

Program: Post Graduate Program in DevOps

Name of Institution: Edureka/ Purdue University

Name: Wezi Bonono Chipeta

Industrial Grade Project 1

Title: Building a CI/CD Pipeline for ABC Technologies

Project GitHub Repository: <https://github.com/Izewdevlabs/igpaug26>

Date: 7th September 2025

Table of Contents

1.	Introduction	5
2.	Business Challenge/Requirement	5
3.	The Goal of the Project	6
4.	Data Flow Architecture/Process Flow	6
5.	Data Explanation and Schema:.....	6
6.	Problem Statements/Tasks	6
7.	Pre-requisites:	7
8.	Approach to Solve:	7
9.	Considerations/ Assumptions	8
10.	DevOps Solution to ABC Technologies.....	8
10.1	Source Code Management	8
10.2	Integration and Build	12
10.2.1	EC2 Setup.....	12
10.2.2	Access of the VM with PuTTY	15
10.2.3	Installation and Configuration of Jenkins	16
10.2.4	Installation of Java	17
10.2.5	Unlocking Jenkins	17
10.2.6	Creating initialAdminPassword.....	18
10.2.7	Creating First Admin User.....	18
10.2.8	Instance Configuration	19
10.2.9	Installation of Maven on the IGP_ABCTech.....	20
10.2.10	Installation of Git on the IGP_ABCTech.....	21
10.2.11	Installation of Maven Plugins in Jenkins	22
10.2.12	Creation of First CI Job- ABCTechnologies- CI	22
10.3	Deployment Using Docker Container	24
10.3.1	Step 1: Installation of Docker on IGP_ABCTech VM	25
10.3.2	Step 2: Giving Privileges to Jenkins User to Run Docker Commands.....	25
10.3.3	Step 3: Store the Dockerhub Credentials in Some ID (mydockerhubcred)	25
10.3.4	Step 4: Installation of Docker Pipeline Plugin to Pass Credentials	25
10.3.5	Step 5: Writing a Dockerfile and Pushing to GitHub Repository	26
10.3.6	ABCTechnologies CI-CD Pipeline and Deployment to Docker	27
10.4	Deployment to Kubernetes	30
10.4.1	Step 1: Update Packages and Install Dependencies	31
10.4.2	Step 2: Install AWS CLI v2	31
10.4.3	Step 3: Install kubectl	31
10.4.4	Step 4: Install eksctl	32
10.4.5	Step 5: Configure AWS CLI on the EC2.....	32
10.4.6	Step 6: Create an EKS Cluster with 2 Worker Nodes	33
10.4.7	Step 7: Writing deployment.yaml, service.yaml, ingress.yaml, namespace.yaml and hpa.yaml manifest files.....	34
10.4.8	Step 8: Pipeline Script with Deployment to Kubernetes	36
10.4.9	Step 9: Getting the Service ELB URL	38
10.4.10	Step 10: Testing the App	39

10.5	Monitoring Using Prometheus and Grafana.....	40
10.5.1	Installation of the Stack	40
10.5.2	Getting Grafana URL + Admin Password	42
10.5.3	Dashboards Monitored.....	44
11.	Summary.....	45

Table of Figures

Figure 1:	Data Flow Architecture/ Process Flow	6
Figure 2:	Edureka's Industry Grade Project File Repository	9
Figure 3:	Directory Structure of the Source Code on the Local Machine (Laptop).....	9
Figure 4:	Local Repo Created with Untracked Files.....	10
Figure 5:	Committed Changes to the Local Repository.....	10
Figure 6:	History of commands to create a local Git repository and have changes committed.....	11
Figure 7:	Executed Commands to Push the Codebase to Remote GitHub Repository	12
Figure 8:	Remote GitHub Repository of the Codebase to Verify Successful Push from Local Repository	12
Figure 9:	VM Name and OS Image Setup	13
Figure 10:	Keypair and Network Setups.....	14
Figure 11:	8GB Secondary Storage.....	14
Figure 12:	Launching the EC2 Instance.....	15
Figure 13:	Access of the EC2 Instance Using PuTTY and Logged in as Ubuntu User.....	15
Figure 14:	Jenkins Installation Commands from Jenkins Website	16
Figure 15:	Successful Jenkins Installation Confirmation.....	16
Figure 16:	TCP Port 8080 Allowed for Inbound Traffic to Allow Jenkins to be Accessed via the Browser	16
Figure 17:	Java Installation Commands from Jenkins Webpage	17
Figure 18:	Java 21.0 Version Confirmed as Installed.....	17
Figure 19:	Request for initialAdminPassword to Unlock Jenkins	18
Figure 20:	Creation of the initialAdminPassword	18
Figure 21:	Creating first Admin user	19
Figure 22:	Replacing Jenkins URL from externalIP:8080 to internalIP:8080.....	20
Figure 23:	History of Maven Installation Commands.....	20
Figure 24:	Apache Maven 3.9.9 Version Confirmation.....	21
Figure 25:	Git Installation Command and Output.....	21
Figure 26:	Jenkins Home Directory and its Associated Files	22
Figure 27:	Simple ABCTechnologies- CI Job Builds #3 & #4 Successful Results.....	23
Figure 27:	Build logs of the ABCTechnologies- CI Job.....	24
Figure 29:	Dockerhub Account with One Repository.....	25
Figure 30:	Adding Dockerfile to Local Repository	26
Figure 31:	History of commands to have the Dockerfile Pushed to the Remote Repository.....	26
Figure 32:	History of Commands to have the Dockerfile Pushed to the Remote Repository.....	27
Figure 33:	Port 18080 Opened for Apache Tomcat in the VM's Security Group with AWS	28
Figure 33:	Snapshots of Docker Image Pushed Dockerhub.....	29
Figure 35:	Webapp View on the Browser.....	29
Figure 36:	Commands for installation of AWS CLI.....	31
Figure 37:	Install kubectl Command.....	31
Figure 38:	Install eksctl Command.....	32
Figure 39:	Created Jenkins-EKS Role	32
Figure 40:	Permission policies attached to Jenkins-EKS Role	32
Figure 41:	AWS CLI Configuration on the EC2	33

Figure 42: AWS EKS Cluster with Nodes Ready to be Scheduled for Workloads	33
Figure 43: Cluster with 2 Worker Nodes Running	34
Figure 44: K8s folder and manifest files pushed to Github Repository “igpaug26”	36
Figure 45: Stage View of the 3 Builds in the Jenkins Dashboard of the ABCTechnologies-CI-CD Job	38
Figure 46: Service ELB URL View in Jenkins.....	39
Figure 47: Service ELB URL Output Using the Terminal.....	39
Figure 48: ABCtechnologies Web app Landing Page (accessed via AWS ELB).....	39
Figure 49: History of Commands Executed to Install the Stack	40
Figure 50: Verification the Monitoring was Installed and Running	42
Figure 51: Getting Grafana Password	42
Figure 52: ELB URL for Grafana	43
Figure 53: Different Kubernetes/ Compute dashboards within Grafana.....	43
Figure 54: Cluster Dashboard (CPU utilization, CPU usage, CPU Quota and Memory)	44
Figure 55: Per Pod Metrics Dashboard (CPU utilization, CPU usage, CPU Quota and Memory).....	44

1. Introduction

This document provides a detailed solution using DevOps for ABC Technologies, an online retailer, which has been using conventional development and deployment approaches which have led to significant losses and issues such as low availability, scalability, and performance, along with difficult builds, maintenance, and slow-release cycles.

In the DevOps solutions provided, five tasks and steps to accomplish them are outlined. Using the provided source code to work with, complete Continuous Integration/ Continuous Deployment (CI/CD) pipeline involving source code management using Git, continuous integration using Jenkins, continuous integration using Docker, container orchestration using Kubernetes, and continuous monitoring using Prometheus and Grafana. Among others, complete code developed such as dockerfile, CI/CD pipeline job scripts for both Docker and Kubernetes deployments are provided. In addition, code for manifest files deployment.yaml, service.yaml, namespace.yaml, ingress.yaml and hpa.yaml are provided. All in all, snapshots of the command and outputs either on the terminal or within Github, Dockerhub, Jenkins, Docker, Grafana and AWS were provided.

This DevOps solution provided for ABC Technologies would ensure that operational benefits would make the service that is: highly available, highly scalable, highly performant; easily built and maintained; developed and deployed quickly; lower production bugs; frequent releases; better customer experience and less time to market. Now, let's explore what business challenges ABC Technologies online retailer is faced with as provided in the project brief.

2. Business Challenge/Requirement

ABC Technologies, a top online retailer, has acquired a large offline retail business with stores worldwide. However, the conventional development and deployment approach has led to significant losses and issues such as low availability, scalability, and performance, along with difficult builds, maintenance, and slow-release cycles.

ABC will acquire the data from all these storage systems and plan to use it for analytics and prediction of the firm's growth and sales prospects. In the first phase ABC must create the servlets to Add a product and Display product details. Add servlet dependencies required to compile the servlets. It is required to create an HTML page which will be used to add a product. Team is using git to keep all the source code.

ABC has decided to use DevOps model and once source code is available in GitHub, it is

required to integrate it with Jenkins and provide continuous build generation for continuous delivery, integrate with Ansible and Kubernetes for deployment. It would be required to use docker hub to pull and push images between ansible and Kubernetes.

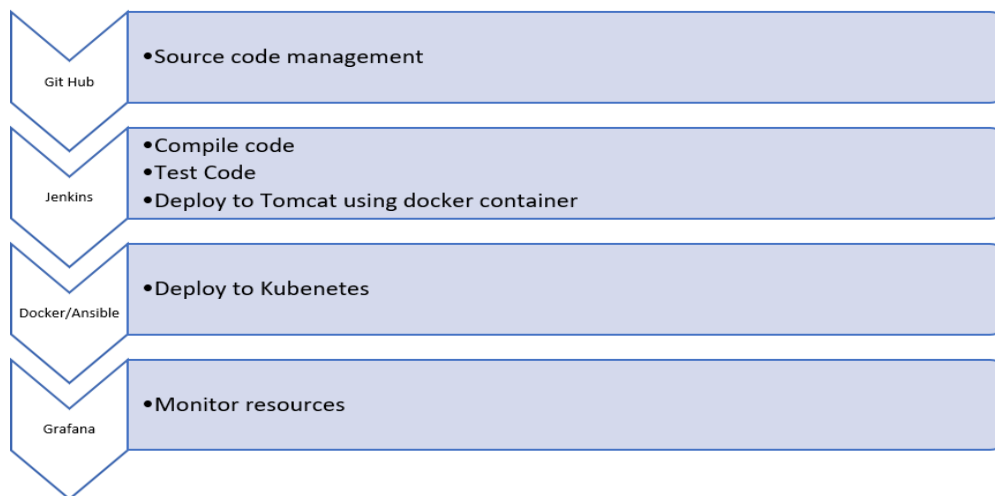
3. The Goal of the Project

According to the project brief, some of the high-level goals of the project comprise implementing CI/CD such as ABC Technologies can be *highly available; highly scalable; highly performant; easily built and maintained and developed and deployed quickly*.

4. Data Flow Architecture/Process Flow

Figure 1 shows the data flow architecture/ process flow architecture to be followed in proving solution to ABC Technologies. According to this architecture data would flow from GitHub as part of Source code management, then to Jenkins where the source code would be compiled, tested and deployed to Tomcat using Docker container. Then, deployment would occur to Kubernetes from Docker. After deployment to Kubernetes, the resources would be monitored using Grafana dashboards.

Figure 1: Data Flow Architecture/ Process Flow



5. Data Explanation and Schema:

Sample Java project had been shared for usage. It was a maven project and had source code (SRC) and test folders created into it. It had a POM.xml file which listed all needed dependencies to execute this project.

6. Problem Statements/Tasks

As per problem statement, it was required to develop a CICD pipeline to automate the software

development, testing, packaging, and deploying, thereby reducing the time to market of app and ensuring good quality service is experienced by end users. In this project it was required to:

- Push the code to GitHub repository
- Create a continuous integration pipeline using Jenkins to compile, test and package the code present in GitHub
- Write docker file to push the war file to tomcat server
- Integrate docker with Ansible and write playbook (*optional according to class of 23 August 2025*)
- Deploy artifacts to Kubernetes cluster
- Monitor resources using Grafana

7. Pre-requisites:

Before building the CICD pipeline, it was necessary to verify following software installed in the working machine. *Java, Maven, Git, Jenkins, Docker, Ansible, Kubernetes, Grafana and Prometheus.*

8. Approach to Solve:

The following were the suggested approach to solving the problem which was followed. However, the class on 23 August 2025 provided additional technical guide on the approach by the instructor.

Task 1: *Clone the project from GitHub link shared in resources to your local machine. Build the code using maven commands.*

Task 2: *Setup git repository and push the source code. Login to Jenkins*

- Create a build pipeline containing a job each
 - One for compiling source code
 - Second for testing source code
 - Third for packing the code
- Execute CICD pipeline to execute the jobs created in step1
- Setup master-slave node to distribute the tasks in pipeline

Task 3: *Write a Docker file to create an Image and container on docker host. Integrate docker host with Jenkins. Create CI/CD job on Jenkins to build and deploy on a container*

- Enhance the package job created in step 1 of task 2 to create a docker image
- In the docker image add code to move the war file to tomcat server and build the image

Task 4: *Deploy Artifacts on Kubernetes; Write pod, service, and deployment manifest file*

Task 5: *Using Prometheus monitors resources like CPU utilization: Total Usage, Usage per core, usage breakdown, Memory, Network on the instance by providing the end points in local host. Install node exporter and add URL to target Prometheus. Using this data login to Grafana and create a dashboard to show the metrics.*

9. Considerations/ Assumptions

Provision of the DevOps solution to ABC Technologies had considerations and assumptions on the resources needed such as:

- An AWS account
- A GitHub account
- MobaXterm / Putty
- Git Bash setup
- Source Code

All the above resources were available. In addition to the above, Docker hub Account was added to the list.

10. DevOps Solution to ABC Technologies

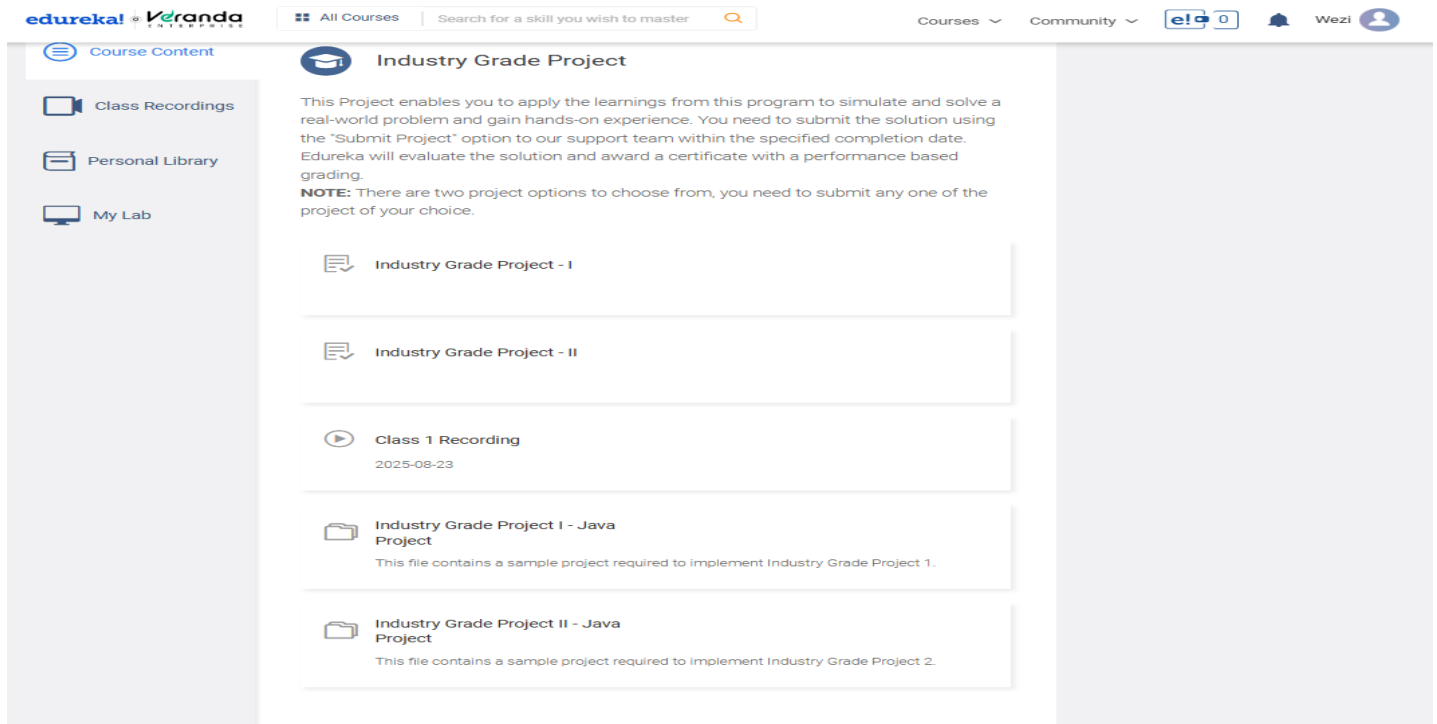
Following sections cover steps that were taken to provide the DevOps solution to the problem faced by ABC Technologies for each task as per the data scheme. The first one was source code management which covered next.

10.1 Source Code Management

Task 1: *Clone the project from GitHub link shared in resources to your local machine. Build the code using maven commands.*

No GitHub link was provided to clone the resources in one's local machine. However, for the ABC Technologies project, a zipped file “*Industry Grade Project 1- Java Project.zip*” was downloaded from Eureka's LMS portal to the local Windows laptop. *Figure 2* shows the setup within LMS portal of the files for the Industry Grade Project (IGP).

Figure 2:Edureka's Industry Grade Project File Repository



The downloaded file was unzipped. Figure 3 shows the screenshot of the directory structure of the folders and files for the project.

Figure 3:Directory Structure of the Source Code on the Local Machine (Laptop)

Name	Date modified	Type	Size
.settings	8/25/2025 6:43 PM	File folder	
src	8/25/2025 6:43 PM	File folder	
.classpath	8/31/2022 7:40 PM	CLASSPATH File	2 KB
.project	9/5/2022 5:12 PM	PROJECT File	1 KB
pom	9/6/2022 11:45 AM	XML Document	3 KB
pom.xml.bak	8/31/2022 7:40 PM	BAK File	1 KB
README	9/6/2022 11:47 AM	Markdown Source...	1 KB

Figure 3 above shows the necessary codebase for the project. As such it was necessary to create a local repository to track the files on the local machine. To track the codebase, Git Bash was used, where the new local repo was created, with files added and able to be tracked. By using Git Bash in Windows, the following commands were executed on CLI:

```
>git init # initiate a git repository
>git status
>git add .
```

Figure 4 shows the screenshot of the commands that ensured that the local repo was created, and files added and able to be tracked but not yet committed.

Figure 4: Local Repo Created with Untracked Files

```
MINGW64/d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies
$ git init
Initialized empty Git repository in D:/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies/.git/

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .classpath
        .project
        .settings/
        README.md
        pom.xml
        pom.xml.bak
        src/

nothing added to commit but untracked files present (use "git add" to track)

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git add .
warning: in the working copy of 'src/main/webapp/WEB-INF/web.xml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/main/webapp/index.jsp', LF will be replaced by CRLF the next time Git touches it

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .classpath
        new file:   .project
        new file:   .settings/org.eclipse.jdt.core.prefs
        new file:   .settings/org.eclipse.m2e.core.prefs
        new file:   README.md
        new file:   pom.xml
        new file:   pom.xml.bak
        new file:   src/main/java/com/abc/RetailModule.java
        new file:   src/main/java/com/abc/dataAccessObject/RetailAccessObject.java
        new file:   src/main/java/com/abc/dataAccessObject/RetailDataImp.java
        new file:   src/main/webapp/WEB-INF/web.xml
        new file:   src/main/webapp/index.jsp
        new file:   src/test/java/com/abc/dataAccessObject/ProductImpTest.java

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$
```

To commit the changes to the local repository, the following commands were ran:

```
>git commit -m "codebase"
>git log --oneline
>git status
```

Figure 5 shows the screenshot of the output after executing the command in which 13 files were changed.

Figure 5: Committed Changes to the Local Repository

```
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git commit -m "codebase"
[master (root-commit) d9f6bdb] codebase
13 files changed, 313 insertions(+)
 create mode 100644 .classpath
 create mode 100644 .project
 create mode 100644 .settings/org.eclipse.jdt.core.prefs
 create mode 100644 .settings/org.eclipse.m2e.core.prefs
 create mode 100644 README.md
 create mode 100644 pom.xml
 create mode 100644 pom.xml.bak
 create mode 100644 src/main/java/com/abc/RetailModule.java
 create mode 100644 src/main/java/com/abc/dataAccessObject/RetailAccessObject.java
 create mode 100644 src/main/java/com/abc/dataAccessObject/RetailDataImp.java
 create mode 100644 src/main/webapp/WEB-INF/web.xml
 create mode 100644 src/main/webapp/index.jsp
 create mode 100644 src/test/java/com/abc/dataAccessObject/ProductImpTest.java

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$
```

To check which commands were executed thus far, history command within Git Bash was used.

Figure 6 below shows the history of commands executed as highlighted from lines 42-48.

Figure 6: History of commands to create a local Git repository and have changes committed

```
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ history
1 exit
2 git
3 git clone https://github.com/CryptowizardsNet/flash-loan-arbitrage-triangular.git flashswap
4 git clone https://github.com/CryptowizardsNet/flash-loan-arbitrage-triangular.git flashswap
5 echo "# wezichipeta" >> README.md
6 git init
7 git add README.md
8 git commit -m "first commit"
9 git branch -M main
10 git remote add origin https://github.com/wezichipeta/wezichipeta.git
11 git push -u origin main
12 git clone https://github.com/atomicals/atomicals-js.git
13 https://github.com/wezichipeta/TeamSphere.git
14 https://github.com/Patty-Hsu/TeamSphere.git
15 pip install PyQt5 PyQt5-tools
16 git init
17 git init
18 git add file.txt
19 echo "Git Rocks" >>file.txt
20 git add file.txt
21 git status
22 cd ..
23 cd /Repos/
24 mkdir Repos
25 git init
26 git --version
27 pwd
28 git clone https://github.com/Izewdevlabs/tumacash
29 git remote add origin https://github.com/Izewdevlabs/tumacash
30 git remote -v
31 git clone https://github.com/Izewdevlabs/privondao
32 git remote remove
33 git remote remove privondao
34 git remote add origin https://github.com/Izewdevlabs/privondao
35 git remote -v
36 echo "# privondao" >> README.md
37 git init
38 git add README.md
39 git commit -m "first commit"
40 git clone https://github.com/krishnaik06/Complete-Python-Bootcamp
41 exit
42 git init
43 git status
44 git add .
45 git status
46 git commit -m "codebase"
47 git log --oneline
48 git status
49 history
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$
```

The above activities concluded Task 1 of the project. Task 2 involved setting up a remote git repository and pushing the source code. In addition, it was required to log into Jenkins. This is covered next.

Task 2: Setup git repository and push the source code. Login to Jenkins Creation of Remote Repository on GitHub

To push the source code from the local repository to GitHub so that it can be accessed by Jenkins, it was required to create a remote repository on GitHub. A public repository was created for the project called “*igpaug26.git*” with the following URL: <https://github.com/Izewdevlabs/igpaug26>. Figure 7 shows the screenshot of the series of commands executed to push the local codebase to remote repo. These comprised:

```
>git remote -v
>git remote add origin https://github.com/Izewdevlabs/igpaug26.git
>git remote -v
>git branch
>git push -u origin master #upload codebase upstream from the master
branch
```

Figure 7 shows the above commands executed and associated outputs.

Figure 7: Executed Commands to Push the Codebase to Remote GitHub Repository

```
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git remote add origin https://github.com/Izewdevlabs/igpaug26.git

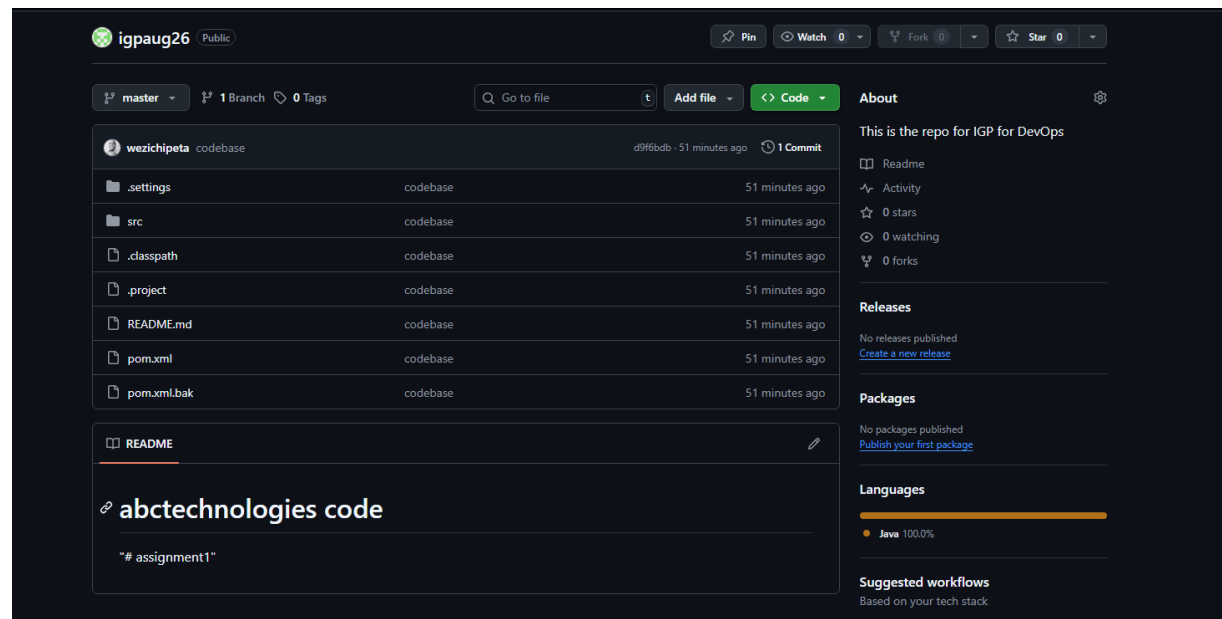
wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git remote -v
origin  https://github.com/Izewdevlabs/igpaug26.git (fetch)
origin  https://github.com/Izewdevlabs/igpaug26.git (push)

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ git push -u origin master
info: please complete authentication in your browser...
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 8 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (29/29), 4.44 KiB | 101.00 KiB/s, done.
Total 29 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Izewdevlabs/igpaug26.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

wezic@IzewDevLabs MINGW64 /d/Purdue DevOps/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ |
```

Having executed the commands above, Figure 8 shows the screenshot of the GitHub “igpaug26.git” repository to verify that the codebase had been pushed successfully from local to GitHub. The successful pushing of the codebase on GitHub marked the end of the source code management part of the pipeline.

Figure 8: Remote GitHub Repository of the Codebase to Verify Successful Push from Local Repository



Having dealt with the source code management of the pipeline, the next step was to deal with the integration of the source code management with Jenkins.

10.2 Integration and Build

10.2.1 EC2 Setup

Integration setup was still part of Task 2 and the tools required comprised Jenkins and Maven.

The first step was to configure a Linux Virtual Machine (VM) within any public cloud of choice (AWS/GCP/Azure) and install Jenkins and Maven there. For this project, Amazon Web Services (AWS) public cloud was used. Using the existing AWS account, the VM was configured with the following Elastic Cloud Compute (EC2) specs as follows:

- Name: *IGP_ABCTech*
- Operating System (OS) Image: Ubuntu Linux 22.04 LTS
- Memory: 2core CPU & 4GB RAM
- Secondary Storage: 8GB

Figures 9 to 12 below show screenshots in the process of configuring the VM in AWS. Figure 8 shows the configuration of EC2 instance name and Linux OS image version. Figure 9 shows the key pair and network setup. Figure 10 shows the secondary storage setup for the VM. Figure 11 shows the successful launch of the EC2 instance and shows it was successfully running. Figure 12 shows access to the EC2 instance using PuTTY and successfully logged as ubuntu user.

Figure 9: VM Name and OS Image Setup

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name

IGP_ABCTech [Add additional tags](#)

▼ Application and OS Images (Amazon Machine Image) Info

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose [Browse more AMIs](#).

Recents

Quick Start

Amazon Linux

aws

macOS

Mac

Ubuntu

ubuntu

Windows

Microsoft

Red Hat

Red Hat

SUSE Linux

SUSE

Debian

debian

[Browse more AMIs](#)

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-0bbdd8c17ed981ef9 (64-bit (x86)) / ami-01b2110eef525172b (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible ▼

Figure 10: Keypair and Network Setups

▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Edureka-DevOps Training ▼

↻ Create new key pair

▼ Network settings Info

Edit

Network Info

vpc-06c7c1698d609f073

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

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

☐ Create security group

☒ Select existing security group

Common security groups Info

Select security groups ▼

launch-wizard-1 sg-0bbf97848604879c7 ✕

VPC: vpc-06c7c1698d609f073

↻ Compare security group rules

Security groups that you add or remove here will be added to or removed from all your network interfaces.

Figure 11: 8GB Secondary Storage

▼ Configure storage Info

Advanced

1x 8 GiB gp2 ▼

Root volume, Not encrypted

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage ✕

Add new volume

The selected AMI contains instance store volumes, however the instance does not allow any instance store volumes. None of the instance store volumes from the AMI will be accessible from the instance

🕒 Click refresh to view backup information

🔄

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

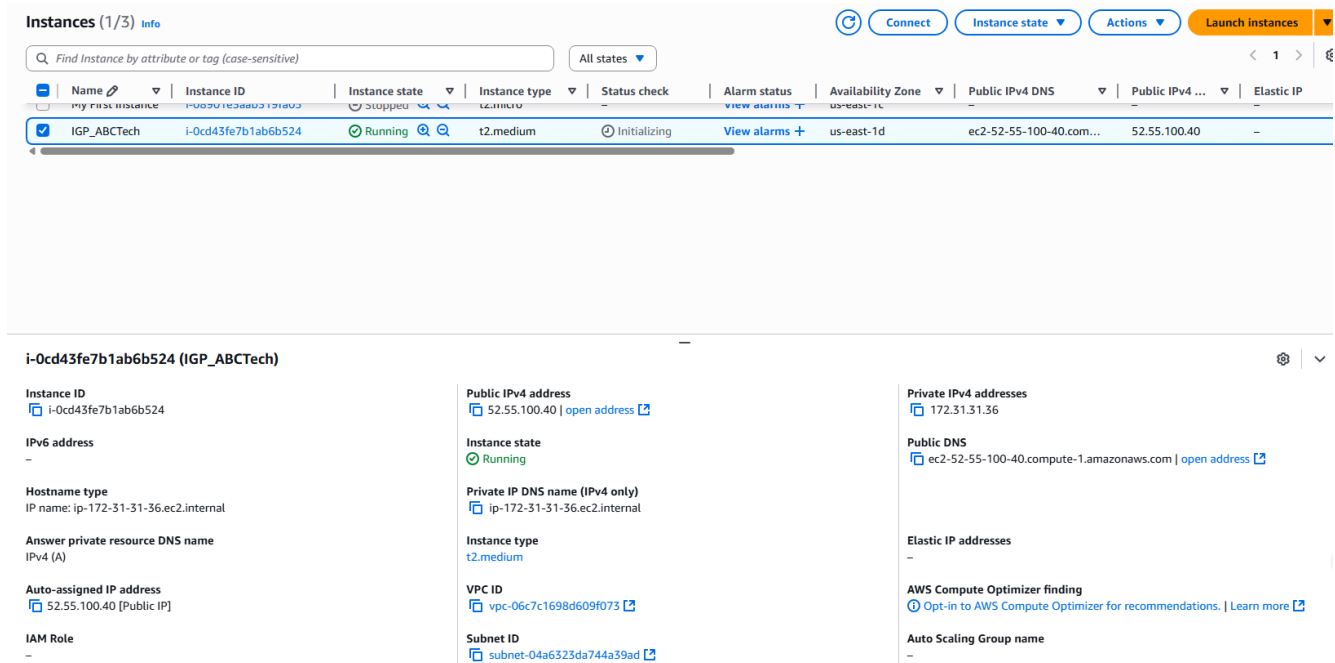
0 x File systems

Edit

► Advanced details Info

Page 14 of 45

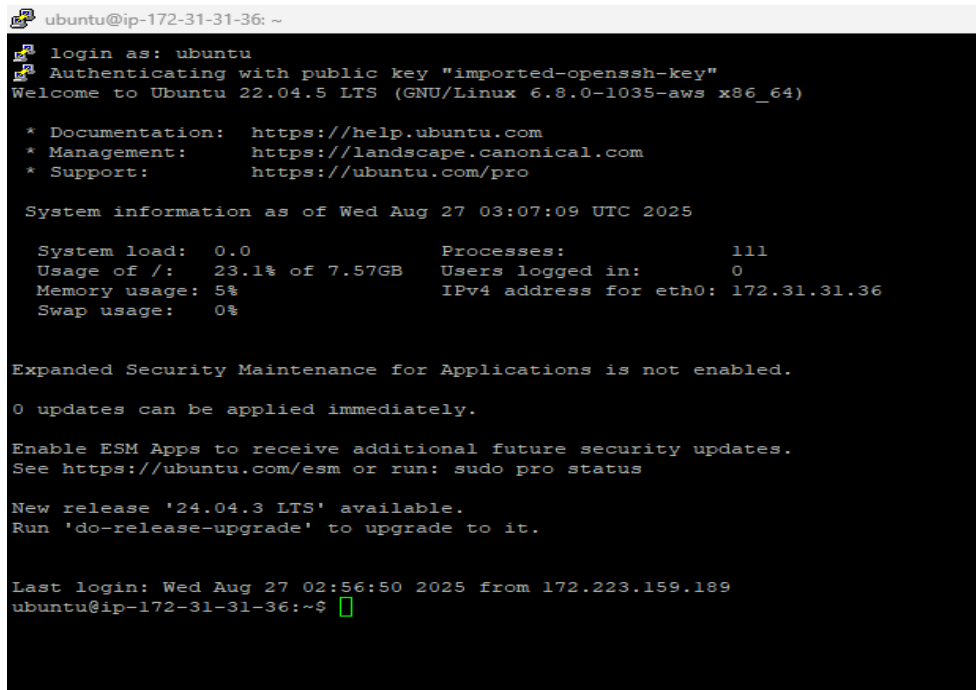
Figure 12: Launching the EC2 Instance



10.2.2 Access of the VM with PuTTY

Having set up and configured the VM, it was necessary to have access. PuTTY was installed and ensured all CLI commands were run using it. Figure Ubuntu user logged in to the VM.

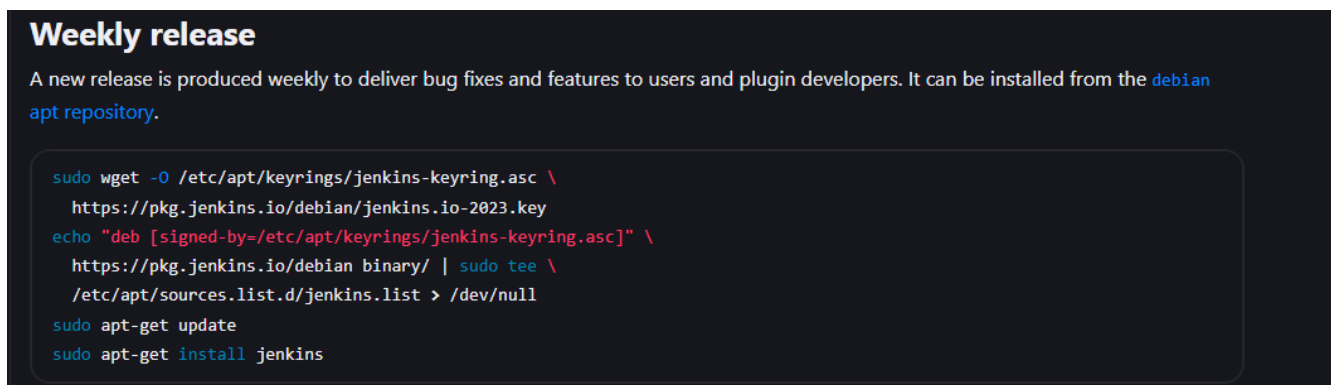
Figure 13: Access of the EC2 Instance Using PuTTY and Logged in as Ubuntu User.



10.2.3 Installation and Configuration of Jenkins

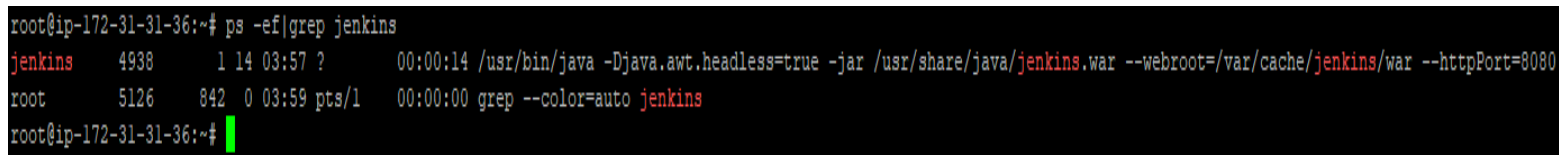
After the ability to access EC2 instance via PuTTY, the next stage involved was to install Jenkins on the VM. However, since Jenkins is written in Java language, there was prerequisite to install Java. Going to the official Jenkins installation webpage, commands were copied from this Jenkins webpage (URL: <https://www.jenkins.io/doc/book/installing/linux/>) to VM's CLI via PuTTY. Figure 14 shows Jenkins installation commands from the Jenkins website.

Figure 14: Jenkins Installation Commands from Jenkins Website



Logged as Ubuntu user, it was necessary to change to root user. To execute Jenkins installation commands, it was necessary to become a root user (>sudo su-), and commands above were used to install Jenkins. Figure 15 shows the screenshot to confirm Jenkins was successfully installed on the VM and was configured to be accessed on port 8080. It is important to note that when Jenkins is installed, Jenkins' user was also created to be able to issue command on the terminal.

Figure 15: Successful Jenkins Installation Confirmation



Since Jenkins was by default configured to run on TCP port 8080, this port had to be opened in the Security Group for inbound traffic to EC2 within AWS. Figure 16 shows the TCP port 8080 allowed for the inbound traffic enabling Jenkins to be accessed via the browser.

Figure 16: TCP Port 8080 Allowed for Inbound Traffic to Allow Jenkins to be Accessed via the Browser

Inbound rules (3)									
Search									
<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description	
<input type="checkbox"/>	-	sgr-04531f93699aa46d5	IPv4	SSH	TCP	22	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-076eb3b539a27b0b2	IPv4	HTTP	TCP	80	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-0ff16a3b860c7e83f	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-	

10.2.4 Installation of Java

Having installed and configured Jenkins, the next step involved installation of Java. Using the same Jenkins URL: <https://www.jenkins.io/doc/book/installing/linux/>, commands were copied and pasted on the CLI of PuTTY. Figure 17 shows the commands used to install Java.

Figure 17: Java Installation Commands from Jenkins Webpage.

Installation of Java

Jenkins requires Java to run, yet not all Linux distributions include Java by default. Additionally, **not all Java versions are compatible with Jenkins**.

There are multiple Java implementations which you can use. **OpenJDK** is the most popular one at the moment, we will use it in this guide.

Update the Debian apt repositories, install OpenJDK 21, and check the installation with the commands:

```
sudo apt update
sudo apt install fontconfig openjdk-21-jre
java -version
openjdk version "21.0.3" 2024-04-16
OpenJDK Runtime Environment (build 21.0.3+11-Debian-2)
OpenJDK 64-Bit Server VM (build 21.0.3+11-Debian-2, mixed mode, sharing)
```

To confirm that Java had been installed successfully, command (`> java --version`) was used.

Figure 18 shows the screenshot of command and its output. Java 21.0 was confirmed as successfully installed.

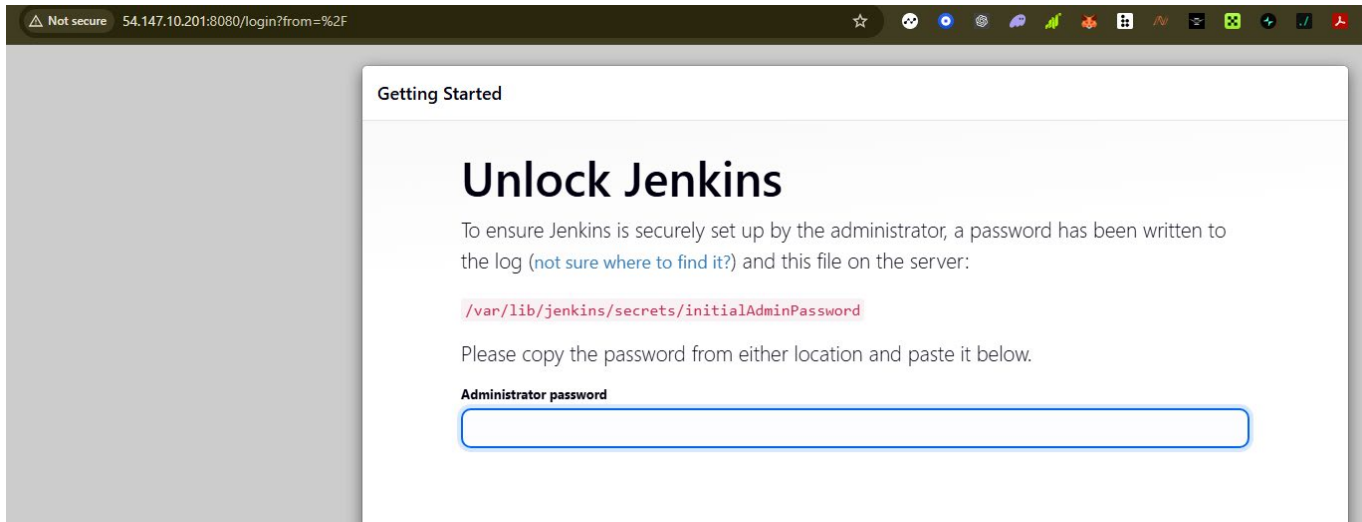
Figure 18: Java 21.0 Version Confirmed as Installed.

```
root@ip-172-31-31-36:~# java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu122.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu122.04.1, mixed mode, sharing)
root@ip-172-31-31-36:~#
```

10.2.5 Unlocking Jenkins

Having installed Jenkins and Java, the next step involved unlocking Jenkins since it would be first time to access it on the VM. The publicIP of the VM was 54.147.10.201. Jenkins was accessed on 54.147.10.201:8080. Accessing Jenkins for the first time required unlocking by creating *initialAdminPassword*. Figure 19 shows the webpage requiring unlocking Jenkins.

Figure 19: Request for initialAdminPassword to Unlock Jenkins



10.2.6 Creating initialAdminPassword

To create the initialAdminPassword the following command was used for the root user:

```
>cat /var/lib/Jenkins/secrets/initialAdminPassword
```

Execution of the above command provided the initialAdminPassword, which was copied and pasted in Jenkins. This unlocked Jenkins. Figure 20 shows the screenshot of the command and output.

Figure 20: Creation of the initialAdminPassword

```
root@ip-172-31-31-36:~# systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2025-08-27 03:57:48 UTC; 36min ago
     Main PID: 4938 (java)
       Tasks: 43 (limit: 4670)
      Memory: 525.7M
         CPU: 21.955s
    CGroup: /system.slice/jenkins.service
            └─4938 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Aug 27 03:57:45 ip-172-31-31-36 jenkins[4938]: lb5cbf57f84f41f8812217097e40cldb
Aug 27 03:57:45 ip-172-31-31-36 jenkins[4938]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Aug 27 03:57:45 ip-172-31-31-36 jenkins[4938]: *****
Aug 27 03:57:45 ip-172-31-31-36 jenkins[4938]: *****
Aug 27 03:57:45 ip-172-31-31-36 jenkins[4938]: *****
Aug 27 03:57:48 ip-172-31-31-36 jenkins[4938]: 2025-08-27 03:57:48.423+0000 [id=56] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstall
Aug 27 03:57:48 ip-172-31-31-36 jenkins[4938]: 2025-08-27 03:57:48.424+0000 [id=56] INFO hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1
Aug 27 03:57:48 ip-172-31-31-36 jenkins[4938]: 2025-08-27 03:57:48.542+0000 [id=38] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
Aug 27 03:57:48 ip-172-31-31-36 jenkins[4938]: 2025-08-27 03:57:48.559+0000 [id=30] INFO hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
Aug 27 03:57:48 ip-172-31-31-36 systemd[1]: Started Jenkins Continuous Integration Server.
root@ip-172-31-31-36:~# cat /var/lib/jenkins/secrets/initialAdminPassword
lb5cbf57f84f41f8812217097e40cldb
root@ip-172-31-31-36:~# ^C
root@ip-172-31-31-36:~#
```

10.2.7 Creating First Admin User

Having unlocked Jenkins, it was now time to create first Admin user. The credentials for the first Admin user were filled in as per Figure 21.

Figure 21: Creating first Admin user

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.516.2 [Skip and continue as admin](#) [Save and Continue](#)

10.2.8 Instance Configuration

The last step in configuring Jenkins involved replacing the external IP of the VM with internal IP. This is being done to ensure Jenkins and EC2 agents communicate securely and efficiently over AWS' private network, without exposing SSH connections to the public Internet. Figure shows the publicIP_vm:8080 (54.147.10.201:8080) replaced by internalIP_vm (172.31.31.36:8080). Figure 22 shows the replacement of Jenkins externalIP:8080 to internalIP:8080 in the configuration of the instance.

Figure 22: Replacing Jenkins URL from externalIP:8080 to internalIP:8080

Getting Started

Instance Configuration

Jenkins URL:

<http://172.31.31.36:8080/>

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.

10.2.9 Installation of Maven on the IGP_ABCTech

Having installed Java, next step involved installation of Maven on the same VM (IGP_ABCTech) where Jenkins was installed. To create a pipeline job on Jenkins, there is a prerequisite to install Maven to compile, build and test artifacts. The following commands below were executed. Figure 23 shows the history of the commands executed to have Maven installed.

```
> wget https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz
> sudo mkdir -p /opt/maven
> sudo tar -xvzf apache-maven-3.9.9-bin.tar.gz -C /opt/maven/ --strip-components=1
> sudo ln -s /opt/maven/bin/mvn /usr/bin/mvn
> mvn --version
```

Figure 23: History of Maven Installation Commands

```
14 wget https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz
15 sudo mkdir -p /opt/maven
16 sudo tar -xvzf apache-maven-3.9.9-bin.tar.gz -C /opt/maven/ --strip-components=1
17 sudo ln -s /opt/maven/bin/mvn /usr/bin/mvn
18 mvn --version
19 history
root@ip-172-31-31-36:~#
```

Having executed the Maven installation commands above, Maven version confirmation command (> mvn -version) revealed Apache Maven 3.9.9. Figure 24 shows the confirmation command. Now, with this Maven installation, Jenkins would be able to communicate with Maven that the former (Jenkins) is available.

Figure 24: Apache Maven 3.9.9 Version Confirmation

```
root@ip-172-31-31-36:~# mvn --version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
```

10.2.10 Installation of Git on the IGP_ABCTech

Having installed Maven that would allow the pipeline job to be created to compile, test and build the artifact, it was a prerequisite to install git on the *IGP_Tech VM* to access the codebase on GitHub. The following commands were used:

```
> apt install git -y
```

Figure 25 shows the git installation command and its output.

Figure 25: Git Installation Command and Output

```
root@ip-172-31-31-36:~# apt install git -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.34.1-lubuntul.15).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ip-172-31-31-36:~#
```

Having installed both Maven and Git, it was necessary to configure Jenkins so that it knows where Java, Maven and git are installed. This was done through Global Tool Configuration by accessing Managing Jenkins. Using this tool home directory for Java (JAVA_HOME), Maven (MAVEN_HOME) and git's URL: (<https://github.com/Izewdevlabs/igpaug26>) were added in Jenkins. Jenkins home directory was /var/lib/Jenkins/. Figure 26 shows the Jenkins Home directory and files that reside in this home directory.

Figure 26: Jenkins Home Directory and its Associated Files

```
root@ip-172-31-31-36:/var/lib/jenkins# ls -ltr
total 84
-rw-r--r-- 1 jenkins jenkins 64 Aug 27 03:57 secret.key
-rw-r--r-- 1 jenkins jenkins 0 Aug 27 03:57 secret.key.not-so-secret
drwxr-xr-x 2 jenkins jenkins 4096 Aug 27 03:57 jobs
-rw-r--r-- 1 jenkins jenkins 171 Aug 27 03:57 jenkins.telemetry.Correlator.xml
drwxr-xr-x 2 jenkins jenkins 4096 Aug 27 03:57 userContent
drwxr-xr-x 89 jenkins jenkins 12288 Aug 27 04:38 plugins
drwxr-xr-x 2 jenkins jenkins 4096 Aug 27 04:39 updates
-rw----- 1 jenkins jenkins 1680 Aug 27 04:39 identity.key.enc
-rw-r--r-- 1 jenkins jenkins 370 Aug 27 04:39 hudson.plugins.git.GitTool.xml
drwxr-xr-x 2 jenkins jenkins 4096 Aug 27 04:39 logs
drwxr-xr-x 3 jenkins jenkins 4096 Aug 27 04:42 users
-rw-r--r-- 1 jenkins jenkins 182 Aug 27 04:47 jenkins.model.JenkinsLocationConfiguration.xml
-rw-r--r-- 1 jenkins jenkins 7 Aug 27 04:47 jenkins.install.UpgradeWizard.state
-rw-r--r-- 1 jenkins jenkins 258 Aug 27 04:56 queue.xml.bak
-rw-r--r-- 1 jenkins jenkins 156 Aug 28 02:37 hudson.model.UpdateCenter.xml
-rw-r--r-- 1 jenkins jenkins 1037 Aug 28 02:37 nodeMonitors.xml
-rw-r--r-- 1 jenkins jenkins 7 Aug 28 02:37 jenkins.install.InstallUtil.lastExecVersion
-rw-r--r-- 1 jenkins jenkins 1660 Aug 28 02:37 config.xml
drwx----- 2 jenkins jenkins 4096 Aug 28 03:09 secrets
-rw-r--r-- 1 jenkins jenkins 1267 Aug 28 03:36 hudson.plugins.emailext.ExtendedEmailPublisher.xml
root@ip-172-31-31-36:/var/lib/jenkins#
```

10.2.11 Installation of Maven Plugins in Jenkins

Before creating any pipeline job within Jenkins, it was necessary to install plugins that could facilitate artifact's compile, test and build tasks. For instance, for Maven to work, there is a need for pom.xml file, in which all the requirements are specified by the developers. Inspection of the pom.xml file revealed maven plugins that needed to be installed. These comprised mvn JUnit and mvn jacoco plugins. In addition, other plugins not specified in the pom.xml file of the project, but useful comprised the following:

- *build pipeline plugin*
- *warning plugin*
- *stage view plugin to allow UI view of the results of the compile, test and package*

10.2.12 Creation of First CI Job- ABCTechnologies- CI

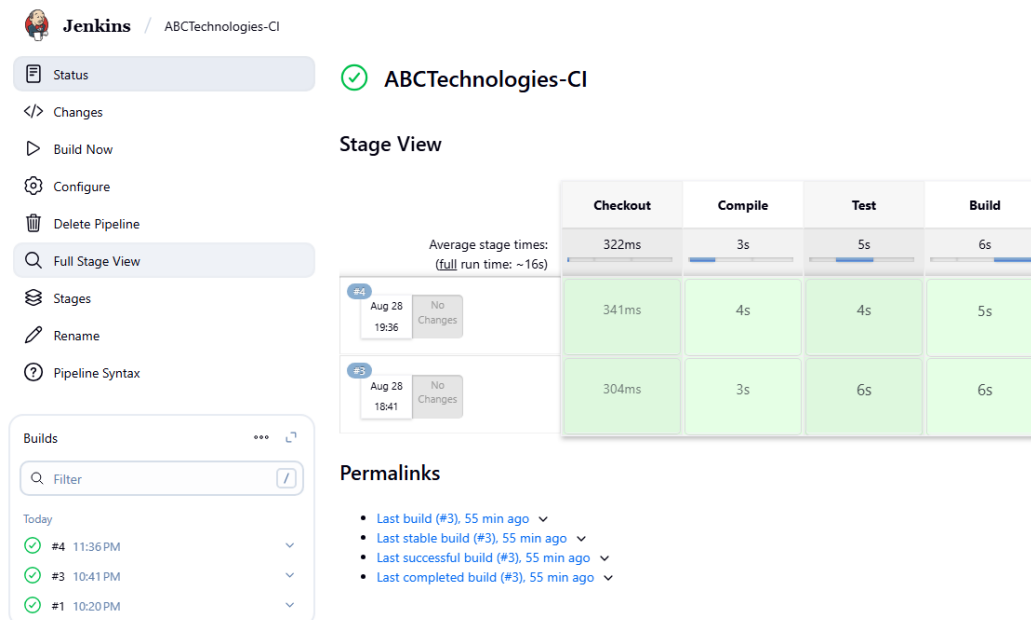
Having installed the necessary plugins, it was worth testing the continuous integration (CI) first to see if all was in good order. The first pipeline CI job involved writing the jenkinsfile written in Groovy language. The following CI script was added to the *ABCTechnologies- CI job*

```
pipeline
{
    agent any
    stages
    {
        stage('Checkout')
        {
```

```
        steps
        {
            git
            'https://github.com/Izewdevlabs/igpaug26.git'
        }
    stage('Compile')
    {
        steps
        {
            sh 'mvn compile'
        }
    }
    stage('Test')
    {
        steps
        {
            sh 'mvn test'
        }
    }
    stage('Build')
    {
        steps
        {
            sh 'mvn package'
        }
    }
}
}
```

Building the above simple CI pipeline job yielded success, evidenced by the output as per stage view within Jenkins shown in Figure 27.

Figure 27: Simple ABCTechnologies- CI Job Builds #3 & #4 Successful Results



By inspecting the build output logs in Figure 27 below , the artifact web app was downloaded in: `/var/lib/Jenkins/workspace/ABCTechnologies-CI/target/ABCtechnologies-1.0.war`

Figure 28: Build logs of the *ABCTechnologies- CI* Job

```
Stage Logs (Build)
Shell Script -- mvn package (self time 5s)
[INFO]
[INFO]
[INFO] --- war:3.2.2:war (default-war) @ ABCtechnologies ---
[INFO] Packaging webapp
[INFO] Assembling webapp [ABCtechnologies] in [/var/lib/jenkins/workspace/ABCTechnologies-CI/target/ABCtechnologies-1.0]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/ABCTechnologies-CI/src/main/webapp]
[INFO] Webapp assembled in [68 msecs]
[INFO] Building war: /var/lib/jenkins/workspace/ABCTechnologies-CI/target/ABCtechnologies-1.0.war
[INFO]
[INFO] --- jacoco:0.8.6:report (jacoco-site) @ ABCtechnologies ---
[INFO] Loading execution data file /var/lib/jenkins/workspace/ABCTechnologies-CI/target/jacoco.exec
[INFO] Analyzed bundle 'RetailModule' with 2 classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.606 s
[INFO] Finished at: 2025-08-28T23:37:14Z
[INFO] -----
```

This marked the end of the integration and building stages of the pipeline, confirming that it was working properly . The next step was to go into deployment of the artifact using Docker to Docker hub, which led to Task 3.

10.3 Deployment Using Docker Container

Task 3: *Write a Docker file Create an Image and container on docker host. Integrate docker host with Jenkins. Create CI/CD job on Jenkins to build and deploy on a container*

- *Enhance the packagejob created in step 1 of task 2 to create a docker image*
- *In the docker image add code to move the war file to tomcat server and build the image*

To be ready to create CI/CD pipeline job on Jenkins, five prerequisite steps had to be followed that comprised the following:

Step1: Installation of the docker on the same VM where Jenkins was installed

Step 2: Giving privileges to Jenkins user not run docker commands

Step 3: Store the dockerhub credentials in Jenkins global credentials that will be referred using

some

ID (mydockerhubcred)

Step 4: Installation of docker pipeline plugin on Jenkins to pass credentials ID

(mydockerhubcred)

Step 5: Writing of the dockerfile and push it to GitHub along with source code

The above steps were followed as follows:

10.3.1 Step 1: Installation of Docker on IGP_ABCTech VM

Having built the ABTechnologies-1.0.war artifact, the next step would be to create a Docker image and have it pushed onto Dockerhub. The first step involved installation of Docker (`> apt install docker.io -y`) on to the IGP_ABCTech VM where Jenkins was installed. The installation of Docker ensured that the default ownership on this file was the root user and the docker group. It was necessary to change ownership of the file from docker group to Jenkins, so that Jenkins users should be able to run Docker commands. This was done by this command:

```
> sudo chown root:jenkins /var/run/docker.sock
```

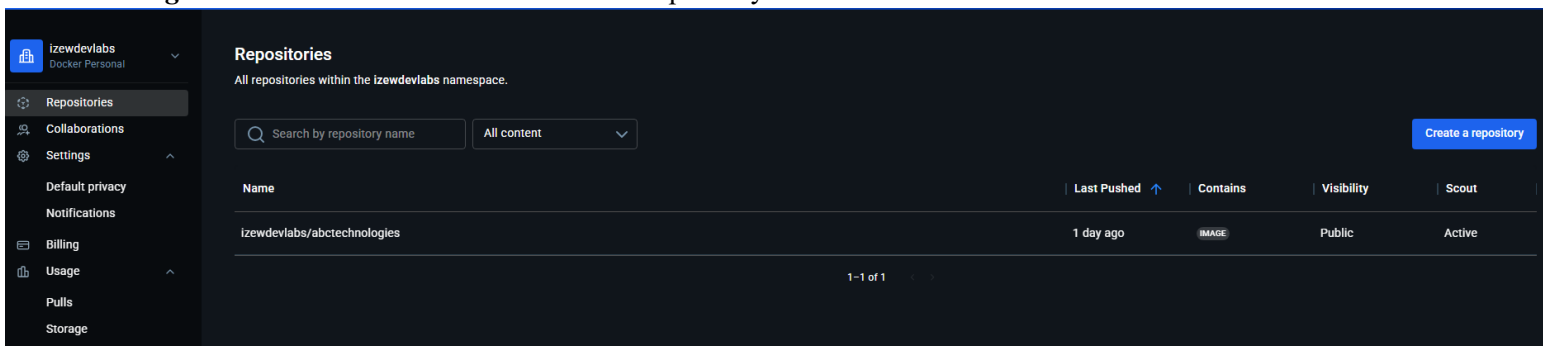
10.3.2 Step 2: Giving Privileges to Jenkins User to Run Docker Commands

To allow Jenkins user to run docker commands, it was required to store the Dockerhub of the repository (URL: <https://hub.docker.com/repositories/izewdevlabs>) credentials in Jenkins.

10.3.3 Step 3: Store the Dockerhub Credentials in Some ID (mydockerhubcred)

This was done in Global Tool Configuration within Managing Jenkins. These credentials would be referred to using an id (*mydockerhubcred*). Figure 29 shows the screenshot of the Dockerhub Account, showing one repository for the project.

Figure 29: Dockerhub Account with One Repository



Once the Dockerhub credentials had been stored in Jenkins, it would be possible for Jenkins' user to log into the Dockerhub account to push an image.

10.3.4 Step 4: Installation of Docker Pipeline Plugin to Pass Credentials

Before creating *ABCTechnologies- CI-CD* pipeline job, it was important to install docker pipeline plugin on Jenkins to pass the credentials Id (*mydockerhubcred*) to Docker. This would ensure

Jenkins' user is able to securely log into Dockerhub account without comprising security. This was done with Managing Jenkins and searching for it among the available plugins

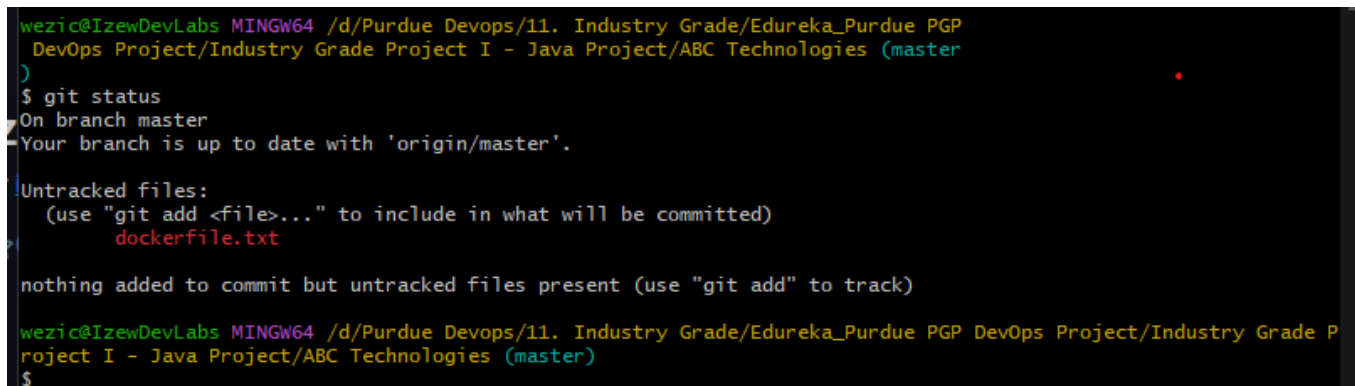
10.3.5 Step 5: Writing a Dockerfile and Pushing to GitHub Repository

To create an image and push it to dockerhub, the dockerfile needed to be written and pushed to GitHub along with the source code. Firstly, the contents of the dockerfile were as follows:

```
FROM iamdevopstrainer/tomcat:base
COPY ABCtechnologies-1.0.war /usr/local/tomcat/webapps/
CMD ["catalina.sh", "run"]
```

This docker file was first added to the local repository, allowed to be tracked and pushed to remotely to GitHub. Figure 30 shows the dockerfile added to the local repository.

Figure 30: Adding Dockerfile to Local Repository



```
wezic@IzewDevLabs MINGW64 /d/Purdue Devops/11. Industry Grade/Edureka_Purdue PGP
DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master
)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

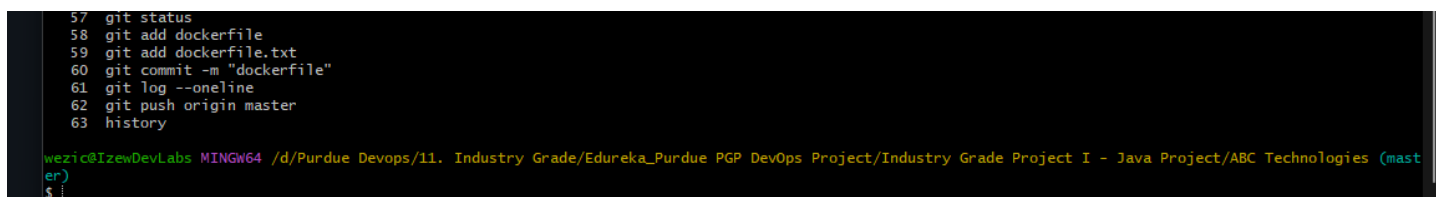
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  dockerfile.txt

nothing added to commit but untracked files present (use "git add" to track)

wezic@IzewDevLabs MINGW64 /d/Purdue Devops/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade P
roject I - Java Project/ABC Technologies (master)
$
```

Figure 31 shows the history of commands used to have the dockerfile pushed to the remote repository.

Figure 31: History of commands to have the Dockerfile Pushed to the Remote Repository

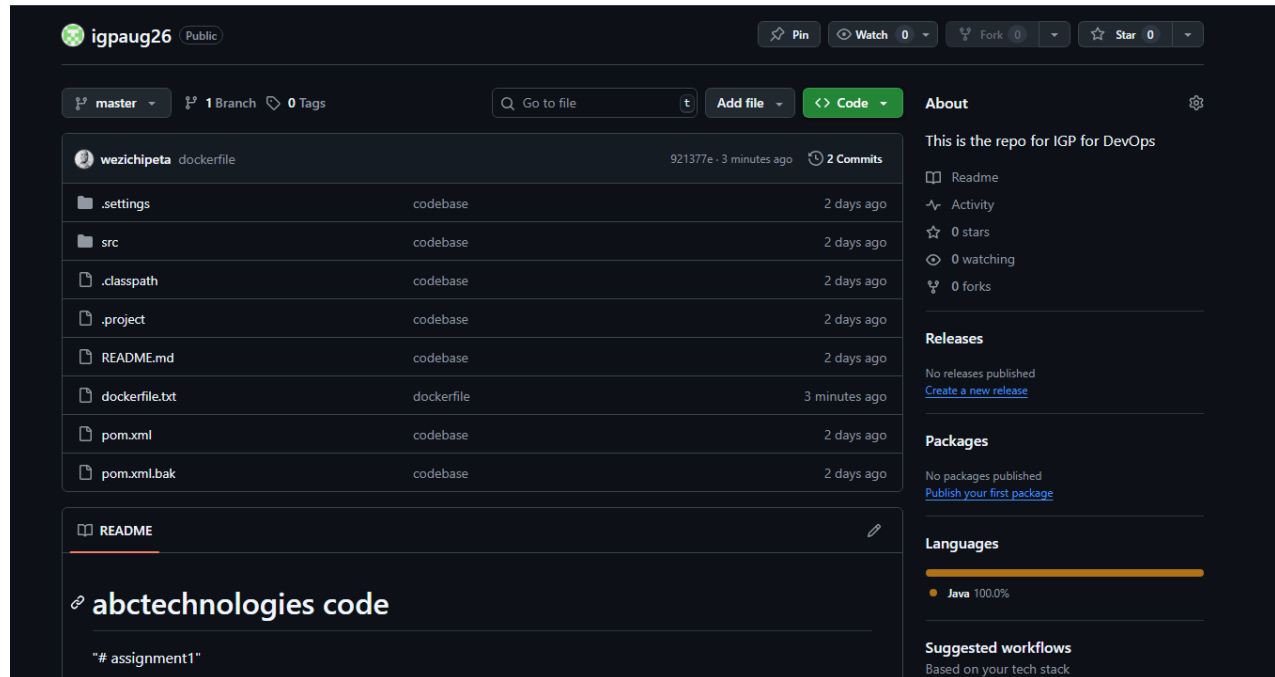


```
57 git status
58 git add dockerfile
59 git add dockerfile.txt
60 git commit -m "dockerfile"
61 git log --oneline
62 git push origin master
63 history

wezic@IzewDevLabs MINGW64 /d/Purdue Devops/11. Industry Grade/Edureka_Purdue PGP DevOps Project/Industry Grade Project I - Java Project/ABC Technologies (master)
$ |
```

Having pushed the dockerfile in the remote repository, Figure 32 shows the snapshot of the GitHub repository that includes the dockerfile.

Figure 32: History of Commands to have the Dockerfile Pushed to the Remote Repository



10.3.6 ABCTechnologies CI-CD Pipeline and Deployment to Docker

After going through the five steps above, the CI/CD pipeline job could now be created with Jenkins. The CI/CD pipeline job called “*ABCTechnologies CI-CD*” was created with the following script.

```
ABCTechnologies CI-CD Script within Jenkins
Deploy to a Docker Container
pipeline {
    agent any
    environment {
        IMAGE_LOCAL = "abctechologies"
        IMAGE_REMOTE = "izewdevlabs/abctechologies"
        TAG = "${BUILD_NUMBER}"
    }
    stages {
        stage('Checkout') { steps { git
            'https://github.com/Izewdevlabs/igpaug26.git' } }
        stage('Compile') { steps { sh 'mvn -B compile' } }
        stage('Test') { steps { sh 'mvn -B test' } }
        stage('Build WAR') { steps { sh 'mvn -B -DskipTests package' } }

        stage('Build Docker Image') {
            steps {
                sh "docker build -t ${IMAGE_LOCAL}:${TAG} ."
                sh "docker tag ${IMAGE_LOCAL}:${TAG} ${IMAGE_REMOTE}:${TAG}"
            }
        }

        stage('Push Docker Image') {
            steps {
```

```
        withCredentials([usernamePassword(credentialsId:
'mydockerhubcred',
                                usernameVariable: 'DH_USER',
                                passwordVariable: 'DH_PASS')]) {
            sh '''
            set -e
            echo "$DH_PASS" | docker login -u "$DH_USER" --password-stdin
            docker push izewdevlabs/abctechnologies:${BUILD_NUMBER}
            '''
        }
    }
}

stage('Deploy as container') {
    steps {
        sh '''
        set -e
        docker rm -f abctechnologies >/dev/null 2>&1 || true
        docker ps --filter 'publish=8080' -q | xargs -r docker rm -f ||
true

        # Optional: pull from Hub to ensure the right tag is present
        docker pull ${env.IMAGE_REMOTE}:${env.TAG} || true

        # Run the container
        docker run -d --name abctechnologies -p 18080:8080
        izewdevlabs/abctechnologies:${BUILD_NUMBER}

        docker ps --filter 'name=abctechnologies'
        '''
    }
}
}
```

The above script was the final one after some debugging because of docker failing to bind 0.0.0.0:8080. This was solved by mapping Apache Tomcat's 8080 inside to another host port 18080. This was a hard coded approach and worked. Port 18080 had to be opened to allow inbound traffic within Security Group of the VM in AWS. Figure 33 shows the screenshot of the port 18080 open.

Figure 33: Port 18080 Opened for Apache Tomcat in the VM's Security Group with AWS

Industrial Grade Project: Building a CI/CD Pipeline for ABC Technologies

sg-0bbf97848604879c7 - launch-wizard-1

Actions

Details

Security group name
launch-wizard-1

Security group ID
sg-0bbf97848604879c7

Description
launch-wizard-1 created 2025-01-08T04:31:29.643Z

VPC ID
vpc-06c7c1698d609f073

Owner
116981803550

Inbound rules count
4 Permission entries

Outbound rules count
1 Permission entry

Inbound rules

Outbound rules

Sharing - new

VPC associations - new

Tags

Inbound rules (4)

Search



Manage tags

Edit inbound rules

<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
<input type="checkbox"/>	-	sgr-04531f93699aa46d5	IPv4	SSH	TCP	22	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-05bf6ccbfed30cb81	IPv4	Custom TCP	TCP	18080	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-076eb3b539a27b0b2	IPv4	HTTP	TCP	80	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0ff16a3b860c7e83f	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

In order to confirm that the docker image was able to be pushed to the Dockerhub

(<https://hub.docker.com/repository/docker/izewdevlabs/abctechologies/general>) for each build.

Figure 33 shows docker images pushed and deployed in a container.

Figure 34: Snapshots of Docker Image Pushed Dockerhub

Repositories / abctechologies / Tags

izewdevlabs/abctechologies

Last pushed 1 day ago · Repository size: 258.6 MB

This image is part of the app project for Post Graduate Program in DevOps

Add a category

General Tags Image Management BETA Collaborators Webhooks Settings

Sort by Newest Filter tags Delete

TAG 28

Last pushed 1 day by izewdevlabs

Digest 73a79513c51f OS/ARCH linux/amd64 Vulnerabilities 7 16 14 2 0 Last pull 1 day Compressed size 258.55 MB

TAG 27

Last pushed 1 day by izewdevlabs

Digest 73a79513c51f OS/ARCH linux/amd64 Vulnerabilities 7 16 14 2 0 Last pull 1 day Compressed size 258.55 MB

Docker commands

To push a new tag to this repository:

```
docker push izewdevlabs/abctechologies:tagname
```

To pull a new tag to this repository:

```
docker pull izewdevlabs/abctechologies:28
```

To confirm that Apache Tomcat had been successfully installed and that the artifact was able to be viewed, it was possible to access the browser with the following VM's address:

PublicIP_vm:port number/artifactName -----> 13.218.96.178:18080/ABCtechnologies-1.0/. Figure

35 shows the artifact (webapp) view on the browser.

Figure 35: Webapp View on the Browser

← → ↻ ⚠ Not secure 13.218.96.178:18080/ABCtechnologies-1.0/

Welcome to ABC technologies

This is retail portal

Add Product

View Product

The above step marked the end of Task 3 and therefore the need to move to Task 4. According to the class session on 23 August 2025, Task 4 would require deployment to Kubernetes cluster (EKS/GKE) depending on public cloud being used. Integrating Docker host with Ansible, writing ansible playbook and integrating Ansible with Jenkins were optional and not emphasized. However, the deployment to Kubernetes was a must. Therefore, Task 4 would comprise this deployment of the artifact to Kubernetes.

10.4 Deployment to Kubernetes

The deployment to Kubernetes was part of Task 4 with steps which would enable deployment of a k8s EKS cluster with two worker nodes in AWS. The modified CI/CD pipeline job with associated script would ensure that the pipeline builds WAR file with Maven, packages into Docker image and pushes to Dockerhub; deploys automatically to Amazon EKS; and accessible publicly via AWS ELB.

Task 4: Deployment to Kubernetes with the following steps.

Step 1: Update Packages and Install Dependencies

Step 2: Install AWS CLI

Step 3: Install kubectl

Step 4: Install eksctl

Step 5: Configure AWS CLI on the EC2

Step 6: Create an EKS cluster with 2 worker nodes

Step 7: Writing deployment.yaml, service.yaml, ingress.yaml, namespace.yaml and hpa.yaml manifest files

Step 8: Pipeline Script with Deployment to Kubernetes

Step 9: Getting the Service ELB URL

Step 10: Testing the App

10.4.1 Step 1: Update Packages and Install Dependencies

```
>sudo apt update && sudo apt -y upgrade
>sudo apt -y install unzip curl git jq apt-transport-https ca-
certificates gnupg lsb-release
```

10.4.2 Step 2: Install AWS CLI v2

```
>curl -sSL "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
-o /tmp/awscliv2.zip
>unzip -q /tmp/awscliv2.zip -d /tmp
>sudo /tmp/aws/install
# verify
>aws --version
```

Figure 36 below shows the output of above commands:

Figure 36: Commands for installation of AWS CLI

```
root@ip-172-31-31-36:~# curl -sSL "https://awscli.amazonaws.com/awscli-exe-linux
-x86_64.zip" -o /tmp/awscliv2.zip
root@ip-172-31-31-36:~# unzip -q /tmp/awscliv2.zip -d /tmp
root@ip-172-31-31-36:~# sudo /tmp/aws/install
You can now run: /usr/local/bin/aws --version
root@ip-172-31-31-36:~# aws --version
aws-cli/2.28.21 Python/3.13.7 Linux/6.8.0-1035-aws exe/x86_64.ubuntu.22
root@ip-172-31-31-36:~#
```

10.4.3 Step 3: Install kubectl

```
>K_VER=$(curl -sL https://dl.k8s.io/release/stable.txt)
>curl -LO "https://dl.k8s.io/release/${K_VER}/bin/linux/amd64/kubectl"
>chmod +x kubectl
>sudo mv kubectl /usr/local/bin/
# verify
>kubectl version --client
```

Figure 37 below shows the output of above commands:

Figure 37: Install kubectl Command

```
root@ip-172-31-31-36:~# K_VER=$(curl -sL https://dl.k8s.io/release/stable.txt)
root@ip-172-31-31-36:~# curl -LO "https://dl.k8s.io/release/${K_VER}/bin/linux/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  138  100  138    0     0  1345      0 --:--:-- --:--:-- --:--:-- 1352
100 57.7M  100 57.7M    0     0 78.2M      0 --:--:-- --:--:-- --:--:-- 114M
root@ip-172-31-31-36:~# chmod +x kubectl
root@ip-172-31-31-36:~# sudo mv kubectl /usr/local/bin/
root@ip-172-31-31-36:~# kubectl version --client
Client Version: v1.34.0
Kustomize Version: v5.7.1
root@ip-172-31-31-36:~#
```

10.4.4 Step 4: Install eksctl

```
>curl -sSL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" \
| sudo tar -xz -C /usr/local/bin eksctl
# verify
>eksctl version
```

Figure 38 below shows the output of above commands:

Figure 38: Install eksctl Command

```
root@ip-172-31-31-36:~# curl -sSL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" \
| sudo tar -xz -C /usr/local/bin eksctl

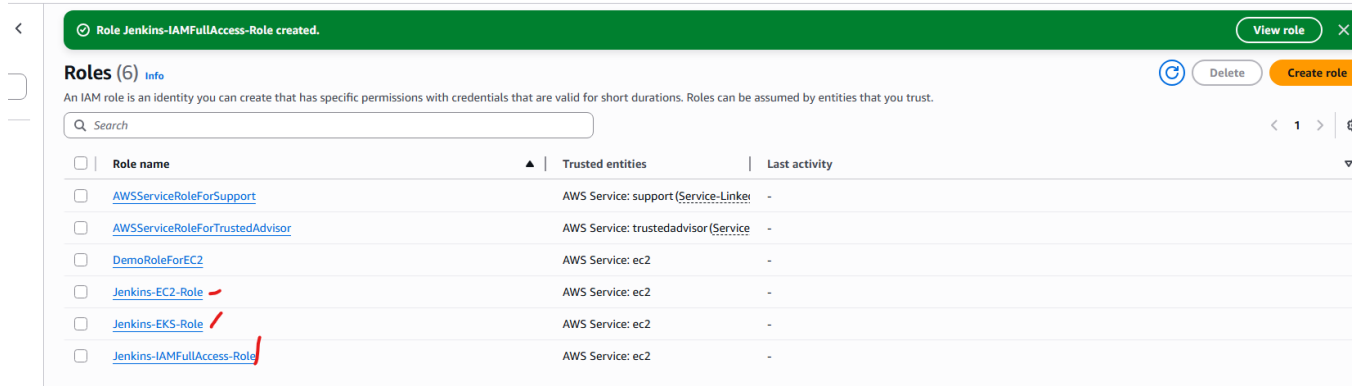
# Verify
eksctl version
0.214.0
root@ip-172-31-31-36:~#
```

10.4.5 Step 5: Configure AWS CLI on the EC2

```
>aws configure
```

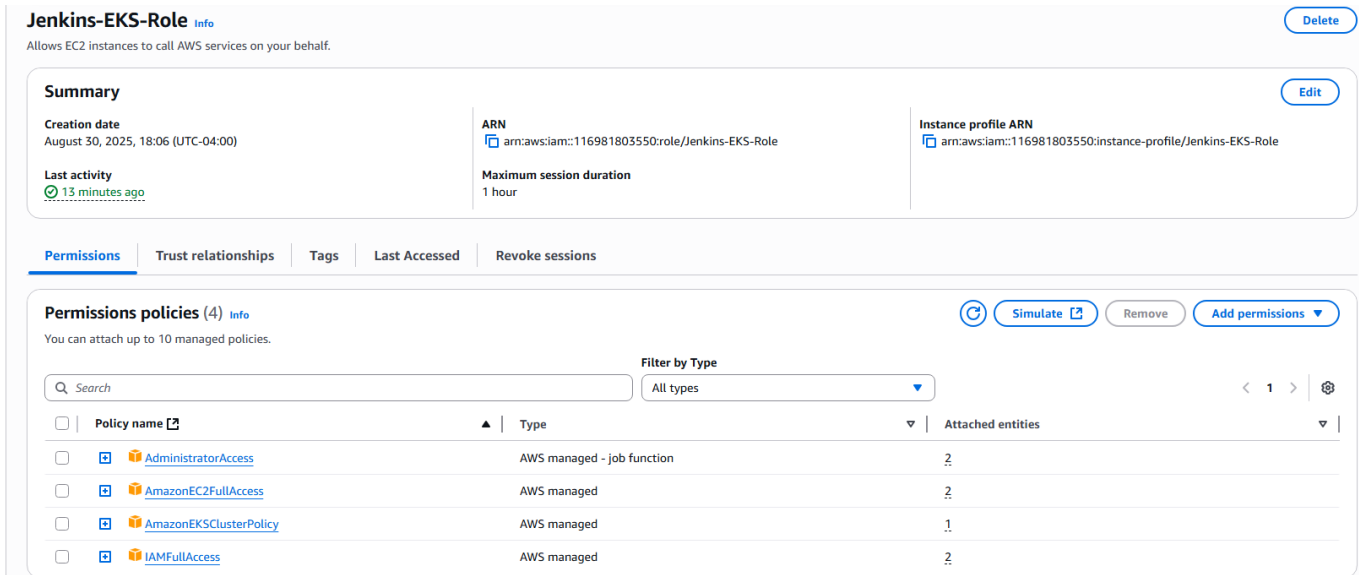
Jenkins- EKS role had to be created and attached to the EC2 instance. Figure 39 shows the Jenkins-EKS Role created that would need to be attached to the EC2 instance.

Figure 39: Created Jenkins-EKS Role



With the Jenkins-EKS Role created, permission policies had to be assigned. Figure 40 shows policies such as *AdministratorAccess*, *AmazonEC2FullAccess*, *AmazonEKSClusterPolicy* & *IAMFullAccess*.

Figure 40: Permission policies attached to Jenkins-EKS Role



Having created these permission policies, the commands below shown were able to be executed by copying information from the Jenkins-EKS Role within AWS. Figure 41 shows the screenshot of the commands and associated outputs.

Figure 41: AWS CLI Configuration on the EC2

```
root@ip-172-31-31-36:~# sed -i '/aws_access_key_id/d;/aws_secret_access_key/d' ~/.aws/credentials 2>/dev/null || true
root@ip-172-31-31-36:~# aws configure set region us-east-1
root@ip-172-31-31-36:~# aws configure set output json
root@ip-172-31-31-36:~# aws sts get-caller-identity
{
  "UserId": "AROARWPFIVIPLLT35EYA4:i-0cd43fe7b1ab6b524",
  "Account": "116981803550",
  "Arn": "arn:aws:sts::116981803550:assumed-role/Jenkins-EKS-Role/i-0cd43fe7b1ab6b524"
}
```

10.4.6 Step 6: Create an EKS Cluster with 2 Worker Nodes

To create an EKS cluster with 2 worker nodes, the following commands were used:

```
eksctl create cluster \
  --name abctech-eks \
  --region us-east-1 \
  --version 1.29 \
  --nodegroup-name ng-general \
  --node-type t3.medium \
  --nodes 2 \
  --nodes-min 2 \
  --nodes-max 4
```

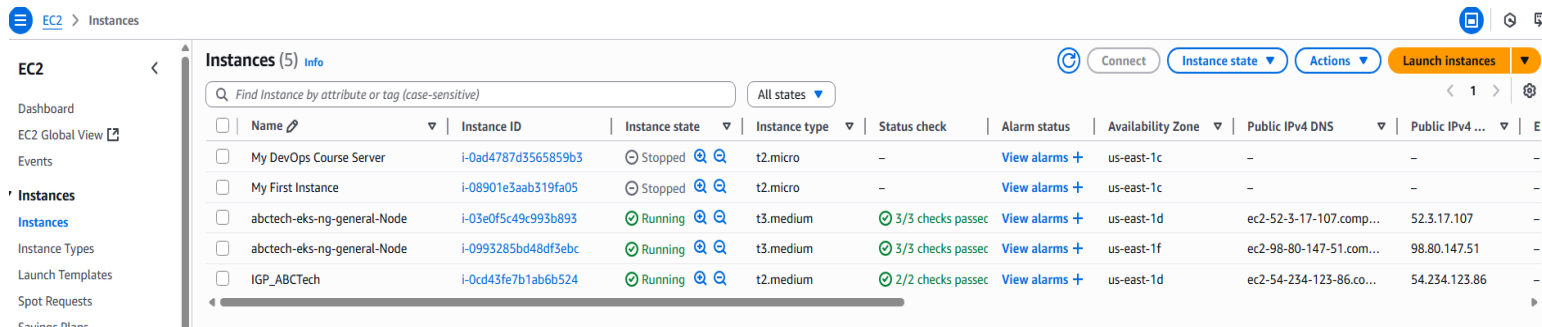
With the above commands, an EKS cluster was created with 2 worker nodes. Figure 42 shows the worker nodes ready to be scheduled for workloads.

Figure 42: AWS EKS Cluster with Nodes Ready to be Scheduled for Workloads

```
root@ip-172-31-31-36:~# aws eks update-kubeconfig --region us-east-1 --name abctech-eks
Added new context arn:aws:eks:us-east-1:116981803550:cluster/abctech-eks to /root/.kube/config
root@ip-172-31-31-36:~# kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-192-168-1-126.ec2.internal       Ready    <none>   6m2s    v1.29.15-eks-3abbec1
ip-192-168-47-181.ec2.internal      Ready    <none>   6m14s    v1.29.15-eks-3abbec1
root@ip-172-31-31-36:~#
```

Within AWS, Figure 43 shows the cluster with 2 worker nodes running. There are 2 x *abctech-eks-ng-general-Node* instances with different instance IDs and IP addresses.

Figure 43: Cluster with 2 Worker Nodes Running



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
My DevOps Course Server	i-0ad4787d3565859b3	Stopped	t2.micro	-	View alarms +	us-east-1c	-	-
My First Instance	i-08901e3aab319fa05	Stopped	t2.micro	-	View alarms +	us-east-1c	-	-
abctech-eks-ng-general-Node	i-03e0f5c49c993b893	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1d	ec2-52-3-17-107.comp...	52.3.17.107
abctech-eks-ng-general-Node	i-0993285bd48df3ebc	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1f	ec2-98-80-147-51.com...	98.80.147.51
IGP_ABCTech	i-0cd43fe7b1ab6b524	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1d	ec2-54-234-123-86.co...	54.234.123.86

10.4.7 Step 7: Writing deployment.yaml, service.yaml, ingress.yaml, namespace.yaml and hpa.yaml manifest files

Having set up the EKS cluster, the next step was to add a local Git repo and add a folder *k8s/* and include the *.yaml* files.

Firstly, the following are the contents of each *.yaml* files

K8s/namespace.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: abctech
```

K8s/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: abctechologies
  namespace: abctech
spec:
  replicas: 2
  selector:
    matchLabels:
      app: abctechologies
  template:
    metadata:
```

```
labels:
  app: abctechnologies
spec:
  containers:
    - name: web
      image: izewdevlabs/abctechnologies:latest # or use
: ${BUILD_NUMBER} in Jenkins
  ports:
    - containerPort: 8080
```

K8s/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: abctechnologies
  namespace: abctech
spec:
  type: LoadBalancer
  selector:
    app: abctechnologies
  ports:
    - port: 80
      targetPort: 8080
```

The following ***.yaml files*** were optional, that would be needed at later stage (but still included)

K8s/ingress.yaml (requires AWS Load Balancer Controller)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: abctechnologies
  namespace: abctech
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: abctechnologies
                port:
```

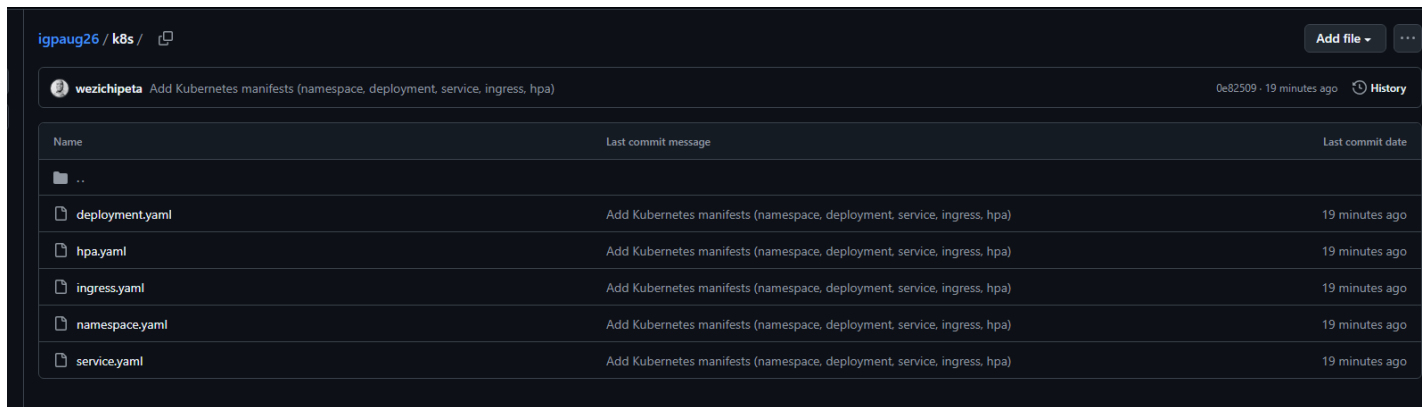
number: 80

K8s/hpa.yaml (Horizontal Pod Autoscaler)

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: abctechnologies
  namespace: abctech
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: abctechnologies
  minReplicas: 2
  maxReplicas: 6
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 60
```

Having written the manifest files, the next step involved adding the folder k8s/ and files to local repository and then pushing it remotely to GitHub. *Figure 44* shows the remote GitHub with the added folder k8s.

Figure 44: K8s folder and manifest files pushed to Github Repository “igpaug26”



Name	Last commit message	Last commit date
..		
deployment.yaml	Add Kubernetes manifests (namespace, deployment, service, ingress, hpa)	19 minutes ago
hpa.yaml	Add Kubernetes manifests (namespace, deployment, service, ingress, hpa)	19 minutes ago
ingress.yaml	Add Kubernetes manifests (namespace, deployment, service, ingress, hpa)	19 minutes ago
namespace.yaml	Add Kubernetes manifests (namespace, deployment, service, ingress, hpa)	19 minutes ago
service.yaml	Add Kubernetes manifests (namespace, deployment, service, ingress, hpa)	19 minutes ago

10.4.8 Step 8: Pipeline Script with Deployment to Kubernetes

Now that all the prerequisites' steps were carried out, the Jenkinsfile was enhanced so that it builds and pushes the Docker image through application of Kubernetes manifests to the EKS cluster. The enhanced pipeline script is shown below.

```
pipeline {
  agent any
  environment {
    IMAGE_LOCAL      = "abctechnologies"
    IMAGE_REMOTE     = "izewdevlabs/abctechnologies"
    TAG              = "${BUILD_NUMBER}"
    AWS_REGION       = "us-east-1"
    CLUSTER_NAME     = "abctech-eks"
    K8S_NS           = "abctech"
    K8S_DEPLOY       = "abctechnologies"
  }
  stages {
    stage('Checkout') { steps { git 'https://github.com/Izewdevlabs/igpaug26.git' } }
    stage('Compile') { steps { sh 'mvn -B compile' } }
    stage('Test') { steps { sh 'mvn -B test' } }
    stage('Build WAR') { steps { sh 'mvn -B -DskipTests package' } }

    stage('Build Docker Image') {
      steps {
        sh "docker build -t ${IMAGE_LOCAL}:${TAG} ."
        sh "docker tag ${IMAGE_LOCAL}:${TAG} ${IMAGE_REMOTE}:${TAG}"
      }
    }

    stage('Push Docker Image') {
      steps {
        withCredentials([usernamePassword(credentialsId: 'mydockerhubcred',
                                         usernameVariable: 'DH_USER',
                                         passwordVariable: 'DH_PASS')]) {
          sh '''
            set -e
            echo "$DH_PASS" | docker login -u "$DH_USER" --password-stdin
            docker push izewdevlabs/abctechnologies:${BUILD_NUMBER}
          '''
        }
      }
    }
  }

  stage('Deploy to EKS') {
    steps {
      sh '''
        set -e

        # point kubectl at your cluster (uses EC2 IAM role)
        aws eks update-kubeconfig --region ${AWS_REGION} --name ${CLUSTER_NAME}

        # apply base manifests (namespace/deploy/service/ingress/hpa if present)
        kubectl apply -f k8s/namespace.yaml
        kubectl apply -f k8s/deployment.yaml
        kubectl apply -f k8s/service.yaml || true
        test -f k8s/ingress.yaml && kubectl apply -f k8s/ingress.yaml || true
        test -f k8s/hpa.yaml && kubectl apply -f k8s/hpa.yaml || true

        # roll the deployment to the exact image tag of this build
        kubectl -n ${K8S_NS} set image deploy/${K8S_DEPLOY}
        web=${IMAGE_REMOTE}:${TAG}

        # wait for rollout
        kubectl -n ${K8S_NS} rollout status deploy/${K8S_DEPLOY} --timeout=180s

        # print service/ingress info
        kubectl -n ${K8S_NS} get pods,svc,ingress
      '''
    }
  }
}
```

```
post {
    always {
        sh '''
            set +e
            echo "=== Docker disk usage BEFORE ==="
            docker system df || true

            docker container prune -f || true
            docker image prune -f || true
            docker builder prune -f || true

            echo "=== Docker disk usage AFTER ==="
            docker system df || true
        '''
    }
}
```

According to this CI/CD pipeline code, Maven built the WAR; Docker built and pushed the image to Dockerhub; Jenkins connected to the EKS cluster; Kubernetes deployed the app via manifests in k8s/; rollout finished with pods in ready to use state. Post stage was added to save resources after each build. To confirm success of the CI/CD pipeline, Figure 45 shows the stage view of the 3 builds in the Jenkins dashboard.

Figure 45: Stage View of the 3 Builds in the Jenkins Dashboard of the ABCTechnologies-CI-CD Job



10.4.9 Step 9: Getting the Service ELB URL

Having deployed the app successfully, it was required to get the service ELB URL. There were two ways to get it. One way was within Jenkins' dashboard. Figure 46 shows the following information for the service.

Industrial Grade Project: Building a CI/CD Pipeline for ABC Technologies

- Name: service/abctechologies
- Type: LoadBalancer
- Cluster- IP: 100.100.79.130
- External-IP: <http://afaa13523188c414c8f97bd9eaaaa8c2-1297544135.us-east-1.elb.amazonaws.com/ABCtechnologies-1.0/>
- Port(s): 80:31103/TCP

Figure 46: Service ELB URL View in Jenkins

NAME	READY	STATUS	RESTARTS	AGE
pod/abctechologies-859458fd66-nhxp6	1/1	Running	0	2s
pod/abctechologies-859458fd66-s9t55	1/1	Running	0	3s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/abctechologies	LoadBalancer	10.100.79.130	afaa13523188c414c8f97bd9eaaaa8c2-1297544135.us-east-1.elb.amazonaws.com	80:31103/TCP	2m26s

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
ingress.networking.k8s.io/abctechologies	<none>	*		80	2m25s

Another way to get the service ELB URL was through the terminal by running the command. Figure 47 shows the output of running this command and displays the same service URL information as the one acquired within Jenkins.

```
>kubectl -n abctech get svc abctechologies
```

Figure 47: Service ELB URL Output Using the Terminal

```
root@ip-172-31-31-36:~# kubectl -n abctech get svc abctechologies
NAME                TYPE                CLUSTER-IP    EXTERNAL-IP                                PORT(S)          AGE
abctechologies      LoadBalancer        10.100.79.130  afaa13523188c414c8f97bd9eaaaa8c2-1297544135.us-east-1.elb.amazonaws.com  80:31103/TCP     12m
root@ip-172-31-31-36:~#
```

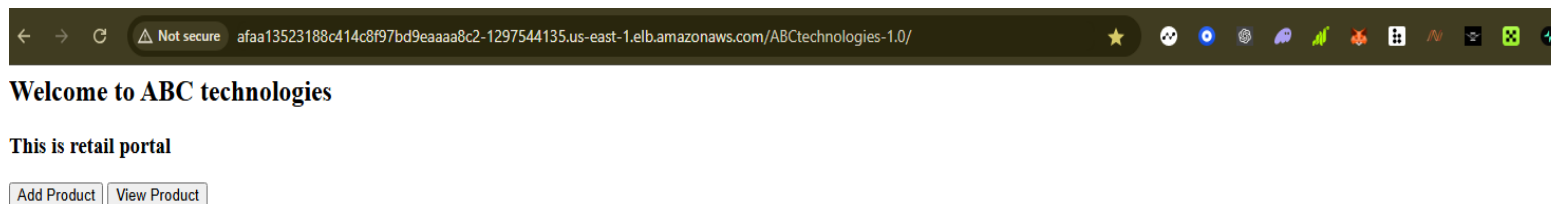
To have app access to the browser, the following URL was used.

<http://<ELB>/ABCtechnologies-1.0/>. Replacing the <ELB> with ‘afaa13523188c414c8f97bd9eaaaa8c2-1297544135.us-east-1.elb.amazonaws.com’, the app could be accessed on this ELB URL via browser:

<http://afaa13523188c414c8f97bd9eaaaa8c2-1297544135.us-east-1.elb.amazonaws.com/ABCtechnologies-1.0/>

Figure 48 shows the web applications landing page.

Figure 48: ABCtechnologies Web app Landing Page (accessed via AWS ELB)



The above activity/ step marked the end of Task 4 and ready to move to Task 5 involving the use of Prometheus to monitor resources. The section of Task 5 is covered next.

10.5 Monitoring Using Prometheus and Grafana

Task 5: Using Prometheus monitors resources like CPU utilization: Total Usage, Usage per core, usage breakdown, Memory , Network on the instance by providing the end points in local host. Install node exporter and add URL to targeting Prometheus. Using this data login to Grafana and create a dashboard to show the metrics.

10.5.1 Installation of the Stack

The fastest way to add monitoring was to install Prometheus and Grafana via Helm. This option would give the following in one installation a stack that would give all this in one installation.

- *Prometheus (metrics DB)*
- *Alertmanager (alerts)*
- *Grafana (dashboards)*
- *kube-state-metrics & node-exporter (cluster/node metrics)*

The following commands were run on Jenkins EC2 (kubectl already pointing to your cluster):

```
# Namespace for monitoring
kubectl create namespace monitoring || true

# Add Helm repo
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update

# Install/upgrade the stack and expose Grafana via LoadBalancer on port 80
helm upgrade --install kps prometheus-community/kube-prometheus-stack \
  -n monitoring \
  --set grafana.service.type=LoadBalancer \
  --set grafana.service.port=80 \
  --set grafana.defaultDashboardsEnabled=true
```

Figure 49 shows the history of commands executed on the Jenkins machine and output that ensured the whole stack was installed.

Figure 49:History of Commands Executed to Install the Stack


```
85 kubectl create namespace monitoring || true
86 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
87 snap install helm
88 helm repo update
89 # Download the install script
90 curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
91 helm version
92 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
93 helm repo update
94 helm upgrade --install kps prometheus-community/kube-prometheus-stack -n monitoring --set grafana.service.type=LoadBalancer --set grafana.service.port=80 --set grafana.defaultDashboardsEnabled=true
95 kubectl -n monitoring get pods
96 kubectl -n monitoring get svc kps-grafana
97 GRAFANA=$(kubectl -n monitoring get svc kps-grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'); echo "http://$GRAFANA/"
98 echo "user: admin"
99 kubectl -n monitoring get secret kps-grafana -o jsonpath='{.data.admin-password}' | base64 -d; echo
100 http://aee84570ca78b4flab594886e98e2773-1735423370.us-east-1.elb.amazonaws.com/
101 kubectl -n monitoring get svc kps-grafana
```

To verify if the stack was installed successfully, the following commands were executed:

```
kubectl -n monitoring get pods
kubectl -n monitoring get svc kps-grafana
```

Figure 50 shows the command and output. It could be verified that the monitoring stack was installed and running. All Prometheus, Alertmanager, Grafana, node-exporter, kube-state-metrics pods were up

Figure 50:Verification the Monitoring was Installed and Running

```
root@ip-172-31-31-36:~# kubectl -n monitoring get pods
NAME                                READY   STATUS    RESTARTS   AGE
alertmanager-kps-kube-prometheus-stack-alertmanager-0  2/2     Running   0           25s
kps-grafana-c97d9b89-vx4wb          2/3     Running   0           32s
kps-kube-prometheus-stack-operator-75465c9c4d-hpvn4    1/1     Running   0           32s
kps-kube-state-metrics-db76ddb-krxwf 1/1     Running   0           33s
kps-prometheus-node-exporter-mftkc  1/1     Running   0           33s
kps-prometheus-node-exporter-shfj7  1/1     Running   0           33s
prometheus-kps-kube-prometheus-stack-prometheus-0     1/2     Running   0           25s
root@ip-172-31-31-36:~# kubectl -n monitoring get svc kps-grafana
NAME      TYPE          CLUSTER-IP      EXTERNAL-IP                                PORT(S)      AGE
kps-grafana  LoadBalancer  10.100.206.65    ae8e4570ca78b4flab594886e98e2773-1735423370.us-east-1.elb.amazonaws.com  80:31558/TCP  58s
http://ae8e4570ca78b4flab594886e98e2773-1735423370.us-east-1.elb.amazonaws.com/
root@ip-172-31-31-36:~# echo "user: admin"
user: admin
root@ip-172-31-31-36:~# kubectl -n monitoring get secret kps-grafana -o jsonpath='{.data.admin-password}' | base64 -d; echo
prom-operator
root@ip-172-31-31-36:~# history
```

10.5.2 Getting Grafana URL + Admin Password

To be able to access Grafana, password for the user admin was required. This was obtained by running the following commands:

```
# ELB URL (appears in ~1-3 mins)
GRAFANA=$(kubectl -n monitoring get svc kps-grafana -o
  jsonpath='{.status.loadBalancer.ingress[0].hostname}'); echo
"http://$GRAFANA/"

# Credentials
echo "user: admin"
kubectl -n monitoring get secret kps-grafana -o jsonpath='{.data.admin-
  password}' | base64 -d; echo
```

Figure 51 shows the credentials retrieved via the above commands. The following were the credentials:

```
Grafana admin user: admin
Grafana password:prom-operator
```

Figure 51:Getting Grafana Password

```
root@ip-172-31-31-36:~# echo "user: admin"
user: admin
root@ip-172-31-31-36:~# kubectl -n monitoring get secret kps-grafana -o jsonpath='{.data.admin-password}' | base64 -d; echo
prom-operator
root@ip-172-31-31-36:~#
```

To get the Grafana URL that would give an AWS ELB, the following command was executed:

```
kubectl -n monitoring get svc kps-grafana
```

Output of this command provided the following as shown in Figure 52:

```
Name: kps-grafana
Type: LoadBalancer
Cluster-IP: 10.100.206.65
External IP: ae8e4570ca78b4flab594886e98e2773-1735423370.us-east-
1.elb.amazonaws.com
```

Ports: 80:31556/TCP

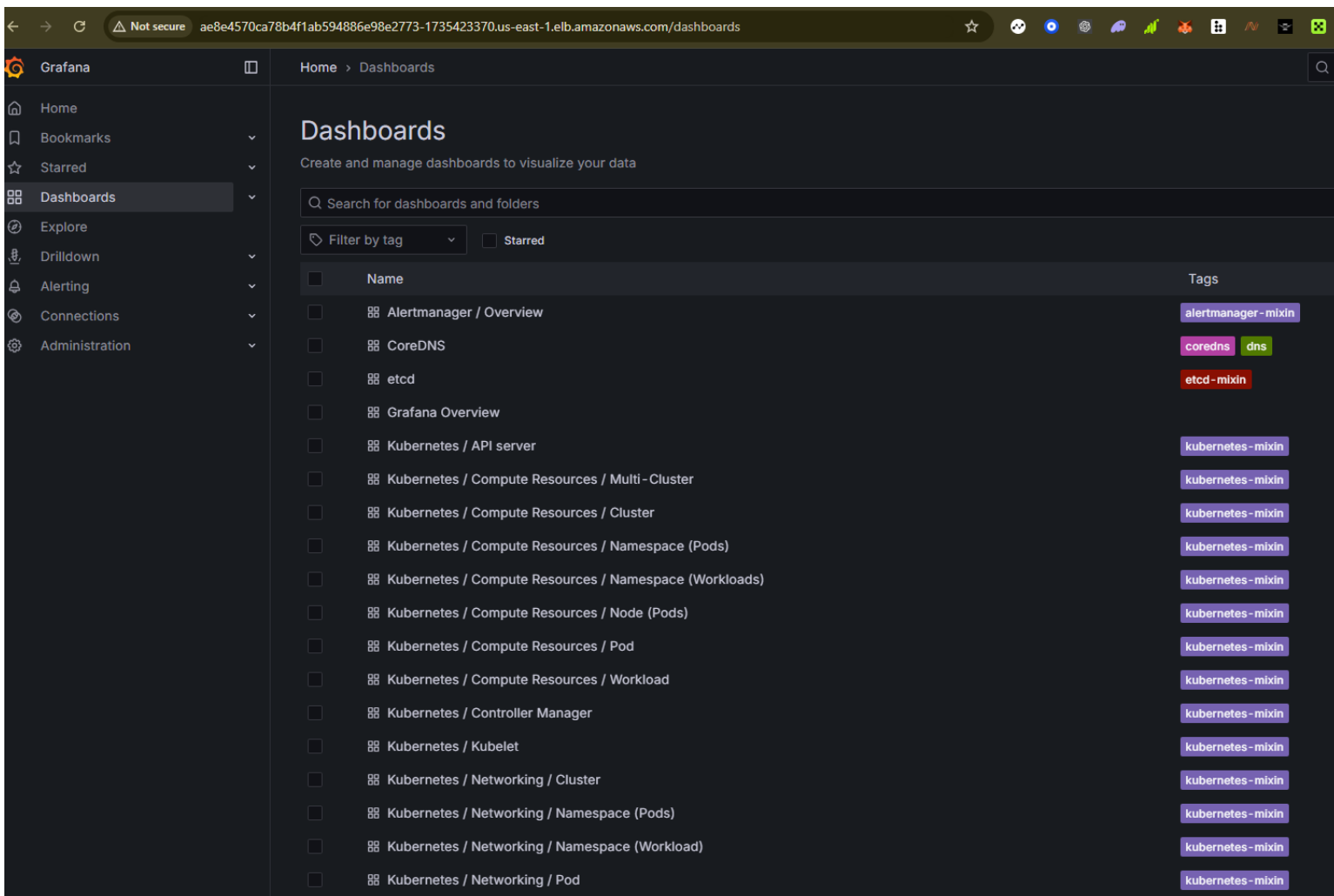
Figure 52:ELB URL for Grafana

```
root@ip-172-31-31-36:~# kubectl -n monitoring get svc kps-grafana
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kps-grafana   LoadBalancer 10.100.206.65  ae8e4570ca78b4flab594886e98e2773-1735423370.us-east-1.elb.amazonaws.com 80:31558/TCP 42h
root@ip-172-31-31-36:~# ^C
root@ip-172-31-31-36:~# █
```

This meant that Grafana was now exposed via an AWS LoadBalancer and could be accessed on the browser. Opening <http://ae8e4570ca78b4flab594886e98e2773-1735423370.us-east-1.elb.amazonaws.com> on the browser gave access to Grafana.

Inside Grafana, one was able to browse through the dashboards and a set of Kubernetes / Compute Resources dashboards (Pods, Nodes, Deployments, etc.) from the chart. Figure 53 shows the screenshot of different Kubernetes/ Compute dashboards.

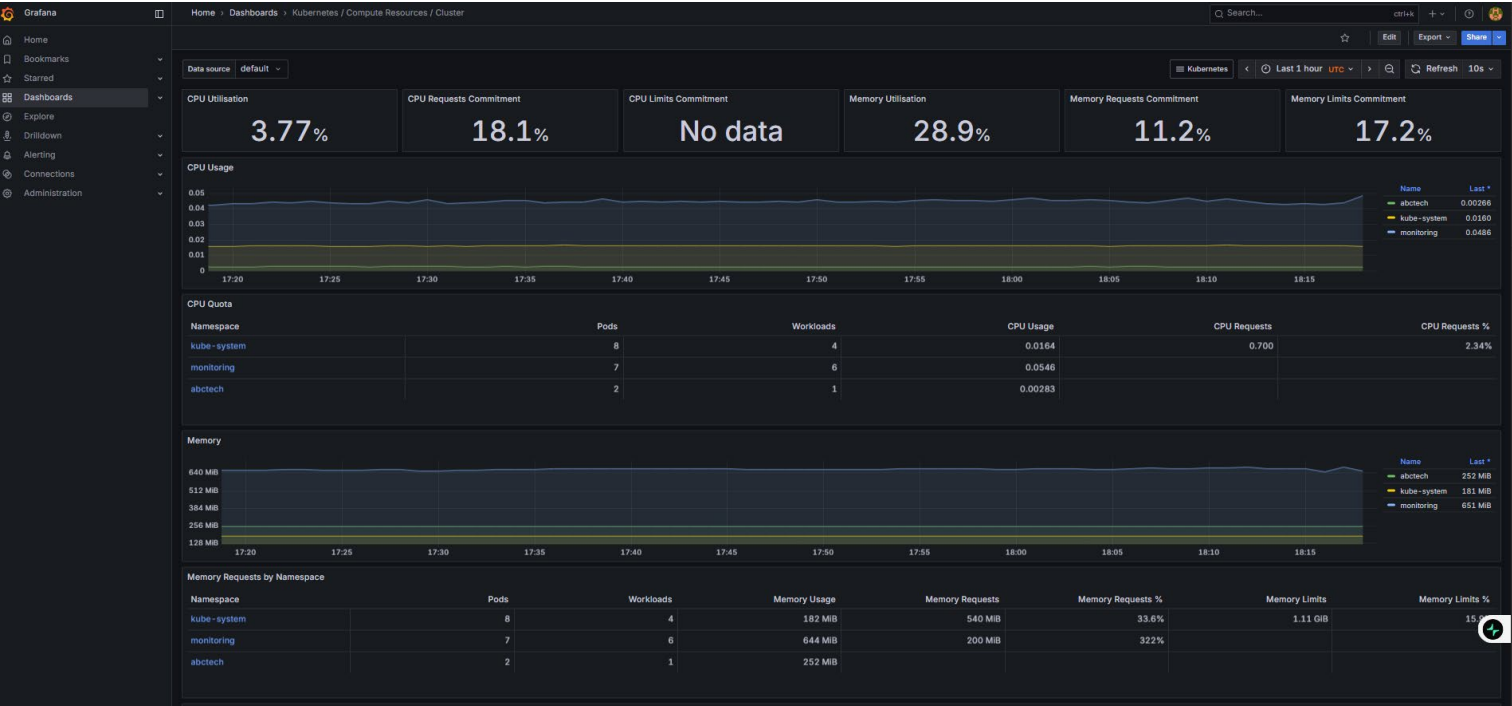
Figure 53: Different Kubernetes/ Compute dashboards within Grafana



10.5.3 Dashboards Monitored

One of the dashboards explored was for the whole cluster, which included CPU utilization, CPU usage, CPU quota and memory. This was done by following: **Kubernetes / Compute Resources / Cluster → CPU/mem of the whole cluster** . Figure 53 shows the screenshot of the dashboard.

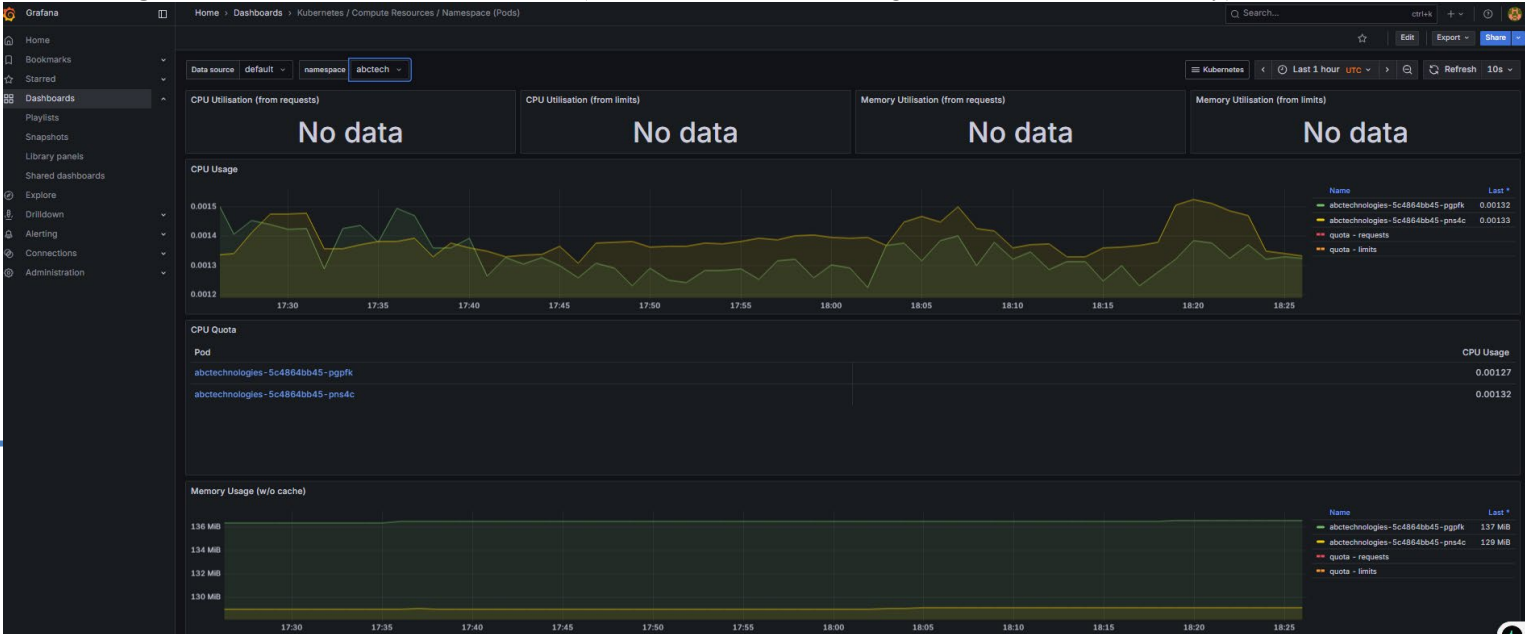
Figure 54: Cluster Dashboard (CPU utilization, CPU usage, CPU Quota and Memory)



The second dashboard was monitoring per pod metrics in a cluster, shown in Figure 55. This was done by following :

Kubernetes / Compute Resources / Pod → Per-pod metrics (including abctechnologies pods).

Figure 55: Per Pod Metrics Dashboard (CPU utilization, CPU usage, CPU Quota and Memory)



11. Summary

This document provided a detailed solution using DevOps where five tasks and steps to accomplish them were outlined. Complete code developed such as dockerfile, CI pipeline job script, CI/CD pipeline job scripts for both docker and Kubernetes deployments were provided. In addition, code for manifest files deployment.yaml, service.yaml, namespace.yaml, ingress.yaml and hpa.yaml were provided. All in all, snapshots of the command and outputs either on the terminal or within Github, Dockerhub, Jenkins, Docker, Grafana and AWS were provided. This DevOps solution for ABC Technologies would ensure that operational benefits below would accrue.

- Highly available
- Highly scalable
- Highly Performant
- Easily built and maintained
- Developed and deployed quickly
- Lower production bugs
- Frequent releases
- Better customer experiences
- Lesser time to market