

## DAY 6 – Java Application Minikube deployment

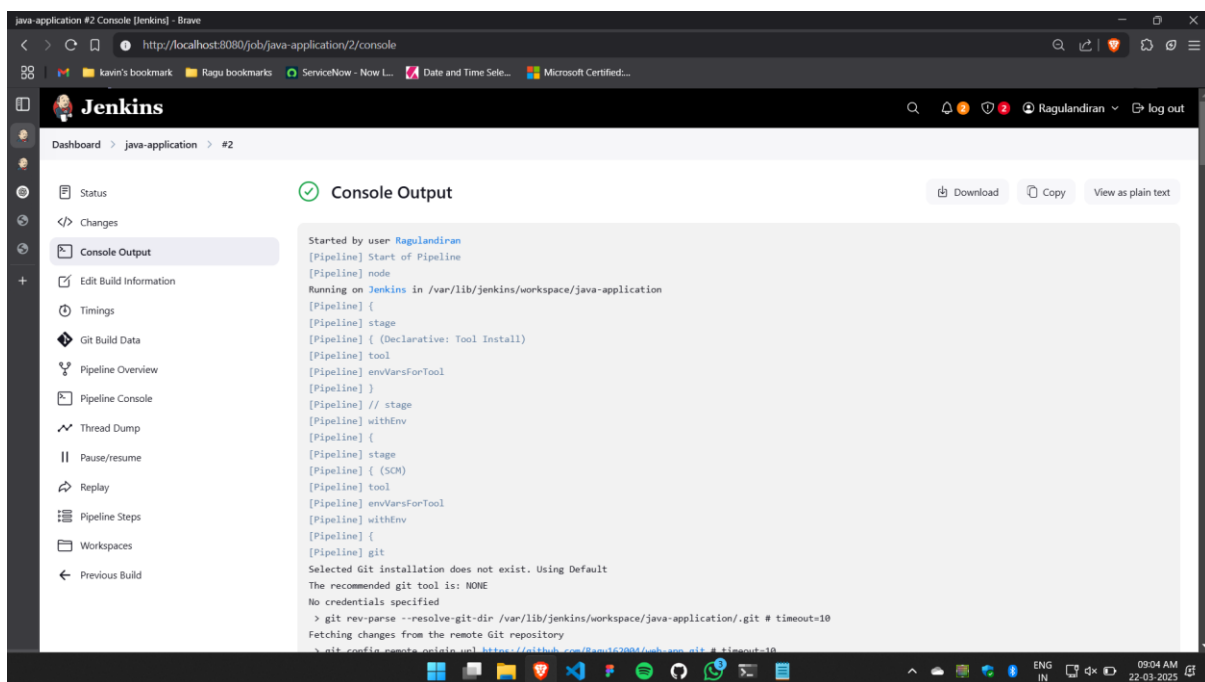
### Minikube Command:

- Minikube start
- Minikube status
- Sudo nano deploy.yaml
- Sudo nano pod.yaml
- Kubectl apply -f pod.yaml
- Kubectl apply -f deploy.yaml
- Minikube service my-service
- For permission to, sudo visudo
- Kubectl get node
- Configuring config file in ~/.kube, Sudo vi config

**Sudo visudo:** update this, jenkins ALL=(ALL) NOPASSWD: ALL

Data updation in config file:

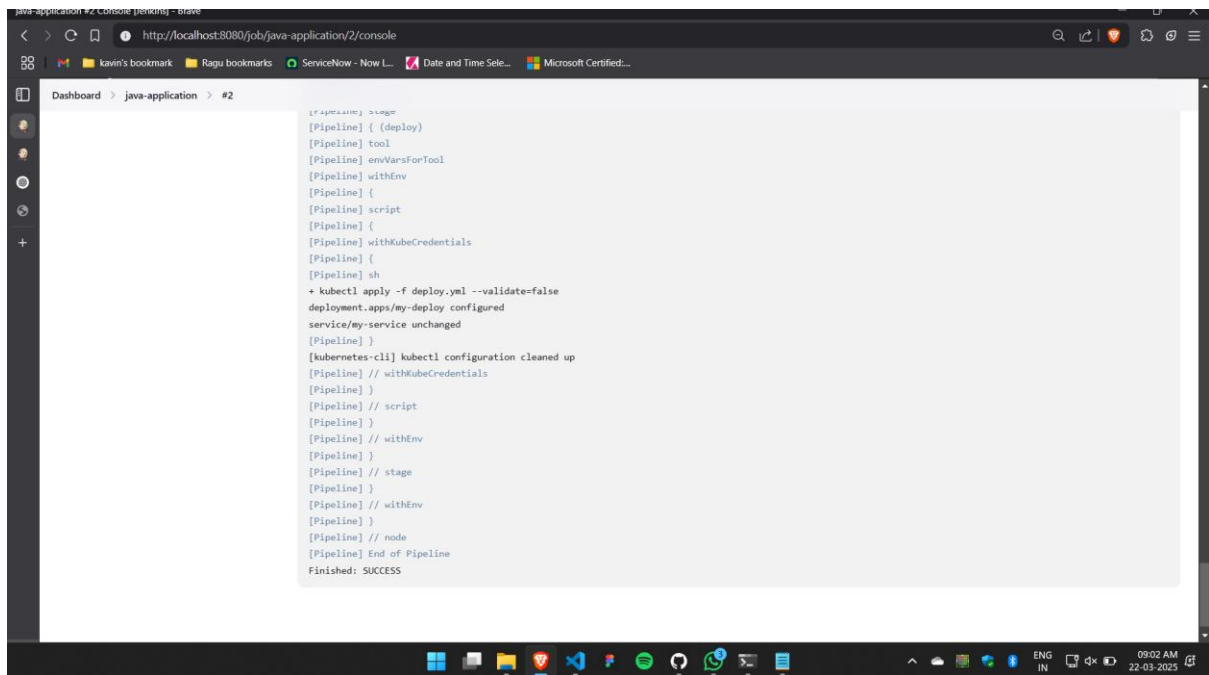
- sudo cat /home/ragu\_ubuntu/.minikube/ca.crt
- sudo cat /home/ragu\_ubuntu/.minikube/ca.crt | base64 -w 0; echo
- sudo cat /home/ragu\_ubuntu/.minikube/profiles/minikube/client.crt | base64 -w 0; echo
- sudo cat /home/ragu\_ubuntu/.minikube/profiles/minikube/client.key | base64 -w 0; echo



The screenshot shows the Jenkins web interface in a browser. The main panel displays the 'Console Output' for a build named 'java-application #2'. The output text is as follows:

```
Started by user Ragulandiran
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/java-application
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (SSH)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] git
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/java-application/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/Ramd62002/ueb.com.git # timeout=10
```

The left sidebar shows the Jenkins navigation menu with options like Status, Changes, Console Output, Edit Build Information, Timings, Git Build Data, Pipeline Overview, Pipeline Console, Thread Dump, Pause/resume, Replay, Pipeline Steps, Workspaces, and Previous Build. The top of the interface shows the Jenkins logo and the user 'Ragulandiran' is logged in.



## Deploy.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deploy

labels:

name: my-deploy

spec:

replicas: 1

selector:

matchLabels:

apptype: web-backend

strategy:

type: RollingUpdate

template:

metadata:

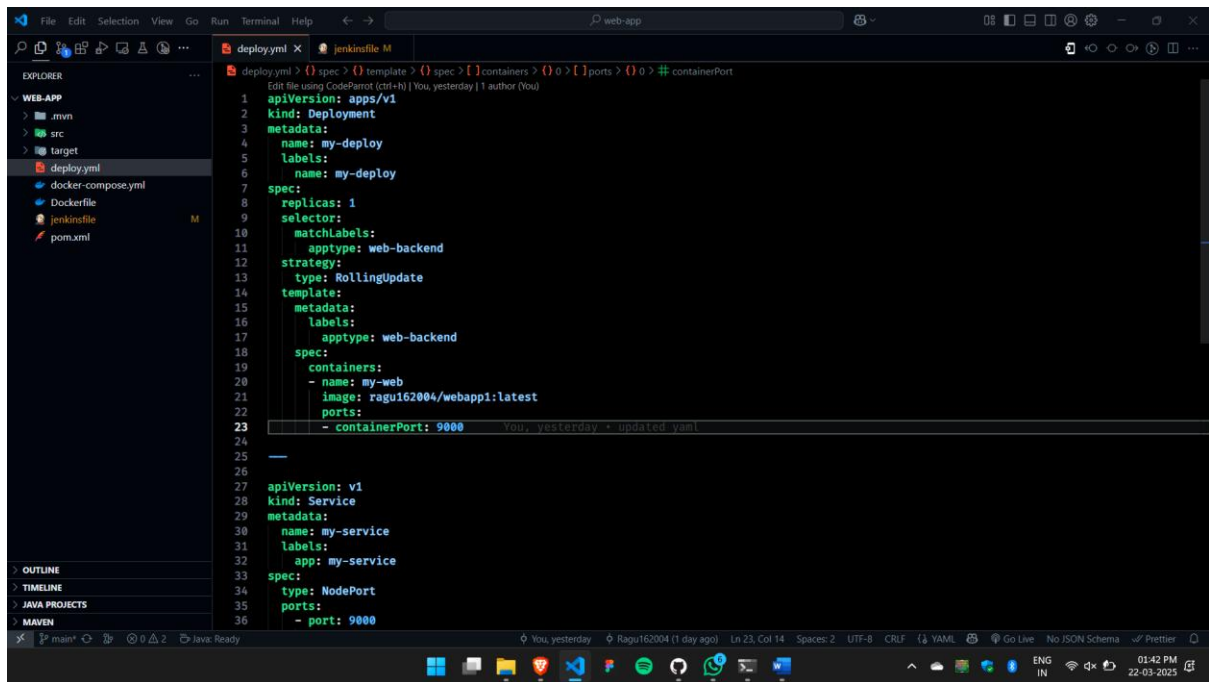
labels:

apptype: web-backend

```
spec:
  containers:
    - name: my-web
      image: ragu162004/webapp1:latest
  ports:
    - containerPort: 9000
```

---

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: my-service
spec:
  type: NodePort
  ports:
    - port: 9000
      targetPort: 8080
      nodePort: 30002
  selector:
    apptype: web-backend
```



## Pipeline Script:

```
pipeline {
    agent any

    tools {maven "maven"}

    stages {
        stage('SCM') {
            steps {
                git branch: 'main', url: 'https://github.com/Ragu162004/web-app.git'
            }
        }

        stage('Build-clean') {
            steps {
                sh 'mvn clean'
            }
        }

        stage('Build-validate') {
            steps {
                sh 'mvn validate'
            }
        }
    }
}
```

```

    }
    stage('Build-complie') {
        steps {
            sh 'mvn compile'
        }
    }
    stage('Build-package') {
        steps {
            sh 'mvn package'
        }
    }
    stage('build to images') {
        steps {
            script {
                sh 'docker build -t ragu162004/webapp1 .'
            }
        }
    }
    stage('push to hub') {
        steps {
            script {
                withDockerRegistry(credentialsId: 'docker_cred', toolName: 'docker', url:
'https://index.docker.io/v1/') {
                    sh 'docker push ragu162004/webapp1'
                }
            }
        }
    }
}

```

}

The screenshot shows the Jenkins Configuration page for a job named 'java-application'. The left sidebar contains a 'Configure' section with tabs for 'General', 'Triggers', 'Pipeline', and 'Advanced'. The 'Pipeline' tab is selected, displaying a Jenkins Pipeline script. The script defines several stages: 'SCM' (fetching code from GitHub), 'Build-clean' (cleaning the workspace), 'Build-validate' (validating the code), 'Build-compile' (compiling the code), 'Build-package' (building the package), 'build to images' (building Docker images), 'push to hub' (pushing images to Docker Hub), and 'deploy' (deploying to Kubernetes). The 'deploy' stage uses 'withKubeCredentials' to manage credentials. At the bottom of the configuration area are 'Save' and 'Apply' buttons.

```
5 stage('SCM') {
6   steps {
7     git branch: 'main', url: 'https://github.com/Ragu12004/web-app.git'
8   }
9 }
10 stage('Build-clean') {
11   steps {
12     sh 'mvn clean'
13   }
14 }
15 stage('Build-validate') {
16   steps {
17     sh 'mvn validate'
18   }
19 }
20 stage('Build-compile') {
21   steps {
22     sh 'mvn compile'
23   }
24 }
25 stage('Build-package') {
26   steps {
27     sh 'mvn package'
28   }
29 }
30 stage('build to images') {
31   steps {
32     script {
33       sh 'docker build -t ragu12004/webapp1 .'
34     }
35   }
36 }
37 stage('push to hub') {
38   steps {
39     script {
40       withDockerRegistry(credentialsId: 'docker_cred', toolName: 'docker', url: 'https://index.docker.io/v1/') {
41         sh 'docker push ragu12004/webapp1'
42       }
43     }
44   }
45 }
46 stage('deploy') {
47   steps {
48     script {
49       withKubeCredentials(kubect1Credentials: [[caCertificate: '', clusterName: 'minikube', contextName: 'minikube', credentialsId: 'kubect1']], {
50         sh 'kubectl apply -f deploy.yml --validate=false'
51       }
52     }
53   }
54 }
55 }
```

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. The left sidebar has a 'Jenkins' header and a 'Dashboard' section with links to 'Snippet Generator', 'Declarative Directive Generator', 'Declarative Online Documentation', 'Steps Reference', 'Global Variables Reference', 'Online Documentation', 'Examples Reference', and 'IntelliJ IDEA GDSDL'. The 'Snippet Generator' tab is selected, showing an 'Overview' section. The 'Overview' section explains the purpose of the Snippet Generator and provides instructions on how to use it. Below the overview, there is a 'Steps' section with a 'Sample Step' dropdown menu. The selected step is 'withKubeCredentials: Configure Kubernetes CLI (kubectl) with multiple credentials'. Below this, there is a form to configure the step, including fields for 'Credentials to use' (a dropdown menu with 'config (kube\_cred)' selected), 'Kubernetes API endpoint' (a text field with 'https://127.0.0.1:32769'), and 'Cluster name' (a text field).

**Overview**

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values)

**Steps**

Sample Step

withKubeCredentials: Configure Kubernetes CLI (kubectl) with multiple credentials

withKubeCredentials

Credentials to use

Credentials

config (kube\_cred)

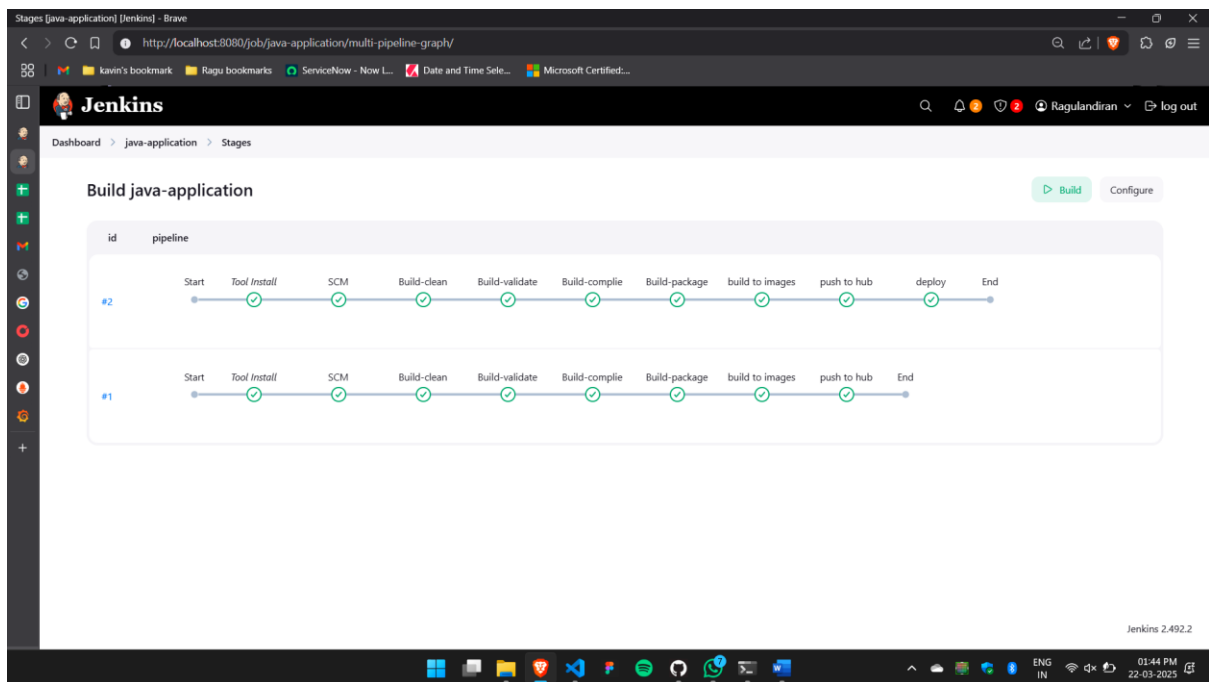
+ Add

Kubernetes API endpoint

https://127.0.0.1:32769

Cluster name

## Stages:



## Stage View of pipeline project:

The image shows the Jenkins Stage View for a job named 'java-application'. The view displays a table of stage execution times for two builds, #2 and #1. The stages are: Declarative: Tool Install, SCM, Build-clean, Build-validate, Build-compile, Build-package, build to images, push to hub, and deploy. The table also includes average stage times and a full run time of ~45s. The Jenkins interface includes a sidebar with navigation links, a top bar with the Jenkins logo and user information, and a bottom status bar showing the Jenkins version (2.492.2) and system time (11:25 AM 22-03-2025).

	Declarative: Tool Install	SCM	Build-clean	Build-validate	Build-compile	Build-package	build to images	push to hub	deploy
Average stage times: (full run time: ~45s)	240ms	3s	3s	2s	4s	4s	945ms	21s	2s
#2 09:01 No Changes	118ms	1s	3s	2s	6s	4s	635ms	18s	2s
#1 08:53 No Changes	362ms	5s	3s	3s	3s	4s	1s	23s	

**Permalinks**

- Last build (#2), 2 hr 24 min ago
- Last stable build (#2), 2 hr 24 min ago
- Last successful build (#2), 2 hr 24 min ago
- Last completed build (#2), 2 hr 24 min ago

- Minikube srv my-service

```
ragu_ubuntu@Kavin: ~
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

ragu_ubuntu@Kavin: $ minikube service my-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service	9000	http://192.168.49.2:30002

```
Starting tunnel for service my-service.
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service		http://127.0.0.1:39141

```
Opening service default/my-service in default browser...
http://127.0.0.1:39141
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
^C Stopping tunnel for service my-service.
ragu_ubuntu@Kavin: $ minikube service my-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service	9000	http://192.168.49.2:30002

```
Starting tunnel for service my-service.
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service		http://127.0.0.1:46509

```
Opening service default/my-service in default browser...
http://127.0.0.1:46509
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

