

## Day 4- Kubernetes - Pods & Services [ in this build java application]

<https://docs.google.com/document/d/1Is4h94KVFliaNxSuBZX9Spui98CYiFOlawjLfYa3wBU/edit> [this is very important.do this must]

<https://github.com/zen-class/zen-class-devops-documentation>

launch an instance & connect:

**Name and tags** [Info](#)

Name

demo-app [Add additional tags](#)

**Recents** **Quick Start**

Amazon Linux **aws**

macOS **Mac**

**Ubuntu** **ubuntu**

Windows **Microsoft**

Red Hat **Red Hat**

SUSE Linux **SUS**

[Browse more AMIs](#)  
Including AMIs from AWS, Marketplace and the Community

**Amazon Machine Image (AMI)**

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type **Free tier eligible**

ami-0c2af51e265bd5e0e (64-bit (x86)) / ami-0c938b21c7e598cd0 (64-bit (Arm))  
Virtualization: hvm ENA enabled: true Root device type: ebs

Select t2.medium.because install docker, Jenkins , Kubernetes cluster in single machine

**▼ Instance type** [Info](#) [Get advice](#)

Instance type

t2.medium

Family: t2 2 vCPU 4 GiB Memory Current generation: true

On-Demand Linux base pricing: 0.0496 USD per Hour

On-Demand Windows base pricing: 0.0676 USD per Hour

On-Demand RHEL base pricing: 0.0784 USD per Hour

On-Demand SUSE base pricing: 0.1496 USD per Hour

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

3 tools are required to launch your Kubernetes cluster

1. Kubectl
2. Eksctl

### 3. Jenkins

#### 1. Awscli

#### 2. Create Kubernetes cluster

#### Kubectrl:

```
ubuntu@ip-172-31-44-53:~$ curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
```

```
ubuntu@ip-172-31-44-53:~$ chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
Client Version: v1.19.6-eks-49a6c0
```

#### Ekctl:

```
ubuntu@ip-172-31-44-53:~$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_${uname -s}_amd64.tar.gz" | tar
xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

```
ubuntu@ip-172-31-44-53:~$ sudo apt update
```

Duplicate the machine to install Jenkins ,docker:

1. install Jenkins [inside Jenkins install awscli & create Kubernetes cluster]
2. install docker

#### install Jenkins:

```
To check for new updates run: sudo apt update
```

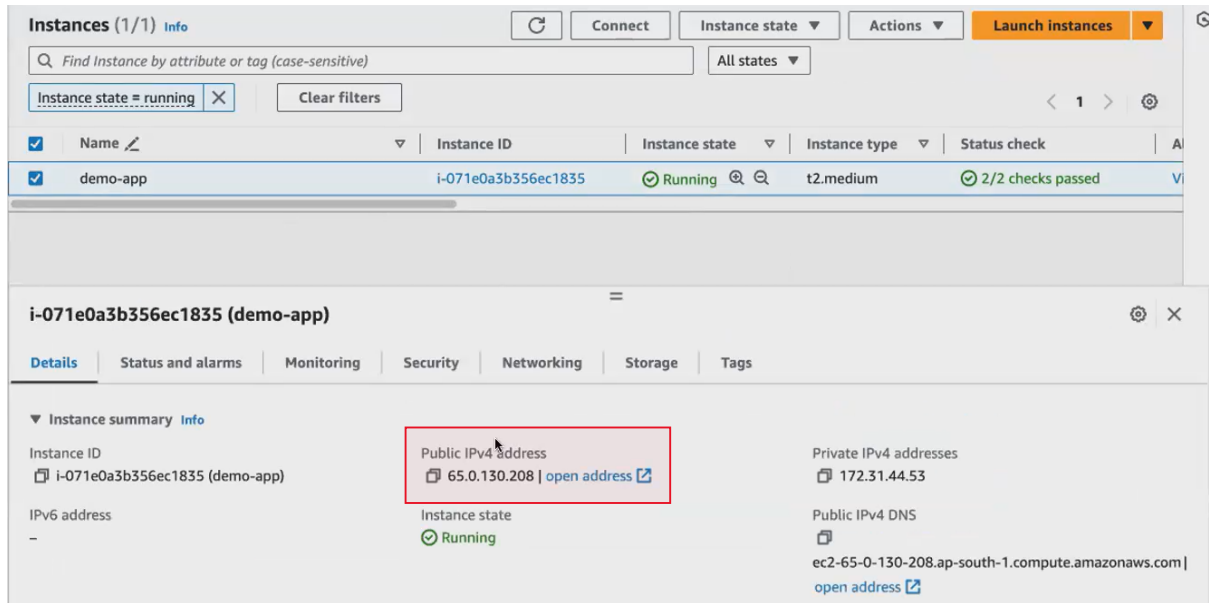
```
ubuntu@ip-172-31-42-111:~$ sudo apt install openjdk-17-jdk -y
```

```
ubuntu@ip-172-31-42-111:~$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

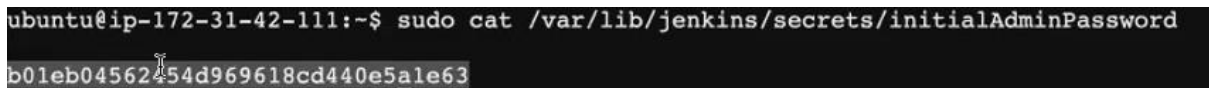
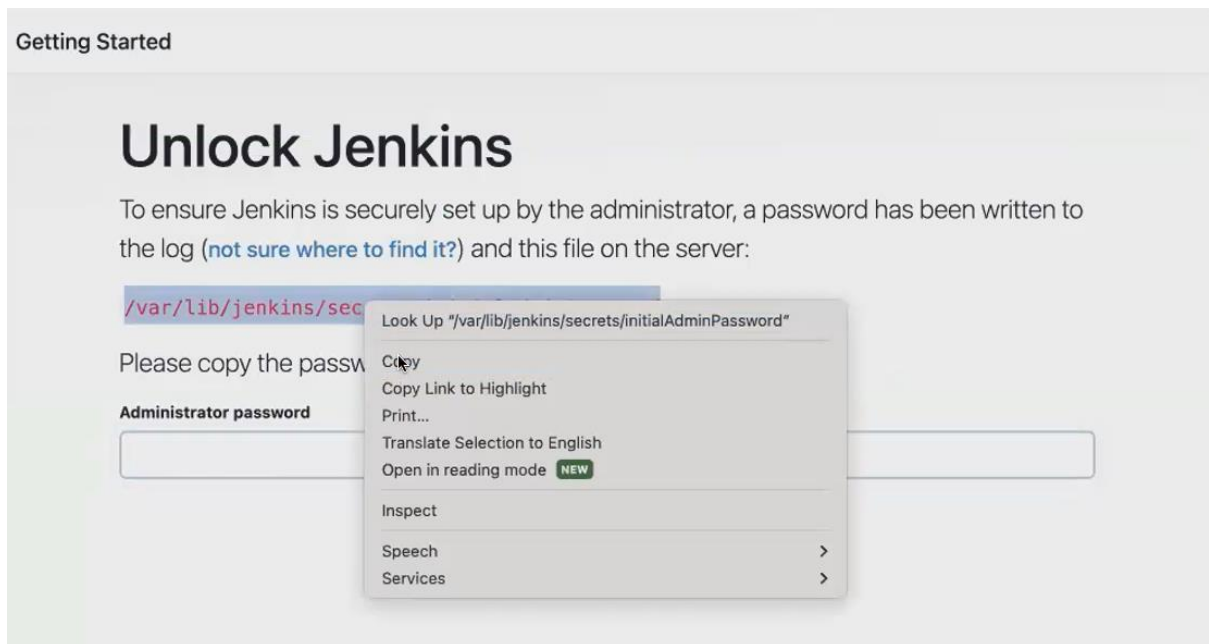
Duplicate the machine: [where we installed Jenkins]

- So now you can see my Jenkins from the Jenkins user and launching my cluster.
- Okay, let it launch. I will take another session.

- So, in this machine we have installed Jenkins. Right, we will go and open up the port number [8080].



Jenkins page opened.



## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

## Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

### Install suggested plugins

Install plugins the Jenkins community finds most useful.

### Select plugins to install

Select and install plugins most suitable for your needs.

Get into the Jenkins user:

1. install awscli
2. create Kubernetes cluster

get into Jenkins user [sudo su -jenkins]

```
ubuntu@ip-172-31-44-53:~$ sudo su - jenkins
jenkins@ip-172-31-44-53:~$
```

Awscli:

```
jenkins@ip-172-31-44-53:~$ aws configure
AWS Access Key ID [None]: AKIASGIII4IGBNATXP47
AWS Secret Access Key [None]: dPWqmF8F0ZGP46Zdgo9sBQAHR25tZvXeLtIjbz2
Default region name [None]: ap-south-1
Default output format [None]: json
jenkins@ip-172-31-44-53:~$
```

Create Kubernetes cluster inside Jenkins user:

- Now you're inside the Jenkins user. from the Jenkins user Only, we have to launch the cluster.
- Only then your Jenkins can identify this cluster.
- Create cluster **name of your cluster**
- which **region** you want to launch a cluster
- **type of worker node** that you want to launch.
- worker node by default will be created with **instance type** t2.small. now the cluster, while the cluster is being created.

```
jenkins@ip-172-31-42-111:~$ eksctl create cluster --name test-cluster --region ap-south-1
2024-07-24 11:41:59 [i] eksctl version 0.187.0
2024-07-24 11:41:59 [i] using region ap-south-1
```

open Jenkins page:

The screenshot shows the Jenkins Setup Wizard interface in a web browser. The browser's address bar displays the URL 65.132.30:8080. The page title is "Getting Started". The main heading is "Create First Admin User". Below this, there are five form fields: "Username" with the value "admin", "Password" with masked characters ".....", "Confirm password" with masked characters ".....", "Full name" with the value "admin", and "E-mail address" with the value "yasmin@guvi.in". At the bottom of the form, there are two buttons: "Skip and continue as admin" and "Save and Continue". The Jenkins version "Jenkins 2.452.3" is visible in the bottom left corner.

Getting Started

## Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.452.3 Skip and continue as admin Save and Continue

The screenshot shows the Jenkins Setup Wizard interface for "Instance Configuration". The page title is "Getting Started". The main heading is "Instance Configuration". Below this, there is a "Jenkins URL:" label followed by a text input field containing the value "http://65.132.30:8080/". Below the input field, there is a paragraph of text explaining the Jenkins URL. The text states: "The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps. The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links."

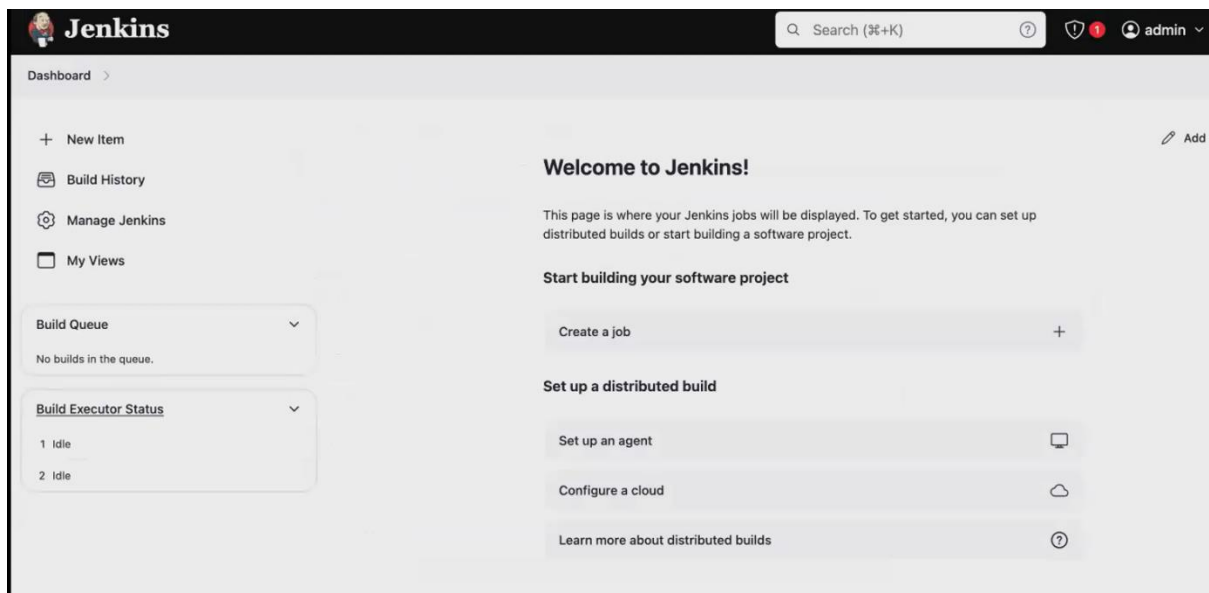
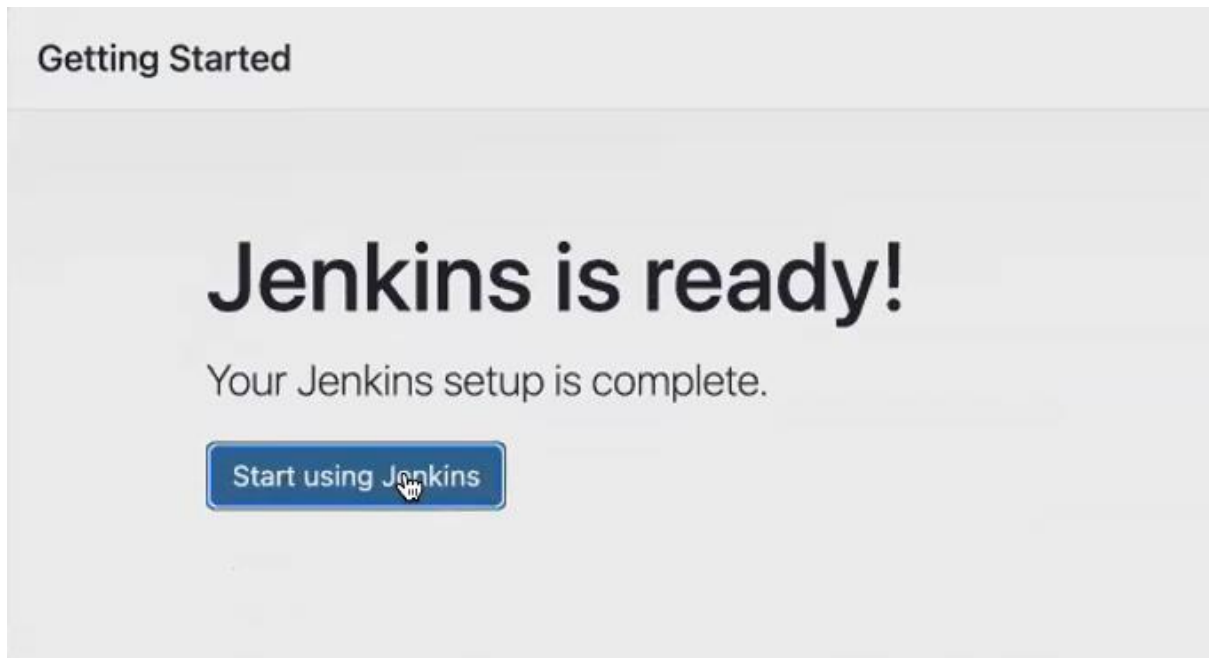
Getting Started

## Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.



**All are doing in the same machine.**

- so once you have launched the cluster, what you have to do.
- our application, will go and look into the application. **Our application is a Java application.**
- so, you have to build your application. That is, you have to artifact. You have to package your application, using Maven so for that what you need. You have to install, Maven.

## Install maven:

- why, I'm installing in this machine.
- My Jenkins have to build the application. My Jenkins should build the Java application via Maven, so I have to give my Jenkins. I have to install maven and my Jenkins machine.
- So, I have installed Maven.
- so, I'm installed Maven, Java, Jenkins. Everything is running.
- We will also install Docker. So do we have to install

```
ubuntu@ip-172-31-42-111:~$ sudo apt install maven -y
```

```
ubuntu@ip-172-31-42-111:~$ mvn --version
Apache Maven 3.6.3
Maven home: /usr/share/maven
```

## Install docker:

```
ubuntu@ip-172-31-42-111:~$ sudo apt install docker.io
```

```
ubuntu@ip-172-31-42-111:~$ sudo usermod -aG docker jenkins
ubuntu@ip-172-31-42-111:~$ sudo usermod -aG docker ubuntu
```

- after installing Docker, make sure you give permission for Jenkins and Ubuntu to the docker user.
- So you don't have to use the pseudo command to run your docker commands.

```
ubuntu@ip-172-31-42-111:~$ sudo systemctl restart jenkins
```



## Create a pipeline in Jenkins:


### Click build project


Enter an item name

demo-cicd

» Required field

---

 **Freestyle project**  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

 **Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**General** Enabled 

Description

My Java Application deployment in EKS Cluster

Plain text [Preview](#)

☐ Discard old builds 

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url 

<https://github.com/yasminjeelani/EKS-CICD.git>

Later on do automation. don't select poll scm



### 1. tools

### 2. environment

### 3. stages

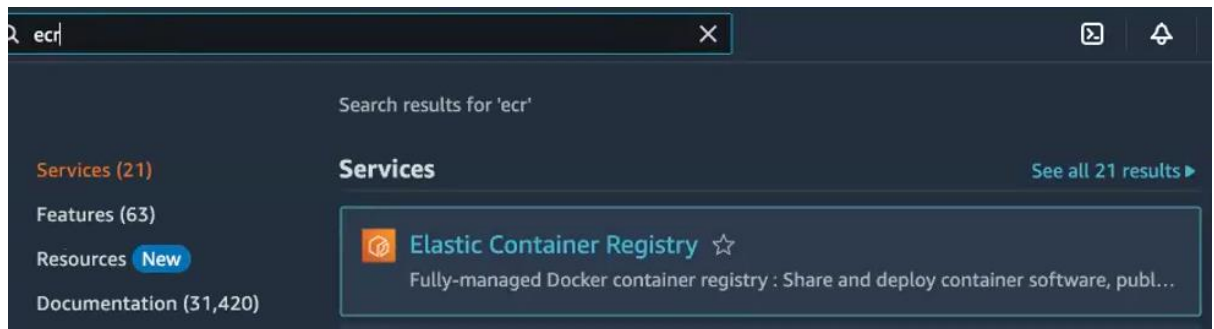
#### 1<sup>st</sup> stage [checkout]

#### 2<sup>nd</sup> stage [built the application using maven]

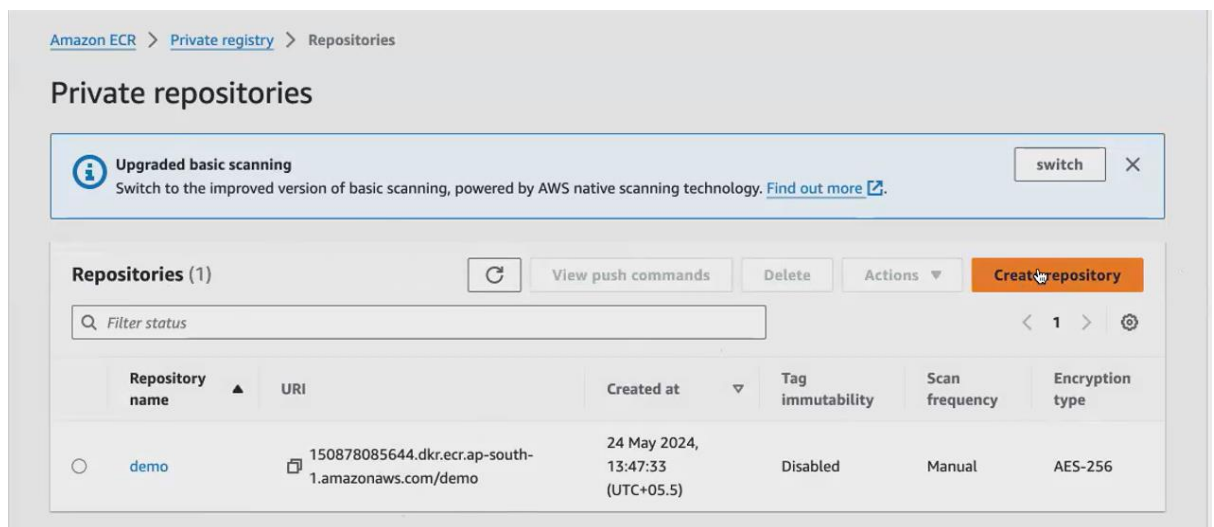
#### 3<sup>rd</sup> stage [build docker ]

#### 4<sup>th</sup> stage [ push the docker image to ecr registry]

1. Your Jenkins file always starts with pipeline. And after the pipeline, what you need.
2. you have to before agent. Before you specify any agent here,
3. we will be specifying the tools. So here I'm using a tool called Maven. So here I'm using a tool called Maven, and I'm naming that tool as Maven 3.
4. I will tell you where to configure this, Maven 3. you have installed Maven in your Jenkins machine, and you should also bring that Plugin. You have to install that Plugin Maven Plugin to your Jenkins
5. next is the environment. So here **I will be using my Ecr registry**. Once the docker build is ready, once the docker is ready. Once your docker image is ready, we will be pushing the docker image to the Ecr registry.
6. So where you get the Ecr registry, you know, aws, you have something. You can also push it to your Docker Hub registry. But since we have already done it, we have already seen configuring the Doctor Hub registry without Jenkins.



This is same as docker repository. We have push the docker image in this repo.



# Create repository

## General settings

### Visibility settings [Info](#)

Choose the visibility setting for the repository.

☒ **Private**

Access is managed by IAM and repository policy permissions.

☐ **Public**

Publicly visible and accessible for image pulls.

### Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.

150878085644.dkr.ecr.ap-south-1.amazonaws.com/

4 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.


### Tag immutability [Info](#)

Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

☒ **Disabled**




 Once a repository has been created, the visibility setting of the repository can't be changed.


Click create repository


 **Created private repository**  
test has been successfully created in private registry

[Amazon ECR](#) > [Private registry](#) > Repositories

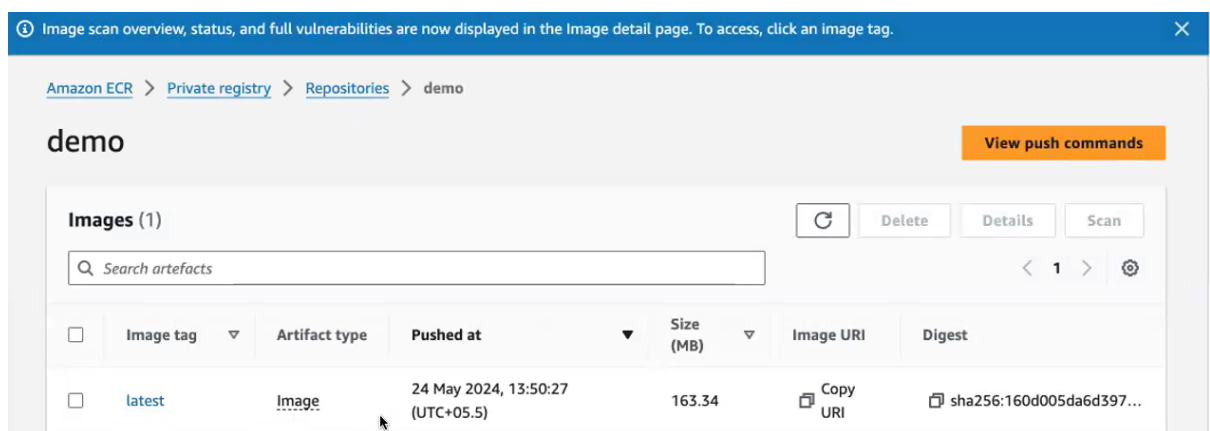
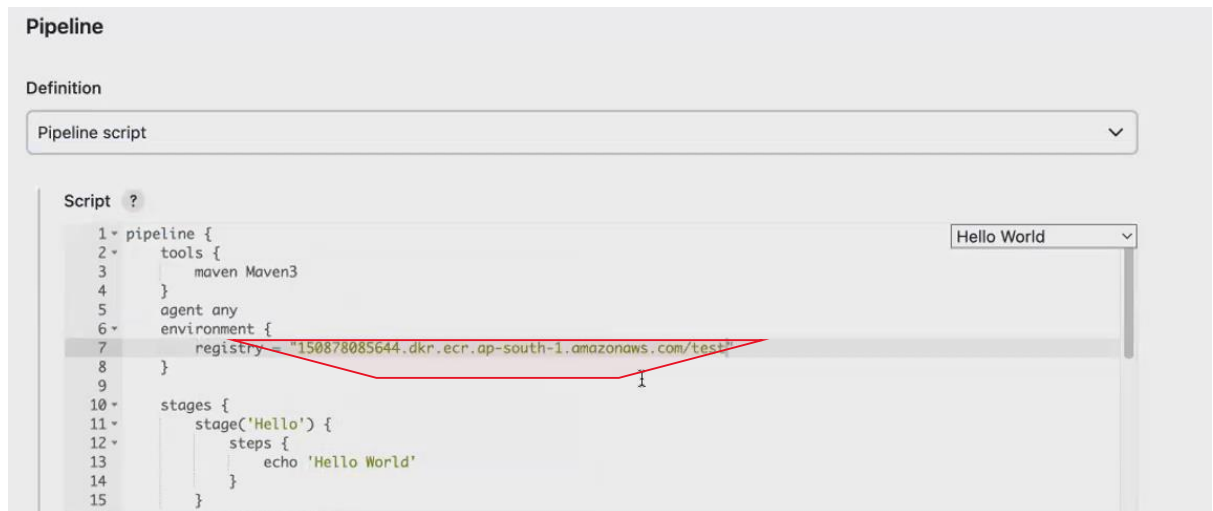
## Private repositories

 **Upgraded basic scanning**  
Switch to the improved version of basic scanning, powered by AWS native scanning technology. [Find out more](#)  switch 

**Repositories (2)**  View push commands Delete Actions Create repository

<input type="checkbox"/>	Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
<input type="checkbox"/>	demo	150878085644.dkr.ecr.ap-south-1.amazonaws.com/demo	24 May 2024, 13:47:33 (UTC+05.5)	Disabled	Manual	AES-256
<input type="checkbox"/>	test	150878085644.dkr.ecr.ap-south-1.amazonaws.com/test <small> Copy URI for test</small>	24 July 2024, 17:23:12 (UTC+05.5)	Disabled	Manual	AES-256

Copy the link and give it in Jenkins file.



7. So, you create a registry and you pass the URL or Ecr registry URL. Here. Okay, so to this registry we will be pushing our docker image.

### 1<sup>st</sup> stage [checkout]:

8. So what is the 1st stage? 1<sup>st</sup> is your checkout stage. 1<sup>st</sup> you have where to check out.
9. Your Jenkins has to fetch the code from the Github Repository. Okay, so here you have to give the checkout link.

How-to get this path .steps given below:

Script ?

```
11- stage('Checkout Github') {
12-   steps {
13-     checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://git
14-   }
15- }
16- stage('Hello') {
17-   steps {
18-     echo 'Hello World'
19-   }
20- }
21- stage('Maven Build') {
22-   steps {
23-     echo 'Hello World'
24-   }
25- }
```

Hello World

Pipeline script

Script ?

```
2- tools {
3-   maven Maven3
4- }
5- agent any
6- environment {
7-   registry = "150878085644.dkr.ecr.ap-south-1.amazonaws.com/test"
8- }
9-
10- stages {
11-   stage('Checkout Github') {
12-     steps {
13-       echo 'Hello World'
14-     }
15-   }
16-   stage('Hello') {
17-     steps {
18-       echo 'Hello World'
19-     }
20-   }
21- }
```

Hello World

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

- ✓ archiveArtifacts: Archive the artifacts
- bat: Windows Batch Script
- build: Build a job
- catchError: Catch error and set build result to failure
- checkout: Check out from version control

## Steps

### Sample Step

checkout: Check out from version control

checkout ?

SCM

Git

Repositories ?

Repository URL ?

https://github.com/yasminjeelani/EKS-CICD.git

! Please enter Git repository.

Credentials ?

- none -

+ Add

\*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

☒ Include in polling? ?

☒ Include in changelog? ?

Generate Pipeline Script

```
checkout scmGit(branches: [[name: '**/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/yasminjeelani/EKS-CICD.git']])
```



Copy the path in 1<sup>st</sup> stage

## 2<sup>nd</sup> stage [build stage]:

- So, we will be building it using maven. So you have to perform a maven build.
- To perform Maven, build what you need. **You have to integrate your Jenkins with Maven.** Since you're already installed it. Okay, you have already installed Maven. I will just enable the main plugin and this Gui.

Script ?

```
11 - stage('Checkout Github') {
12 -     steps {
13 -         checkout scmGit(branches: [[name: '*/main']], extensions: [])
14 -     }
15 - }
16
17 - stage('Maven Build') {
18 -     steps {
19 -         sh 'mvn clean package'
20 -     }
21 - }
```

I've already installed. If you haven't installed Maven, you can give installed automatically, it will install it in your Jenkins instance.

## Give maven plugin in Jenkins:

Open in new tab. set up the Maven tool in our Jenkins

The screenshot shows the Jenkins 'Manage Jenkins' interface. On the left sidebar, 'Manage Jenkins' is selected. The main content area has a yellow warning banner about security. Below it, the 'System Configuration' section is visible, with the 'Tools' option highlighted by a red box. The 'Tools' option includes a wrench icon and the text 'Tools' and 'Configure tools, their locations and automatic installers.'.



Add Git ▾

### Gradle installations

Add Gradle

### Ant installations

Add Ant

### Maven installations

Add Maven

Save

Apply

## Maven installations

Add Maven

☰ Maven

Name

MAVEN\_HOME

  
  
☐ Install automatically ?

Add Maven

Save

Apply

**3<sup>rd</sup> stage [build docker]:**

**then we will be building the application using docker.**

**Instead to give docker image name give ecr registry**

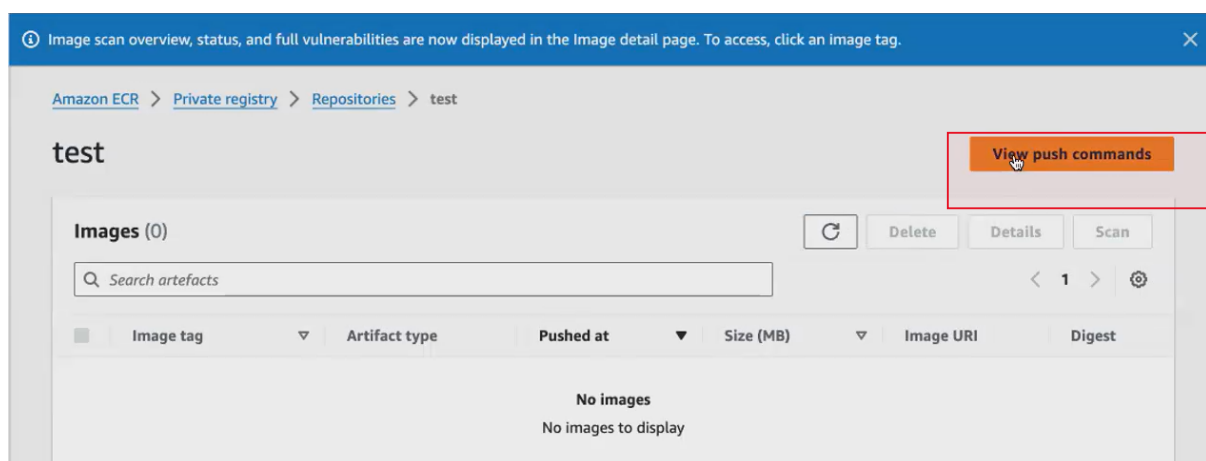
```
stage('Maven Build') {
    steps {
        sh 'mvn clean package'
    }
}
stage('Docker Build') {
    steps {
        sh 'docker build -t registry'
    }
}
```

## 4<sup>th</sup> stage [ push the docker image to ecr registry]:

```
26     }
27     stage('Push Docker Image to ECR') {
28     steps {
29         sh 'docker build -t registry'
30     }
31 }
32 }
```

- so we will be pushing the docker image to the Ecr registry.
- So to before pushing your docker image to the docker hub registry. What do you do?
- You have to do a docker login
- after doing the docker login only we will be able to push the docker image to the docker Hub register.
- This Ecr belongs to Aws. It is. It doesn't belong to your docker hub. I
- it belongs to aws. So, using the Aws cli command, you will be logging into the Ecr registry after successful logging to ecr registry.
- you will be pushing your application; you will be pushing your docker image to the Ecr registry.

## Docker login:



Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting started with Amazon ECR](#).

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry authentication](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 150878085644.dkr.ecr.ap-south-1.amazonaws.com
```

Copy and paste the login in Jenkins file

Docker build already given.

Docker push command copy and paste it in Jenkins file

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 150878085644.dkr.ecr.ap-south-1.amazonaws.com/test:latest
```

```
stage('Docker Build') {  
  steps {  
    sh 'docker build -t registry'  
  }  
}  
stage('Push Docker Image to ECR') {  
  steps {  
    sh 'aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 150878085644.dkr.ecr.ap-south-1.amazonaws.com'  
    sh 'docker push 150878085644.dkr.ecr.ap-south-1.amazonaws.com/test:latest'  
  }  
}
```

### 5<sup>th</sup> stage [deploying the application to the Kubernetes cluster]:

For deployment we have deployment file.in github Yasmin have deployment file and service file in single file. **Service file used to access the application via browser.**



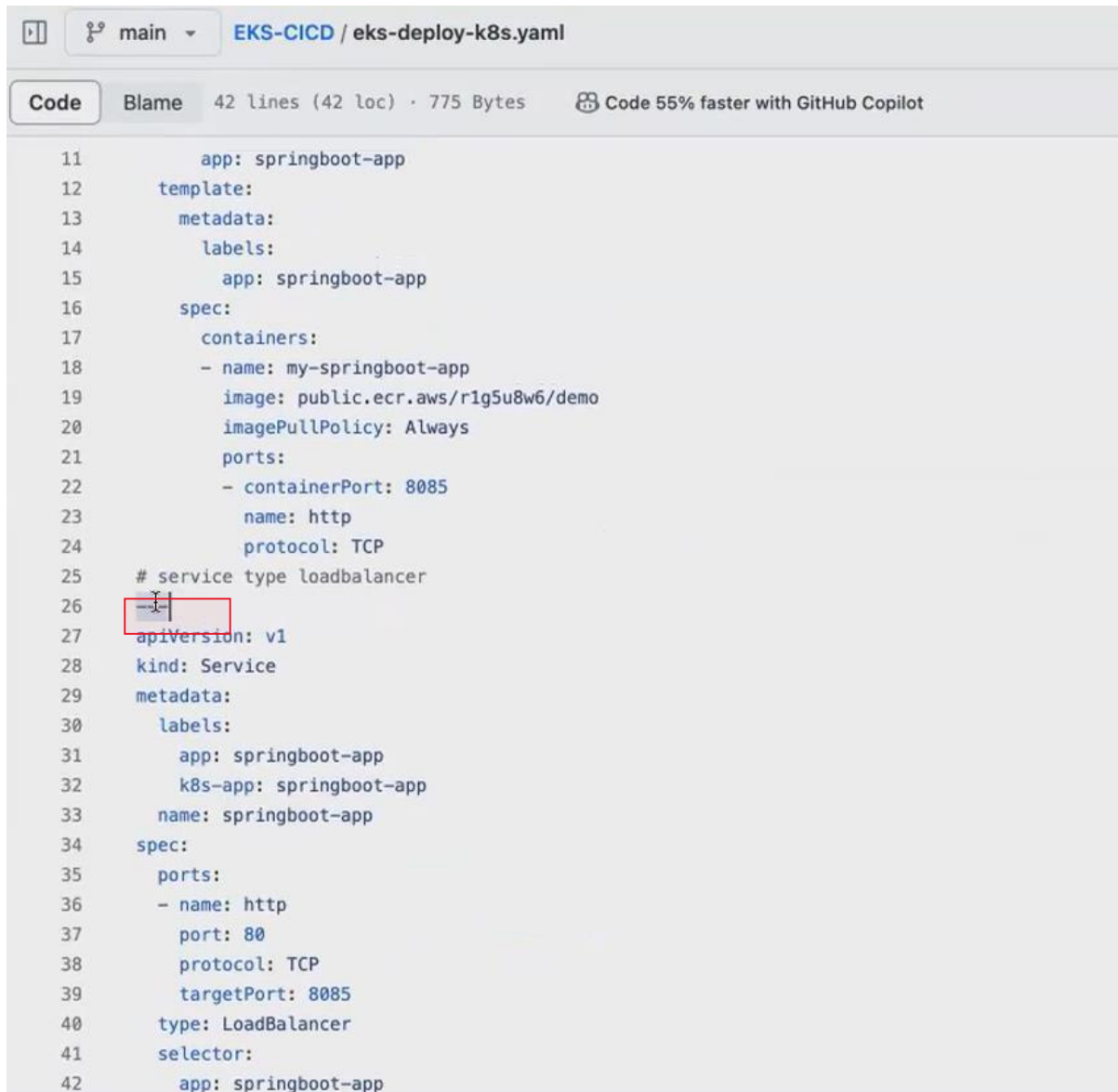
# EKS-CICD

Publicmain1 Branch0 TagsGo to file

**yasminjeelani** Update eks-deploy-k8s.yaml

manifestsupdatedsrc/mainupdatedtarget/classesupdatedDockerfileupdatedJenkinsfile-K8SUpdate Jenkinsfile-K8Seks-deploy-from-ecr.yamlupdated[eks-deploy-k8s.yaml](#)Update eks-deploy-k8s.yamlpom.xmlupdated

Both files separated by ---



```
11     app: springboot-app
12   template:
13     metadata:
14       labels:
15         app: springboot-app
16     spec:
17       containers:
18         - name: my-springboot-app
19           image: public.ecr.aws/r1g5u8w6/demo
20           imagePullPolicy: Always
21           ports:
22             - containerPort: 8085
23               name: http
24               protocol: TCP
25   # service type loadbalancer
26   ---
27   apiVersion: v1
28   kind: Service
29   metadata:
30     labels:
31       app: springboot-app
32       k8s-app: springboot-app
33     name: springboot-app
34   spec:
35     ports:
36       - name: http
37         port: 80
38         protocol: TCP
39         targetPort: 8085
40     type: LoadBalancer
41     selector:
42       app: springboot-app
```

- Docker image and you have pushed it to the Ecr registry. Assume that inside this registry you have your docker image.

Amazon ECR > Private registry > Repositories

## Private repositories

**Upgraded basic scanning**  
Switch to the improved version of basic scanning, powered by AWS native scanning technology. [Find out more](#)

switch X

Repositories (2) [View push commands](#) [Delete](#) [Actions](#) [Create repository](#)

Filter status

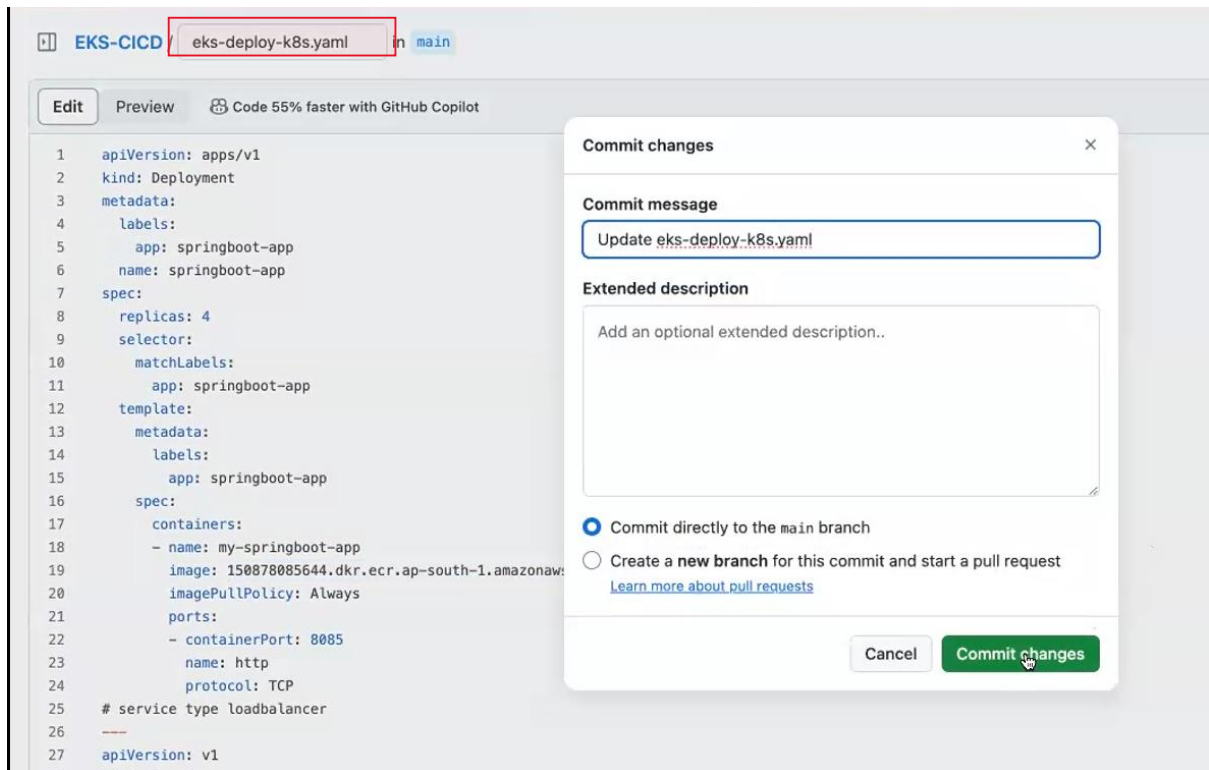
Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
demo	150878085644.dkr.ecr.ap-south-1.amazonaws.com/demo	24 May 2024, 13:47:33 (UTC+05.5)	Disabled	Manual	AES-256
test	150878085644.dkr.ecr.ap-south-1.amazonaws.com/test	24 July 2024, 17:23:12 (UTC+05.5)	Disabled	Manual	AES-256

- copy this registry name. Go to your deployment file and paste it here.

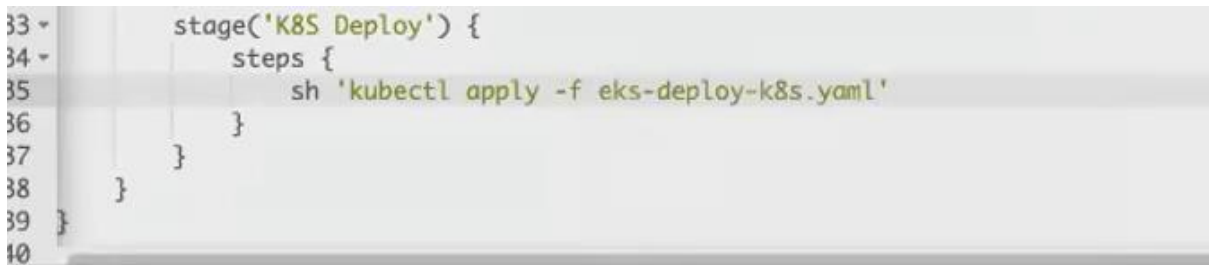
Edit Preview Code 55% faster with GitHub Copilot

```

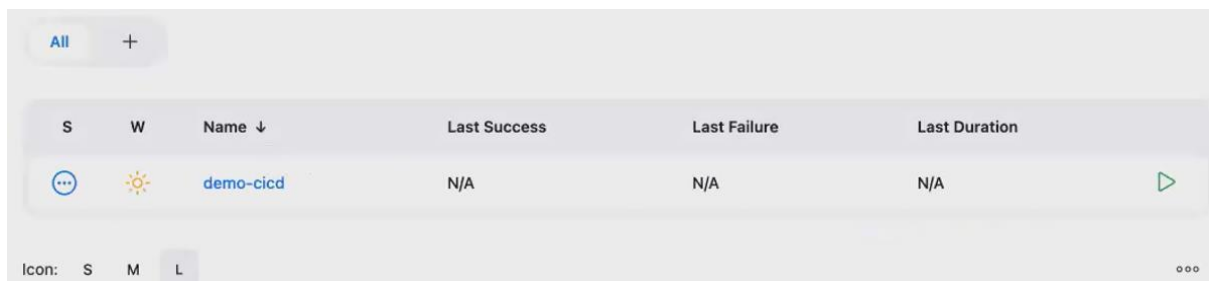
4 labels:
5   app: springboot-app
6   name: springboot-app
7 spec:
8   replicas: 4
9   selector:
10    matchLabels:
11     app: springboot-app
12   template:
13     metadata:
14       labels:
15        app: springboot-app
16   spec:
17     containers:
18     - name: my-springboot-app
19       image: 150878085644.dkr.ecr.ap-south-1.amazonaws.com/test
20       imagePullPolicy: Always
  
```



**Then give the deployment command with deployment file name [eks-deploy-k8s.yaml]**



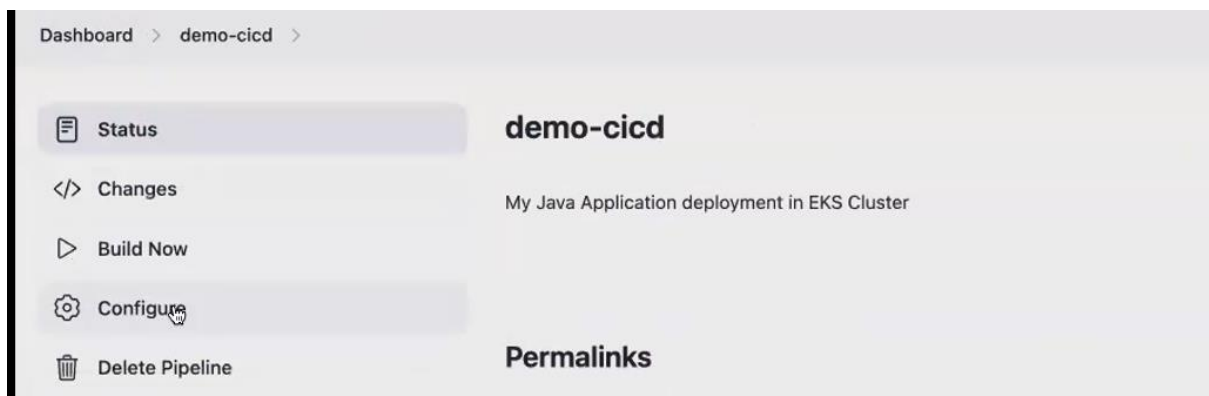
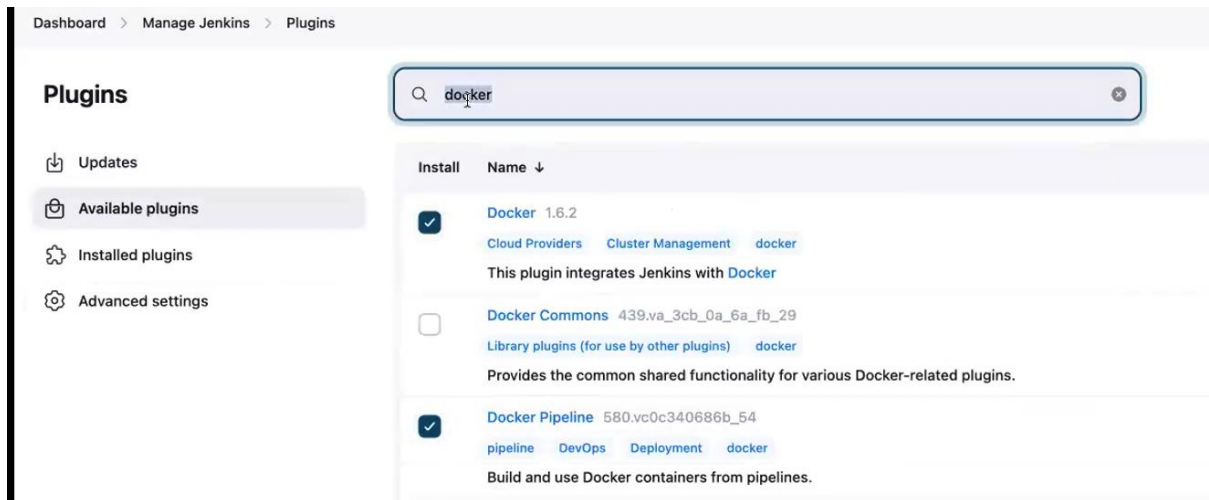
**Let save this file**



**Install plugins**

**Goto dashboard → manage Jenkins → plugins → available Jenkins → search docker and Kubernetes cli → select this 2 → click install**





We write the Jenkins file in github only.in Jenkins page it's not recommended .so select pipeline script from scm[source code management]

## Pipeline

### Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

`https://github.com/yasminjeelani/EKS-CICD.git`

Please enter Git repository.

Branch Specifier (blank for 'any') ?

\*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

Script Path ?

Jenkinsfile-K8S


☒ Lightweight checkout ?


[Pipeline Syntax](#)

Save

Apply

 Status

 Changes Build scheduled

 Build Now

 Configure

**demo-cicd**

My Java Application deployment in EKS Cluster

```

+ kubectl apply -f eks-deploy-k8s.yaml
deployment.apps/springboot-app created
service/springboot-app created
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

```

ubuntu@ip-172-31-42-111:~$ sudo su - jenkins
jenkins@ip-172-31-42-111:~$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/springboot-app-789989b54-6vc44  1/1     Running   0           4m1s
pod/springboot-app-789989b54-8pzh7  1/1     Running   0           4m1s
pod/springboot-app-789989b54-nq9hs  1/1     Running   0           4m1s
pod/springboot-app-789989b54-r7xk8  1/1     Running   0           4m1s

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP
service/kubernetes                  ClusterIP     10.100.0.1    <none>
service/springboot-app              LoadBalancer  10.100.63.90  af2815daddb7c4c4ab3a9d46664cbb8d

NAME                                READY    UP-TO-DATE   AVAILABLE   AGE
deployment.apps/springboot-app      4/4      4             4           4m1s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/springboot-app-789989b54  4          4          4       4m1s
jenkins@ip-172-31-42-111:~$

```

**Copy the load balancer ip and paste it in browser**

## My Awesome Spring Boot App Running on EKS Cluster

First Name:

Last Name:

Successfull!

• yasmin L

**In Jenkins file:**

**Instead of giving docker build -t my docker image name I have given as docker.dot build.**

**So docker.build is nothing but a docker plugin.**

Okay, you install docker pipeline. if you have installed docker pipeline, you can simply give docker plugins will be enabled in your Jenkins. You can simply use this function. Docker.build what will happen? It will by default it will execute your docker build command.



```
main EKS-CICD / Jenkinsfile-K8S
Blame 53 lines (45 loc) · 1.32 KB Code 55% faster with GitHub Copilot
}
agent any
environment {
    registry = "150878085644.dkr.ecr.ap-south-1.amazonaws.com/test"
}

stages {
    stage('Cloning Git') {
        steps {
            checkout([$class: 'GitSCM', branches: [[name: '*/main']], doGenerateSubmoduleConfigurations: false, e
        }
    }
    stage('Build') {
        steps {
            sh 'mvn clean install'
        }
    }
    // Building Docker images
    stage('Building image') {
        steps{
            script {
                dockerImage = docker.build registry
            }
        }
    }
}
```

- **Mvn claen - mvn clean package command. Okay, what the mvn clean will do. It will remove the target folder. It will remove all the jar file. That is that it got created previously, and it will freshly create the artifact.**
- **So now, this artifact will be inside of Ec2.**

```
Started by user admin
Obtained Jenkinsfile-K8S from git https://github.com/yasminjeelani/EKS-CICD.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/demo-cicd
```

## Everything done in this work folder.

```
buntu@ip-172-31-42-111:~$ cd /var/lib/jenkins/workspace/demo-cicd
buntu@ip-172-31-42-111:/var/lib/jenkins/workspace/demo-cicd$ ls
Dockerfile Jenkinsfile-K8S eks-deploy-from-ecr.yaml eks-deploy-k8s.yaml manifests pom.xml src target
buntu@ip-172-31-42-111:/var/lib/jenkins/workspace/demo-cicd$ cd target/
buntu@ip-172-31-42-111:/var/lib/jenkins/workspace/demo-cicd/target$ ls
classes generated-sources maven-archiver maven-status springbootApp.jar springbootApp.jar.original
buntu@ip-172-31-42-111:/var/lib/jenkins/workspace/demo-cicd/target$
```



```
main EKS-CICD / Jenkinsfile-K8S
Blame 53 lines (45 loc) · 1.32 KB Code 55% faster with GitHub Copilot
}
agent any
environment {
    registry = "150878085644.dkr.ecr.ap-south-1.amazonaws.com/test"
}

stages {
    stage('Cloning Git') {
        steps {
            checkout([$class: 'GitSCM', branches: [[name: '*/main']], doGenerateSubmoduleConfigurations: false, e
        }
    }
    stage('Build') {
        steps {
            sh 'mvn clean install'
        }
    }
    // Building Docker images
    stage('Building image') {
        steps {
            script {
                dockerImage = docker.build registry
            }
        }
    }
}
```

## Repo Link -

<https://github.com/yasminjeelani/EKS-CICD.git>

repo link for full devops course—

<https://github.com/zen-class/zen-class-devops-documentation>

**documentation for this application by Yasmin--**

**<https://docs.google.com/document/d/1Is4h94KVFliaNxSuBZX9Spui98CYiFOIawjLfYa3wBU/edit>**

