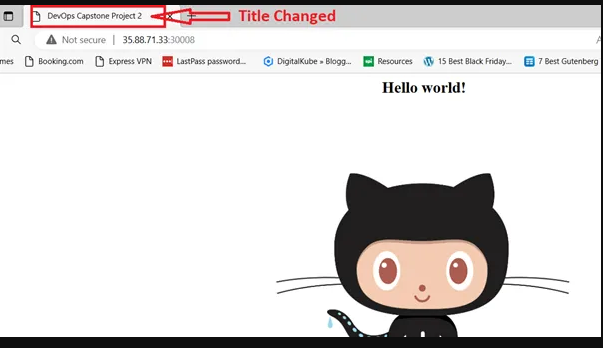
**Step 5: Pipeline will be shown to you.**

****

**Step 6: Go** to **“Browser”** & **again refresh** the **IP Address** of **“Machine-2” & “Machine-4”.**

**You will notice that in “title”, now “Hello Intellipaat” is showing.**





**🡨------------------------------------Project 2 Completed--------------------------🡪**