DevOps Project

Problem Statement:

Create an end to end CI/CD pipeline in AWS platform using Jenkins as the orchestration tool, Github as the SCM, Maven as the Build tool, Deploy in a docker instance and create a Docker image, Store the docker image in ECR, Achieve Kubernetes deployment using the ECR image. Build a sample java web app using maven.

Tools & Technologies

Tool/Technology	Purpose
Git & GitHub	Version control and source code repository
Jenkins	Continuous Integration and Deployment
Maven	Java build automation
Docker	Containerization of the application
Amazon ECR	Secure Docker image storage
Amazon EKS	Kubernetes cluster for deployment
kubectl	CLI tool to interact with Kubernetes

Architecture Overview

1. Source Code: Java web application hosted on GitHub
2. CI/CD Pipeline: Jenkins pulls the code, builds it using Maven, and creates a Docker image
3. Image Storage: Docker image is pushed to Amazon ECR
4. Deployment: Kubernetes (EKS) pulls the image from ECR and deploys it as pods
5. Access: A LoadBalancer service exposes the app to the internet via an external IP

Step-by-Step Implementation

Step 1: CI/CD with GitHub, Jenkins, Maven & Tomcat (Optional)

1. Install Jenkins on an EC2 instance or local instance
2. Set up Jenkins with Git and Maven plugins
3. Install Tomcat server for initial deployment
4. Create a Jenkins job to:
   - Pull code from GitHub
   - Build using Maven
   - Deploy to Tomcat

Step 2: CI/CD with GitHub, Jenkins, Maven & Docker

1. Install Docker on the new EC2-host
2. Add a Dockerfile to your Java project
3. Update Jenkins job to:
   - Build the project with Maven
   - Create a Docker image

Step 3: Push Docker Image to Amazon ECR

1. Create an ECR repository in AWS
2. Authenticate Jenkins with ECR using AWS CLI
3. Update Jenkins job to:
   - Tag the Docker image
   - Push it to ECR

Step 4: Deploy to Kubernetes using Amazon EKS

1. Create an EKS cluster via AWS Console or CLI
2. Configure kubectl to connect to your EKS cluster
3. Write Kubernetes manifests:
   - deployment.yaml for app deployment
   - service.yaml for LoadBalancer
4. Update Jenkins job to:
   - Apply Kubernetes manifests using kubectl

CI/CD Automation Flow

1. Developer pushes code to GitHub
2. Jenkins detects the change via webhook
3. Jenkins:
   - Pulls the latest code
   - Builds with Maven
   - Creates a Docker image
   - Pushes the image to ECR
   - Applies Kubernetes manifests
4. Kubernetes:
   - Pulls the image from ECR
   - Deploys pods and services
5. Application is accessible via LoadBalancer IP

Validation & Testing

- Run kubectl get services to retrieve the LoadBalancer IP
- Access the app at http://<external-ip>:8080/webapp
- Make a new commit to GitHub and verify that the app updates automatically

Sample Commands

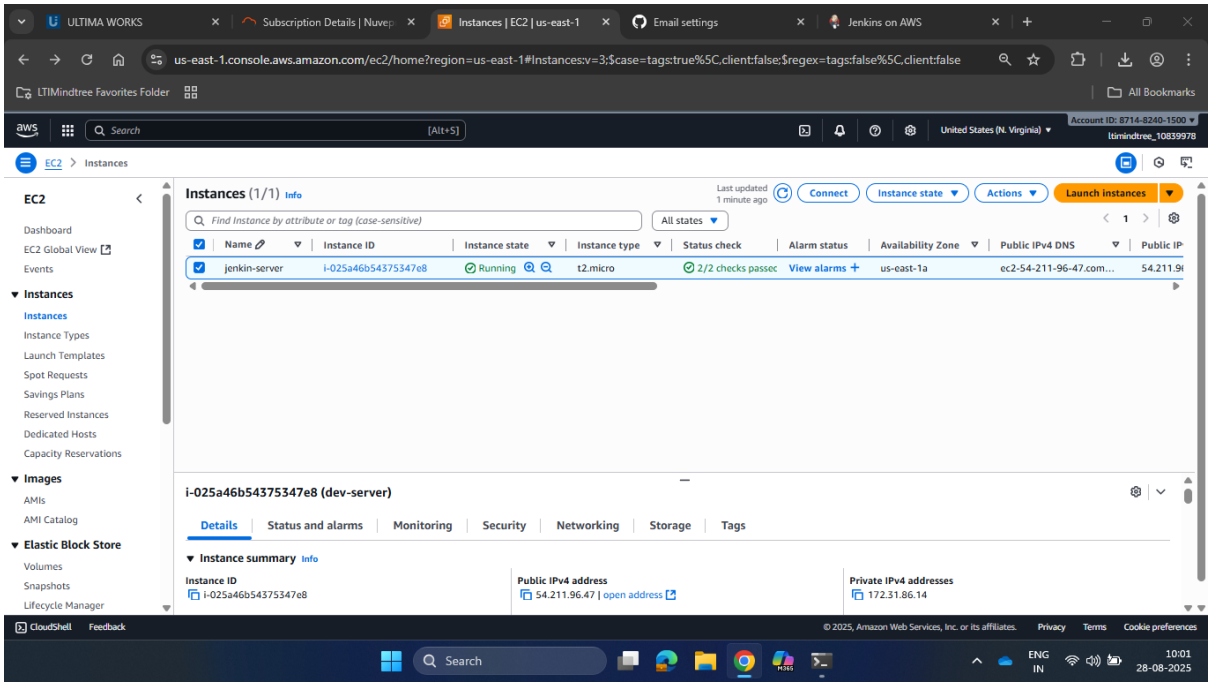
kubectl get service regapp-service
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

Outcome

- ✅ End-to-end CI/CD pipeline fully automated
- ✅ Scalable deployment using Kubernetes
- ✅ Secure image storage with Amazon ECR
- ✅ Real-time updates from GitHub reflected in the deployed app

Solution:

Launching EC2 instance for jenkins-server hosting



Installing Jenkins on Ec2-instance.

```
[root@jenkins-server ~]# wget -O /etc/yum.repos.d/jenkins.repo \  https://pkg.jenkins.io/redhat-stable/jenkins.repo
--2025-08-28 04:47:00--  http://%20/
```

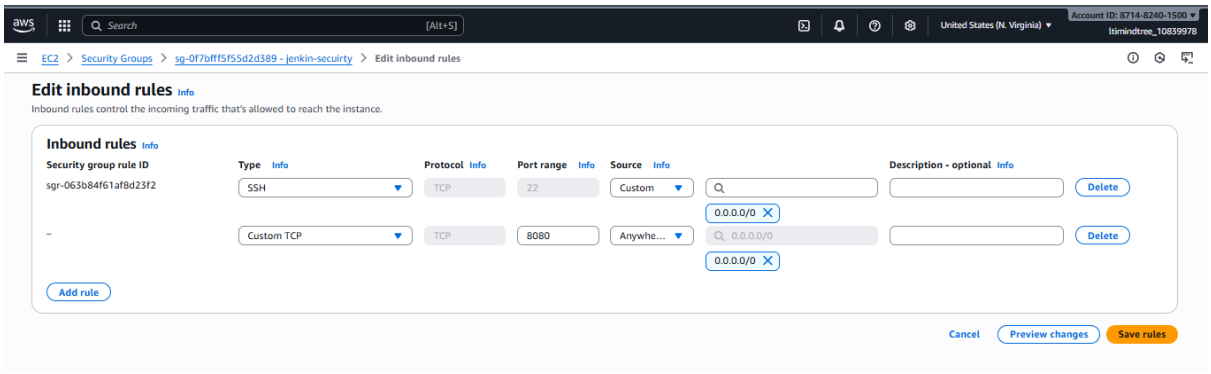
```
[root@jenkins-server ~]# yum install java-21-amazon-corretto -y
```

```
[root@ip-172-31-10-10 ~]# sudo yum install jenkins -y
Last metadata expiration check: 0:00:26 ago on Thu Aug 28 08:59:40 2025.
Dependencies resolved.
================================================================================
 Package                 Architecture            Version                 Repository              Size
================================================================================
Installing:
 jenkins                 noarch                  2.516.2-1.1             jenkins                 83 M

Transaction Summary
================================================================================
```

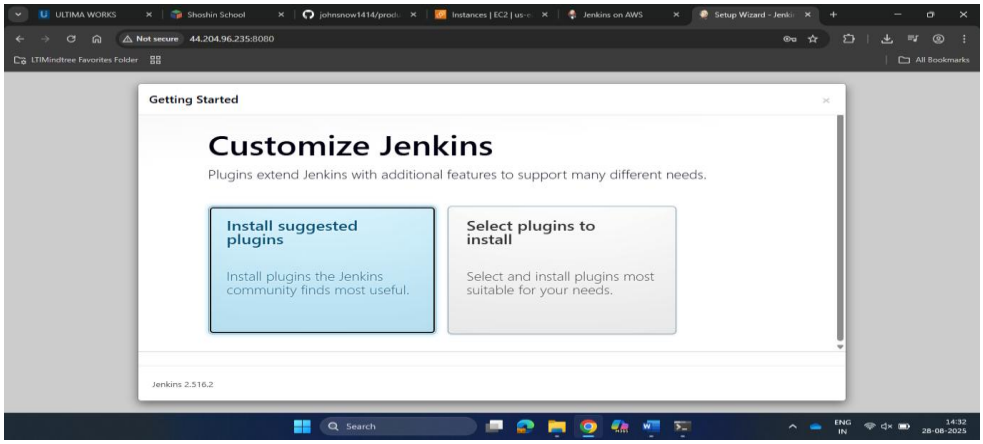
Now we can see that Jenkins is running perfectly

```
Complete!
[root@jenkins-server ~]# systemctl enable jenkins
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
[root@jenkins-server ~]# systemctl start jenkins
[root@jenkins-server ~]# systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: disabled)
     Active: active (running) since Thu 2025-08-28 04:49:05 UTC; 6s ago
   Main PID: 26512 (java)
      Tasks: 45 (limit: 1111)
     Memory: 373.9M
        CPU: 17.055s
     CGroup: /system.slice/jenkins.service
             └─26512 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenki>

Aug 28 04:48:59 jenkins-server jenkins[26512]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Aug 28 04:48:59 jenkins-server jenkins[26512]: ************************************************************
Aug 28 04:48:59 jenkins-server jenkins[26512]: ************************************************************
Aug 28 04:48:59 jenkins-server jenkins[26512]: ************************************************************
Aug 28 04:49:05 jenkins-server jenkins[26512]: 2025-08-28 04:49:05.856+0000 [id=32]       INFO       jenkins.InitReac>
Aug 28 04:49:05 jenkins-server jenkins[26512]: 2025-08-28 04:49:05.879+0000 [id=24]       INFO       hudson.lifecycle>
Aug 28 04:49:05 jenkins-server systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Aug 28 04:49:06 jenkins-server jenkins[26512]: 2025-08-28 04:49:06.115+0000 [id=48]       INFO       h.m.DownloadServ>
Aug 28 04:49:06 jenkins-server jenkins[26512]: 2025-08-28 04:49:06.117+0000 [id=48]       INFO       hudson.util.Retr>
Aug 28 04:49:10 jenkins-server jenkins[26512]: 2025-08-28 04:49:10.986+0000 [id=70]       WARNING        h.n.DiskSpace>
lines 1-20/20 (END)
```

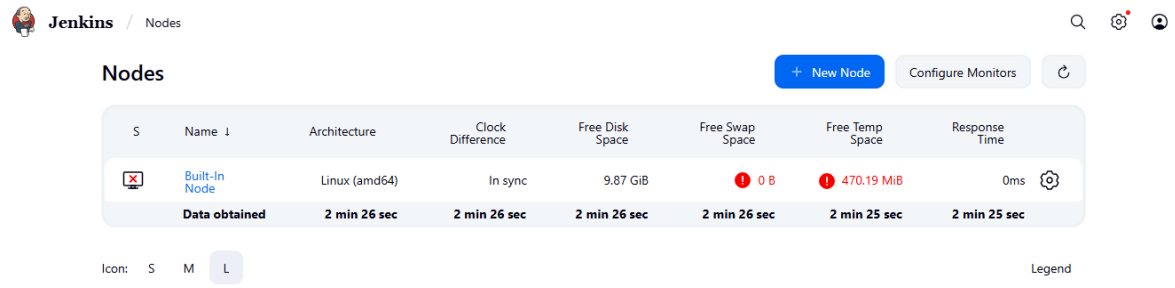
Enable port 8080 on jenkin-EC2 server



Now on browser hit http://<ip-addres of server>:8080



Now we are getting error as less space is allocated in temp storage

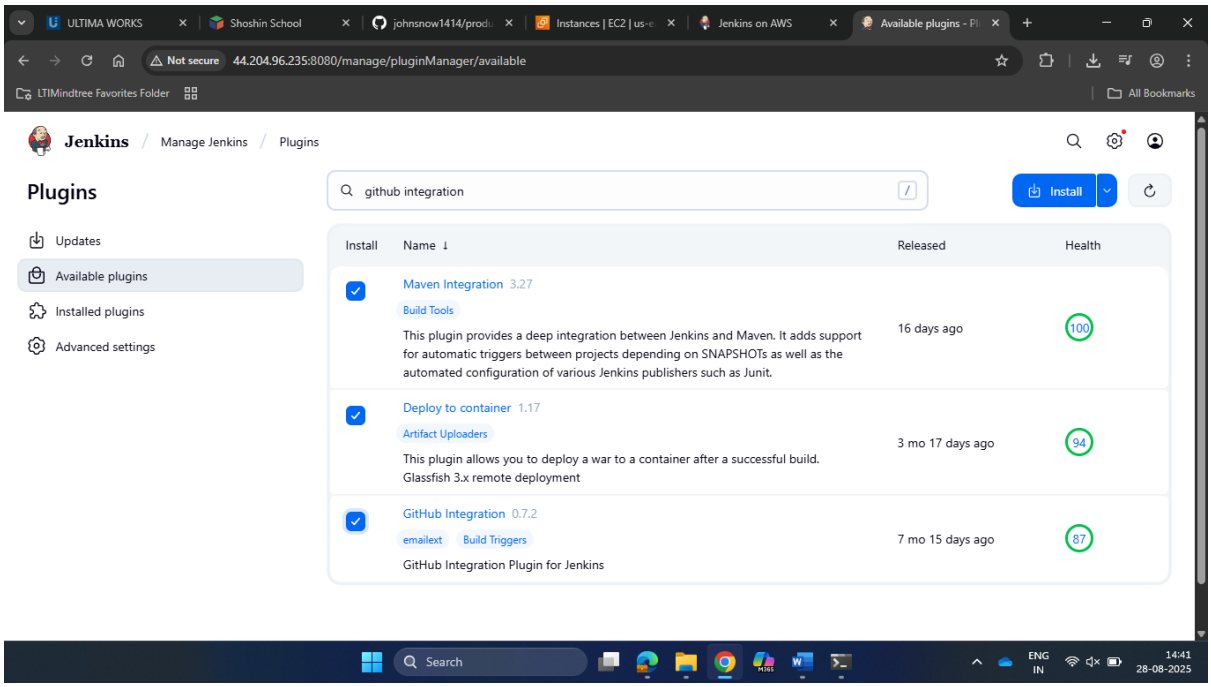


Now increasing the temp space and set it 2GB

```
[root@ip-172-31-10-10 ~]# df -h temp
df: temp: No such file or directory
[root@ip-172-31-10-10 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  440K  190M   1% /run
/dev/xvda1       12G  2.3G  9.7G  19% /
tmpfs           475M  4.5M  471M   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
[root@ip-172-31-10-10 ~]# df -h /tmp
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           475M  4.5M  471M   1% /tmp
[root@ip-172-31-10-10 ~]# mount -o remount,size=2G /tmp
[root@ip-172-31-10-10 ~]# df -h /tmp
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           2.0G  4.5M  2.0G   1% /tmp
[root@ip-172-31-10-10 ~]# vim /etc/fstab
[root@ip-172-31-10-10 ~]# mount -a
[root@ip-172-31-10-10 ~]# systemctl daemon-reload
[root@ip-172-31-10-10 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  444K  190M   1% /run
/dev/xvda1       12G  2.3G  9.7G  19% /
tmpfs           2.0G  4.5M  2.0G   1% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
```

Now installing plugins in Jenkins as required

- Maven integration (to build java application)
- Github integration
- Deploy to container



Now on Jenkin-Server install maven, git

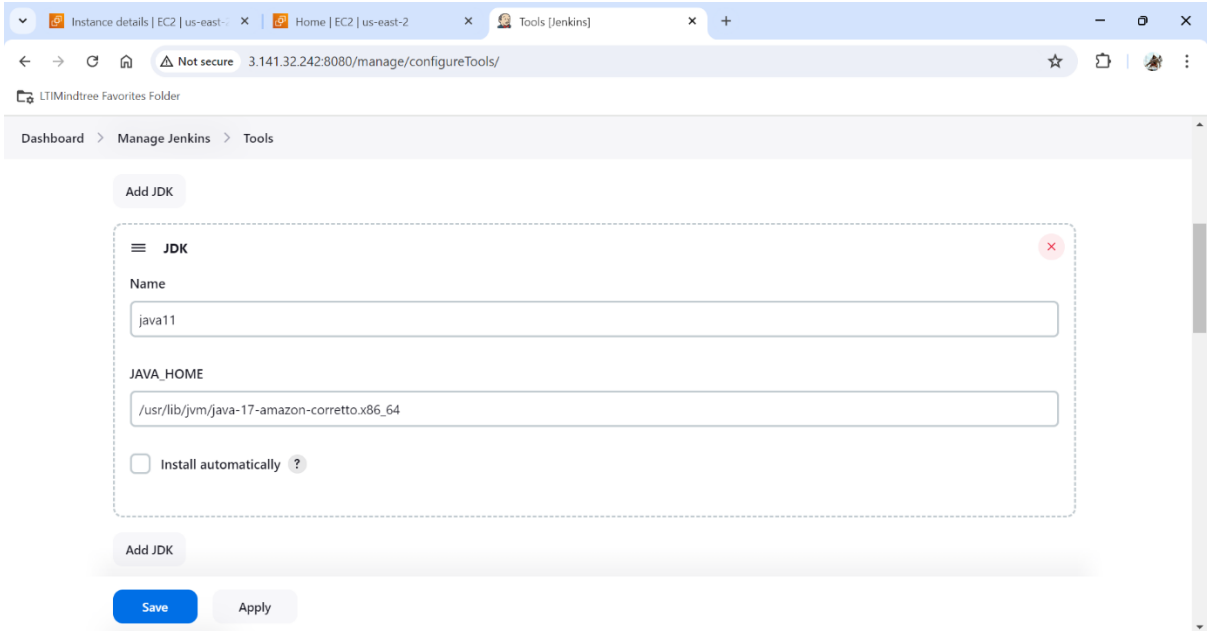
```
[root@ip-172-31-10-10 ~]# yum install git -y
```

```
[root@ip-172-31-10-10 ~]# yum install maven
```

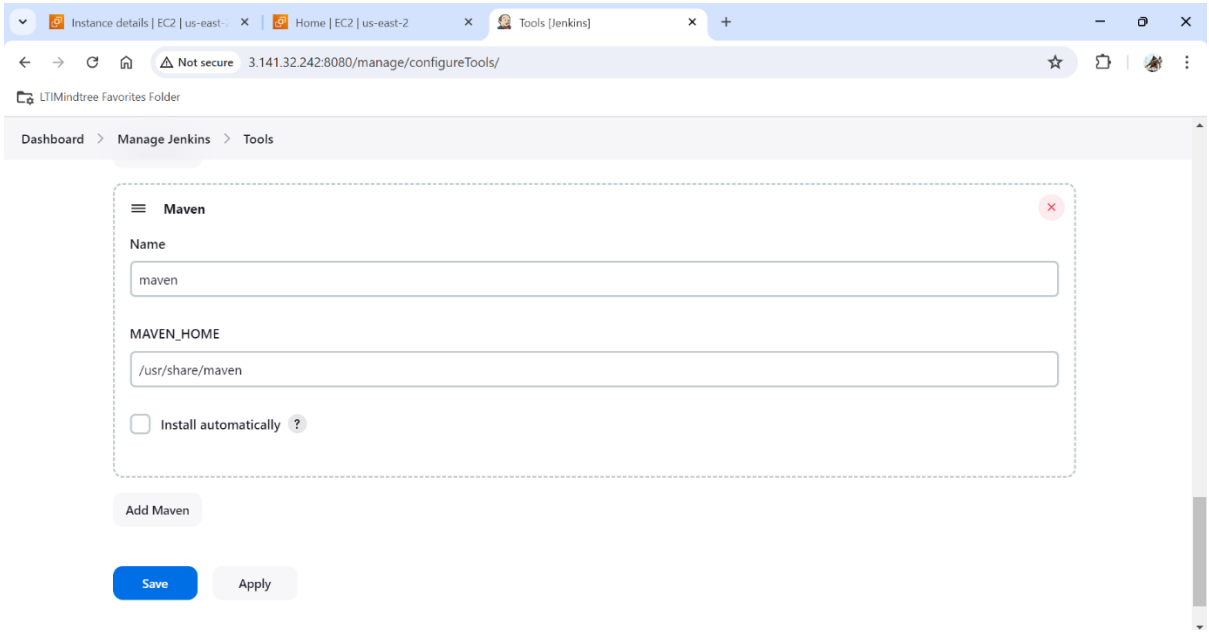
Now run command mvn -h -> it will give directory address of maven and Java

```
[root@ip-172-31-10-10 ~]# mvn -v
Apache Maven 3.8.4 (Red Hat 3.8.4-3.amzn2023.0.5)
Maven home: /usr/share/maven
Java version: 17.0.16, vendor: Amazon.com Inc., runtime: /usr/lib/jvm/java-17-amazon-corretto.x86_64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.1.147-172.266.amzn2023.x86_64", arch: "amd64", family: "unix"
[root@ip-172-31-10-10 ~]#
```

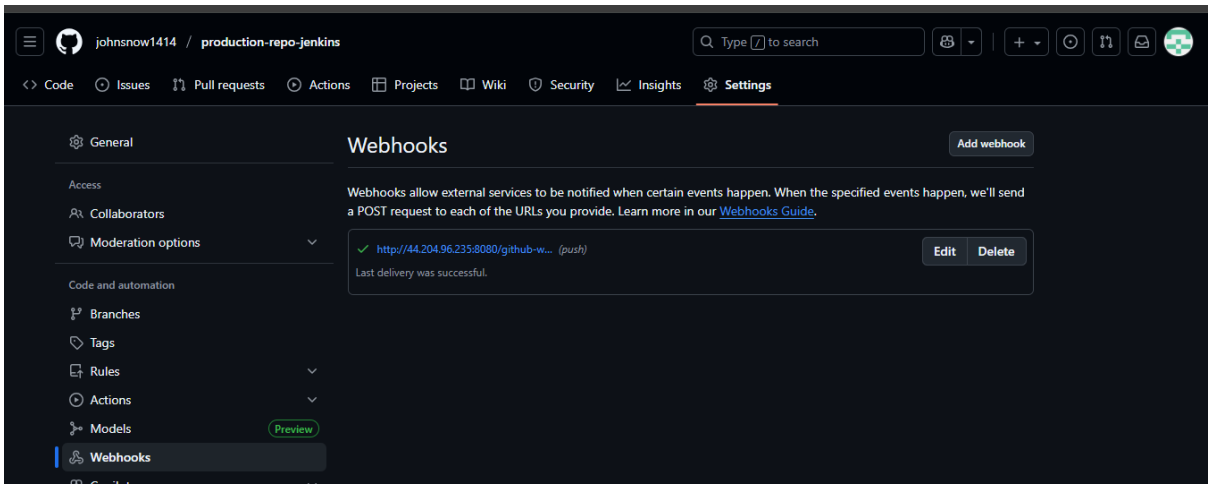
Giving java path here



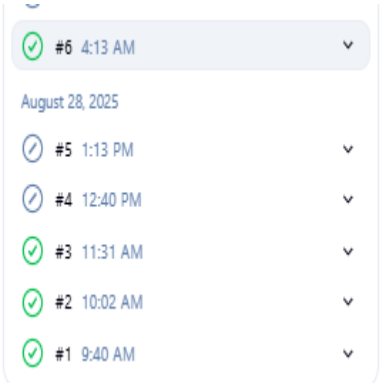
Giving maven path

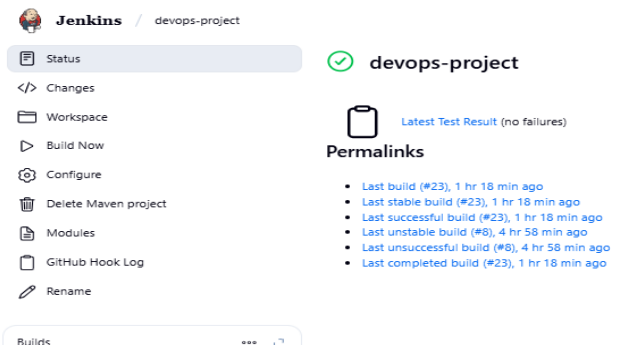


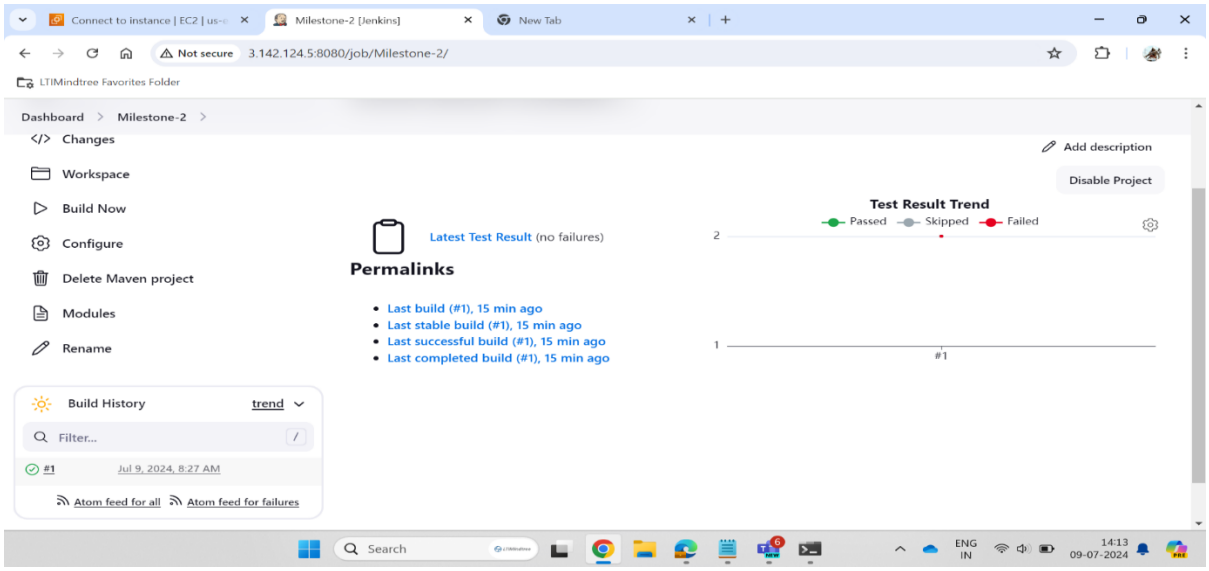
Setting up web hooks for our git repository -> use secret key(token) generated in Jenkins



Now we build in Jenkins



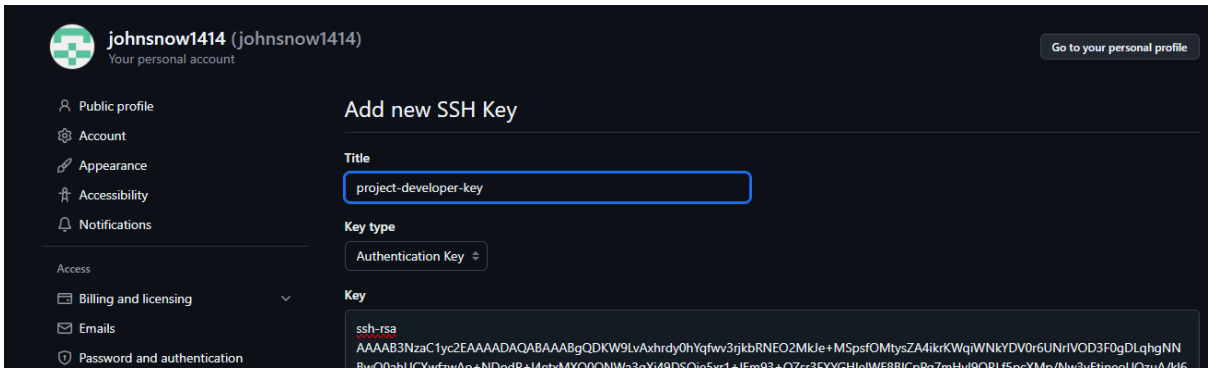




We can see the build result

Now Setting up developer instance -> lauch ec2 mahine

Adding ssh key to our github



Adding remote origin in our dev server

And pulling the code fomr our repository

```
[root@dev-server data]# git remote -v
[root@dev-server data]# git remote add origin git@github.com:johnsnow1414/production-repo-jenkins.git
[root@dev-server data]# git remote -v
origin  git@github.com:johnsnow1414/production-repo-jenkins.git (fetch)
origin  git@github.com:johnsnow1414/production-repo-jenkins.git (push)
[root@dev-server data]# git pull origin main
The authenticity of host 'github.com (140.82.114.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 141, done.
remote: Counting objects: 100% (141/141), done.
remote: Compressing objects: 100% (75/75), done.
Receiving objects: 100% (141/141), 22.35 KiB | 11.17 MiB/s, done.
Resolving deltas: 100% (36/36), done.
remote: Total 141 (delta 36), reused 109 (delta 23), pack-reused 0 (from 0)
From github.com:johnsnow1414/production-repo-jenkins
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main
[root@dev-server data]# ls
Dockerfile  README.md  pom.xml  regapp-deploy.yml  regapp-service.yml  server  webapp
[root@dev-server data]#
```

Making some changes and commiting

```
[root@dev-server data]# vim README.md
[root@dev-server data]# git add README.md
[root@dev-server data]# git commit -m "first"
```

Pushing some changes to our git repository

```
[root@dev-server data]# git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:johnsnow1414/production-repo-jenkins.git
   9199ea0..be599c9  main -> main
```

Configuring TOMCAT: launch ec2-instance for tomcat

Installing tomcat

```
[root@tom-cat-server ~]# wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.108/bin/apache-tomcat-9.0.108.tar.gz
--2025-08-28 10:55:51--  https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.108/bin/apache-tomcat-9.0.108.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

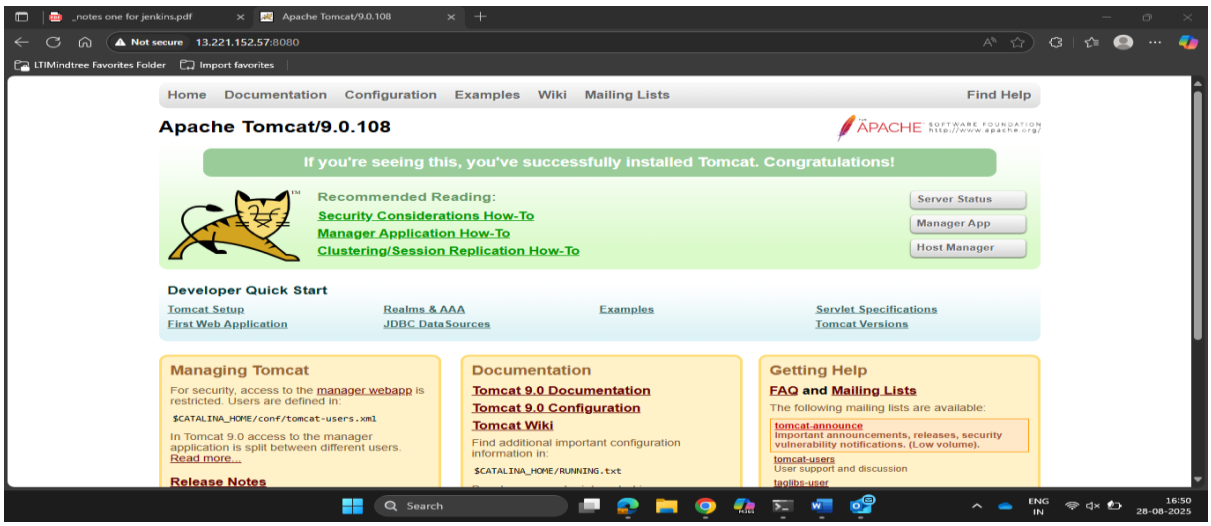
Configuring tomcat -server

```
[root@tom-cat-server bin]# chmod +x startup.sh
[root@tom-cat-server bin]# chmod +x shutdown.sh
[root@tom-cat-server bin]# find -name context.xml
[root@tom-cat-server bin]# cd ..
[root@tom-cat-server apache-tomcat-9.0.108]# find -name context.xml
./conf/context.xml
./webapps/docs/META-INF/context.xml
./webapps/examples/META-INF/context.xml
./webapps/host-manager/META-INF/context.xml
./webapps/manager/META-INF/context.xml
[root@tom-cat-server apache-tomcat-9.0.108]# vim ./webapps/examples/META-INF/context.xml
[root@tom-cat-server apache-tomcat-9.0.108]# vim ./webapps/host-manager/META-INF/context.xml
[root@tom-cat-server apache-tomcat-9.0.108]# vim ./webapps/host-manager/META-INF/context.xml
[root@tom-cat-server apache-tomcat-9.0.108]# vim ./webapps/manager/META-INF/context.xml
[root@tom-cat-server apache-tomcat-9.0.108]# cd conf
[root@tom-cat-server conf]# ls
catalina.policy      context.xml           jaspic-providers.xsd  server.xml       tomcat-users.xsd
catalina.properties  jaspic-providers.xml  logging.properties    tomcat-users.xml  web.xml
[root@tom-cat-server conf]# vim tomcat-users.xml
```

Start -tomcat

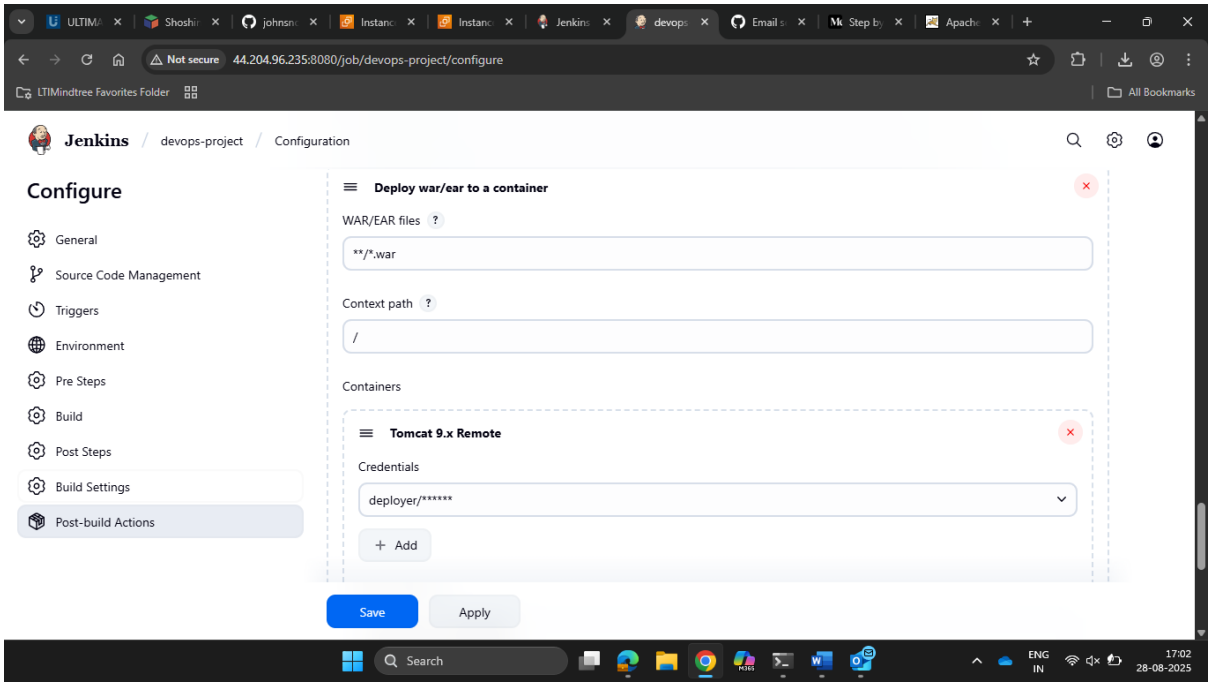
```
[root@tom-cat-server bin]# ./startup.sh
Using CATALINA_BASE:   /root/apache-tomcat-9.0.108
Using CATALINA_HOME:   /root/apache-tomcat-9.0.108
Using CATALINA_TMPDIR: /root/apache-tomcat-9.0.108/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /root/apache-tomcat-9.0.108/bin/bootstrap.jar:/root/apache-tomcat-9.0.108/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
[root@tom-cat-server bin]#
```

We can now see the tomcat server working on <public-ip-ec2>:8080



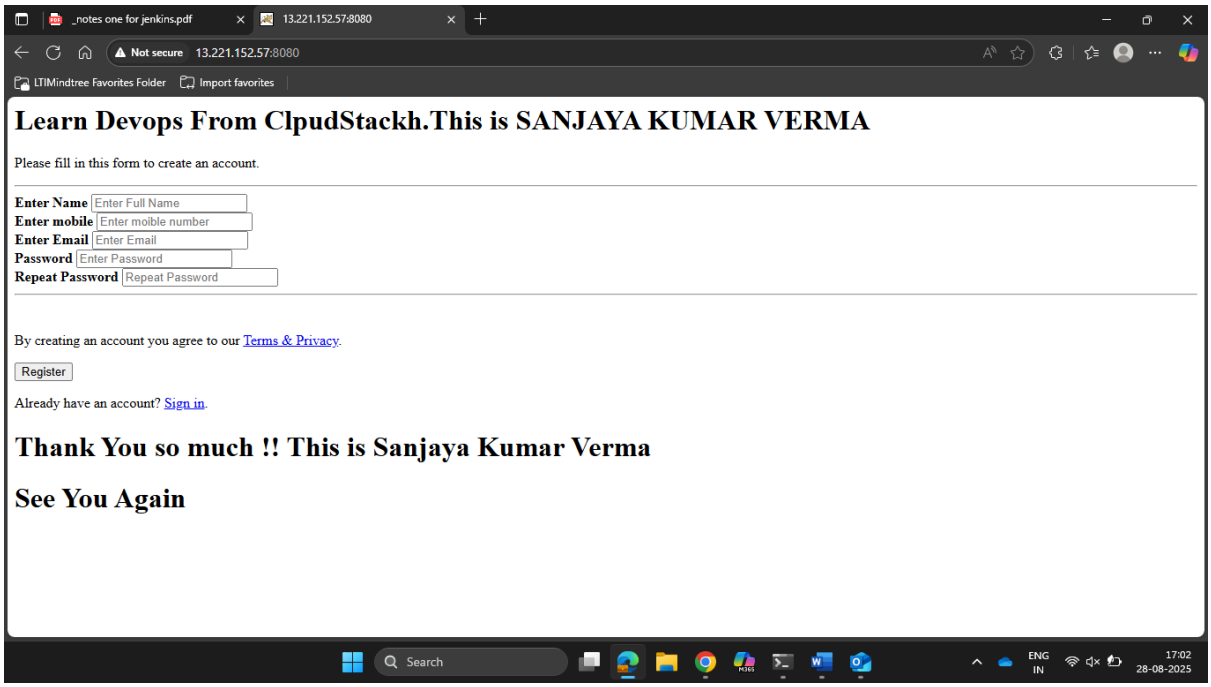
Now configure TOMCAT on Jenkins

Creating .war file for tomcat server



After configure we will build

Now we can see our page on tomcat server



NOW setting up docker instance

Configuring IAM user

```
[root@docker-server .aws]# aws configure
AWS Access Key ID [None]: AKIAROFD5A6AYWS6KU6L
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: table
```

Exchange ssh key from Jenkins to docker and docker to Jenkins

Do it both the instance

```
# Explicitly disable PasswordAuthentication. By presetting it, we
# avoid the cloud-init set_passwords module modifying sshd_config and
# restarting sshd in the default instance launch configuration.
PasswordAuthentication yes
PermitEmptyPasswords no
```

```
[root@docker-server ~]# vim /etc/ssh/sshd_config
[root@docker-server ~]# systemctl restart sshd
[root@docker-server ~]#
```

Now copy ssh-key of both instance to each other

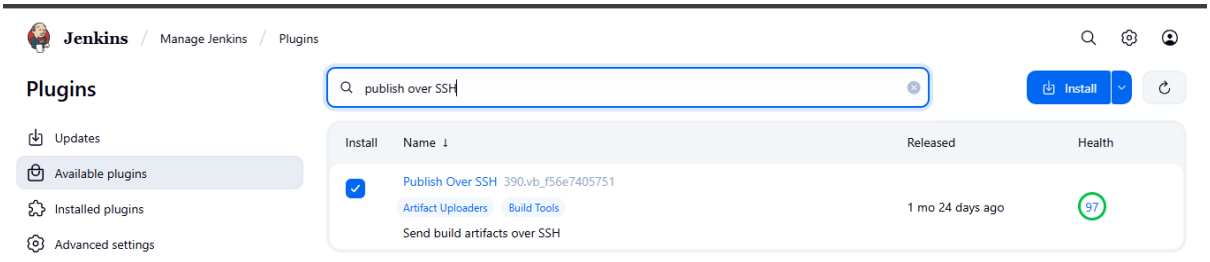
```
[root@jenkins-server ~]# ssh-copy-id root@172.31.5.172
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.10.10's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.10.10'"
and check to make sure that only the key(s) you wanted were added.
```

Now Configuring Docker in Jenkins

So that we can send files over ssh

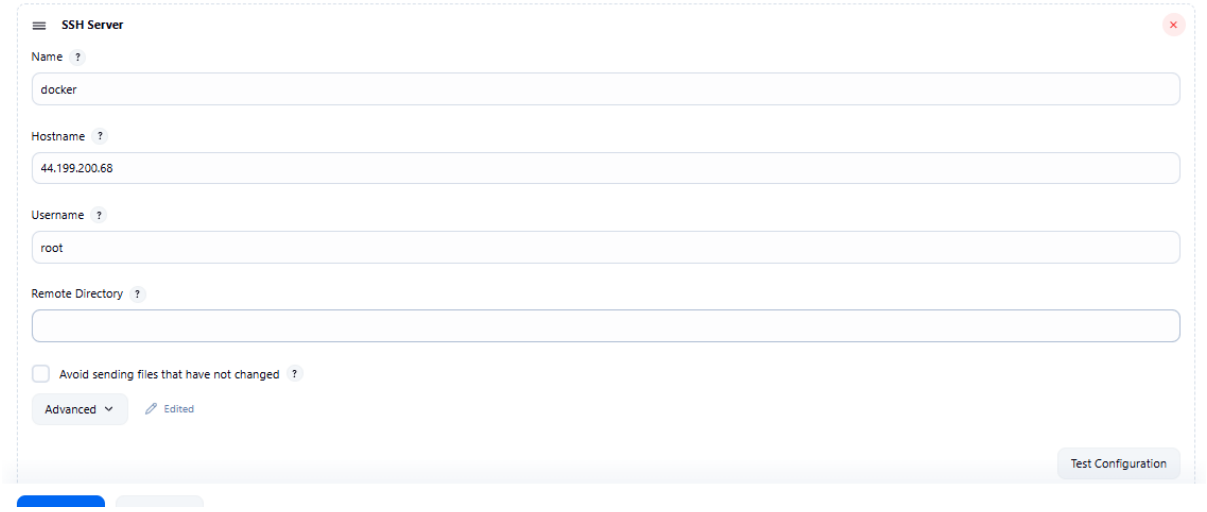
Install Plugins **publish over ssh**



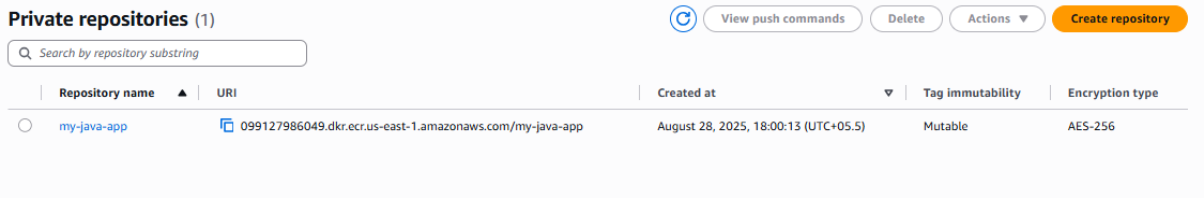
In manage Jenkins

Got to system

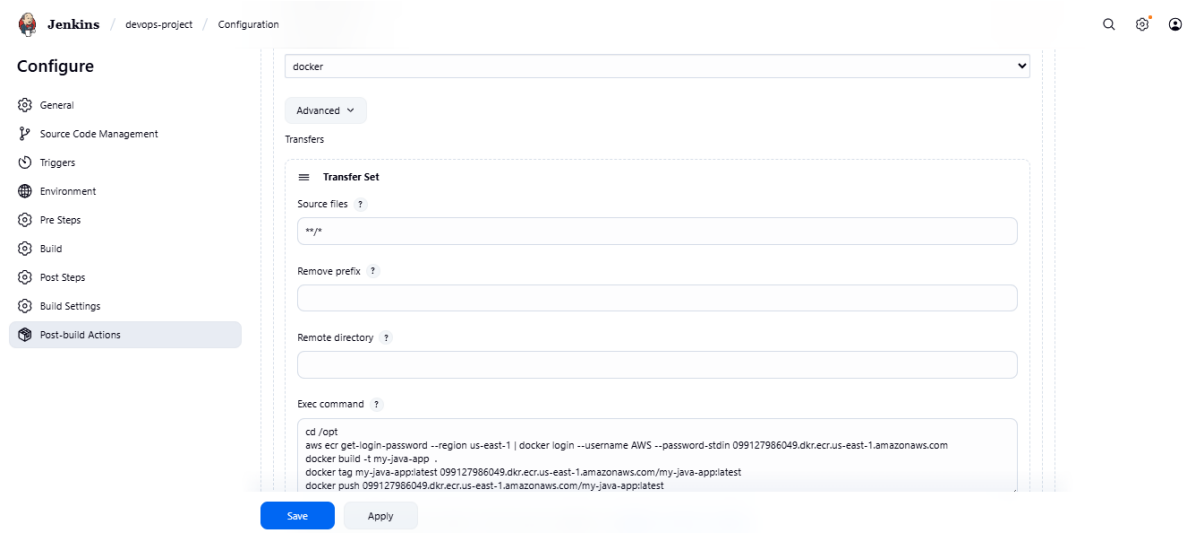
Add ssh SERVER



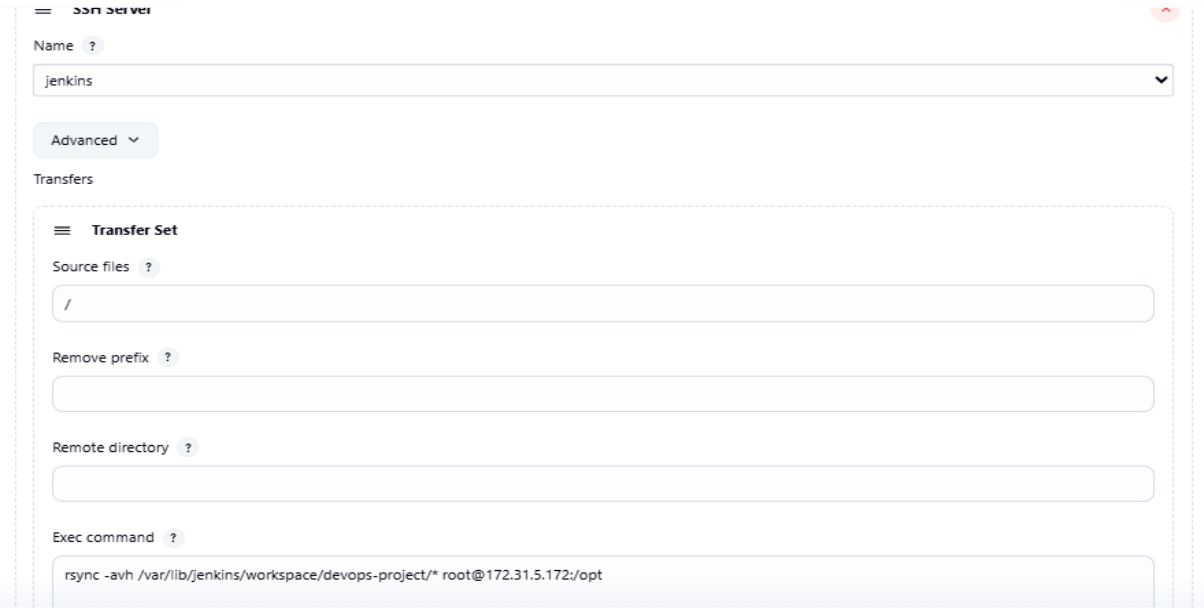
In ECR create private repository in AWS



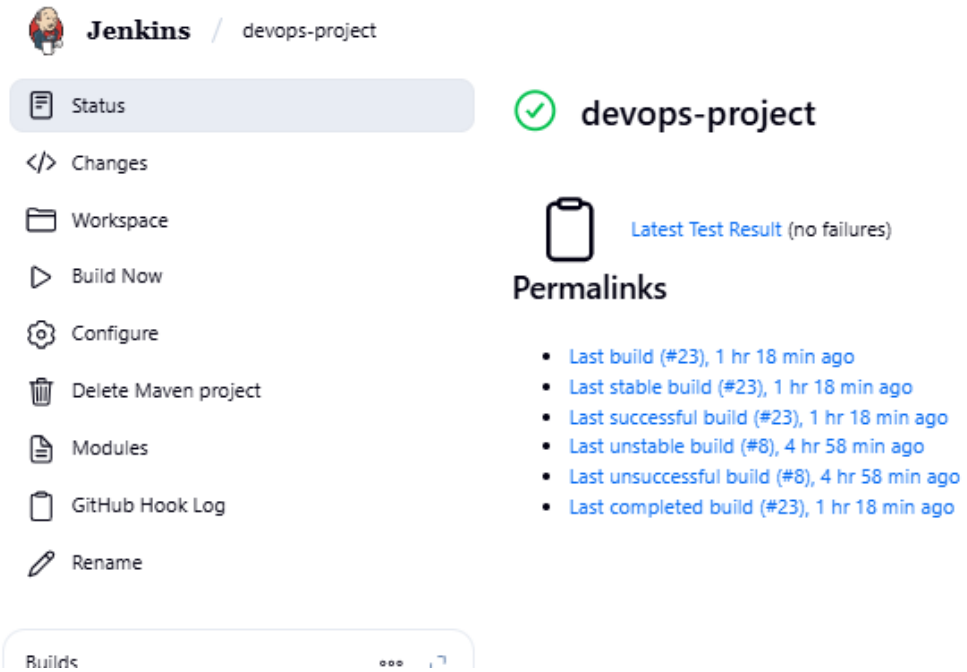
Configure SSH in post build action



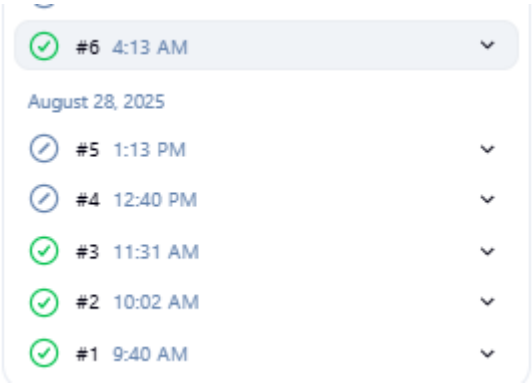
Set up ssh server for Jenkins to send file from workspace to docker



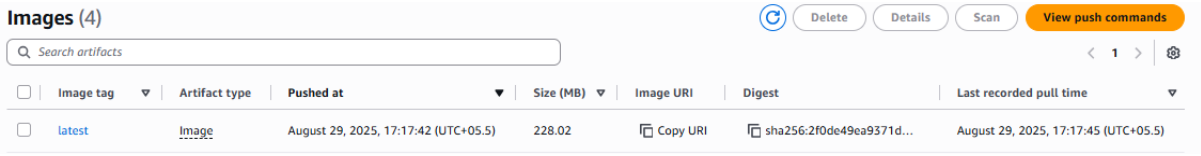
Now we will run build in Jenkins



As we can see the build is scuccesful



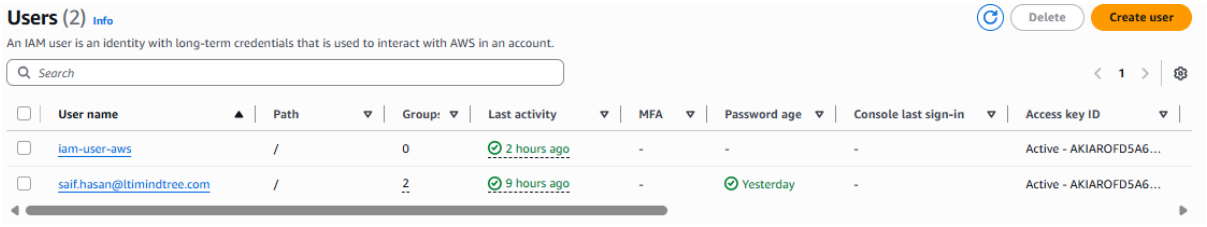
now we see that after successful build ->the Image is available in ECR



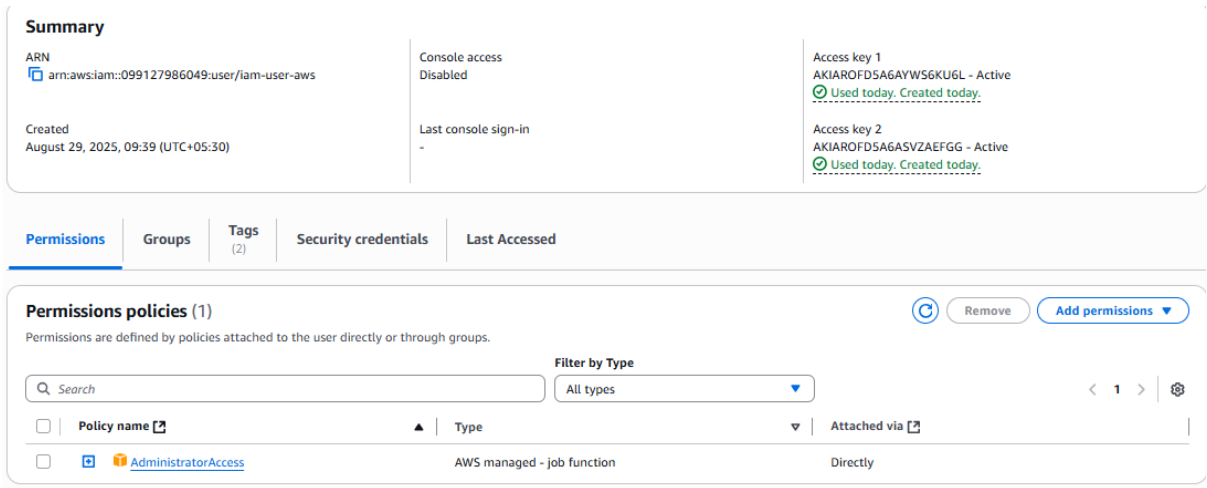
EKS :-

- First launch a EC2 instance
- create Iam user
- create role
- Attach role to ec2

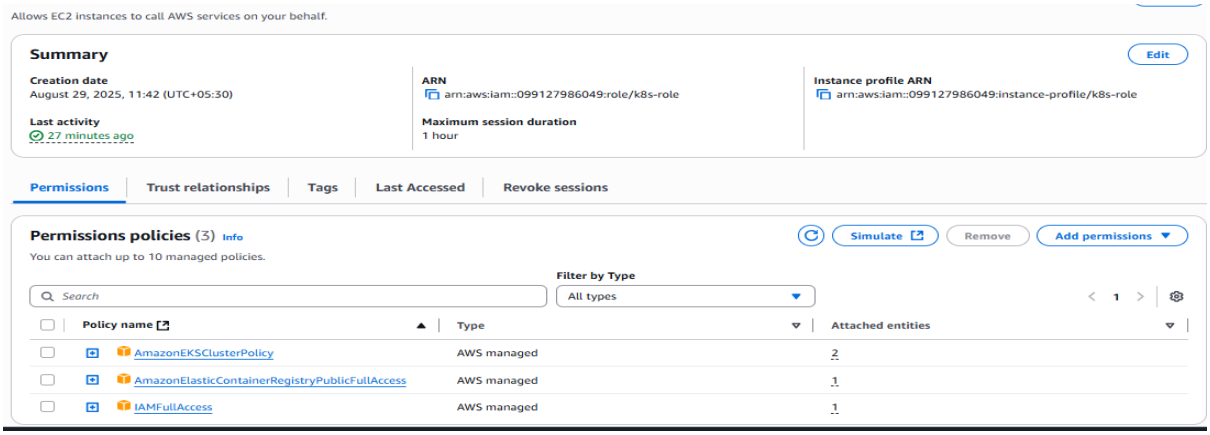
Creating IAM user



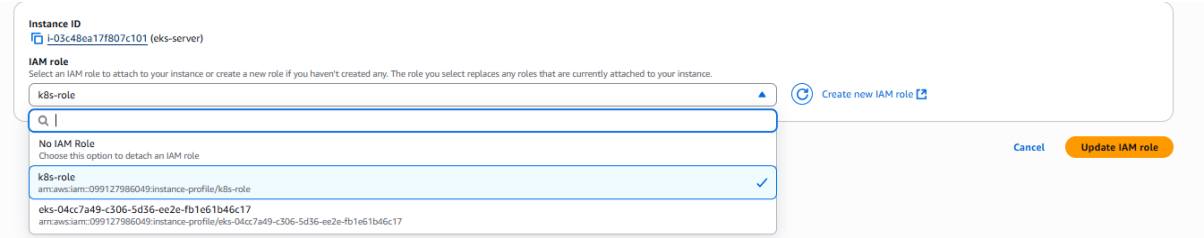
With administrator access



Now create Role with attached polices



Attach the role to Ec2 instance for EKS



Configure AWS on EKS instance

```
[root@eks-setup-machine ~]# yum install awscli* -y
```

```
[root@eks-setup-machine ~]# aws configure
AWS Access Key ID [None]: AKIAROFD5A6A7PYOW7A5
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: table
```

Download eksctl using curl cmd

```
[root@eks-setup-machine .aws]# curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.
gz" | tar xz -C /tmp
```

```
[root@eks-setup-machine tmp]# sudo mv /tmp/eksctl /usr/local/bin
[root@eks-setup-machine tmp]# ls
```

Installing kubernetes

```
[root@eks-setup-machine tmp]# eksctl version
0.214.0
[root@eks-setup-machine tmp]# 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 53.7M  100 53.7M    0     0  70.1M      0 --:--:-- --:--:-- --:--:-- 70.1M
[root@eks-setup-machine tmp]# sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
[root@eks-setup-machine tmp]# kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
[root@eks-setup-machine tmp]# ssh-keygen
Generating public/private rsa key pair.
```

```
[root@eks-setup-machine tmp]# curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.google
-release/release/stable.txt)/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0  70.1M      0 --:--:-- --:--:-- --:--:-- 70.1M
[root@eks-setup-machine tmp]# sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
[root@eks-setup-machine tmp]# kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
[root@eks-setup-machine tmp]# ssh-keygen
```

Creating the Cluster

```
[root@eks-setup-machine tmp]#
[root@eks-setup-machine tmp]# eksctl create cluster --name my-cluster --region us-east-1 --version 1.32 --vpc-public-sub
nets subnet-0b899f2e728e7b521,subnet-089faf81e0b5d0ecf --without-nodegroup
2025-08-29 06:27:27 [ℹ]  eksctl version 0.214.0
2025-08-29 06:27:27 [ℹ]  using region us-east-1
```

Creating the node group

```
[root@eks-setup-machine tmp]# eksctl create nodegroup \
  --cluster my-cluster \
  --region us-east-1 \
  --name my-node-group \
  --node-ami-family Ubuntu2204 \
  --node-type t2.small \
  --subnet-ids subnet-0b899f2e728e7b521,subnet-089faf81e0b5d0ecf \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key /root/.ssh/id_rsa.pub
2025-08-29 06:45:42 [ℹ]  will use version 1.32 for new nodegroup(s) based on control plane version
```

We will now be able to see nodes

```
[root@eks-setup-machine ~]# kubectl get node
NAME                           STATUS   ROLES    AGE     VERSION
ip-172-31-2-21.ec2.internal    Ready    <none>   6h40m   v1.32.5
ip-172-31-22-119.ec2.internal  Ready    <none>   6h40m   v1.32.5
ip-172-31-29-50.ec2.internal   Ready    <none>   6h40m   v1.32.5
```

Set password for root and enable password based authetication

```
[root@eks-setup-machine tmp]# passwd root
Changing password for user root.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
[root@eks-setup-machine tmp]# vim /etc/ssh/sshd_config
[root@eks-setup-machine tmp]# systemctl restart sshd
[root@eks-setup-machine tmp]# systemctl enable sshd
```

Now in copy Jenkins ssh key to EKS-ec2 instance

```
[root@jenkins-server ~]# ssh-copy-id root@172.31.5.172
```

ssh key added to eks-ec2 instance

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.10.10's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.10.10'"
and check to make sure that only the key(s) you wanted were added.
```

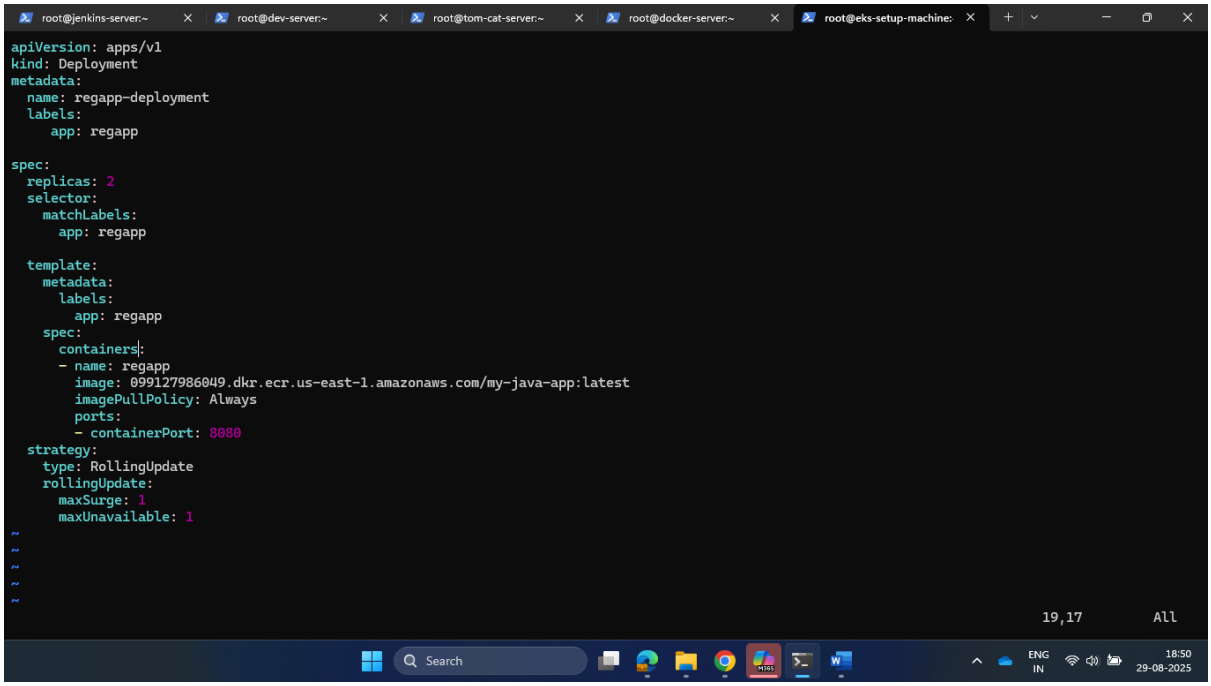
After adding ssh key.

Jenkin is ready to make connection with EKS server.

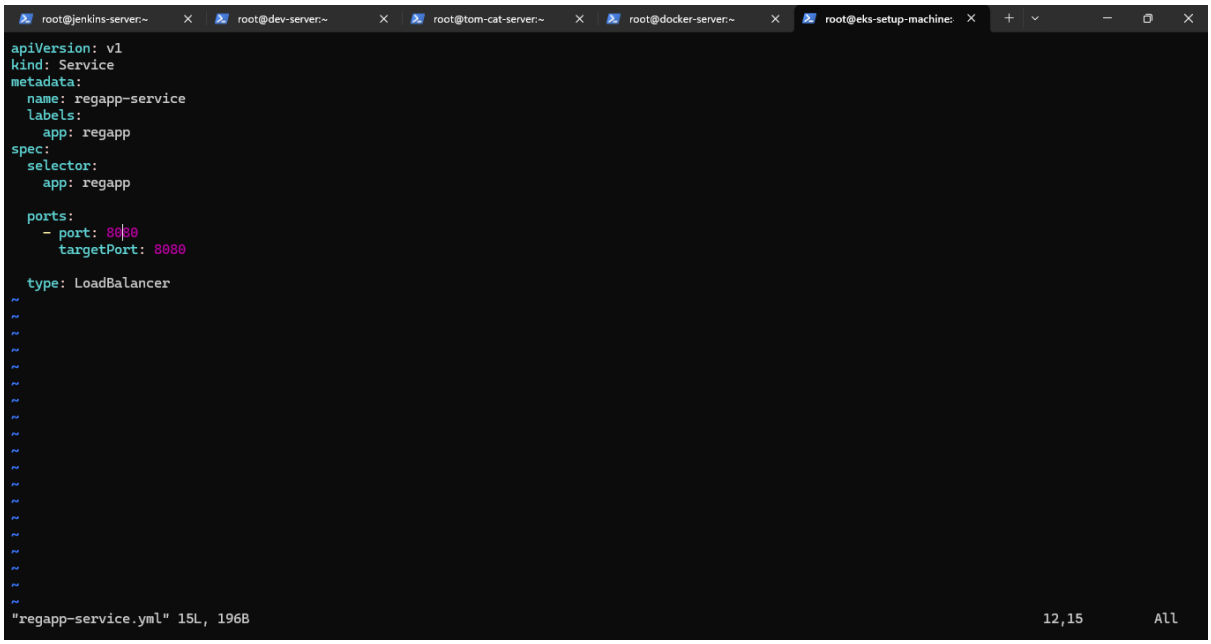
Now we will create pod.

Creating regapp-deploy.yml for deploying pod

vim reagapp-deploy.yml

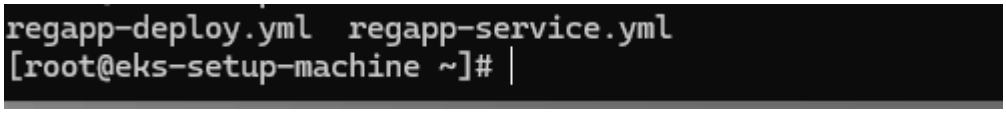


vim reagapp-service.yml



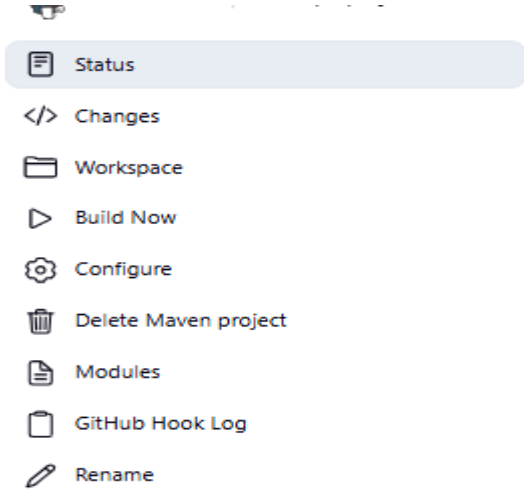
This file is for load balancer

Now the two file are created in eks-ec2-server

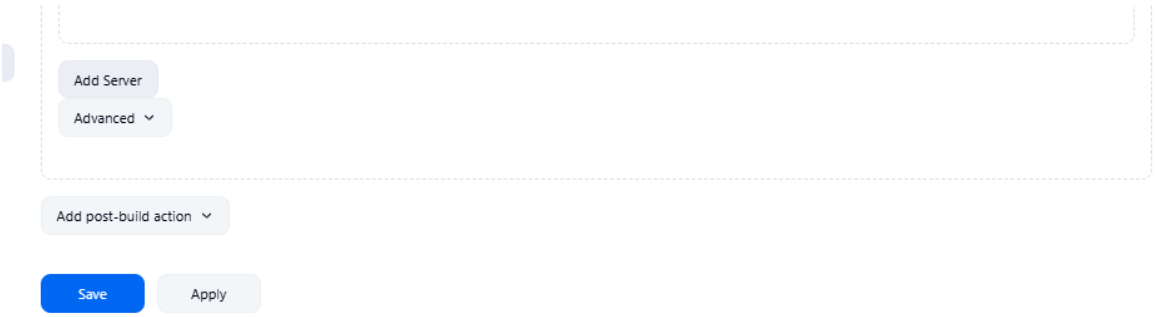


Now configure SSH server for EKS in Jenkins

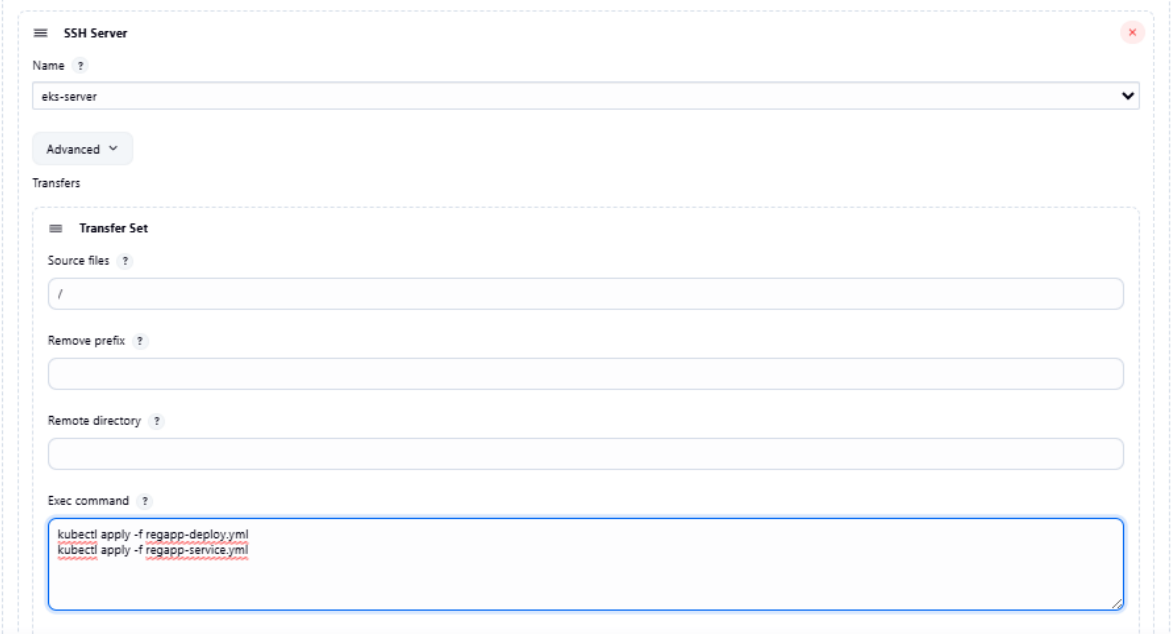
Click on configure



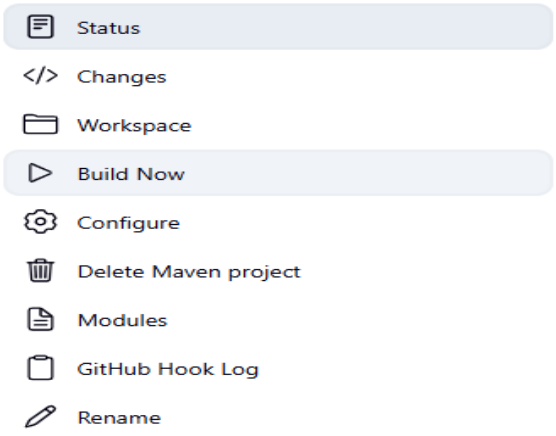
Click on Add server



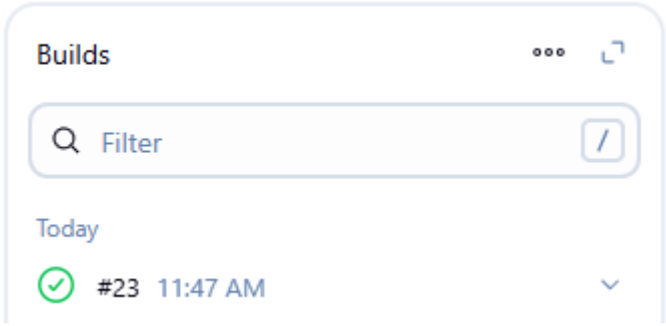
Now setup ssh server for eks -> run kubectl apply -f regapp-deploy.yml



Build now



Now we will build and we can see the build is successful



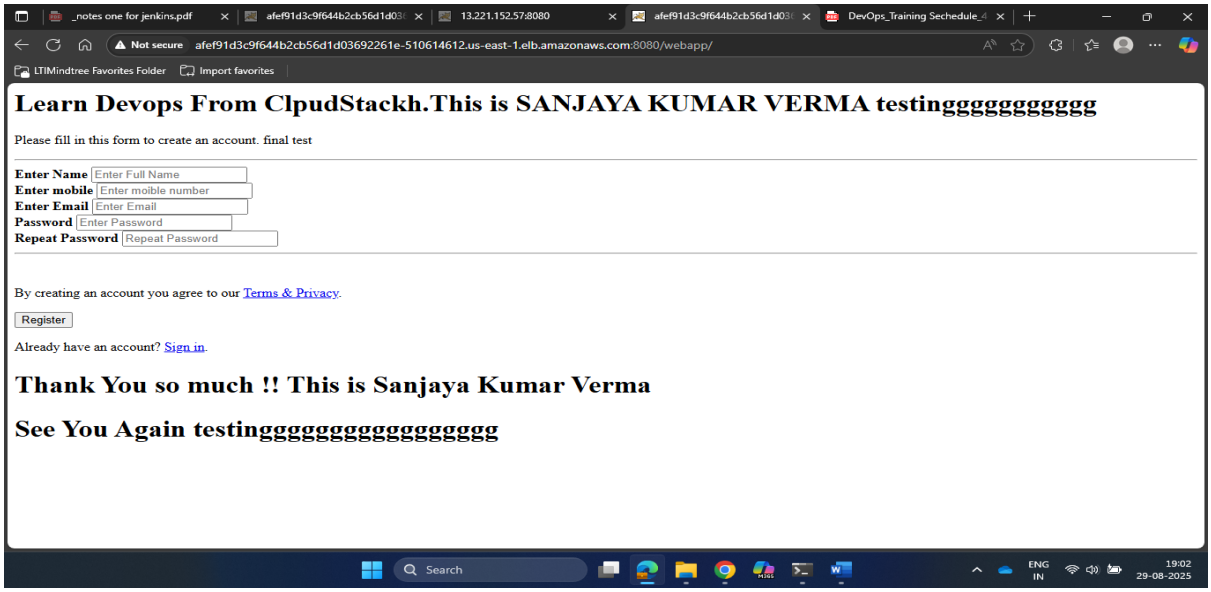
Now run kubectl get pod -> to see the pods

```
[root@eks-setup-machine ~]# kubectl get pod
NAME                                  READY   STATUS    RESTARTS   AGE
regapp-deployment-5f79cbfd8d-4ccvq    1/1     Running   0          101m
regapp-deployment-5f79cbfd8d-mgpxp    1/1     Running   0          101m
[root@eks-setup-machine ~]#
```

Run kubectl get svc -> to get load balancer external ip

```
[root@eks-setup-machine ~]# kubectl get svc
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)
kubernetes      ClusterIP      10.100.0.1      <none>                                                                   443/TCP
regapp-service  LoadBalancer   10.100.68.232   afef91d3c9f644b2cb56d1d03692261e-510614612.us-east-1.elb.amazonaws.com   8080:31326/TCP
[root@eks-setup-machine ~]#
```

Now copy the Externel-IP and paste it in browser and add 8080/webapp
Now wen see the application running



Now testing that the pipeline is successful or not by committing some changes in our source code.

Now logging in developer ec2 instance

Now pulling changes

```
[root@dev-server developer-code]# ls
java-code-with-maven
[root@dev-server developer-code]# cd java-code-with-maven/
[root@dev-server java-code-with-maven]#
[root@dev-server java-code-with-maven]# git pull origin main
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 30 (delta 12), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (30/30), 6.12 KiB | 783.00 KiB/s, done.
From github.com:johnsnow1414/production-repo-jenkins
 * branch            main       -> FETCH_HEAD
   fc1d304..bf1db2f  main       -> origin/main
Updating fc1d304..bf1db2f
Fast-forward
 Dockerfile                      | 3 +--
 webapp/src/main/webapp/index.jsp | 6 +++---
 2 files changed, 4 insertions(+), 5 deletions(-)
[root@dev-server java-code-with-maven]# git status
On branch main
Your branch is up to date with 'origin/main'.
```

Making some changes in index.jsp

```
[root@dev-server webapp]# vim index.jsp
```

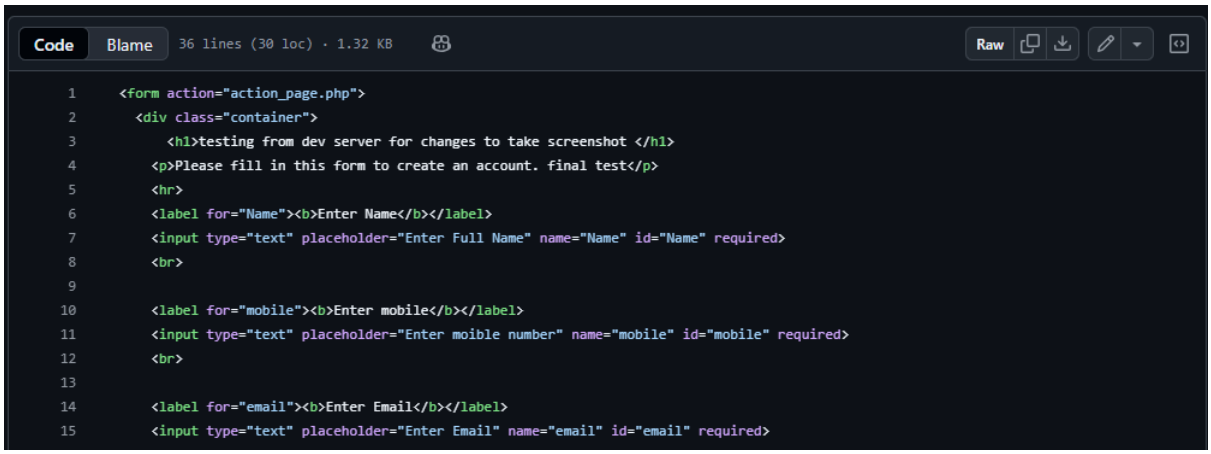
root@jenkins-server:~ | root@dev-server:~/develo | root@tom-cat-server:~ | root@docker-server:~ | root@eks-setup-machine:~

```
<form action="action_page.php">
  <div class="container">
        <h1>testing from dev server for changes to take screenshot </h1>
    <p>Please fill in this form to create an account. final test</p>
    <hr>

    <label for="Name"><b>Enter Name</b></label>
    <input type="text" placeholder="Enter Full Name" name="Name" id="Name" required>
    <br>

    <label for="mobile"><b>Enter mobile</b></label>
    <input type="text" placeholder="Enter moible number" name="mobile" id="mobile" required>
    <br>

    <label for="email"><b>Enter Email</b></label>
    <input type="text" placeholder="Enter Email" name="email" id="email" required>
    <br>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="psw" id="psw" required>
    <br>
```

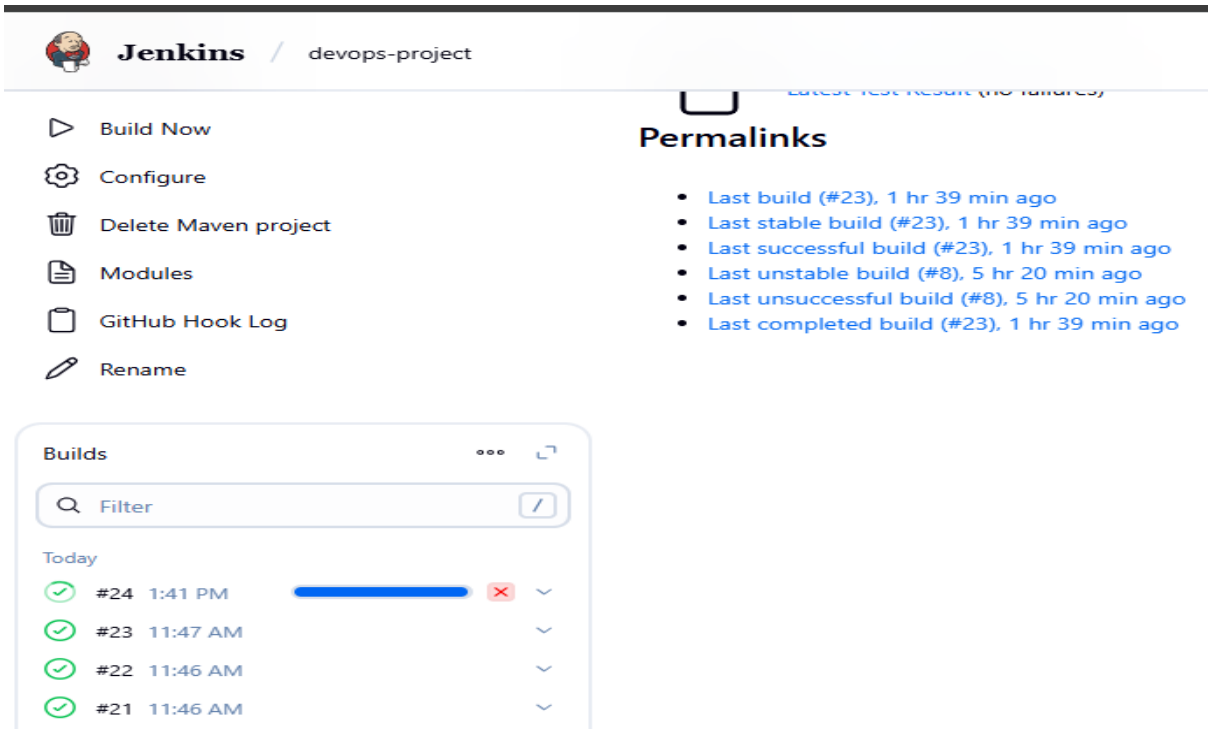
Committing the changes in our repo from our dev-machine

```
[root@dev-server webapp]# vim index.jsp
[root@dev-server webapp]# git add .
[root@dev-server webapp]# git commit -m "test commit"
[main 85258a0] test commit
 1 file changed, 1 insertion(+), 2 deletions(-)
[root@dev-server webapp]# git push origin main
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 588 bytes | 588.00 KiB/s, done.
Total 7 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:johnsnow1414/production-repo-jenkins.git
   bf1db2f..85258a0  main -> main
[root@dev-server webapp]#
```

We can see that the changes are reflecting in our repository on GitHub



We can now see that build is running automatically



Now we can see that the changes are reflected in our webpage