

Name -Parmeshwar*

Assignment (GITHUB, JENKINS, Gradle, DOCKER & KUBERNETES)

Problem Statement: Automated Deployment of Web application

Q1) Design and develop tasks required to build CICD pipeline using ALL learned technologies to deploy WEB application on cloud platform as a set of microservices on containerized platform such as DOCKER. Deploy a web application on Docker container using GRADLE as build tool to install, test, package application. Push the code to Docker HUB and make it publicly available on internet. Pull this code from Docker Hub & Deploy it on KUBERNETES Cluster & create a pod with 3 Replicas. Access the Webpage from Kubernetes Worker Node using Node port service. Provision Infrastructure on AWS.

Step 1) created 3 instances

i) Kubernets master

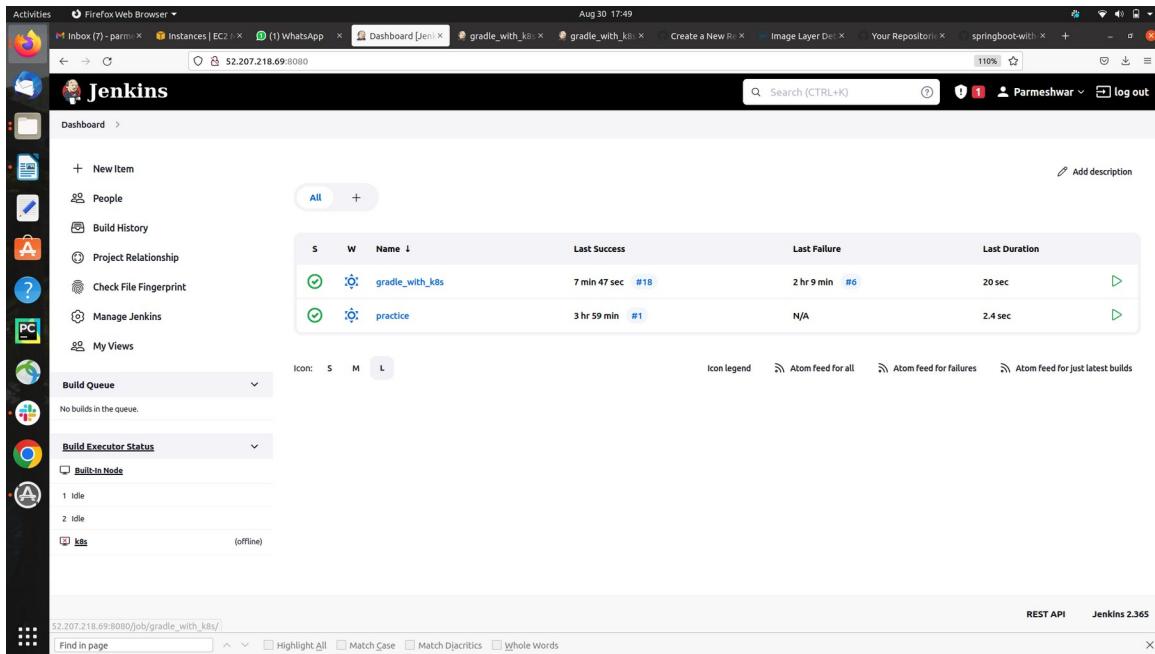
i) Kubernets WorkNode

i) Jenkins master

The screenshot shows the AWS Management Console interface for the EC2 service. The left sidebar has a tree view with 'Instances' selected, showing sub-options like 'Instances', 'AMIs', and 'Elastic Block Store'. The main content area is titled 'Instances (3) Info' and displays a table of three EC2 instances:

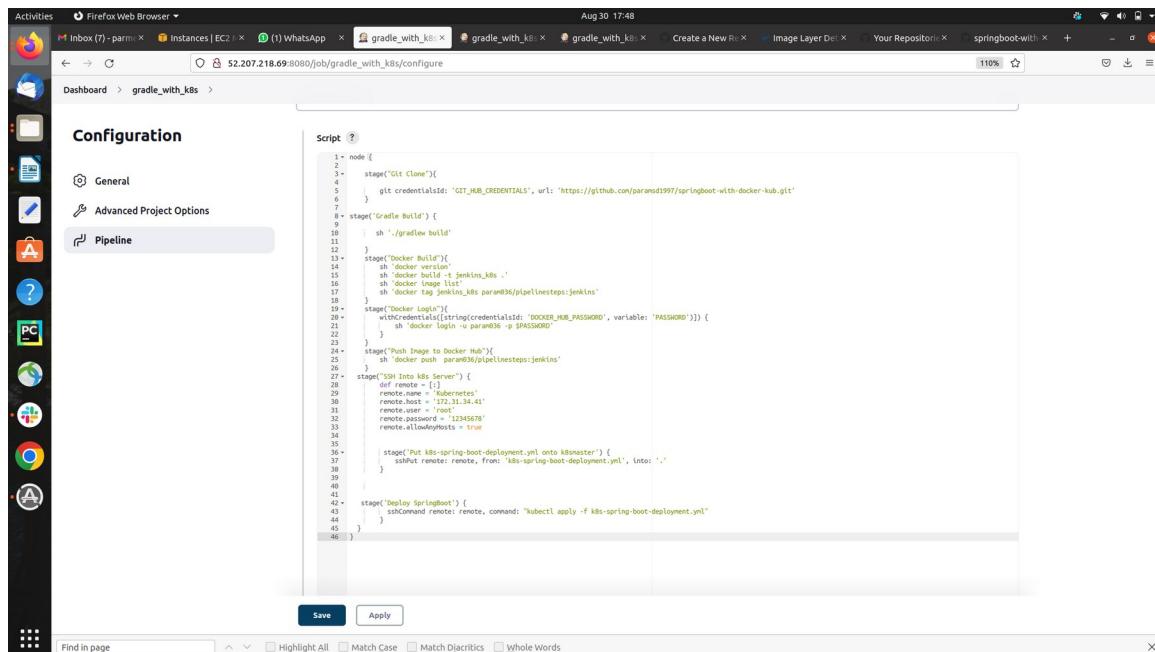
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
k8 Master	i-0fc17dcf7e3f562e3	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-54-224-24-31.o
jenkinMaster	i-07df570df259b3c6c	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-52-207-218-69.
WN	i-0e055e9d2b9b0aa4c	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-34-207-204-81.

step 2) created the job



The screenshot shows the Jenkins dashboard at 52.207.218.69:8080. The left sidebar contains links for New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, and My Views. The main area displays a table of jobs with columns for Status (S), Workstation (W), Name, Last Success, Last Failure, and Last Duration. Two jobs are listed: 'gradle_with_k8s' and 'practice'. 'gradle_with_k8s' has a green status icon, a last success of 7 min 47 sec, a last failure of #18 (2 hr 9 min ago), and a last duration of 20 sec. 'practice' has a green status icon, a last success of 3 hr 59 min, a last failure of N/A, and a last duration of 2.4 sec. Below the table, there are sections for Build Queue (empty), Build Executor Status (1 idle, 2 idle, k8s offline), and Built-in Node (1 idle). At the bottom, there is a search bar and links for REST API and Jenkins 2.365.

step 3) pipeline script is written for execution

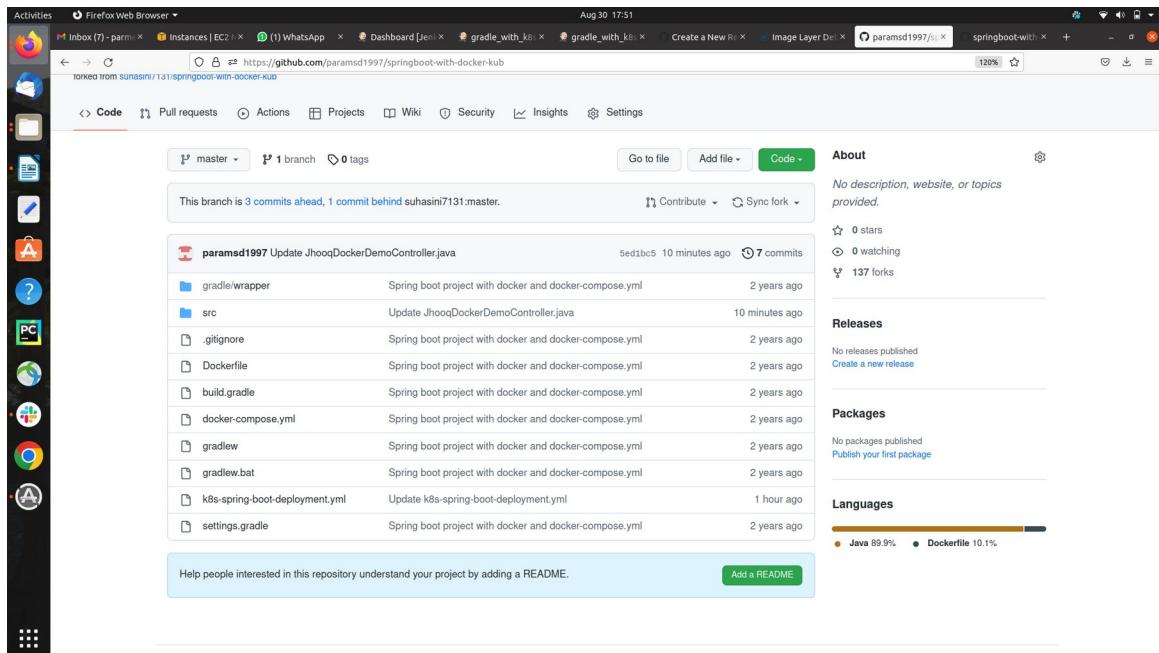


The screenshot shows the Jenkins Pipeline configuration for the 'gradle_with_k8s' job at 52.207.218.69:8080/job/gradle_with_k8s/configure. The left sidebar shows General and Advanced Project Options. The right panel is titled 'Script' and contains the Jenkinsfile code:

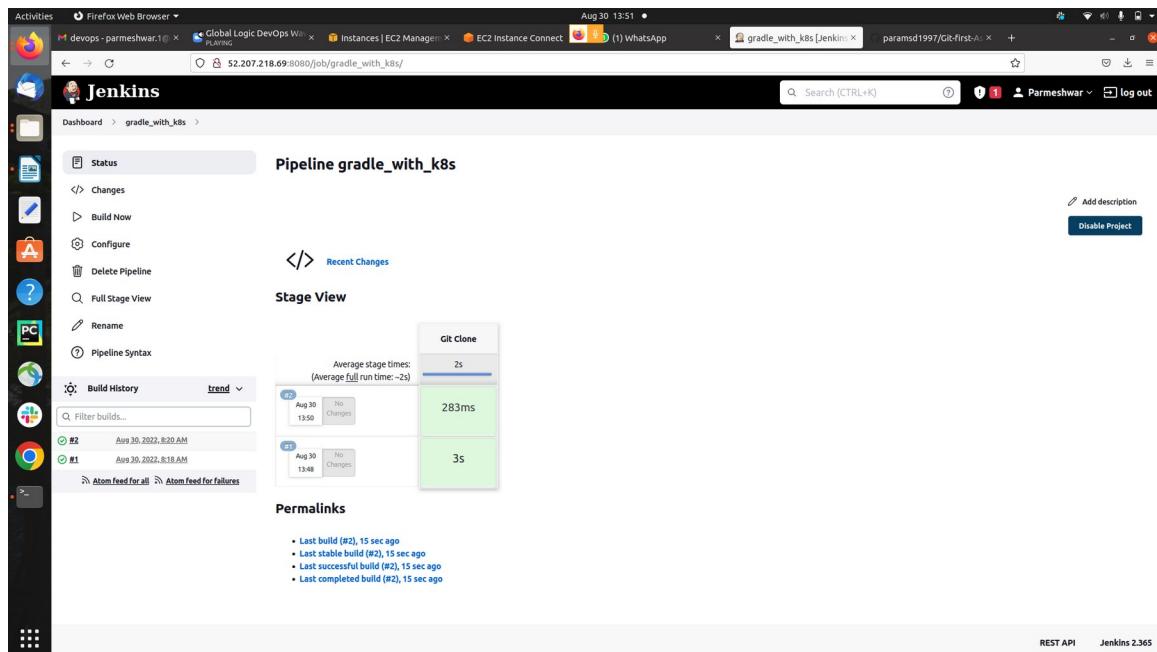
```
1 - node {
2     stage('Git Clone'){
3         git credentialsId: 'GIT_K8S_CREDENTIALS', url: 'https://github.com/parnidan97/springboot-with-docker-kub.git'
4     }
5 }
6 }
7 }
8 stage('Gradle Build') {
9     sh './gradlew build'
10 }
11 }
12 }
13 stage('Docker Build'){
14     sh 'docker version'
15     sh 'docker login -u jenkins_k8s -p Jenkins_K8s'
16     sh 'docker image list'
17     sh 'docker tag jenkins_k8s parnidan97/pipelinetests:jenkins'
18 }
19 stage('Docker Logon'){
20     withCredentials([
21         string(credentialsId: 'DOCKER_K8S_PASSWORD', variable: 'PASSWORD')
22     ]) {
23         sh "docker login -u $PARAM936 -p $PASSWORD"
24     }
25 }
26 stage('Push Image to Docker Hub'){
27     sh 'docker push parnidan97/pipelinetests:jenkins'
28 }
29 stage('Get Into K8s Server') {
30     def remote = []
31     remote_name = 'Kubernetes'
32     remote_ip = '172.24.34.41'
33     remote_user = 'root'
34     remote_password = '12345678'
35     remote_allwaysfresh = true
36 }
37 stage('Put k8s-spring-boot-deployment.yaml onto k8smaster') {
38     sshPut remote, from: 'k8s-spring-boot-deployment.yaml', into: '.'
39 }
40 }
41 stage('Deploy Springboot') {
42     sshCommand remote, command: "kubectl apply -f k8s-spring-boot-deployment.yaml"
43 }
44 }
45 }
46 }
```

At the bottom, there are 'Save' and 'Apply' buttons, and a search bar.

step 4) Git repo Code



step 5) git clone successfully



The screenshot shows the Jenkins Pipeline configuration page. The pipeline is defined by a single stage:

```
node {
    stage("Git Clone"){
        git credentialsId: 'GIT_HUB_CREDENTIALS', url: 'https://github.com/paramsd1997/springboot-with-docker-kub.git'
    }
}
```

Below the script, there is a checkbox labeled "Use Groovy Sandbox". At the bottom, there are "Save" and "Apply" buttons.

The screenshot shows the Jenkins console output for a build. The output shows the execution of the pipeline script:

```
Started by user Parmeshwar
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/gradle_with_k8s
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Pipeline] git
The recommended git tool is: NONE
using credential GIT HUB CREDENTIALS
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/gradle_with_k8s/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/paramsd1997/springboot-with-docker-kub.git # timeout=10
Fetching upstream changes from https://github.com/paramsd1997/springboot-with-docker-kub.git
> git --version # timeout=10
> git --version # 'git' version 2.37.1'
using GIT_ASKPASS to set credentials first deployment with the k8s
> git fetch --tags --progress -- https://github.com/paramsd1997/springboot-with-docker-kub.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision e90ee2550bb2c668:3002d6af9762f196a498bb2 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f e90ee2550bb2c668:3002d6af9762f196a498bb2 # timeout=10
> git branch -a -v --no-abbrev-v # timeout=10
> git branch -D master # timeout=10
> git checkout -b master e90ee2550bb2c668:3002d6af9762f196a498bb2 # timeout=10
Commit message: "Update k8s-spring-boot-deployment.yaml"
> git rev-list --no-walk e90ee2550bb2c668:3002d6af9762f196a498bb2 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

step 5) Gradle Build Successfully

The screenshot shows the Jenkins interface for the pipeline 'gradle_with_k8s'. On the left, there's a sidebar with various Jenkins-related icons. The main area displays the 'Stage View' for the pipeline. It shows three stages: 'Git Clone' (299ms), 'Gradle Build' (52s), and 'Docker Build' (283ms). Below the stages, a table lists three build history entries: #3 (Aug 30, 2022, 9:51 AM), #2 (Aug 30, 2022, 8:20 AM), and #1 (Aug 30, 2022, 8:18 AM). Each entry indicates 'No Changes'. At the bottom, there's a 'Permalinks' section with links to the last four builds.

Successfully builted

The screenshot shows the Jenkins configuration screen for the pipeline 'gradle_with_k8s'. The left sidebar has tabs for 'General', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is selected. The main area is titled 'Definition' and shows a 'Pipeline script' input field containing Groovy code. The code defines a pipeline with three stages: 'Git Clone', 'Gradle Build', and 'Docker Build'. The 'Gradle Build' stage includes a command to run 'gradlew build'. The 'Docker Build' stage uses Docker to build an image and tag it as 'jhoqq-docker-demo suha7131/dockerkub:jhoqq-docker-demo'. At the bottom, there's a checkbox for 'Use Groovy Sandbox' which is checked, and buttons for 'Save' and 'Apply'.

```

[Pipeline] stage
[Pipeline] {
[Pipeline] sh
+ ./gradlew build
Downloading https://services.gradle.org/distributions/gradle-6.4.1-bin.zip
.....10%.....20%.....30%.....40%.....50%.....60%.....70%.....80%.....90%.....100%
Welcome to Gradle 6.4.1!
Here are the highlights of this release:
- Support for building, testing and running Java Modules
- Precompiled script plugins for Groovy DSL
- Single dependency lock file per project
For more details see https://docs.gradle.org/6.4.1/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava
> Task :processResources
> Task :classes
> Task :bootJar
> Task :jar SKIPPED
> Task :assemble
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

> Task :test
2022-08-30 09:51:52.921 INFO 28344 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolExecutor : Shutting down ExecutorService 'applicationTaskExecutor'

> Task :check
> Task :build

BUILD SUCCESSFUL in 51s
5 actionable tasks: 5 executed
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

step 6) Login to Docker hub and push to Docker hub

Configuration

Script :

```

1 > node {
2   >   stage('Git Clone'){
3     >     git credentialsId: 'GIT_HUB_CREDENTIALS', url: 'https://github.com/paramsd1997/springboot-with-docker-hub.git'
4   }
5   >   stage('Gradle Build'){
6     >     sh './gradlew build'
7   }
8   >   stage('Docker Build'){
9     >     sh 'docker build -t jenkins_k8s .'
10    >   }
11    >   stage('Docker Login'){
12      >     withDockerRegistry(credentialsId: 'DOCKER_HUB_PASSWORD', variable: 'PASSWORD') {
13        >       sh 'docker login -u $PARAMSD1997 -p $PASSWORD'
14      }
15    }
16    >   stage('Push Image to Docker Hub'){
17      >     sh 'docker push jenkins_k8s:latest'
18    }
19  }
20
21  > stage("Put file into k8s Server"){
22    >   def remote = []
23    >   remote_name = "jenkins"
24    >   remote_ip = "172.31.34.41"
25    >   remote_user = "root"
26    >   remote_port = "22445678"
27    >   remote_allowAnyHosts = true
28  }
29
30  > stage("Put k8s-spring-boot-deployment.yaml onto k8smaster"){
31    >   sh"""
32      kubectl replace --force deployment k8smaster --from=yaml --to= yaml
33    """
34  }
35
36  > stage("Put k8s-spring-boot-deployment.yaml onto k8smaster"){
37    >   sh"""
38      kubectl replace --force deployment k8smaster --from=yaml --to= yaml
39    """
40  }

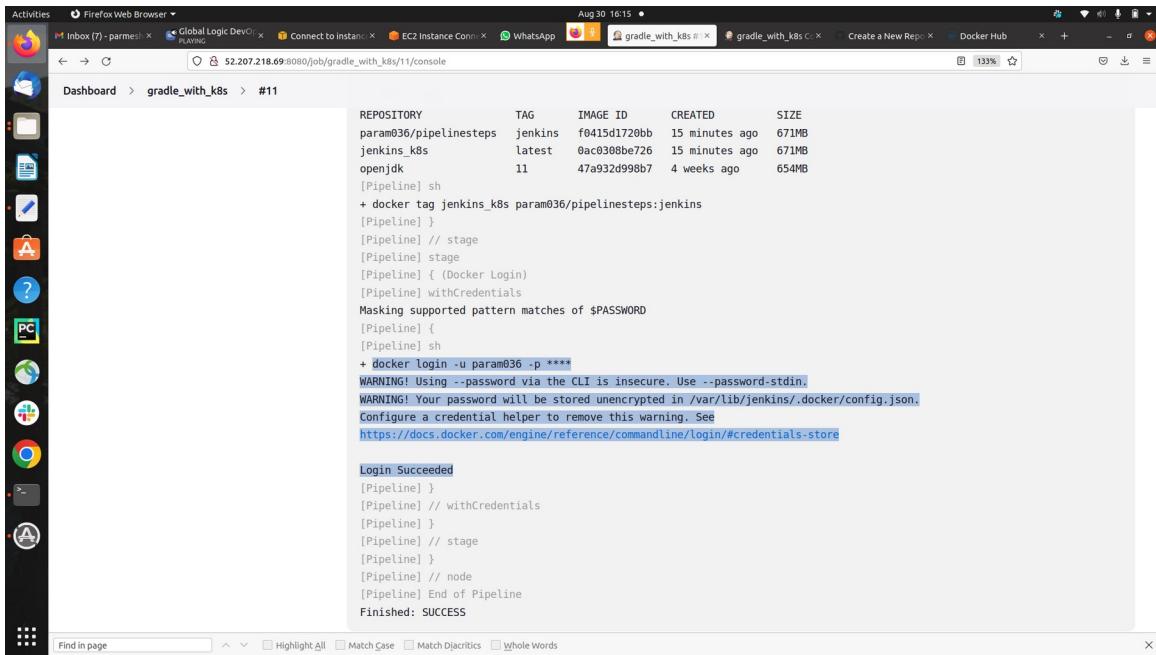
```

Pipeline Syntax

Use Groovy Sandbox [?](#)

Save **Apply**

login Succeed



```
Aug 30 16:15 • 52.207.216.69:8080/job/gradle_with_k8s/11/console
```

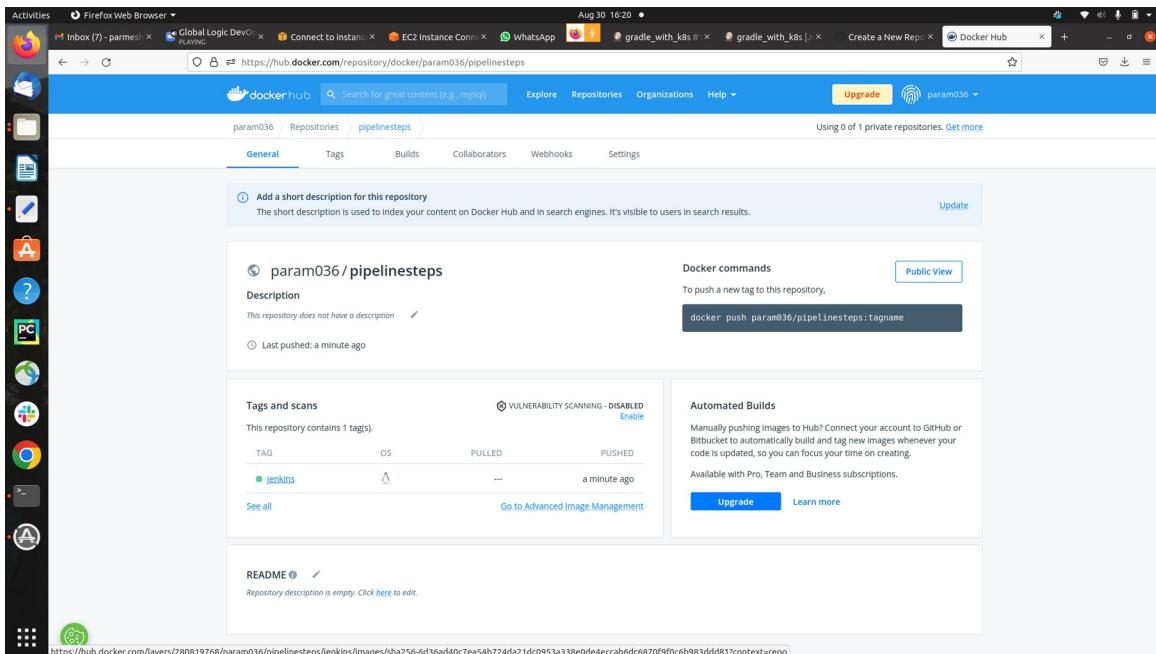
```
Dashboard > gradle_with_k8s > #11
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
param036/pipelinesteps	jenkins	f0415d1720bb	15 minutes ago	671MB
jenkins_k8s	latest	0ac0308be726	15 minutes ago	671MB
openjdk	11	47a932d998b7	4 weeks ago	654MB

```
[Pipeline] sh
+ docker tag jenkins_k8s param036/pipelinesteps:jenkins
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Docker Login)
[Pipeline] withCredentials
Masking supported pattern matches of $PASSWORD
[Pipeline] {
[Pipeline] sh
+ docker login -u param036 -p ****
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

[Pipeline] Login Succeeded
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Image Successfully Pushed to docker hub



Aug 30 16:20 • https://hub.docker.com/repository/docker/param036/pipelinesteps

param036 Repositories pipelinesteps

General Tags Builds Collaborators Webhooks Settings

Add a short description for this repository
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

param036 / pipelinesteps

Description
This repository does not have a description

Last pushed: a minute ago

Docker commands
To push a new tag to this repository.

docker push param036/pipelinesteps:tagname

Tags and scans
This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
jenkins		---	a minute ago

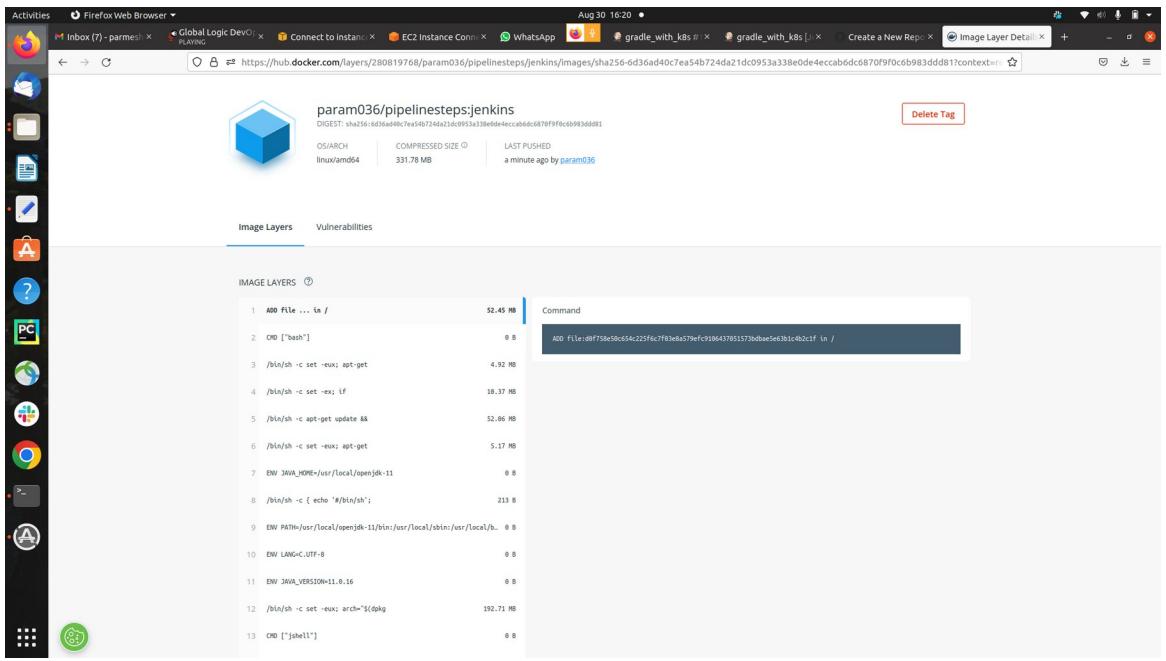
See all Go to Advanced Image Management

VULNERABILITY SCANNING - DISABLED
Enable

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

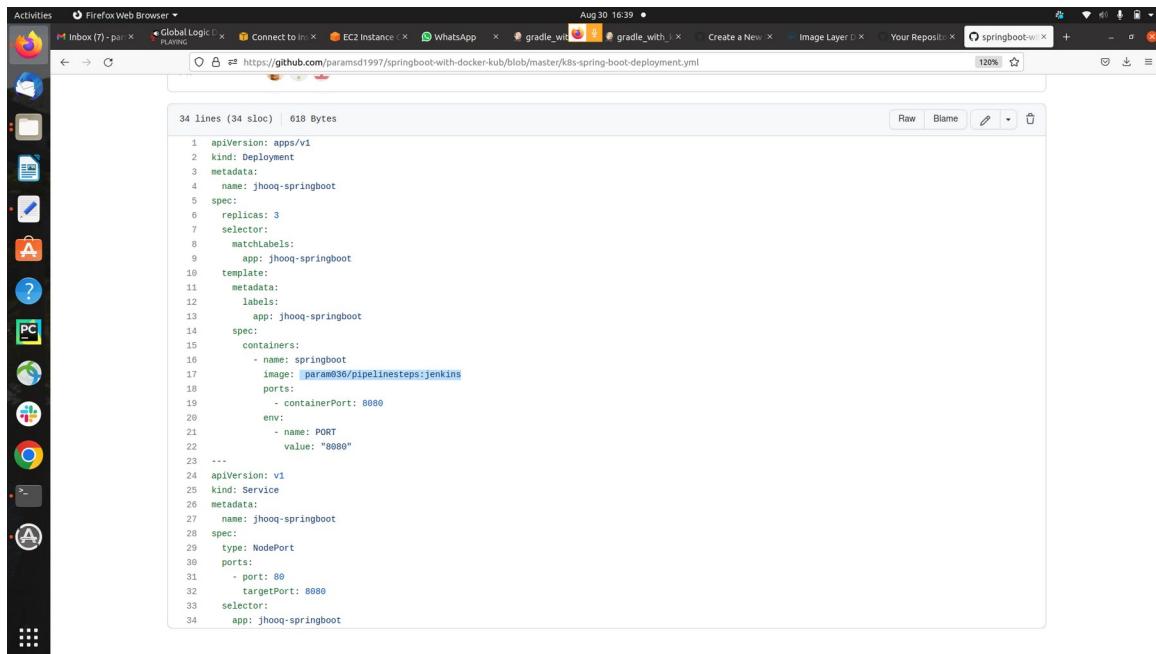
Available with Pro, Team and Business subscriptions.

README
Repository description is empty. Click here to edit.



The screenshot shows a Jenkins Pipeline interface. On the left, there's a sidebar with various configuration options like "Configure", "Delete Pipeline", "Full Stage View", "Rename", and "Pipeline Syntax". Below that is a "Build History" section with a dropdown for "trend" and a search bar. The main area is titled "Stage View" and displays a table of build stages. The columns are: Git Clone, Gradle Build, Docker Build, Docker Login, and Push Image to Docker Hub. Each row represents a build (e.g., #13, #12, #11, #10, #9, #8, #7, #6, #5, #4, #3, #2) with its timestamp (Aug 30, 16:18 or 15:54). The "Push Image to Docker Hub" column for build #12 is highlighted with a red background and the text "293ms failed".

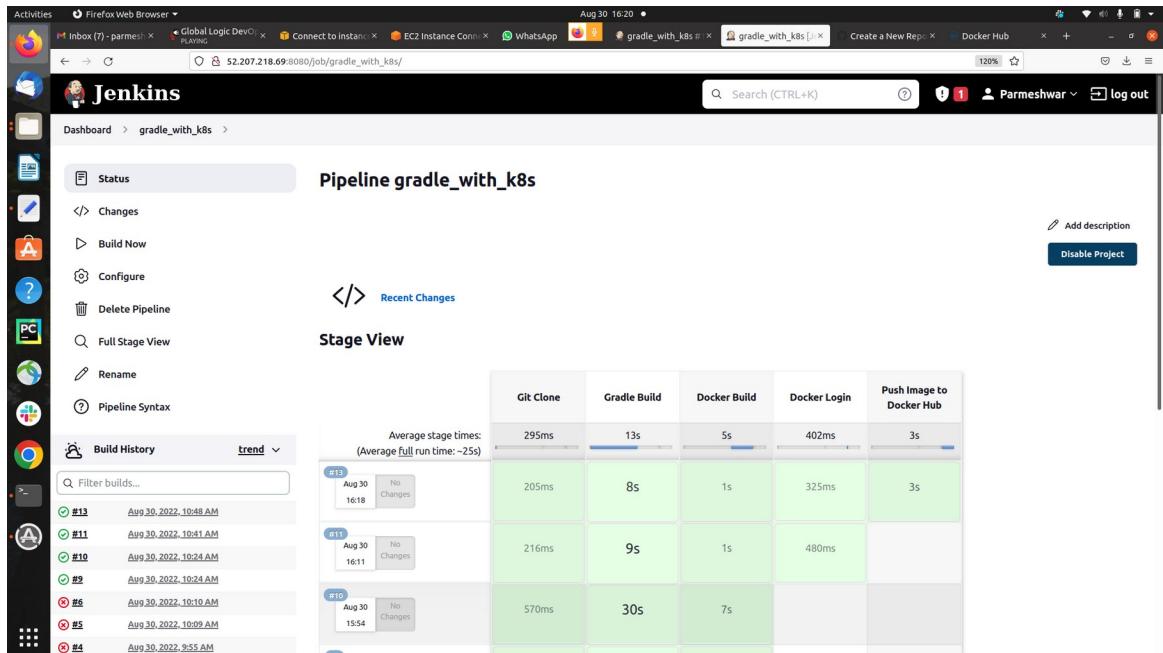
step 8) Yml file for the creating the deployment with 3 replicas



The screenshot shows a Firefox browser window with multiple tabs open. The active tab displays a GitHub page containing a YAML configuration file named `springboot-with-docker-kub/blob/master/k8s-spring-boot-deployment.yaml`. The code defines a Deployment and a Service for a Spring Boot application named `jhoog-springboot`. The Deployment has 3 replicas and uses a Jenkins image as the container. The Service is of type NodePort, exposing port 80. The GitHub interface shows 34 lines of code with 618 bytes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jhoog-springboot
spec:
  replicas: 3
  selector:
    matchLabels:
      app: jhoog-springboot
  template:
    metadata:
      labels:
        app: jhoog-springboot
    spec:
      containers:
        - name: springboot
          image: param36/pipelinetests:jenkins
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: "8080"
...
apiVersion: v1
kind: Service
metadata:
  name: jhoog-springboot
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: jhoog-springboot
```

step 8) Access the SSH of K8s MASTer



The screenshot shows a Firefox browser window displaying a Jenkins pipeline status page. The URL is `52.207.218.69:8080/job/gradle_with_k8s/`. The page title is "Pipeline gradle_with_k8s". On the left, there's a sidebar with options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. Below that is a "Build History" section with a "trend" dropdown and a filter input. The main area shows a "Stage View" table with five columns: Git Clone, Gradle Build, Docker Build, Docker Login, and Push Image to Docker Hub. The table shows four stages with their average run times: #13 (295ms), #11 (13s), #10 (5s), and #9 (402ms). The last stage (#10) is highlighted in green. Above the table, it says "Average stage times: (Average full run time: ~25s)". The Jenkins logo and user information are visible at the top right.

```

Aug 30 16:53 • 52.207.218.69:8080/job/gradle_with_k8s/14/console

Activities   Firefox Web Browser
Inbox (7) - par... Global Logic | Instances | EC2 Instance | WhatsApp | gradle_with_k8s/14/console
Dashboard > gradle_with_k8s > #14
/0/130/8e1db: Preparing
826c3ddbb29c: Preparing
b626401ef603: Preparing
9b55156abf26: Preparing
293d5db30c9f: Preparing
03127cd479b: Preparing
9c742cd6c7a5: Preparing
293d5db30c9f: Waiting
9c742cd6c7a5: Waiting
03127cd479b: Waiting
b626401ef603: Layer already exists
826c3ddbb29c: Layer already exists
7b7f3078e1db: Layer already exists
b03e080fd60a: Layer already exists
9b55156abf26: Layer already exists
03127cd479b: Layer already exists
293d5db30c9f: Layer already exists
9c742cd6c7a5: Layer already exists
jenkins: digest: sha256:6d36aa04b7cea54b724da21dc0053a338e0de4eccab6dc6870ff9fc6b983ddd81 size: 2087
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] [SSH Into k8s Server]
[Pipeline] 
[Pipeline] // stage
[Pipeline] 
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

REST API Jenkins 2.365

Step 9) Now Moving the yml file to the k8s server using SSH

	Git Clone	Gradle Build	Docker Build	Docker Login	Push Image to Docker Hub	SSH Into k8s Server	Put k8s-spring-boot-deployment.yml onto k8smaster
Average stage times: (Average full run time: ~21s)	289ms	12s	4s	364ms	1s	20ms	1s
#15 Aug 30, 2022, 11:28 AM	255ms	8s	1s	327ms	549ms	17ms	1s
#14 Aug 30, 2022, 11:22 AM	285ms	9s	1s	325ms	543ms	23ms	
#13 Aug 30, 2022, 10:48 AM	205ms	8s	1s	325ms	3s		
#12 Aug 30, 2022, 10:41 AM							
#10 Aug 30, 2022, 10:24 AM							
#9 Aug 30, 2022, 10:24 AM							
#6 Aug 30, 2022, 10:10 AM							

The screenshot shows a Firefox browser window with the URL 52.207.218.69:8080/job/gradle_with_k8s/configure. The page title is "Configuration". On the left, there's a sidebar with "General", "Advanced Project Options", and "Pipeline" selected. The main content area displays a Groovy script for a Jenkins pipeline:

```
1 node {
2     stage("Git Clone"){
3         git credentialId: "GIT_HUB_CREDENTIALS", url: "https://github.com/paramd1997/springboot-with-docker-kub.git"
4     }
5     stage("Gradle Build") {
6         sh "/gradlew build"
7     }
8     stage("Docker Build") {
9         sh 'docker version'
10        sh 'docker build -t jenkins_k8s .'
11        sh 'docker ps --list'
12        sh 'docker tag jenkins_k8s paramd1997/pipelinetests:jenkins'
13    }
14    stage("Docker Login") {
15        withCredentials([usernamePassword(credentialsId: "DOCKER_HUB_PASSWORD", variable: "PASSWORD")]) {
16            sh 'docker login -u paramd1997 -p $PASSWORD'
17        }
18    }
19    stage("Push Image to Docker Hub") {
20        sh 'docker push paramd1997/pipelinetests:jenkins'
21    }
22    stage("Deploy into k8s Server") {
23        def remote = [:]
24        remote.name = 'Kubernetes'
25        remote.address = '10.10.31.34:41'
26        remote.user = 'root'
27        remote.password = '12345678'
28        remote.allowInsecure = true
29
30        stage('Put k8s-spring-boot-deployment.yaml onto k8smaster') {
31            sshPut remote: remote, from: 'k8s-spring-boot-deployment.yaml', into: '.'
32        }
33        stage('Deploy SpringBoot') {
34            sshCommand remote: remote, command: "kubectl apply -f k8s-spring-boot-deployment.yaml"
35        }
36    }
37}
```

At the bottom, there are "Save" and "Apply" buttons.

Public IP Of the Kubernetes M/c

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various services like EC2 Dashboard, Events, Tags, Limits, Instances, Images, and Elastic Block Store. The main area displays a table of instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
k8 Master	i-Ofc17dcf7e3f562e3	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-54-224-24-31.c...
jenkinMaster	i-07df370df259b3c6c	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-52-207-218-69...
WN	i-0e055e9d2b9b0aa4c	Running	t2.medium	2/2 checks passed	No alarms	+ us-east-1b	ec2-34-207-204-81...

Details for the selected instance (i-Ofc17dcf7e3f562e3) are shown in a modal:

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Instance summary						
Instance ID i-Ofc17dcf7e3f562e3 (k8 Master)	Public IPv4 address 54.224.24.31 [open address]	Private IPv4 addresses 172.31.34.41				
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-54-224-24-31.compute-1.amazonaws.com [open address]				
Hostname type IP name: ip-172-31-34-41.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-34-41.ec2.internal	Elastic IP addresses -				
Answer private resource DNS name IPv4 (A)	Instance type t2.medium	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations.				
Auto-assigned IP address 54.224.24.31 [Public IP]	VPC ID vpc-0fbfeeb51e596b094a					

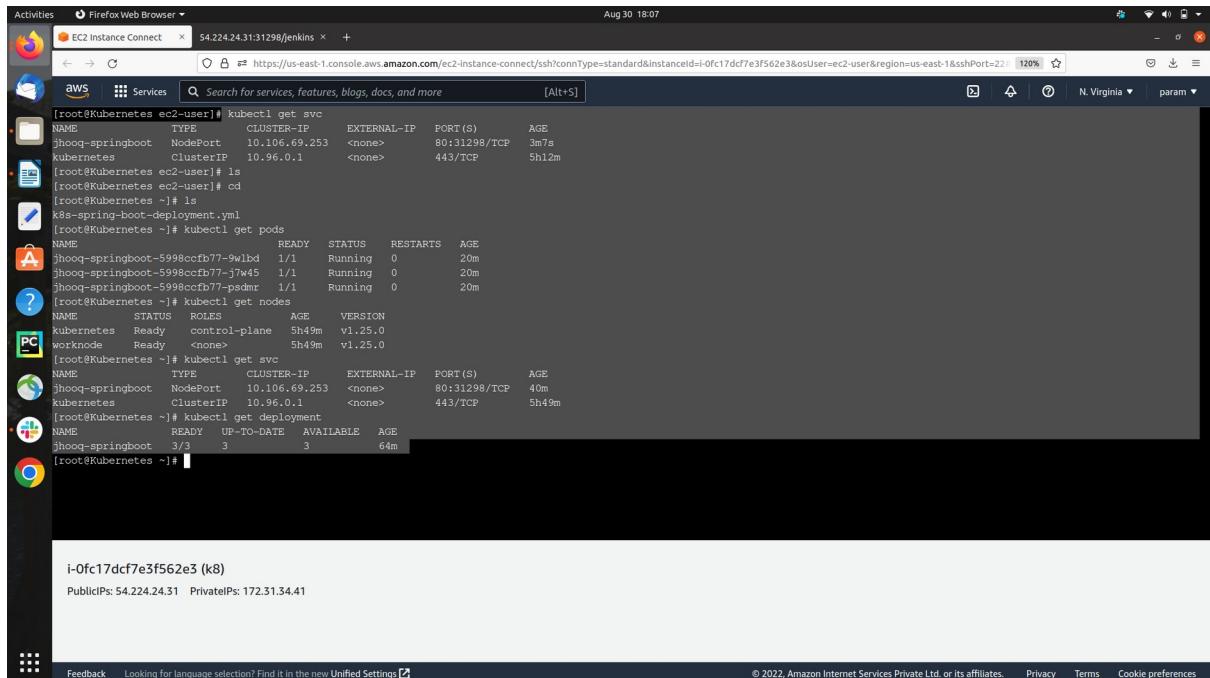
The screenshot shows an EC2 Instance Connect terminal session. The user is connected to the 'k8 Master' instance (i-Ofc17dcf7e3f562e3). The terminal output is as follows:

```
[root@Kubernetes ec2-user]# kubectl get svc
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
k8s       ClusterIP  10.96.0.1    <none>        443/TCP   5h12m
[root@Kubernetes ec2-user]# ls
[root@Kubernetes ec2-user]# cd /k8s
[root@Kubernetes ec2-user]# cd k8s-spring-boot-deployment.yaml
[root@Kubernetes ~]#
```

The bottom of the terminal window shows the instance details:

i-Ofc17dcf7e3f562e3 (k8)
PublicIPs: 54.224.24.31 PrivateIPs: 172.31.34.41

Step 10) Created the pods and container successfully and the NodePort also



The screenshot shows a terminal window within an EC2 Instance Connect session. The user has run several Kubernetes commands:

```
[root@Kubernetes ec2-user]# kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
jhooq-springboot   NodePort   10.106.69.253   <none>       80:31298/TCP   3m7s
kubernetes       ClusterIP  10.96.0.1    <none>       443/TCP       5h12m

[root@Kubernetes ec2-user]# ls
[root@Kubernetes ec2-user]# cd
[root@Kubernetes ~]# ls
k8s-spring-boot-deployment.yaml
[root@Kubernetes ~]# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
jhooq-springboot-5998ccf77-9w1bd   1/1    Running   0          20m
jhooq-springboot-5998ccf77-j7w45   1/1    Running   0          20m
jhooq-springboot-5998ccf77-psdmu   1/1    Running   0          20m

[root@Kubernetes ~]# kubectl get nodes
NAME            STATUS   ROLES   AGE   VERSION
kubernetes      Ready    control-plane   5h49m   v1.25.0
worknode        Ready    <none>    5h49m   v1.25.0

[root@Kubernetes ~]# kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
jhooq-springboot   NodePort   10.106.69.253   <none>       80:31298/TCP   40m
kubernetes       ClusterIP  10.96.0.1    <none>       443/TCP       5h49m

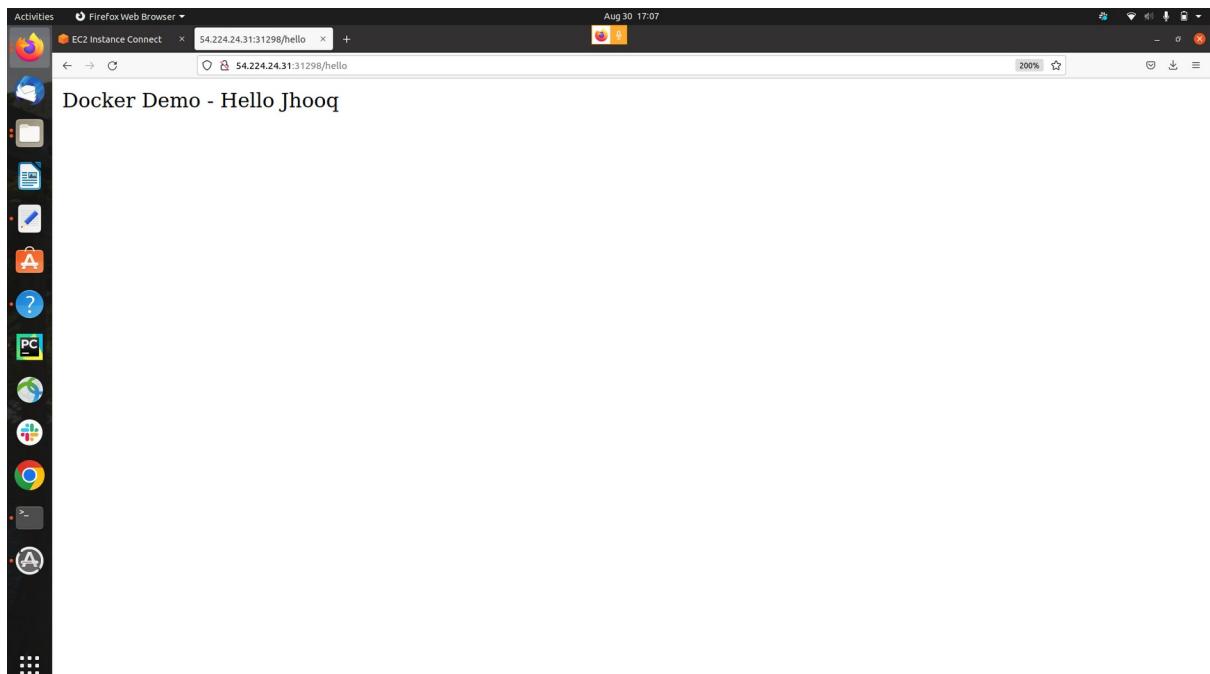
[root@Kubernetes ~]# kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
jhooq-springboot   3/3     3           3           64m

[root@Kubernetes ~]#
```

At the bottom of the terminal, it shows the instance ID and public/private IP addresses:

i-0fc17dcf7e3f562e3 (k8s)
Public IPs: 54.224.24.31 Private IPs: 172.31.34.41

Step 11) Successfully Accessing the Source Code Using the NodePort Server



THANK YOU !!