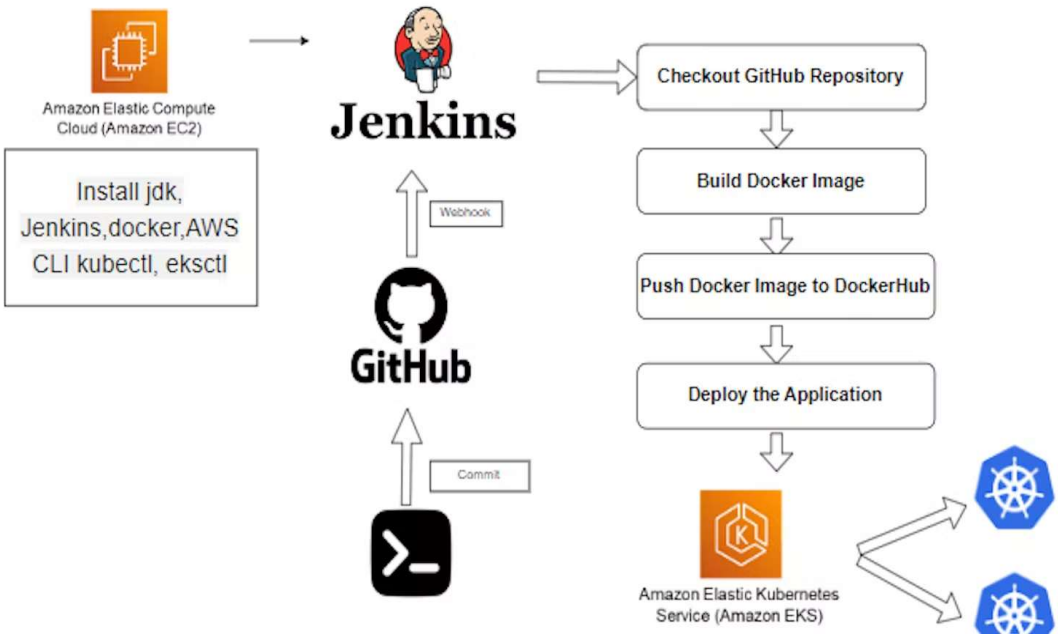
Jenkins CICD with Amazon EKS

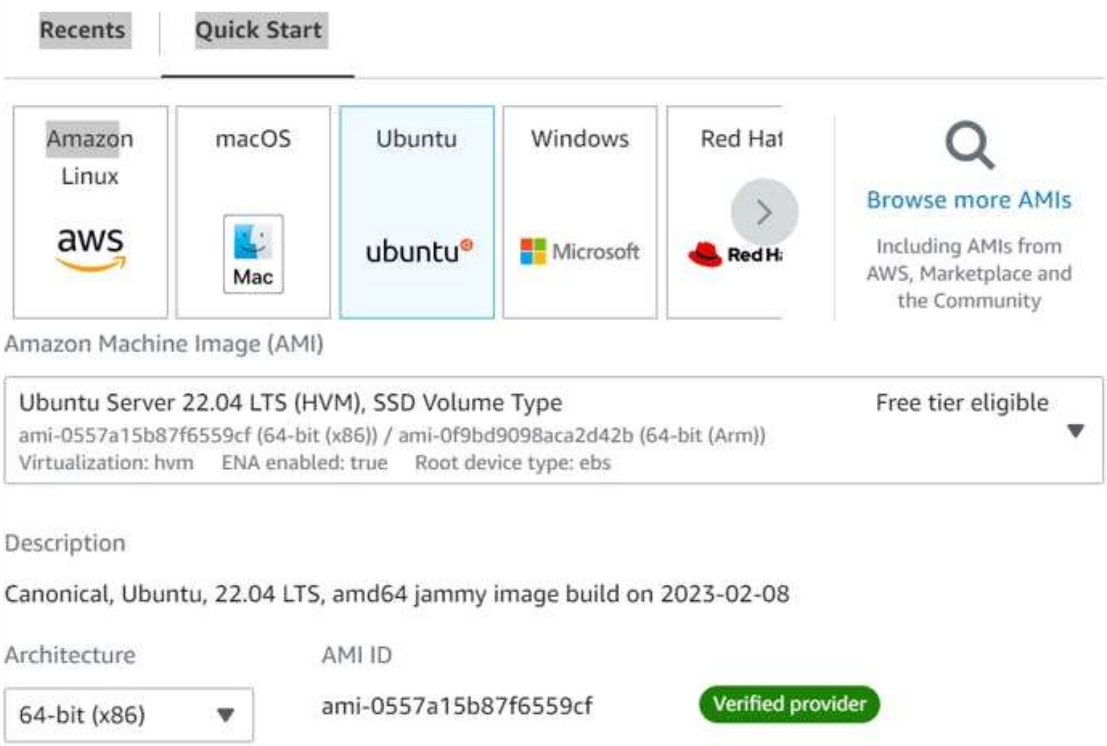


Setup an AWS EC2 Instance

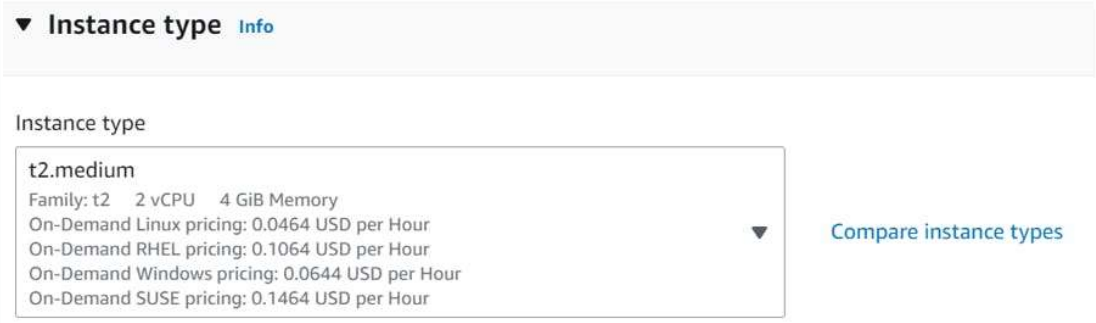
Login to an AWS account using a user with admin privileges and ensure your region is set to `us-east-1` `N. Virginia.`

Move to the EC2 console. Click `Launch Instance.`

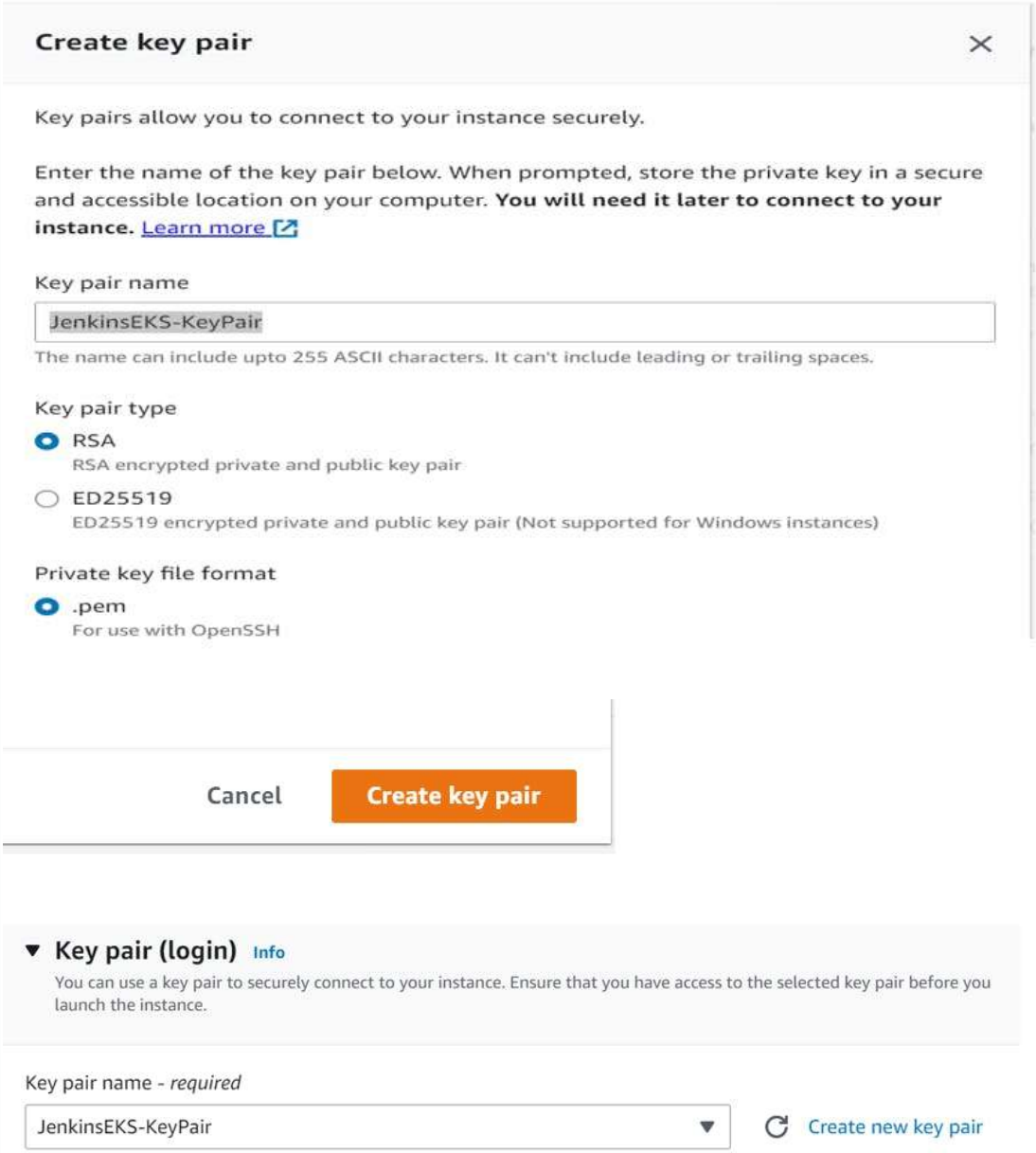
For `name` use `Jenkins-EC2.`



Select **t2.medium** because we will be installing **Jenkins** and t2.micro will not be sufficient enough to set up Jenkins.



Create and download the key pair(private key and public key)



Configure Security Group - This is an important step because here we need to **add Custom TCP Port 8080**, if you do not add this port then you will not be able to access Jenkins using the public IP address of the AWS EC2 instance.

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

- (●) Create security group
- ( ) Select existing security group

Security group name - *required*

Jenkin-EC2-SG

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&;{}!$*

Description - *required* Info

Jenkin-EC2-SG

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) — Remove

Type Info	Protocol Info	Port range Info
ssh	TCP	22

Source type Info	Source Info	Description - *optional* Info
Custom	Add CIDR, prefix list or secur 0.0.0.0/0 ✕	e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 8080, 0.0.0.0/0) — Remove

Type Info	Protocol Info	Port range Info
Custom TCP	TCP	8080

Source type Info	Source Info	Description - *optional* Info
Custom	Add CIDR, prefix list or secur 0.0.0.0/0 ✕	e.g. SSH for admin desktop

Click on launch Instance and once EC2 Instance started, connect to it with EC2 Instance Connect.

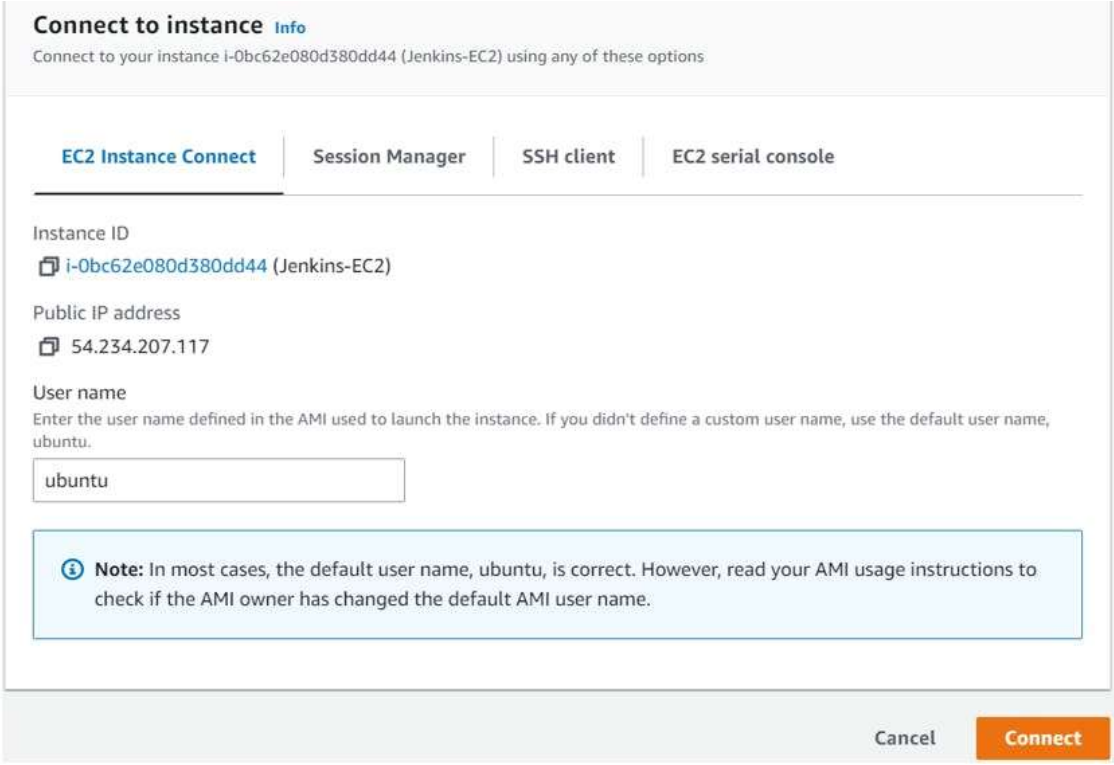
EC2 > Instances > i-0bc62e080d380dd44

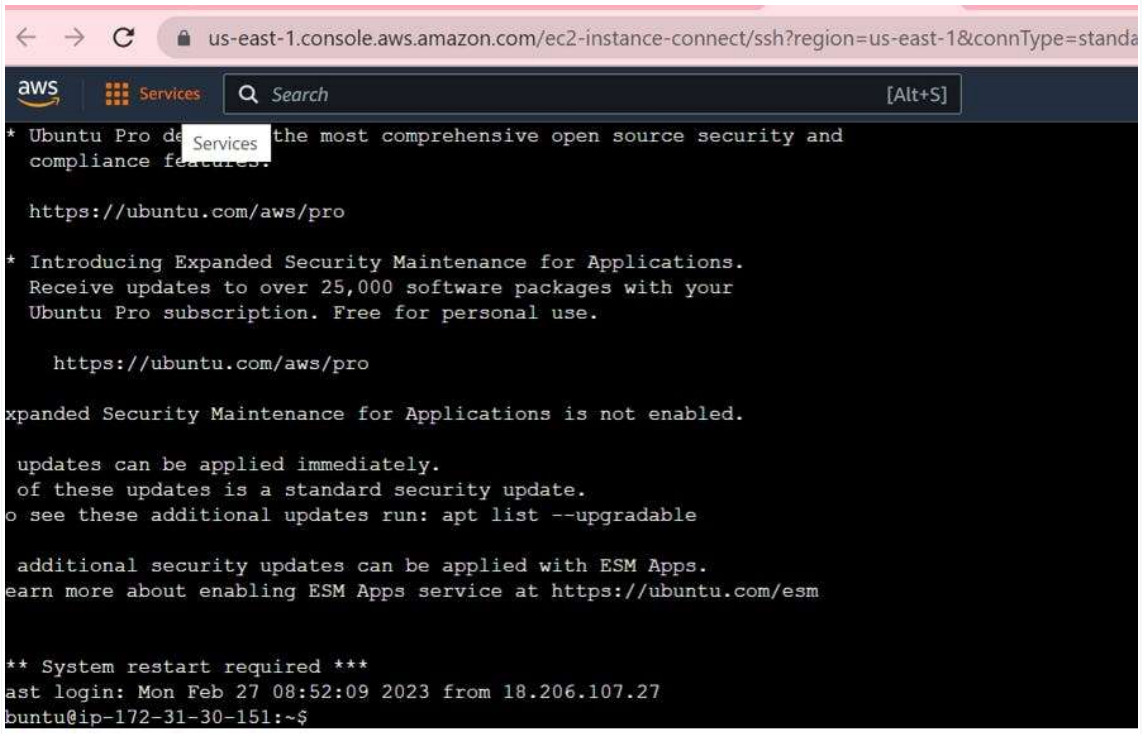
Instance summary for i-0bc62e080d380dd44 (Jenkins-EC2) Info

Updated less than a minute ago

Connect | Instance state ▼ | Actions ▼

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0bc62e080d380dd44 (Jenkins-EC2)	54.234.207.117 | open address	172.31.30.151
IPv6 address	**Instance state**	**Public IPv4 DNS**
–	Running	ec2-54-234-207-117.compute-1.amazonaws.co | open address





Install JDK on AWS EC2 Instance

Before following the below steps, assume you have already launched the ubuntu-based ec2 medium instance.

Check if you have java already installed on your EC2 machine by running the following command -

COPY
COPY

```
sudo apt-get update

java --version
```

```
ubuntu@ip-172-31-30-151:~$ sudo apt-get update
```

```
ubuntu@ip-172-31-30-151:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install default-jre              # version 2:1.11-72build2, or
sudo apt install openjdk-11-jre-headless  # version 11.0.17+8-1ubuntu2~22.04
sudo apt install openjdk-17-jre-headless  # version 17.0.5+8-2ubuntu1~22.04
sudo apt install openjdk-18-jre-headless  # version 18.0.2+9-2~22.04
sudo apt install openjdk-19-jre-headless  # version 19.0.1+10-1ubuntu1~22.04
sudo apt install openjdk-8-jre-headless   # version 8u352-ga-1~22.04
```

If this command indicates that Java is not found, then it's not installed and you can proceed with the next steps.

You can install java by running the following command.

COPY
COPY

```
apt install openjdk-11-jre-headless -y

java --version
```

```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-30-151:~$ java --version
openjdk 11.0.17 2022-10-18
OpenJDK Runtime Environment (build 11.0.17+8-post-Ubuntu-1ubuntu222.04)
OpenJDK 64-Bit Server VM (build 11.0.17+8-post-Ubuntu-1ubuntu222.04, mixed mode, sharing)
ubuntu@ip-172-31-30-151:~$
```

Install and Setup Jenkins

Step 1: Install Jenkins

Follow the steps for installing Jenkins on the EC2 instance.

COPY
COPY

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

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

sudo apt-get update

sudo apt-get install jenkins

sudo systemctl status jenkins
```

```
ubuntu@ip-172-31-30-151:~$ sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
ubuntu@ip-172-31-30-151:~$
```

```
ubuntu@ip-172-31-30-151:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg
```

```
ubuntu@ip-172-31-30-151:~$ sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt
/sources.list.d/jenkins.list'
```

```
ubuntu@ip-172-31-30-151:~$  sudo apt update
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
```

```
ubuntu@ip-172-31-30-151:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree... Done
```

```
ubuntu@ip-172-31-30-151:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-02-23 12:23:13 UTC; 1min 13s ago
   Main PID: 4973 (java)
      Tasks: 43 (limit: 4689)
     Memory: 1.2G
        CPU: 45.099s
```

Step 2: Setup Jenkins

Now go to `AWS dashboard -> EC2 -> Instances(running)and click on Jenkins-EC2`

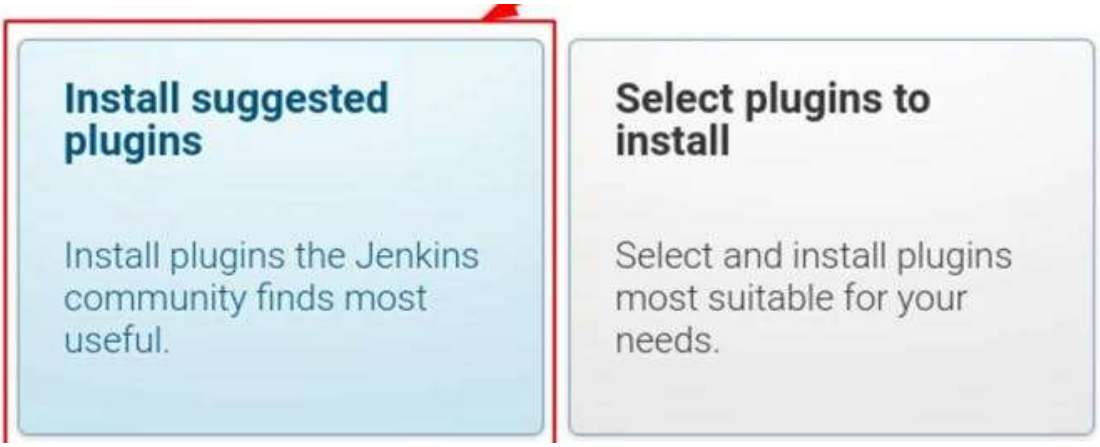
Copy Public IPv4 address.

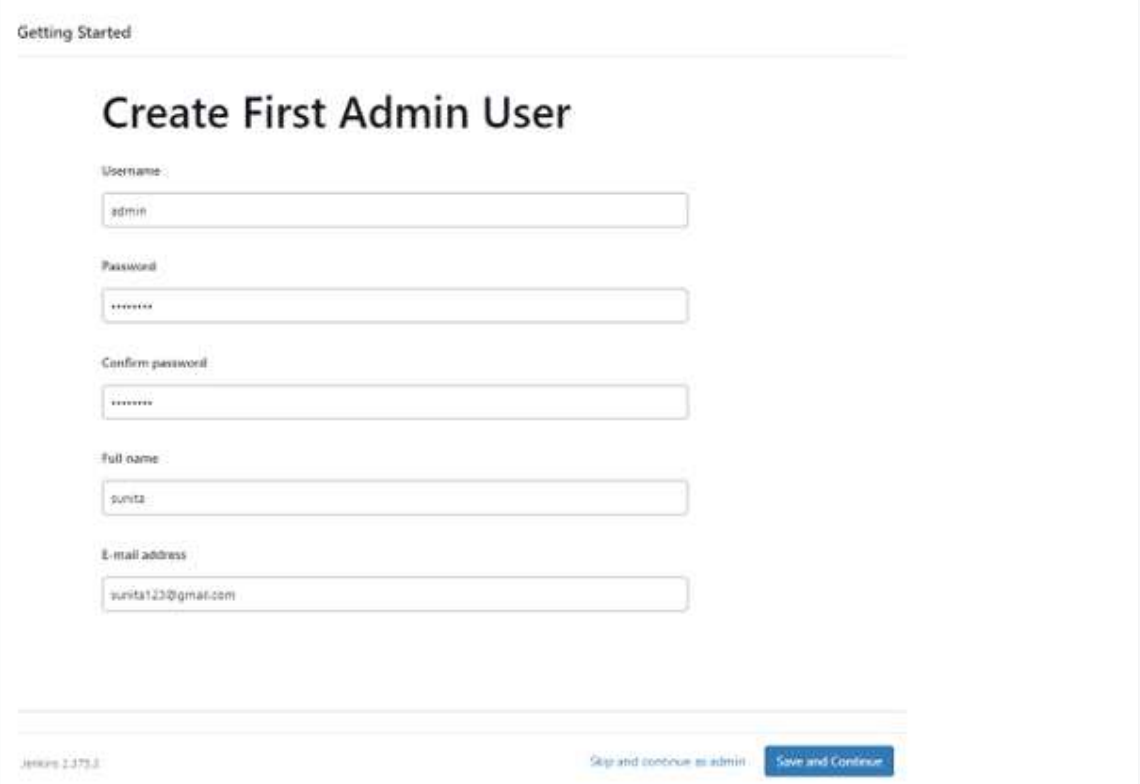
Alright now we know the public IP address of the EC2 machine, so now we can access Jenkins from the browser using the public IP address followed by port 8080.

If you are installing Jenkins for the first time then you need to supply the **initialAdminPassword** and you can obtain it from

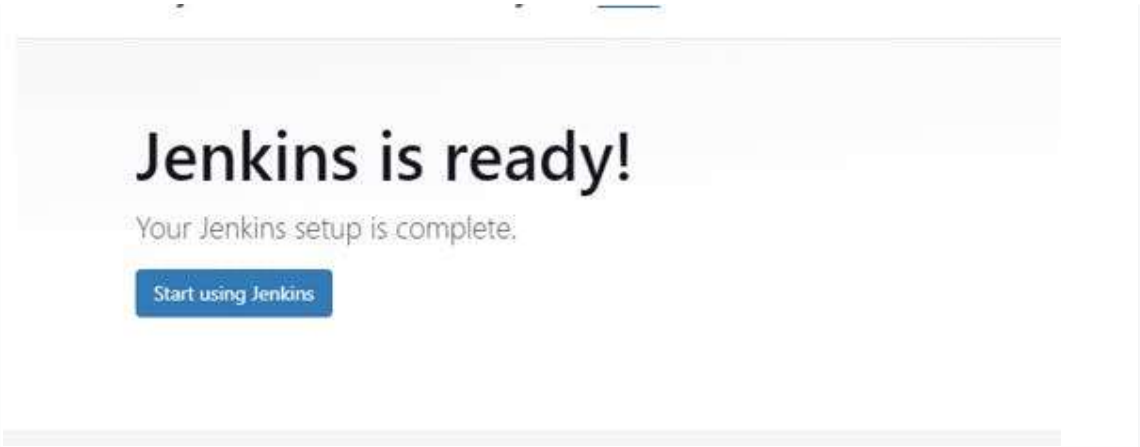
```
ubuntu@ip-172-31-30-151:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
adc00600a863414bb9f01402695f55b9
ubuntu@ip-172-31-30-151:~$
```

After completing the installation of the suggested plugin you need to set the ***First Admin User*** for Jenkins.





And now your Jenkins is ready for use



Install Docker

The docker installation will be done by the ***Jenkins*** user because now it has root user privileges.

Add jenkins user to Docker group. Jenkins will be accessing the Docker for building the application Docker images, so we need to add the Jenkins user to the docker group.

COPY
COPY

```
sudo apt install docker.io

docker --version

docker ps

sudo usermod -aG docker jenkins
```

```
sudo apt install docker.io
```

```
jenkins@ip-172-31-30-151:~$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu4
jenkins@ip-172-31-30-151:~$
```

```
jenkins@ip-172-31-30-151:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
jenkins@ip-172-31-30-151:~$
```

Install and Setup AWS CLI

Now we need to set up the AWS CLI on the EC2 machine so that we can use ***eksctl*** in the later stages

Let us get the installation done for ***AWS CLI***

COPY
COPY

```
sudo apt install awscli

aws --version
```

```
sudo apt install awscli
```

```
jenkins@ip-172-31-30-151:~$ aws --version
aws-cli/1.22.34 Python/3.10.6 Linux/5.15.0-1028-aws botocore/1.23.34
jenkins@ip-172-31-30-151:~$
```

Okay now after installing the AWS CLI, let's configure the ***AWS CLI*** so that it can authenticate and communicate with the AWS environment.

COPY
COPY

```
aws configure
```

To configure the AWS the first command we are going to run is

Once you execute the above command it will ask for the following information -

1. AWS Access Key ID [None]:
2. AWS Secret Access Key [None]:
3. Default region name [None]:
4. Default output format [None]:

You can click on the ***Create New Access Key*** and it will let you generate - `AWS Access Key ID`, `AWS Secret Access Key`.

(Note: - Always remember you can only download your access id and secret once, if you misplace the secret and access then you need to recreate the keys again.

```
jenkins@ip-172-31-30-151:~$ aws configure
AWS Access Key ID [****************BLP4]: AKIAZ
AWS Secret Access Key [****************lVjL]: F
Default region name [None]: us-east-1
Default output format [None]:
jenkins@ip-172-31-30-151:~$
```

Install and Setup Kubectl

Moving forward now we need to set up the kubectl also onto the EC2 instance where we set up the Jenkins in the previous steps.

Here is the command for installing kubectl

COPY
COPY

```
curl -LO [https://storage.googleapis.com/kubernetes-release/release/$(curl](https://storage.googleapis.com/kubernetes-release/release/$(curl) -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl/bin/linux/amd64/kubectl)

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version
```

```
jenkins@ip-172-31-30-151:~$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 45.7M  100 45.7M    0     0  43.3M      0  0:00:01  0:00:01 --:--:-- 43.4M
jenkins@ip-172-31-30-151:~$ chmod +x ./kubectl
jenkins@ip-172-31-30-151:~$ sudo mv ./kubectl /usr/local/bin
jenkins@ip-172-31-30-151:~$
```

Verify the kubectl installation by running the command `kubectl version` and you should see the following output.

```
Client Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.1", GitCommit:"8f94681cd294aa8cfd3407b8191f6c70214973a4", GitTreeState:"clean", BuildDate:"2023-01-18T15:58:16Z", GoVersion:"go1.19.5", Compiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v4.5.7
Server Version: version.Info{Major:"1", Minor:"24+", GitVersion:"v1.24.10-eks-48e63af", GitCommit:"9176fb99b52f8d5ff73d67fea27f3a638f679f8a", GitTreeState:"clean", BuildDate:"2023-01-24T19:17:48Z", GoVersion:"go1.19.5", Compiler:"gc", Platform:"linux/amd64"}
WARNING: version difference between client (1.26) and server (1.24) exceeds the supported minor version skew of +/-1
```

Install and Setup eksctl

Okay, the first command which we are gonna run to install the ***eksctl***

Download and extract the latest release of `eksctl` with the following command.

COPY
COPY

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(
uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

Move the extracted binary to `/usr/local/bin`.

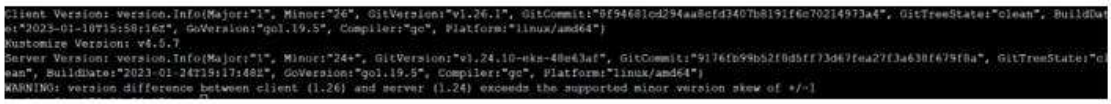
COPY
COPY

```
sudo mv /tmp/eksctl /usr/local/bin
```

Test that your installation was successful with the following command.

COPY
COPY

```
eksctl version
```



Creating an Amazon EKS cluster using eksctl

Now in this step, we are going to create Amazon EKS cluster using `eksctl`

You need the following in order to run the eksctl command

1. **Name of the cluster :** --name first-eks-cluster1
2. **Version of Kubernetes :** --version 1.24
3. **Region :** --region us-east-1

4. **Nodegroup name/worker nodes :** --nodegroup-name worker-nodes
5. **Node Type :** --nodegroup-type t2.micro
6. **Number of nodes:** --nodes 2

Here is the eksctl command -

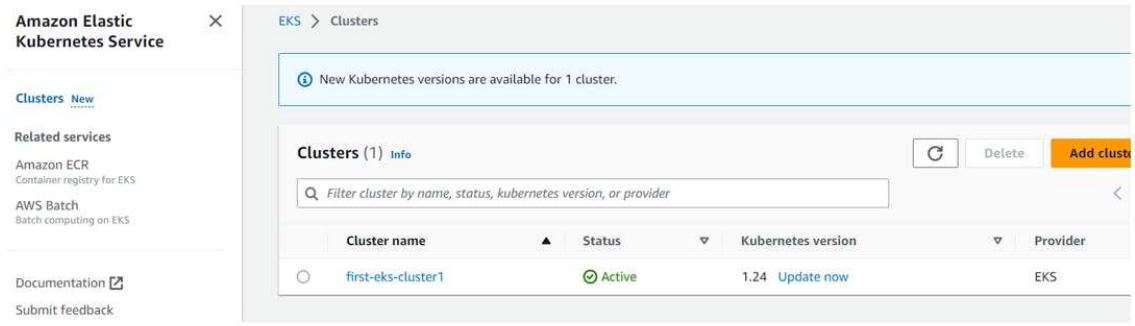
COPY
COPY

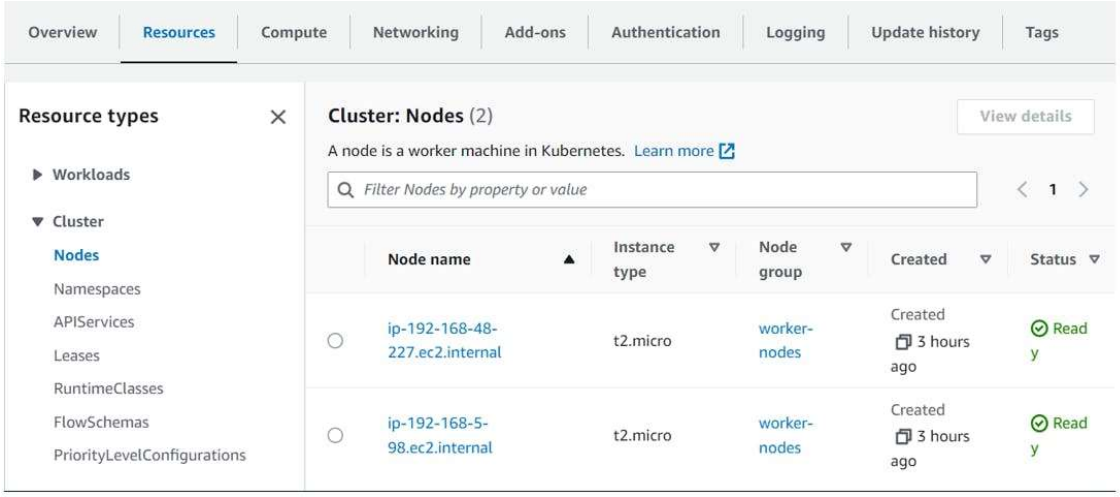
```
eksctl create cluster --name first-eks-cluster1 --version 1.24 --region us-east-1
--nodegroup-name worker-nodes --node-type t2.micro --nodes 2
```

It took me 20 minutes to complete this EKS cluster. If you get any error for not having sufficient data for mentioned availability zone then try it again.

Verify the EKS kubernetes cluster on AWS Console.

You can go back to your AWS dashboard and look for `Elastic Kubernetes Service -> Clusters`



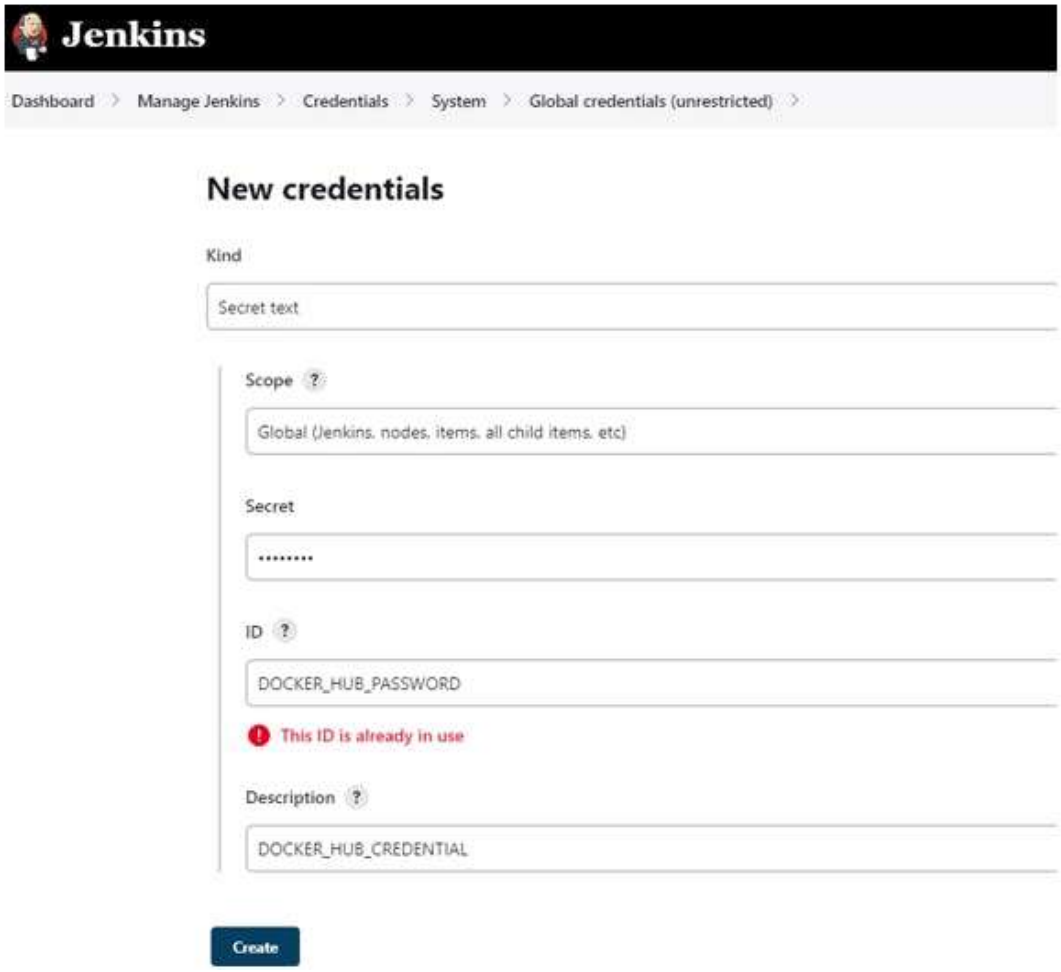


Add Docker and GitHub Credentials on Jenkins

Step 1: Setup Docker Hub Secret Text in Jenkins

You can set the docker credentials by going into -

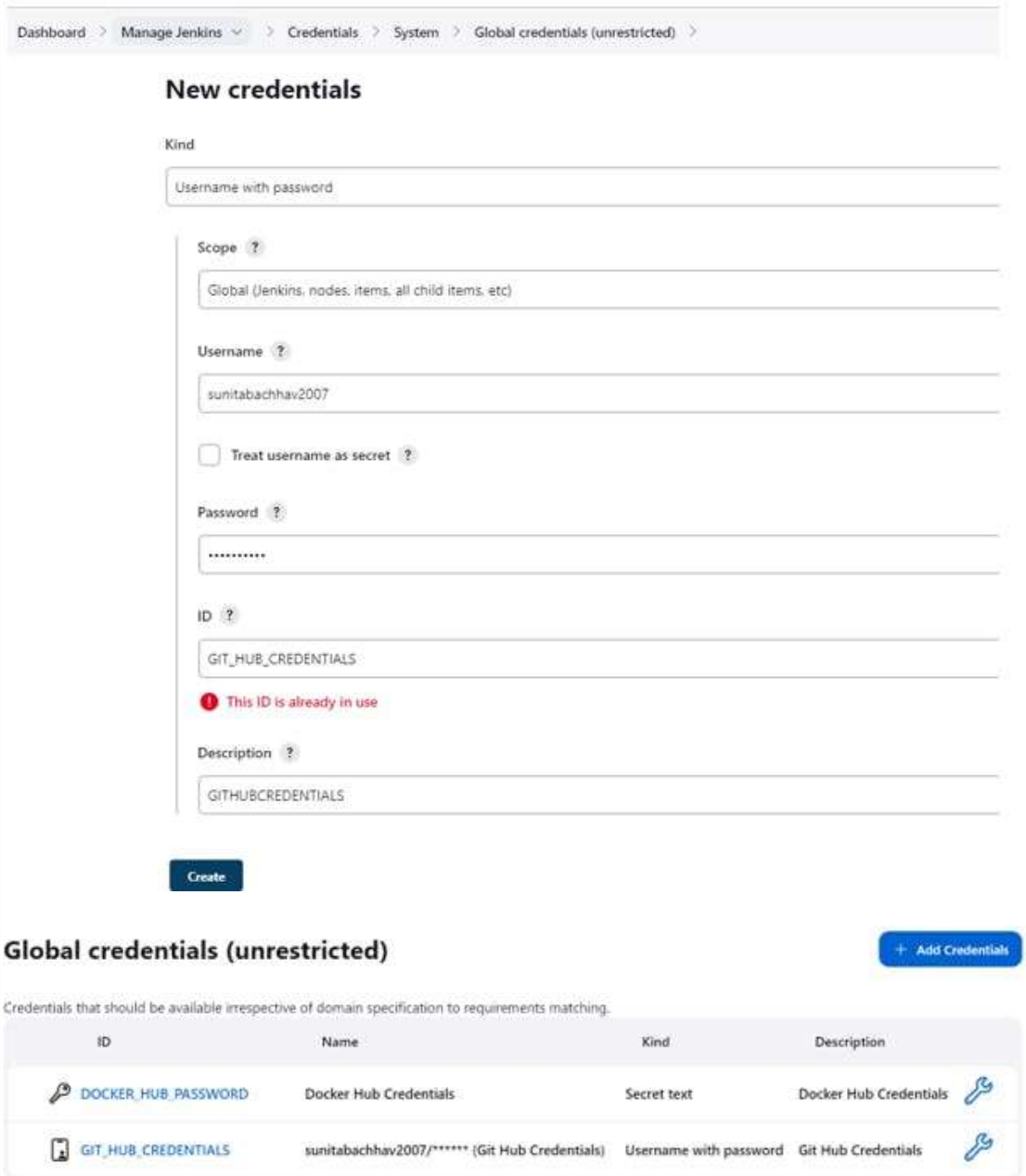
Goto -> Jenkins -> Manage Jenkins -> Manage Credentials -> Stored scoped to jenkins -> global -> Add Credentials



Step 2. Setup GitHub Username and password into Jenkins

Now we add one more username and password for GitHub.

Goto -> Jenkins -> Manage Jenkins -> Manage Credentials -> Stored scoped to jenkins -> global -> Add Credentials

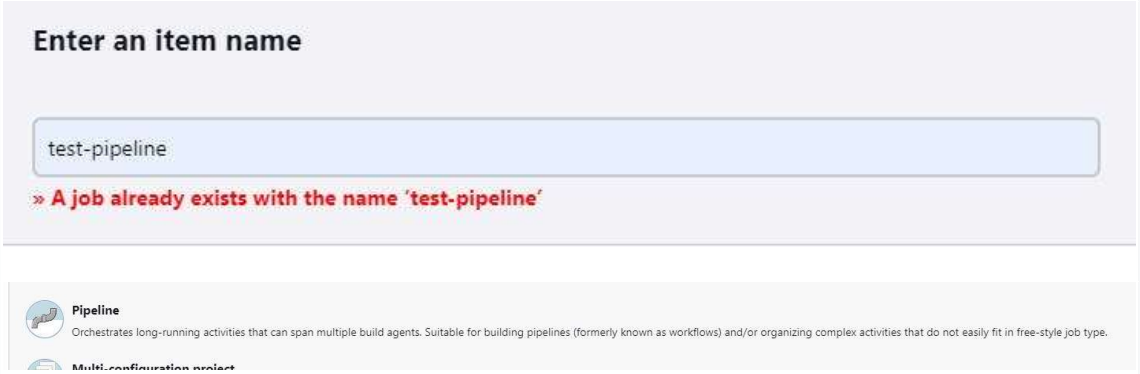


Build, deploy and test CI/CD pipeline

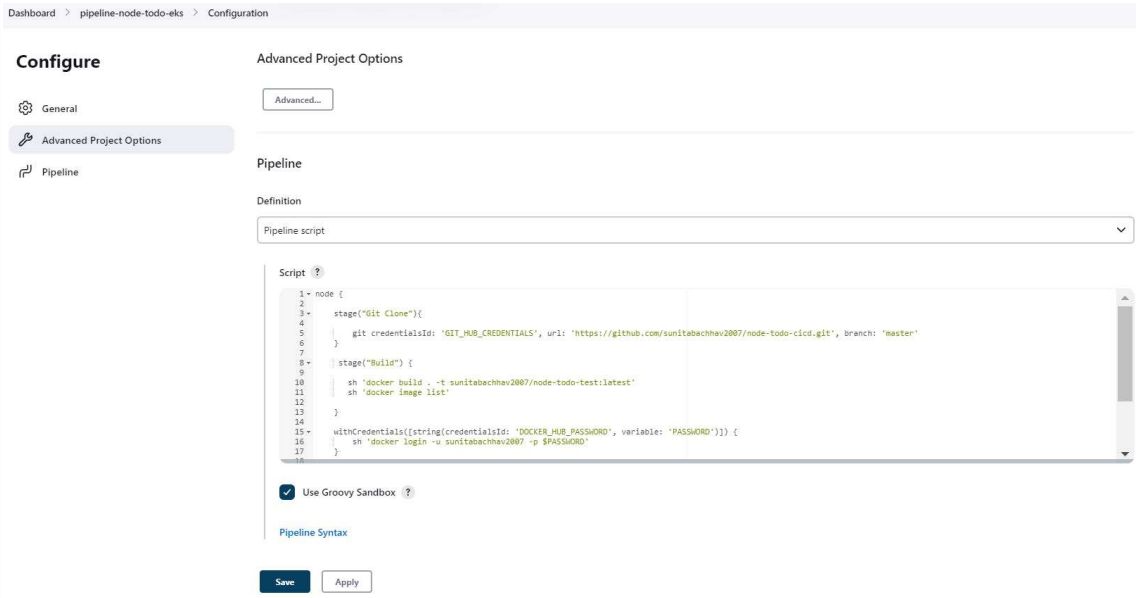
Okay, now we can start writing out the Jenkins pipeline for deploying the Node.js Application into the Kubernetes Cluster.

Create new Pipeline: Goto Jenkins Dashboard or Jenkins home page click on ***New Item***

Pipeline Name: Now enter Jenkins pipeline name and select Pipeline



Add pipeline script: `Goto -> Configure` and then pipeline section.



Copy the below script and paste into Pipeline Script.

COPY

COPY

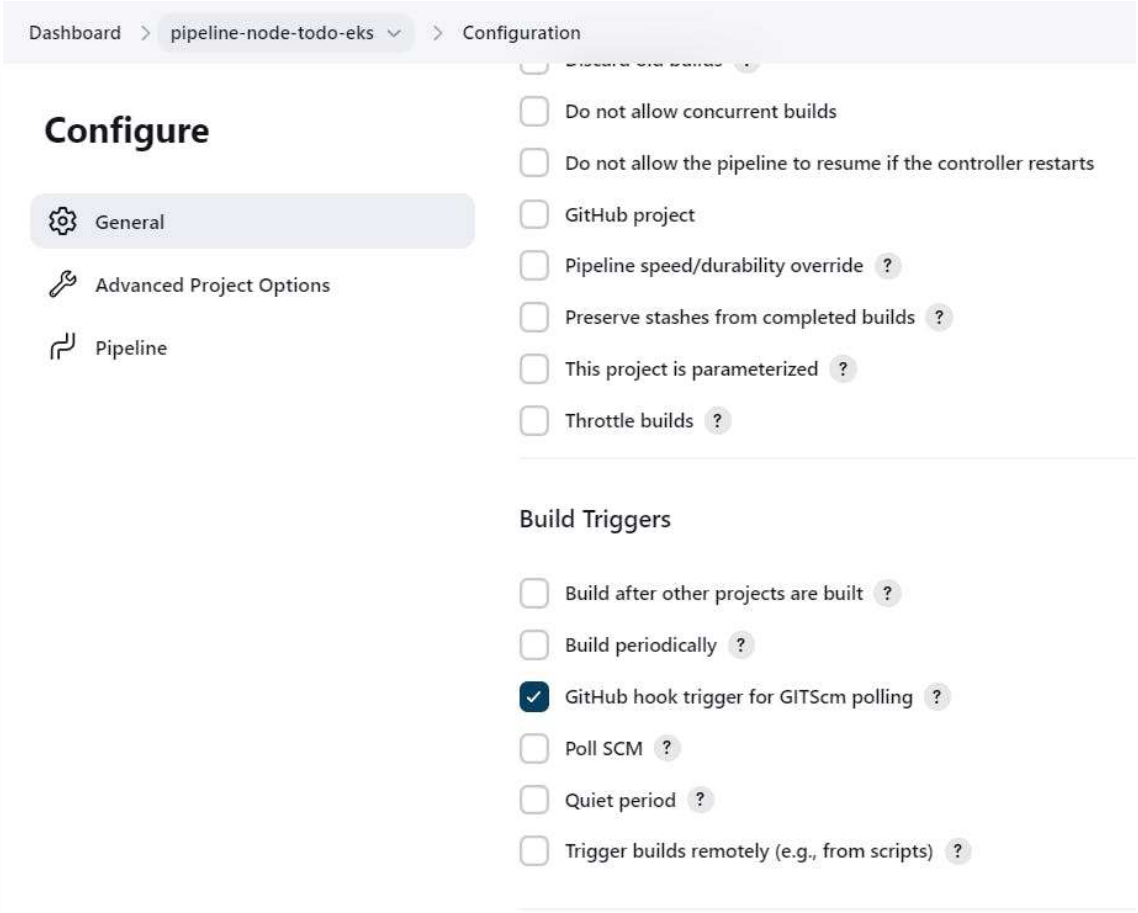
```
node {

    stage("Git Clone"){

        git credentialsId: 'GIT_HUB_CREDENTIALS', url:
'https://github.com/sunitabachhav2007/node-todo-cicd.git', branch: 'master'

    }

    stage("Build") {

        sh 'docker build . -t sunitabachhav2007/node-todo-test:latest'

        sh 'docker image list'

    }

    withCredentials([string(credentialsId: 'DOCKER_HUB_PASSWORD',
variable: 'PASSWORD')]) {

        sh 'docker login -u sunitabachhav2007 -p $PASSWORD'
```

```
    }



    stage("Push Image to Docker Hub"){

        sh 'docker push sunitabachhav2007/node-todo-test:latest'

    }



    stage("kubernetes deployment"){

        sh 'kubectl apply -f deployment.yml'

    }

}
```

To set up Jenkins - GitHub Webhook

Now, go to the "**Build Triggers**" tab.

Here, choose the "**GitHub hook trigger for GITScm pulling**" option, which will listen for triggers from the given GitHub repository, as shown in the image below.

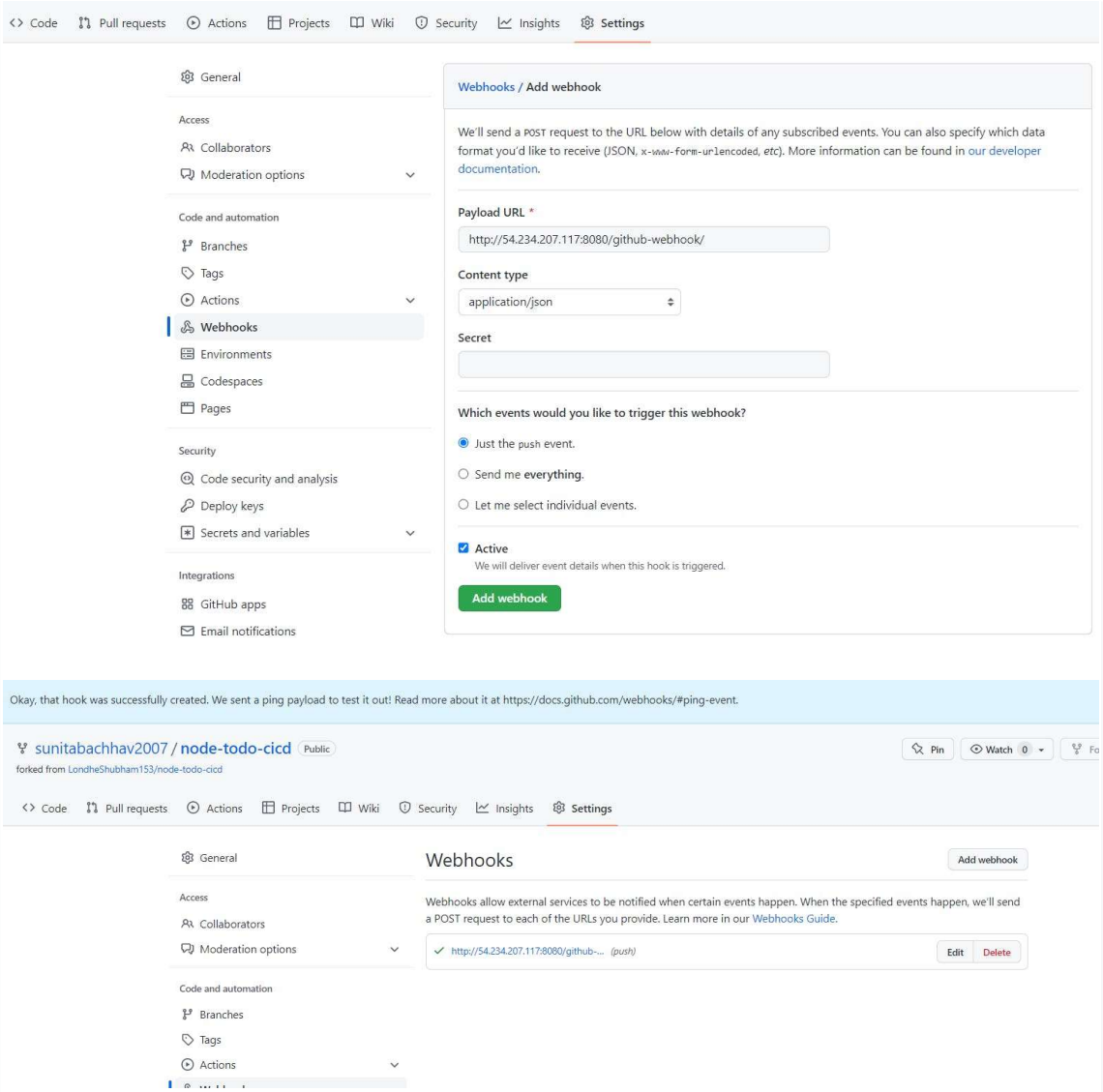


Jenkins GitHub Webhook is **used to trigger the action whenever Developers commit something into the repository**. It can automatically build and deploy applications.

Switch to your GitHub account, go to "**Settings**" option. Here, select the "**Webhooks**" option and then click on the "**Add Webhook**"

It will provide you the blank fields to add the Payload URL where you will paste your Jenkins address, Content type, and other configuration.

Go to your Jenkins tab and copy the URL then paste it in the text field named "**Payload URL**", as shown in the image below. Append the "**/github-webhook/**" at the end of the URL.



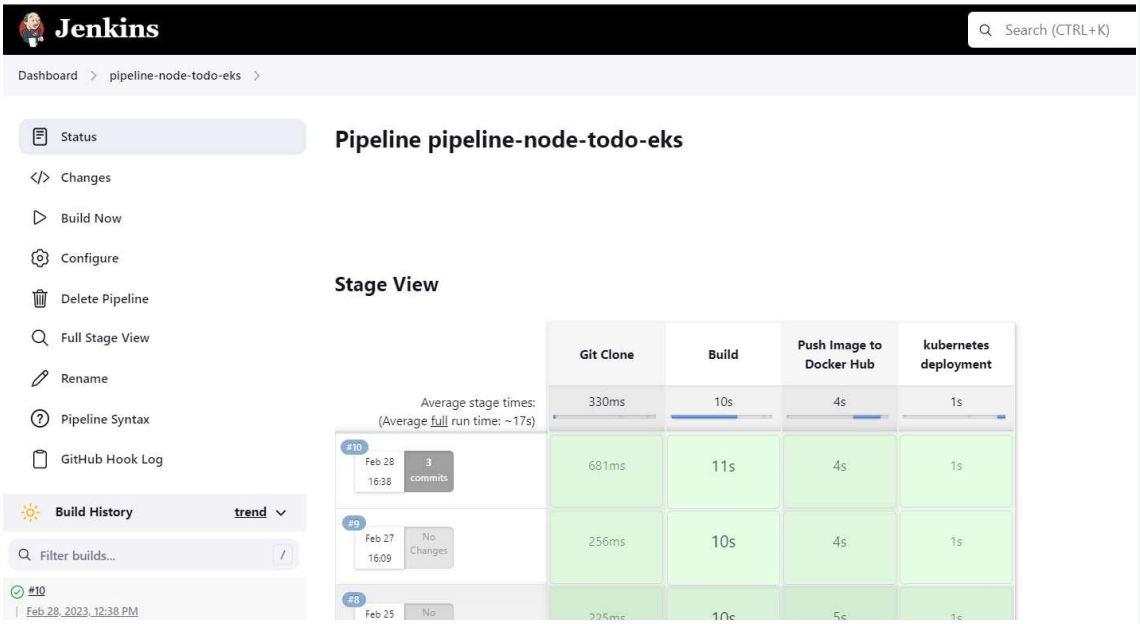
You completed Jenkins GitHub Webhook. Now for any commit in the GitHub repository, Jenkins will trigger the event specified

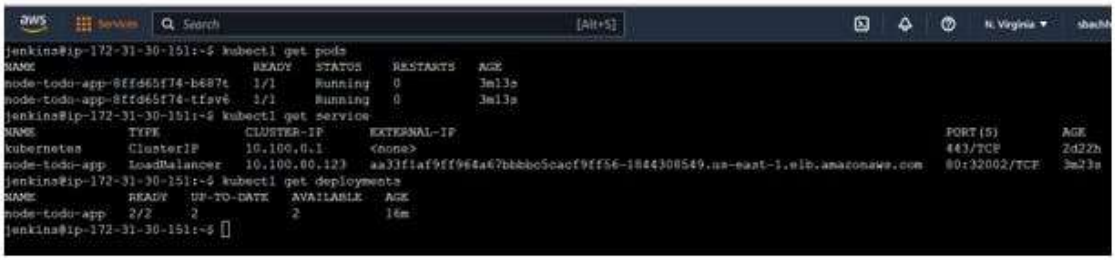
```
views > todo.ejs > html > head > title
<!DOCTYPE html>

<html>

    <head>
        <title>My Awesome todolist. Created today on 28th Feb 2023</title>
        <style>
            a {
                text-decoration: none;
                color: black;
            }

        </style>
    </head>
```

```
PS C:\Sunita\DevOps-Project\CICD-EKS-node-todo-app\node-todo-cicd> git remote -v
origin  https://github.com/sunitabachhav2007/node-todo-cicd.git (fetch)
origin  https://github.com/sunitabachhav2007/node-todo-cicd.git (push)
PS C:\Sunita\DevOps-Project\CICD-EKS-node-todo-app\node-todo-cicd>
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                   powershell
PS C:\Sunita\DevOps-Project\CICD-EKS-node-todo-app\node-todo-cicd> git commit -m "Changed message with today's date"
[master d8562a3] Changed message with today's date
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Sunita\DevOps-Project\CICD-EKS-node-todo-app\node-todo-cicd> git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 387 bytes | 387.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/sunitabachhav2007/node-todo-cicd.git
   19461fb..d8562a3  master -> master
PS C:\Sunita\DevOps-Project\CICD-EKS-node-todo-app\node-todo-cicd>
```

After pushing code to Github repository





You can access the rest endpoint from a browser using the EXTERNAL-IP address.

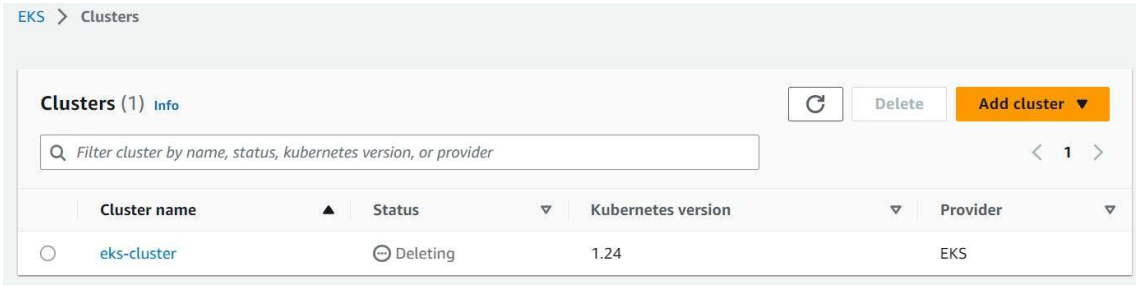
Clean up

Copy Deployment.yml file (From Github Repository) to EC2 server and run with below command.

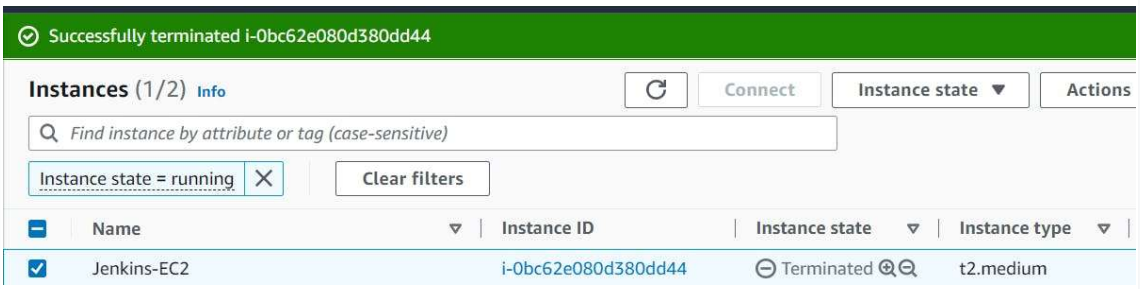
COPY
COPY

```
kubectl delete -f deployment.yml
```

Delete EKS Cluster from AWS Console.



Terminate EC2 Instance.



Conclusion

We have successfully **deployed our Node.js App in Amazon EKS cluster using AWS EC2, Jenkins, Docker, Kubernetes, GitHub, Webhook.**