

Project Report
on
Production Ready DevOps Environment



Submitted in partial fulfillment for the award of
Post Graduate Diploma in High Performance Computing
System Administration from **C-DAC ACTS (Pune)**

Guided by :
Mr. Ashutosh Das

Presented by:

Aboli Waikos	PRN: 230340127001
Sanika Bhanage	PRN: 230340127006
Siddhesh Rupesh Kachkure	PRN: 230340127010
Vikas Rai	PRN: 230340127013
Indrajeet Kumar	PRN: 230340127036

Centre of Development of Advanced Computing(C-DAC), Pune



CERTIFICATE

TO WHOMS IT MAY CONCERN

Aboli Waikos

Sanika Bhanage

Siddhesh Rupesh Kachkure

Vikas Rai

Indrajeet Kumar

This is to certify that they have successfully completed their project on

Production Ready DevOps Environment

**Under the Guidance of
Mr.Ashutosh Das**

Project Guide

Project Supervisor

**HOD ACTS
Mr.Aditya Sinha**



ACKNOWLEDGEMENT

This project “Production Ready DevOps Environment” was a great learning experience for us. And we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of **Mr.Ashutosh Das** for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to Mr. Kaushal Sharma (Manager ACTS training Centre), CDAC, for his guidance and support whenever necessary while doing this course Post Graduate Diploma in **High Performance Computing System Administration (PG-DHPCSA)** through C-DAC ACTS, Pune.

Our most heartfelt thank goes to **Ms. Swati Salunkhe** (Course Coordinator, PG-DHPCSA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

From:

Aboli Waikos	PRN: 230340127001
Sanika Bhanage	PRN: 230340127006
Siddhesh Rupesh Kachkure	PRN: 230340127010
Vikas Rai	PRN: 230340127013
Indrajeet Kumar	PRN: 230340127036

TABLE OF CONTENTS

1. Abstract
2. Introduction:
 - 2.1. Background and Motivation
 - 2.2. Objectives of the Project
 - 2.3. Scope of the Project
 - 2.4. Research Methodology
3. Literature Review:
 - 3.1. DevOps Principles and Benefits
 - 3.2. Continuous Integration and Continuous Deployment (CI/CD)
 - 3.3. Introduction to Git, Jenkins, Maven, Tomcat, Docker, Ansible, Prometheus, Grafana and Kubernetes
4. System Requirements:
 - 4.1 User Interface
 - 4.2 Software Requirement
5. Design and Architecture:
 - 5.1. Overview of the CI/CD Pipeline Architecture
 - 5.2. Role of Each DevOps Tool in the Pipeline
6. Implementation Steps:
 - 6.1. Setting Up Git Repository and Branching Strategy
 - 6.2. Give Configuring Jenkins for Automated Builds and Tests
 - 6.3. Deploying Applications on Tomcat Server
 - 6.4. Containerization with Docker
 - 6.5. Configuration Management using Ansible
 - 6.6. Monitoring and Visualization with Grafana
7. Results and Findings:
8. Conclusion:
 - 8.1. Achievements of the Project
 - 8.2. Benefits of the Implemented DevOps Environment
 - 8.3. Future Potential and Expansion of the Environment
9. Recommendations:
 - 9.1. Enhancing Security in the Pipeline
 - 9.2. Extending Automated Testing Coverage
 - 9.3. Backup and Recovery Strategies for Kubernetes
 - 9.4. Documentation and Knowledge Sharing
10. References:

1. Abstract

In the rapidly evolving domain of software engineering, the adoption of DevOps practices has become integral to efficient development, high-quality software delivery, and rapid deployment. This project presents a comprehensive implementation of a Production Ready DevOps Environment, strategically designed to harmonize a range of tools and methodologies. Focusing on Continuous Integration/Continuous Deployment (CI/CD) pipelines, the project exemplifies collaboration, automation, and scalability inherent in DevOps principles.

The report delves into the strategic integration of key DevOps tools, including Git, Jenkins, Maven, Tomcat, Docker, Ansible, Kubernetes, and Grafana. By dissecting each tool's role, the project demonstrates their collective impact on the software development lifecycle, spanning coding, deployment, and monitoring. This holistic approach fosters a harmonious interaction between development and operations, fostering iterative progress and continuous refinement.

The project's architecture not only champions automation but also addresses common challenges faced by both development and operations teams. Docker's containerization principles ensure consistent environments, mitigating the disparity between development and production environments. Kubernetes' orchestration capabilities enable dynamic management of containerized applications, supporting scalability, load distribution, and fault tolerance.

Furthermore, Ansible, a configuration management tool, streamlines environment setup through automation, ensuring reproducibility and consistency. Real-time insights into system performance and application health are facilitated through Grafana's integration, empowering data-driven decision-making.

To conclude, this project encapsulates modern DevOps practices through a synergistic implementation. Through automation, containerization, orchestration, and real-time monitoring, the executed Production Ready DevOps Environment exemplifies the symbiotic relationship between advanced tools and the evolving landscape of software development. The project's outcomes contribute to the ongoing discourse on DevOps excellence, offering a tangible framework for organizations to adapt and optimize as they pursue streamlined software delivery.

2. Introduction

2.1. Background and Motivation

In the landscape of software development, challenges often arise in terms of agility, collaboration, and efficiency. Traditional practices have struggled to keep up with the demands of rapid deployment and seamless integration between development and operations. This has led to the emergence of DevOps principles, aimed at bridging the gap and fostering a culture of continuous integration and continuous deployment.

2.2. Objectives of the Project

The primary goal of this project is to establish a robust and efficient DevOps environment by leveraging a suite of essential tools. By integrating Git, Jenkins, Maven, Tomcat, Docker, Ansible, Kubernetes, and Grafana, we aim to achieve automation, consistency, scalability, and real-time monitoring within the software development lifecycle.

2.3. Scope of the Project

This project encompasses the design and implementation of a comprehensive DevOps ecosystem. The scope includes creating a sophisticated CI/CD pipeline, employing containerization with Docker, orchestrating deployments using Kubernetes, and incorporating real-time monitoring through Grafana. Furthermore, the project covers automated testing, configuration management, and the deployment of applications.

2.4. Research Methodology

The research methodology employed for this project is a blend of thorough literature review, hands-on experimentation, and industry best practices. By referring to official documentation, online tutorials, and direct practical implementations of tools like Jenkins, Ansible, and Kubernetes, we ensure a well-informed and effective approach.

3. Literature Review

3.1. DevOps Principles and Benefits

DevOps embodies a set of principles that bridge the gap between software development and IT operations. This approach fosters collaboration, communication, and automation, aiming to deliver high-quality software more rapidly and efficiently. The core principles of DevOps include:

3.2. Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration (CI) is the practice of frequently integrating code changes into a shared repository. Automated tests are run to verify the correctness of these changes, allowing for early detection of issues. Continuous Deployment (CD) takes this a step further by automatically deploying code changes to production environments after passing automated tests.

3.3. Introduction to Git, Jenkins, Maven, Tomcat, Docker, Ansible, Prometheus, Grafana and Kubernetes

- **Git:** A distributed version control system that tracks code changes, enabling collaboration and version management among developers.

- **Jenkins:** An automation server that orchestrates the CI/CD pipeline, automating builds, tests, and deployments.

- **Maven:** A build automation tool that manages dependencies and compiles Java projects, ensuring consistent and repeatable builds.

- **Tomcat:** A servlet container for deploying Java web applications, providing a runtime environment for Java-based web services.

- **Docker:** A platform for containerization that packages applications and their dependencies into isolated containers for consistent deployment.

- **Ansible:** An open-source automation tool for configuring and managing infrastructure, ensuring uniformity across environments.

- **Kubernetes:** An open-source container orchestration platform that automates deployment, scaling, and management of containerized applications.

- **Prometheus:** An open-source monitoring system with a dimensional data model, flexible query language, efficient time series database and modern alerting approach.

- **Grafana:** A monitoring and visualization tool that creates interactive dashboards to monitor application and system metrics.

4. System Requirements

4.1 User Interface

1.EC2 Instance Amazon Linux

4.2 Software Requirement

1. GIT
2. Jenkins
3. Docker
4. Ansible
5. Prometheus
6. Grafana
7. Apache Maven
8. Tomcat

5. Design and Architecture

5.1. Overview of the CI/CD Pipeline Architecture

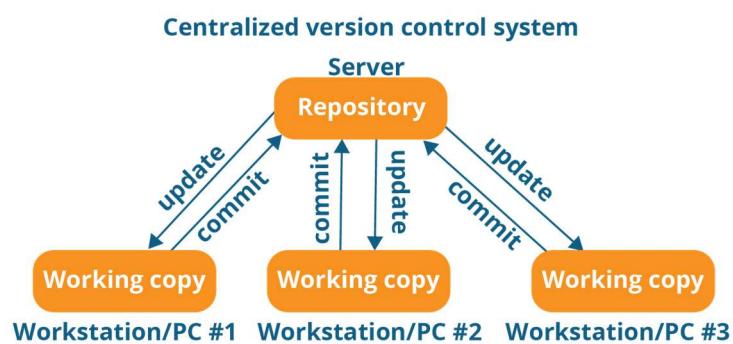
The CI/CD pipeline architecture is the backbone of a streamlined DevOps workflow, enabling continuous integration, automated testing, and seamless deployment. It consists of a series of interconnected stages, from code commits to production releases, each designed to ensure code quality, stability, and efficiency.



Why CI/CD?

CI/CD practices help development and operations teams accelerate time-to-market, improve quality and reduce costs by automating the software build, test, delivery and deployment functions and eliminating manually intensive, error-prone processes. CI/CD practices help developers incorporate code changes, bug fix, etc. quickly, and they help operations teams deploy and update software, quickly and easily.

5.2. Role of Each DevOps Tool in the Pipeline



- **Git for Version Control**

Git serves as the cornerstone of version control, allowing developers to collaborate on code changes, track revisions, and manage different branches. Its distributed nature facilitates seamless branching and merging, ensuring a structured

approach to code development.

- **Jenkins for Automation**

Jenkins automates repetitive tasks in the software development lifecycle. It integrates with version control systems like Git and triggers automated builds, tests, and deployments based on code changes. This reduces manual intervention, accelerates feedback loops, and ensures consistency in the development process.



- **Maven for Dependency Management**

Maven streamlines the management of project dependencies, ensuring a standardized build process across the team. It automatically downloads and configures project dependencies, plugins, and libraries, eliminating compatibility issues and enhancing the overall development experience.



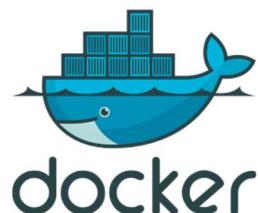
- **Tomcat for Application Deployment**

Tomcat, a widely used servlet container, facilitates the deployment of Java web applications. Its role in the CI/CD pipeline involves receiving built artifacts and making them accessible to end-users through hosting and serving the application.



- **Docker for Containerization**

Docker introduces containerization, encapsulating an application and its dependencies into a portable unit. This ensures consistent environments across development, testing, and production, mitigating the "it works on my machine" problem and enhancing reproducibility.



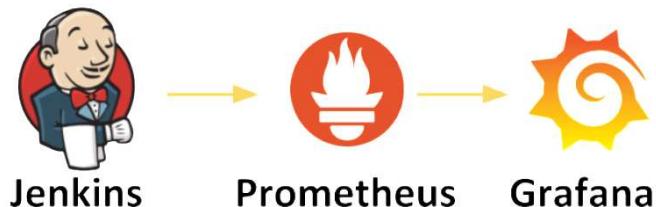
- **Ansible for Configuration Management**

Ansible automates configuration changes across multiple servers, ensuring consistent setups. In the pipeline, Ansible is used to deploy configurations required for the application to run, providing a uniform environment for various stages.



- **Prometheus & Grafana for Monitoring and Visualization**

Grafana facilitates real-time monitoring and visualization of application and Prometheus for system metrics. It helps teams gain insights into performance, identify bottlenecks, and make data-driven decisions, contributing to improved reliability and user experience.



- **Kubernetes for Orchestration (further development)**

Kubernetes orchestrates the deployment, scaling, and management of containerized applications. It ensures high availability, load balancing, and self-healing capabilities, making it an essential tool for large-scale and distributed applications.



By orchestrating the interplay of these DevOps tools, the CI/CD pipeline ensures a cohesive, automated, and efficient journey from code development to production deployment.

6. Implementation Steps

6.1. Setting Up Git Repository and Branching Strategy

The project commences with the creation of a Git repository to facilitate version control and collaboration among team members. A well-defined branching strategy, such as Git Flow, is established to manage feature development, bug fixes, and releases. This strategy ensures clear separation of code changes and promotes a structured development process.

6.2. Configuring Jenkins for Automated Builds and Tests

- Defining Jenkins Jobs and Pipelines

Jenkins is configured to automate build, test, and deployment processes.

Jenkins jobs and pipelines are defined to orchestrate these workflows. Pipelines include stages for source code checkout, build compilation, automated testing, and deployment to different environments.

- Integration with Git and Maven

Jenkins is integrated with the Git repository to trigger pipeline execution upon code changes. Maven, a build automation tool, is used to manage project dependencies, compile source code, and generate executable artifacts.

- Automated Testing Strategies

Automated tests, including unit tests and integration tests, are integrated into the pipeline. These tests validate code quality and functionality at each development stage, ensuring early detection of defects and enabling rapid feedback.

6.3. Deploying Applications on Tomcat Server

- WAR File Deployment

Applications developed are packaged as WAR (Web Application Archive) files. Tomcat, a servlet container, is utilized to deploy these applications. The deployment process involves uploading the WAR files to Tomcat and managing their lifecycle.

- Configuration and Deployment Automation

To ensure consistency and repeatability, deployment processes are automated. Scripts or configuration management tools are employed to handle environment-specific configurations, minimizing manual intervention and potential errors.

6.4. Containerization with Docker

- Building Docker Images

Docker is used to containerize applications and their dependencies. Docker images are created, incorporating the necessary components for application execution. These images are versioned and stored in a Docker registry.

6.5. Configuration Management using Ansible

Ansible is employed to automate configuration changes across various environments. Ansible playbooks define tasks for configuring servers, installing dependencies, and managing application-specific settings.

6.6. Monitoring and Visualization with Prometheus & Grafana

- Integrating Prometheus for Metrics Collection

Prometheus, a monitoring tool, is integrated to collect application and system metrics. Prometheus scrapes metrics from various sources and stores them for analysis.

- Creating Dashboards for Application and System Metrics

Grafana is employed to visualize metrics collected by Prometheus. Interactive dashboards are designed to display real-time insights into application performance, resource utilization, and health.

>> Setup GitHub Code Repository

The screenshot shows a GitHub repository named 'sample-code'. The repository has 1 branch and 0 tags. The README.md file contains the text 'siddheshkachkure update for using CICD WebPage'. The repository details include a 'Code' button, a 'Clone' section with options for HTTPS, SSH, and GitHub CLI, and a URL <https://github.com/siddheshkachkure/sample-code>. The repository has 0 stars, 1 watching, and 0 forks. It also lists releases, packages, and activity.

<https://github.com/siddheshkachkure/sample-code.git>

Git – Source course Management

Jenkins – to create CI/CD Pipelines

Maven – as a build tool

Ansible – for configuration management under deployment

Docker – as a target environment to host our applications

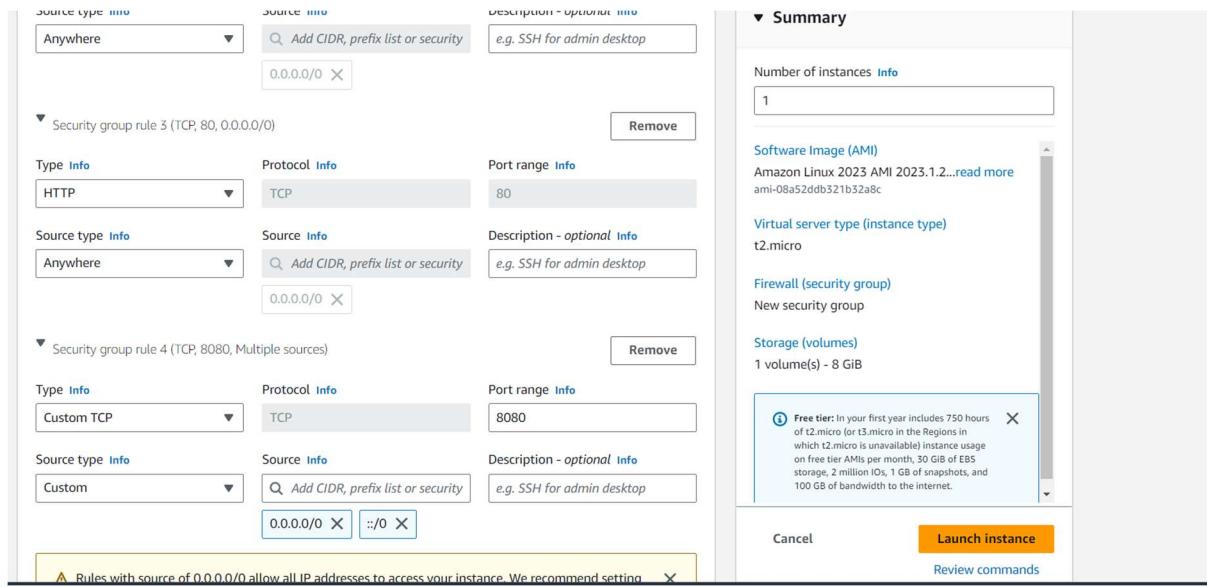
Kubernetes – to manage our Docker Containers

AWS – all the above environment to set up on AWS

The screenshot shows the AWS CloudFormation console. A new stack is being created with the name 'Jenkins Server'. The 'Summary' section shows 1 instance, using the 'Amazon Linux 2023.1.2...' AMI, t2.micro instance type, and a new security group. A tooltip provides information about the free tier. The 'Launch instance' button is visible at the bottom right.

>>Install Jenkins on AWS EC2

Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems. As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.



##Pre-requisites

1. EC2 Instance

- With Internet Access
- Security Group with Port `8080` open for internet

Total EC2 Instance Used –

Instances (5) Info								
Launch instances								
Find instance by attribute or tag (case-sensitive)								
Instance state = running	X	Clear filters						
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Ava-	
<input type="checkbox"/>	Docker_host	i-0830e34fa0c223b37	Running	t2.micro	2/2 checks passed	No alarms	+	
<input type="checkbox"/>	Jenkins_Server	i-0dfe61070557c4d9d	Running	t2.micro	2/2 checks passed	No alarms	+	
<input type="checkbox"/>	k8s_Managem...	i-028a46e91f022cd13	Running	t2.small	Initializing	No alarms	+	
<input type="checkbox"/>	Tomcat_Server	i-08664c04292bff49a	Running	t2.micro	2/2 checks passed	No alarms	+	

2. Java 11 should be installed

```
[user@lib:~/vm/java-1.8.0-amazon-corretto.x86_64/src.zip
[root@ip-172-31-32-11 ~]# vi .bash_profile
[root@ip-172-31-32-11 ~]# echo $JAVA_HOME

[root@ip-172-31-32-11 ~]# exit
logout
[ec2-user@ip-172-31-32-11 ~]$ sudo su -
Last login: Wed Aug 23 08:17:27 UTC 2023 on pts/0
[root@ip-172-31-32-11 ~]# curl -O https://amazon-corretto.s3.amazonaws.com/1.8.0/jdk-1.8.0-amazon-corretto.x86_64
[root@ip-172-31-32-11 ~]# sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
  sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
--2023-08-23 09:18:59--  https://pkg.jenkins.io/redhat-stable/jenkins.repo
Resolving pkg.jenkins.io (pkg.jenkins.io)... 146.75.34.133, 2a04:e42:78::645
Connecting to pkg.jenkins.io (pkg.jenkins.io)|146.75.34.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 85
Saving to: '/etc/yum.repos.d/jenkins.repo'

/etc/yum.repos.d/jenkins.repo          100%[=====] 85 --.-KB/s    in 0s

2023-08-23 09:18:59 (5.40 MB/s) - '/etc/yum.repos.d/jenkins.repo' saved [85/85]

[root@ip-172-31-32-11 ~]# sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
[root@ip-172-31-32-11 ~]# yum install jenkins -y
jenkins-stable
Dependencies resolved.

=====
Package           Architecture      Version       Repository   Size
=====
Installing:
jenkins          noarch        2.40.1-3.1.1  jenkins      94 M
Transaction Summary
Install 1 Package

Total download size: 94 M
Installed size: 94 M
```

The screenshot shows a terminal window within the AWS Lambda function editor. The terminal is displaying a .bash_profile script. The script includes logic to source .bashrc if it exists, sets environment variables for Java_HOME and PATH, and exports the PATH. The bottom of the terminal shows a cursor at the end of the export command and a status bar indicating 'INSERT' mode.

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64
PATH=$PATH:$HOME/bin:$JAVA_HOME

export PATH[]

-- INSERT --
```

>>Install Jenkins

You can install jenkins using the rpm or by setting up the repo. We will set up the repo so that we can update it easily in the future.

```
#sudowget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
# sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

```

# amazon-linux-extras install java-openjdk11
# yum install epel-release
# yum install fontconfig java-11-openjdk
# yum install jenkins
# systemctl start jenkins.service
# systemctl status jenkins.service

```

>>Setup Jenkins to start at boot,

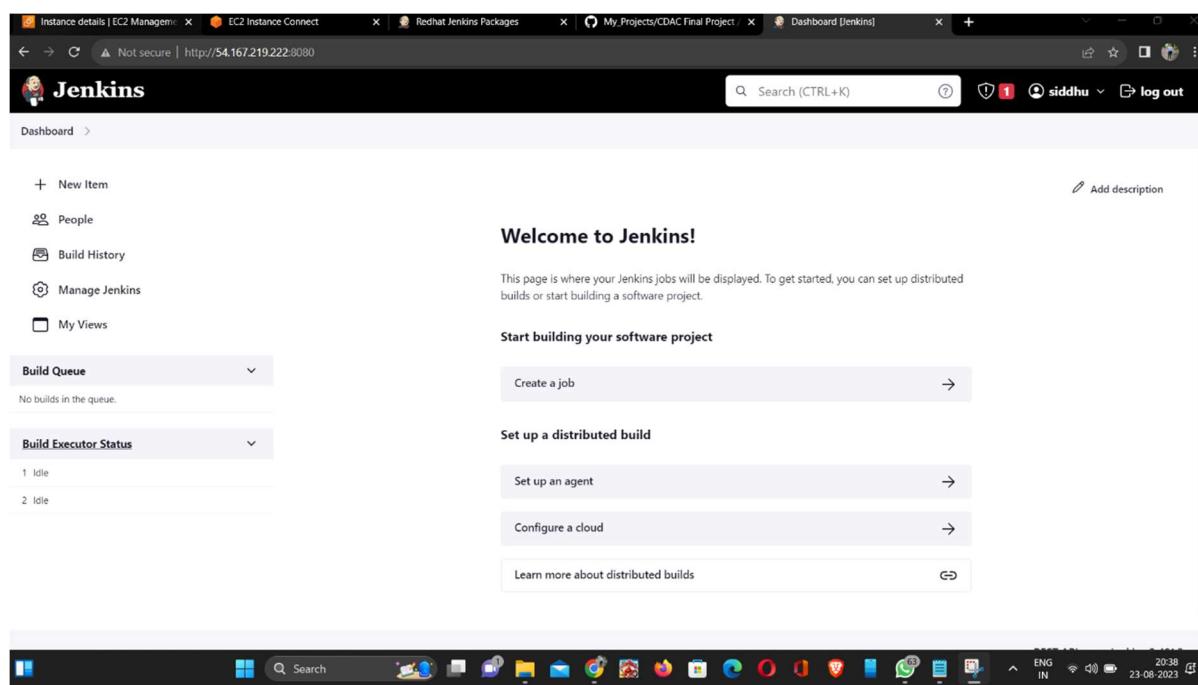
Accessing Jenkins

By default Jenkins runs at port `8080`, You can access Jenkins at

>><http://YOUR-SERVER-PUBLIC-IP:8080>

* Configure Jenkins

- The default Username is `admin`
- Grab the default password
- Password Location: `/var/lib/jenkins/secrets/initialAdminPassword`
- `Skip` Plugin Installation; We can do it later
- Change admin password
 - `Admin` > `Configure` > `Password`
- Configure `java` path
 - `Manage Jenkins` > `Global Tool Configuration` > `JDK`
- Create another admin user id



Test Jenkins Jobs

1. Create "new item"
2. Enter an item name – 'My-First-Project'
 - Chose 'Freestyle' project
3. Under the Build section
 - Execute shell: echo "Welcome to Jenkins Demo"
4. Save your job
5. Build job
6. Check "console output"

The screenshot shows the Jenkins Global Tool Configuration interface. At the top, there's a header for 'JDK installations' with a sub-header 'List of JDK installations on this system'. Below this is a 'Add JDK' button. A dashed box highlights a 'JDK' entry with the name 'JAVA_HOME' and the path '/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64'. There's also an unchecked checkbox for 'Install automatically'. At the bottom of this section is another 'Add JDK' button. Below this is a 'Git installations' section with a 'Save' button (which is blue) and an 'Apply' button.

Configure Git pulgin on Jenkins

Git is one of the most popular tools for version control system. you can pull code from git repositories using jenkins if you use github plugin.

1. Install git packages on jenkins server

- ```
yum install git -y
```
- \* Setup Git on jenkins console \*
- Install git plugin without restart
    - 'Manage Jenkins' > 'Jenkins Plugins' > 'available' > 'github'
    - Configure git path
      - 'Manage Jenkins' > 'Global Tool Configuration' > 'git'

### Install & configure Maven build tool on Jenkins

Maven is a code build tool which used to convert your code to an artifact. this is a widely used plugin to build in continuous integration

### Install Maven on Jenkins

1. Download maven packages <https://maven.apache.org/download.cgi> onto Jenkins server. In this case, I am using /opt/maven as my installation directory

- Link : <https://maven.apache.org/download.cgi>

Creating maven directory under /opt

```
mkdir /opt/maven
```

```
cd /opt/maven
```

Downloading maven version 3.6.0

```
wget http://mirrors.estointernet.in/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz
```

```
tar -xvzf apache-maven-3.6.1-bin.tar.gz
```

2. Setup M2\_HOME and M2 paths in .bash\_profile of the user and add these to the path variable

```
vi ~/.bash_profile
```

```
M2_HOME=/opt/maven/apache-maven-3.6.1
```

```
M2=$M2_HOME/bin
```

```
PATH=<Existing_PATH>:$M2_HOME:$M2
```

The screenshot shows a terminal window titled "CloudShell" with the AWS logo. The command `tar -xvzf apache-maven-3.6.1-bin.tar.gz` was run, extracting files into the current directory. The terminal output shows the names of the extracted JAR files, including various versions of core, resolver, and transport components. The terminal also shows the user's path as [root@ip-172-31-32-11 opt]#.

```
apache-maven-3.9.4/lib/slf4j-api-1.7.36.jar
apache-maven-3.9.4/lib/commons-lang3-3.12.0.jar
apache-maven-3.9.4/lib/maven-core-3.9.4.jar
apache-maven-3.9.4/lib/maven-modelmetadata-3.9.4.jar
apache-maven-3.9.4/lib/maven-artifact-3.9.4.jar
apache-maven-3.9.4/lib/maven-resolver-provider-3.9.4.jar
apache-maven-3.9.4/lib/maven-resolver-impl-1.9.14.jar
apache-maven-3.9.4/lib/maven-resolver-named-locks-1.9.14.jar
apache-maven-3.9.4/lib/maven-resolver-spi-1.9.14.jar
apache-maven-3.9.4/lib/org_eclipse_sisu_inject-0.3.5.jar
apache-maven-3.9.4/lib/plexus-component-annotations-2.1.0.jar
apache-maven-3.9.4/lib/maven-compat-3.9.4.jar
apache-maven-3.9.4/lib/maven-resolver-provider-api-3.5.3.jar
apache-maven-3.9.4/lib/org_eclipse_sisu_plexus-0.3.5.jar
apache-maven-3.9.4/lib/commons-cli-1.5.0.jar
apache-maven-3.9.4/lib/wagon-http-3.5.3.jar
apache-maven-3.9.4/lib/wagon-http-shared-3.5.3.jar
apache-maven-3.9.4/lib/httpclient-4.5.14.jar
apache-maven-3.9.4/lib/commons-codec-1.11.jar
apache-maven-3.9.4/lib/wagon-file-3.5.3.jar
apache-maven-3.9.4/lib/jcl-over-slf4j-1.7.36.jar
apache-maven-3.9.4/lib/maven-resolver-connector-basic-1.9.14.jar
apache-maven-3.9.4/lib/maven-resolver-transport-file-1.9.14.jar
apache-maven-3.9.4/lib/maven-resolver-transport-https-1.9.14.jar
apache-maven-3.9.4/lib/httpcore-4.4.16.jar
apache-maven-3.9.4/lib/maven-resolver-transport-wagon-1.9.14.jar
apache-maven-3.9.4/lib/sl4j-provider-3.9.4.jar
apache-maven-3.9.4/lib/jansi-2.4.0.jar
[root@ip-172-31-32-11 opt]# ll
total 9120
drwxr-xr-x 6 root root 98 Aug 24 04:52 apache-maven-3.9.4
-rw-r--r-- 1 root root 9336368 Aug 3 07:56 apache-maven-3.9.4-bin.tar.gz
drwxr-xr-x 4 root root 33 Aug 7 22:39 aws
drwxr-xr-x 6 root root 99 Aug 24 04:52 maven
[root@ip-172-31-32-11 opt]#
```

\* Checkpoint

1. logout and login to check maven version

```
mvn --version
```

So far we have completed the installation of maven software to support maven plugin on the jenkins console. Let's jump onto Jenkins to complete the remaining steps.

```

aws Services Search [Alt+S]
.bash_profile
Get the aliases and functions
if [-f ~/._bashrc]; then
 . ~/._bashrc
fi

User specific environment and startup programs
JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64
M2_HOME=/opt/maven
M2=$M2/bin/
PATH=$PATH:$HOME/bin:$JAVA_HOME

export PATH
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --

```

### \* Setup maven on Jenkins console

#### 1. Install maven plugin without restart

- 'Manage Jenkins' > 'Jenkins Plugins' > 'available' > 'Maven Invoker'
- 'Manage Jenkins' > 'Jenkins Plugins' > 'available' > 'Maven Integration'

#### 2. Configure maven path

- 'Manage Jenkins' > 'Global Tool Configuration' > 'Maven'

**Status**

**Build #1 (Aug 24, 2023, 6:21:01 AM)**

Keep this build forever

Started 25 min ago Took 20 min

Add description

Changes

Console Output

Edit Build Information

Delete build '#1'

Git Build Data

Redeploy Artifacts

Test Result

See Fingerprints

Module Builds

| Module        | Time    |
|---------------|---------|
| Maven Project | 4.5 sec |
| Server        | 20 min  |
| Webapp        | 10 sec  |

REST API Jenkins 2.401.3

## Setup Tomcat Server

>>Tomcat installation on EC2 instance

Pre-requisites> EC2 instance with Java 11

>>Install Apache Tomcat

1. Download tomcat packages from <https://tomcat.apache.org/download-10.cgi> onto

/opt on EC2 instance >Create tomcat directory

```
cd /opt
```

```
wget https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.12/bin/apache-tomcat-10.1.12.tar.gz
```

```
tar -xvzf /opt/apache-tomcat-10.1.12.tar.gz
```

> Note: you may get below error while starting tomcat incase if you dont install Java

'Neither the JAVA\_HOME nor the JRE\_HOME environment variable is defined At least one of these environment variable is needed to run this program'

```
bash: /bin/: Is a directory
[root@tomcatserver tomcat]# pwd
/opt/tomcat
[root@tomcatserver tomcat]# ls
BUILDING.txt CONTRIBUTING.md LICENSE NOTICE README.md RELEASE-NOTES RUNNING.txt bin conf lib logs temp webapps work
[root@tomcatserver tomcat]# cd bin/
[root@tomcatserver bin]# ls
bootstrap.jar catalina.sh commons-daemon-native.tar.gz configtest.sh digest.sh migrate.bat setclasspath.sh startup.bat tomcat-native.tar.gz version.bat
catalina-tasks.xml ciphers.bat commons-daemon.jar daemon.sh makebase.bat migrate.sh shutdown.bat startup.sh tool-wrapper.bat version.sh
catalina.bat ciphers.sh configtest.bat digest.bat makebase.sh setclasspath.bat shutdown.sh tomcat-juli.jar tool-wrapper.sh
[root@tomcatserver bin]# vi catalina.sh
[root@tomcatserver bin]# ./startup.sh
Using CATALINA_BASE: /opt/tomcat
Using CATALINA_HOME: /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME: /usr
Using CLASSPATH: /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
[root@tomcatserver bin]# ^C
[root@tomcatserver bin]# find / -name context.xml
/opt/tomcat/conf/context.xml
/opt/tomcat/webapps/docs/META-INF/context.xml
/opt/tomcat/webapps/examples/META-INF/context.xml
/opt/tomcat/webapps/host-manager/META-INF/context.xml
/opt/tomcat/webapps/manager/META-INF/context.xml
[root@tomcatserver bin]# vi /opt/tomcat/webapps/docs/META-INF/context.xml
[root@tomcatserver bin]# vi /opt/tomcat/webapps/examples/META-INF/context.xml
[root@tomcatserver bin]# vi /opt/tomcat/webapps/host-manager/META-INF/context.xml
[root@tomcatserver bin]# ./shutdown.sh
Using CATALINA_BASE: /opt/tomcat
Using CATALINA_HOME: /opt/tomcat

i-0d549cc90756f382d (Tomcat_Server)
PublicIPs: 54.80.59.36 PrivateIPs: 172.31.35.68
```

## >>Check point :

access tomcat application from browser on port 8080

- http://<Public\_IP>:8080

Using unique ports for each application is a best practice in an environment. But tomcat and Jenkins runs on ports number 8080. Hence lets change tomcat port number to 8090. Change port number in conf/server.xml file under tomcat home

```
cd /opt/apache-tomcat-<version>/conf
```

```
update port number in the "connecter port" field in server.xml
```

```
restart tomcat after configuration update
```

## >>Check point :

Access tomcat application from browser on port 8080

- http://<Public\_IP>:8080

1. now application is accessible on port 8080. but tomcat application doesnt allow to login from browser. changing a default parameter in context.xml does address this issue

```
search for context.xml
```

```
find / -name context.xml
```

2. above command gives 3 context.xml files. comment (<!--& -->) 'Value ClassName' field on files which are under webapp directory.

After that restart tomcat services to effect these changes.

At the time of writing this lecture below 2 files are updated.

```
/opt/tomcat/webapps/host-manager/META-INF/context.xml
/opt/tomcat/webapps/manager/META-INF/context.xml
```

\* Restart tomcat services

### 3. Update users information in the tomcat-users.xml file

goto tomcat home directory and Add below users to conf/tomcat-users.xml file

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user username="admin" password="admin" roles="manager-gui, manager-
script, manager-jmx, manager-status"/>
<user username="deployer" password="deployer" roles="manager-script"/>
<user username="tomcat" password="s3cret" roles="manager-gui"/>
```

```
The users below are wrapped in a comment and are therefore ignored. If you
wish to configure one or more of these users for use with the manager web
application, do not forget to remove the <!...> that surrounds them. You
will also need to set the passwords to something appropriate.
-->
<!--
<user username="admin" password=<must-be-changed>" roles="manager-gui"/>
<user username="robot" password=<must-be-changed>" roles="manager-script"/>
-->
<!--
The sample user and role entries below are intended for use with the
examples web application. They are wrapped in a comment and thus are ignored
when reading this file. If you wish to configure these users for use with the
examples web application, do not forget to remove the <!...> that surrounds
them. You will also need to set the passwords to something appropriate.
-->
<!--
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password=<must-be-changed>" roles="tomcat"/>
<user username="both" password=<must-be-changed>" roles="tomcat,role1"/>
<user username="role1" password=<must-be-changed>" roles="role1"/>
-->
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>
<user username="admin" password="admin" roles="manager-script"/>
<user username="admin" password="admin" roles="manager-gui"/>
</tomcat-users>
"tomcat-users.xml" 63L, 3120B
```

i-0d549cc90756f382d (Tomcat\_Server)

PublicIPs: 54.80.59.36 PrivateIPs: 172.31.35.68

Restart service and try to login to tomcat application from the browser. This time it should be Successful

The screenshot shows the Apache Tomcat Web Application Manager. At the top, there's a navigation bar with tabs like 'Manager', 'List Applications', 'HTML Manager Help', 'Manager Help', and 'Server Status'. Below the header, there's a section titled 'Tomcat Web Application Manager' with a message box containing 'Message: OK'. The main area is divided into two sections: 'Applications' and 'Deploy'.

**Applications:**

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

**Deploy:**

Deploy directory or WAR files located on server

Context Path:   
 Version (for parallel deployment):   
 XML Configuration file path:   
 WAR or Directory path:

## Deploy a war file on Tomcat VM using Jenkins

The screenshot shows the Jenkins dashboard. At the top, there's a search bar and a user profile. The main area has a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. The 'Build History' section shows two recent builds:

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	Deploy_onsiddhu_tomcat_server	35 sec #2	N/A	26 sec
✓	☀️	siddhu_maven	7 hr 33 min #1	N/A	20 min

Below the build history, there's a 'Build Queue' section stating 'No builds in the queue.' The 'Build Executor Status' section shows '1 Idle' and '2 Idle'. At the bottom right, there are links for 'REST API' and 'Jenkins 2.401.3'.

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Thu Aug 24 10:39:14 2023 from 18.206.107.28
[ec2-user@tomcatserver ~]$ sudo su -
Last login: Thu Aug 24 10:44:29 UTC 2023 on pts/0
[root@tomcatserver ~]# cd /opt/tomcat/
bin/ conf/ lib/ logs/ temp/ webapps/ work/
[root@tomcatserver ~]# cd /opt/tomcat/webapps/
[root@tomcatserver webapps]# ls
ROOT docs examples host-manager manager webapp webapp.war
[root@tomcatserver webapps]# []

```

i-0d549cc90756f382d (Tomcat\_Server)  
PublicIPs: 54.80.59.36 PrivateIPs: 172.31.35.68

## Deploy on VM through PollSCM

Make change in Deploy\_onsiddhu\_tomcat\_server> in Build trigger section > Poll SCM > Schedule for > \*\*\*\*\* (every min-hour-day-week-month) apply & save >

## >>Installing Docker on Amazon Linux server

### 1. Amazon Linux EC2 Instance

### >>Installation Steps

#### 1. Install docker and start docker services

```
yum install docker -y
docker --version
service docker start
service docker status
```

#### 2 . Create a user called dockeradmin

```
useradd dockeradmin
passwd dockeradmin
```

3. add a user to docker group to manage docker

```
usermod -aG docker dockeradmin
```

### >>Validation test

1. Create a tomcat docker container by pulling a docker image from the public docker registry

```
docker run -d --name test-tomcat-server -p 8090:8080 tomcat:latest
yum install docker -y
docker ps -a
systemctl start docker.service
docker ps -a
docker images
docker pull tomcat:latest
docker images
hostnamectl set-hostname dockerhost
su
docker images
docker rmi tomcat
clear
ls
docker --version
docker ps -a
docker images
docker pull tomcat:latest
useradd dockeradmin
passwd dockeradmin
usermod -aG docker dockeradmin
id dockeradmin
ip a
vi /etc/ssh/sshd
vi /etc/ssh/sshd_config
service sshd reload
cd /home/dockeradmin/
ls
su - dockeradmin
cd
useradd ansadmin
passwd ansadmin
ip a
ls
su - dockeradmin
docker login
```

```

usermod -aG docker ansadmin
su - ansadmin
vi /etc/sudoers
docker login
su - ansadmin
docker images
docker ps -a
ls
docker images
su - ansadmin
su - dockeradmin
sudo vi /etc/sudoers

```

After this Configure for docker host. Jenkins

The screenshot shows the Jenkins 'SSH Server' configuration dialog. It includes fields for Name (set to 'dockerhost'), Hostname (set to '172.31.37.176'), Username (set to 'dockeradmin'), and a Remote Directory field (empty). There is also a checkbox for 'Avoid sending files that have not changed'. At the bottom, there are 'Advanced' and 'Edited' buttons.

The screenshot shows the Jenkins job configuration page under 'Post-build Actions'. A 'Send build artifacts over SSH' action is selected. The configuration for this action includes an 'SSH Publishers' section with 'Name' set to 'dockerhost', and a 'Transfers' section where 'Source files' are set to 'webapp/target/\*.war', 'Remove prefix' is set to 'webapp/target', and 'Remote directory' is set to '.'. At the bottom of the dialog are 'Save' and 'Apply' buttons.

```
Deploy on Docker host server using Jenkins
Jenkins Job name: `Deploy_on_Docker_Host`
Pre-requisites
```

1. Jenkins server
1. Docker-host Server

## Integration between Docker-host and Jenkins

Install "publish Over SSH"

- 'Manage Jenkins' > 'Manage Plugins' > 'Available' > 'Publish over SSH'

Enable connection between Docker-host and Jenkins

- 'Manage Jenkins' > 'Configure System' > 'Publish Over SSH' > 'SSH Servers'

- SSH Servers:

- Name: 'docker-host'
- Hostname :`<ServerIP>`
- username: 'dockeradmin'

- 'Advanced' > chose 'Use password authentication, or use a different key'
- password: `\*\*\*\*\*`

### Steps to create "Deploy\_on\_Docker\_Host" Jenkin job

#### From Jenkins home page select "New Item"

- Enter an item name: 'Deploy\_on\_Docker\_Host'
- Copy from: 'Deploy\_on\_Tomcat\_Server'

- \*Source Code Management:\*

- Repository: 'https://github.com/yankils/hello-world.git'
- Branches to build : `\*/master`
- \*Poll SCM\* : - `\* \* \* \*`

- \*Build:\*

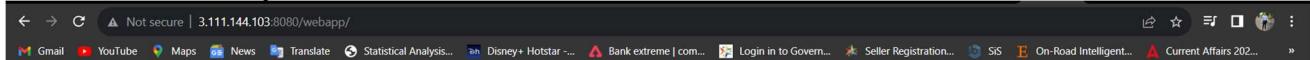
- Root POM: `pom.xml`
- Goals and options: 'clean install package'

- \*Post-build Actions\*

- Send build artifacts over SSH
- \*SSH Publishers\*
- SSH Server Name: 'docker-host'
- 'Transfers' > 'Transfer set'

- Source files: `webapp/target/\*.war`
- Remove prefix: `webapp/target`
- Remote directory: `//home//ansadmin` or `..`

Save and run the job now.



## GROUP PROJECT OF CDAC-ACTS PG-HPCSA Students

[SIDDHESH, ABOLI, SANIKA, VIKAS, INDRAJEET]

Please fill in this form to create an account.

Enter Name	Enter Full Name
Enter mobile	Enter mobile number
Enter Email	Enter Email
Password	Enter Password
Repeat Password	Repeat Password

By creating an account you agree to our [Terms & Privacy](#).

[Register](#)

Already have an account? [Sign in](#).

### Here is my sample webpage

Here is the CI/CD used WebPage

THANK YOU

## >> Now Get Ready a Ansible Server

### >>Ansible Installation

Ansible is an open-source automation platform. It is very, very simple to set up and yet powerful. Ansible can help you with configuration management, application deployment, task automation.

### >>Pre-requisites

1. An AWS EC2 instance (on Control node)

>> Installation steps:

>>on Amazon EC2 instance

2. Install python and python-pip

```
yum install python
yum install python-pip
```

3. Install ansible using pip check for version

```
pip install ansible
ansible --version
```

4. Create a user called ansadmin (on Control node and Managed host)

*useradd ansadmin*

*passwd ansadmin*

5. Below command grant sudo access to ansadmin user. But we strongly recommended using "visudo" command if you are aware vi or nano editor. (on Control node and Managed host)

*echo "ansadmin ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers*

6. Log in as aansadmin user on master and generate ssh key (on Control node)

*sudosu - ansadmin*

*ssh-keygen*

7. Copy keys onto all ansible managed hosts (on Control node)

*ssh-copy-id ansadmin@<target-server>*

8. Ansible server used to create images and store on docker registry. Hence install docker, start docker services and add ansadmin to the docker group.

*yum install docker*

*service docker start*

*service docker start*

*usermod -aG docker ansadmin*

9. Create a directory /etc/ansible and create an inventory file called "hosts" add control node and managed hosts IP addresses to it.

>> Validation test

10. Run ansible command as ansadmin user it should be successful (Master)

*ansible all -m ping*

```

aws | Services | Search [Alt+S] | Mumbai | SIDDHESH KACHKURE
[aws@ansibleserver ~]# su - ansadmin
Last login: Sun Aug 27 14:52:44 UTC 2023 from 127.0.0.1 on pts/2
[ansadmin@ansibleserver ~]$ ansible all -m ping
[WARNING]: Platform linux on host 172.31.37.176 is using the discovered Python interpreter at /usr/bin/python3.9, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.15/reference_appendices/interpreter_discovery.html for more information.
172.31.37.176 | SUCCESS => {
 "ansible_facts": [
 "discovered_interpreter_python": "/usr/bin/python3.9"
],
 "changed": false,
 "ping": "pong"
}
[WARNING]: Platform linux on host localhost is using the discovered Python interpreter at /usr/bin/python3.9, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.15/reference_appendices/interpreter_discovery.html for more information.
localhost | SUCCESS => {
 "ansible_facts": [
 "discovered_interpreter_python": "/usr/bin/python3.9"
],
 "changed": false,
 "ping": "pong"
}
[ansadmin@ansibleserver ~]$

```

i-0c39520e51b17a557 (Ansible\_Server)  
PublicIPs: 43.205.110.38 PrivateIPs: 172.31.38.165

>> Now next the entire CI-CD Pipeline is automated using ansible  
 >> Jenkins job to deploy a war file on Docker container using Ansible using Playbooks

>> Configure Jenkins  
 >> Ansible integration with Jenkins

>>Part-01 Integration Setps

Install "publish Over SSH"

- 'Manage Jenkins' > 'Manage Plugins' > 'Available' > 'Publish over SSH'

Enable connection between Ansible and Jenkins

- 'Manage Jenkins' > 'Configure System' > 'Publish Over SSH' > 'SSH Servers'

- SSH Servers:
  - Hostname: '<ServerIP>'
  - username: 'admin'
  - password: 'admin'

Test the connection "Test Connection"

```

aws | Services | Search [Alt+S] | Mumbai | SIDDHESH KACHKURE
[aws@ansibleserver ~]# su - ansadmin
Last login: Sun Aug 27 14:52:44 UTC 2023 from 127.0.0.1 on pts/2
[ansadmin@ansibleserver ~]$ ansible all -m ping
[WARNING]: Platform linux on host 172.31.37.176 is using the discovered Python interpreter at /usr/bin/python3.9, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.15/reference_appendices/interpreter_discovery.html for more information.
172.31.37.176 | SUCCESS => {
 "ansible_facts": [
 "discovered_interpreter_python": "/usr/bin/python3.9"
],
 "changed": false,
 "ping": "pong"
}
[ansadmin@ansibleserver ~]$

```

i-0c39520e51b17a557 (Ansible\_Server)  
PublicIPs: 43.205.110.38 PrivateIPs: 172.31.38.165

>> Deploy on a docker container using Ansible

>>\* Jenkins Job name: \* `Deploy\_on\_Container\_using\_ansible\_playbooks`

## ## Pre-requisites

1. Jenkins server
2. Docker-host server
3. Ansible server

The screenshot shows the Jenkins job configuration page for the job 'Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks'. The 'General' tab is selected. The 'Description' field contains the text 'Deploy on Docker Container using Ansible Playbooks'. Under the 'Advanced' section, there are several checkboxes:
 

- Discard old builds
- GitHub project
- This project is parameterized
- Execute concurrent builds if necessary

 A 'Enabled' toggle switch is turned on.

Not secure | 15.206.157.157:8080/job/Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks/configure

Gmail YouTube Maps News Translate Statistical Analysis... Disney+ Hotstar... Bank extreme | com... Login in to Govern... Seller Registration... SIS On-Road Intelligent... Current Affairs 202...

Dashboard > Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks > Configuration

Source Code Management

### Configure

General

Source Code Management

Build Triggers

Build Environment

Pre Steps

Build

Post Steps

Build Settings

Post-build Actions

None

Git

Repositories ?

Repository URL ?  
https://github.com/siddheshkachure/sample-code.git

Credentials ?  
- none -

Add Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?  
\*/master

Add Branch

Save Apply

Not secure | 15.206.157.157:8080/job/Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks/configure

Gmail YouTube Maps News Translate Statistical Analysis... Disney+ Hotstar... Bank extreme | com... Login in to Govern... Seller Registration... SIS On-Road Intelligent... Current Affairs 202...

Dashboard > Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks > Configuration

### Configure

General

Source Code Management

Build Triggers

Build Environment

Pre Steps

Build

Post Steps

Build Settings

Post-build Actions

### Build Triggers

Build whenever a SNAPSHOT dependency is built ?  
 Schedule build when some upstream has no successful builds ?

Trigger builds remotely (e.g., from scripts) ?

Build after other projects are built ?

Build periodically ?

GitHub hook trigger for GITScm polling ?

Poll SCM ?

Schedule ?  
\*\*\*\*\*

**⚠ Do you really mean "every minute" when you say "\*\*\*\*\*"? Perhaps you meant "H \* \* \* \* " to poll once per hour**  
Would last have run at Monday, August 28, 2023 at 11:30:22 AM Coordinated Universal Time; would next run at Monday, August 28, 2023 at 11:30:22 AM Coordinated Universal Time.

Ignore post-commit hooks ?

Save Apply

### Build

Root POM ?  
pom.xml

Goals and options ?  
clean install package

Advanced ▾

1. 'Dockerfile' under \*/opt/docker\* on Ansible server

```
Pull tomcat latest image from dockerhub
From tomcat
Maintainer
MAINTAINER "Project Group"
copy war file on to container
COPY ./webapp.war /usr/local/tomcat/webapps
```

2. Create `create.siddhuproject.yml` unser \*/opt/docker\* on Ansible server in ansadmin

```
cat create.siddhuproject.yml
```

```

```

```
- hosts: all
 become: true
```

```
tasks:
```

```
- name: stop current running conatiner
 command: docker stop siddhuproject-container
ignore_errors: yes

- name: remove stopped container
 command: docker rm siddhuproject-container
ignore_errors: yes

- name: remove docker image
 command: docker rmi siddheshkachkure/siddhuproject-image
ignore_errors: yes

- name: build docker image using war file
command: docker build -t siddhuproject-image
args:
chdir: /opt/docker

- name: pull docker image from dockerhub
 command: docker pull siddheshkachkure/siddhuproject-image:latest

- name: create container using siddhuproject-image
 command: docker run -d --name siddhuproject-container -p 8080:8080
siddheshkachkure/siddhuproject-image:latest
```

```

< → C ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=ap-south-1&connType=standard&instanceId=i-0c39520e51b17a557&cosUser=ec2-user&sshPort=22#/
Gmail YouTube Maps News Translate Statistical Analysis... Disney+ Hotstar... Bank extreme | com... Login to Govern... Seller Registration... SiS On-Road Intelligent... Current Affairs 202...
aws Services Search [Alt+S]
hosts: all
become: true
tasks:
- name: create docker image using war file
 command: docker build -t siddhuproject-image:latest .
 args:
 chdir: /opt/docker
- name: create tag to image
 command: docker tag siddhuproject-image siddheshkachkure/siddhuproject-image
- name: push image on to dockerhub
 command: docker push siddheshkachkure/siddhuproject-image
- name: remove docker images form ansible server
 command: docker rmi siddhuproject-image:latest siddheshkachkure/siddhuproject-image
 ignore_errors: yes
"create.siddhuproject-image.yml" [readonly] 20L, 562B
i-0c39520e51b17a557 (Ansible_Server)
PublicIPs: 43.205.110.38 PrivateIPs: 172.31.38.165
7, 44 All

```

3. Create `create.siddhuproject-image.yml` under `\*/opt/docker` on Ansible server

```

cat create.siddhuproject-image.yml

- hosts: all
 become: true

 tasks:

 - name: create docker image using war file
 command: docker build -t siddhuproject-image:latest .
 args:
 chdir: /opt/docker

 - name: create tag to image
 command: docker tag siddhuproject-image siddheshkachkure/siddhuproject-image

 - name: push image on to dockerhub
 command: docker push siddheshkachkure/siddhuproject-image

 - name: remove docker images form ansible server
 command: docker rm siddhuproject-image:latest siddheshkachkure/siddhuproject-image
 ignore_errors: yes
Integration between Ansible-control-node and Jenkins

```

```

< → C ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=ap-south-1&connType=standard&instanceId=i-0c39520e51b17a557&osUser=ec2-user&sshPort=22#/
Gmail YouTube Maps News Translate Statistical Analysis... Disney+ Hotstar... Bank extreme | com... Login in to Govern... Seller Registration... SiS On-Road Intelligent... Current Affairs 202...
aws Services Search [Alt+S]
hosts: all
become: true
tasks:
- name: stop current running container
 command: docker stop siddhuproject-container
 ignore_errors: yes
- name: remove stopped container
 command: docker rm siddhuproject-container
 ignore_errors: yes
- name: remove docker image
 command: docker rmi siddheshkachkure/siddhuproject-image
 ignore_errors: yes
- name: build docker image using war file
command: docker build -t siddhuproject-image
args:
chdir: /opt/docker
- name: pull docker image from dockerhub
 command: docker pull siddheshkachkure/siddhuproject-image:latest
1,1 Top

```

CloudShell Feedback Language © 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

## Install "publish Over SSH"

- 'Manage Jenkins' > 'Manage Plugins' > 'Available' > 'Publish over SSH'

Enable connection between Ansible-control-node and Jenkins

- 'Manage Jenkins' > 'Configure System' > 'Publish Over SSH' > 'SSH Servers'

### - SSH Servers:

- Name: 'ansible-server'
- Hostname: '<ServerIP>'
- username: 'ansadmin'

- 'Advanced' > chose 'Use password authentication, or use a different key'
- password: '\*\*\*\*\*'

## Steps to create "Deploy\_on\_Container\_using\_ansible" Jenkin job  
## From Jenkins home page select "New Item"

- Enter an item name: 'Deploy\_on\_Container\_using\_ansible'

- Copy from: 'Deploy\_on\_Container'

### - \*Source Code Management:\*

- Repository: 'https://github.com/yankils/hello-world.git'
- Branches to build : '/master'
- \*Poll SCM\* : - `\* \* \* \*`

- \*Build:\*
  - Root POM: `pom.xml`
  - Goals and options: `clean install package`

- \*Post-build Actions\*

- Send build artifacts over SSH
  - \*SSH Publishers\*
  - SSH Server Name: `ansible-server`
  - 'Transfers' > 'Transfer set'
    - Source files: `webapp/target/\*.war`
    - Remove prefix: `webapp/target`
    - Remote directory: `//opt//docker`
    - Exec command:
 

```
ansible-playbook -i /opt/docker/hosts
/opt/docker/create.siddhuproject-image.yml --limit localhost;
ansible-playbook -i /opt/docker/hosts
/opt/docker/create.siddhuproject.yml --limit 172.31.37.176
```

Save and run the job now.

Deploy\_on\_Docker.Container\_using\_Ansible\_Playbooks > Configuration

**ure**

**Code Management**

**iggers**

**vironment**

**os**

**ips**

**ettings**

**ild Actions**

**SSH Publishers**

SSH Server Name ?

ansibleserver

Advanced ▾

**Transfers**

Transfer Set ?

Source files ?

webapp/target/\*.war

Remove prefix ?

webapp/target

Remote directory ?

//opt//docker

Exec command ?

```
ansible-playbook -i /opt/docker/hosts /opt/docker/create.siddhuproject-image.yml --limit localhost;
ansible-playbook -i /opt/docker/hosts /opt/docker/create.siddhuproject.yml --limit 172.31.37.176
```

All of the transfer fields (except for Exec timeout) support substitution of Jenkins environment variables

Advanced ▾

**Save** **Apply**

So now on Jenkins server we have installed git run following commands on the Jenkins\_Server

Now you can see here are only 3 time the image is pushed on docker hub

The screenshot shows the Docker Hub interface. At the top, there's a purple header with the text "DockerCon 2023: Our annual developer event is back – online & in person. Learn more." Below the header is a blue navigation bar with the Docker Hub logo, a search bar, and links for "Explore", "Repositories", "Organizations", "Help", and "Upgrade". A user profile icon for "siddheshkachkure" is also present. The main content area shows a repository card for "siddheshkachkure / siddhuproject-image". The card indicates it's an "Inactive" repository with 0 stars, 3 commits, and is "Public". Below the card is a promotional banner for creating an organization and managing repositories. At the bottom of the page is a decorative footer bar.

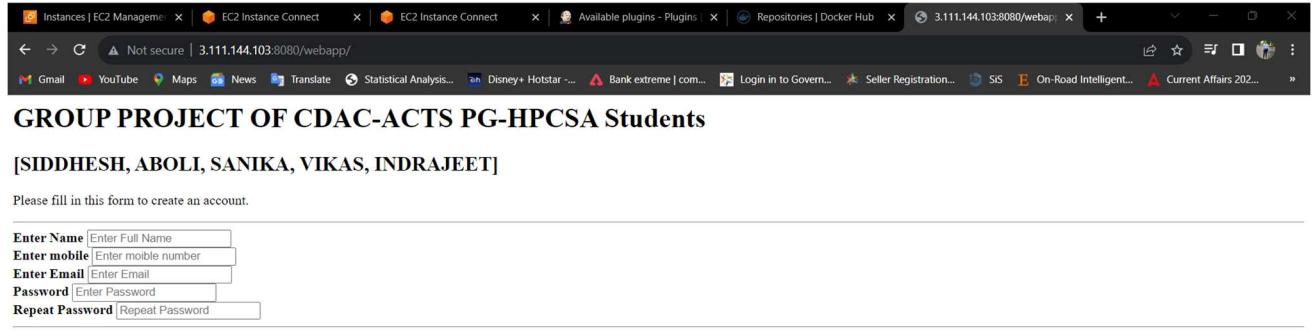
We can see here in Docker-Host the previous images

The screenshot shows a terminal session in an AWS CloudShell window. The terminal output includes:

```
>Last login: Mon Aug 28 10:53:13 2023 from 13.233.177.4
[ec2-user@dockerhost ~]$ sudo su -
Last login: Mon Aug 28 10:53:17 UTC 2023 on pts/0
Last failed login: Mon Aug 28 11:59:43 UTC 2023 from 94.76.224.227 on ssh:notty
There was 1 failed login attempt since the last successful login.
[root@dockerhost ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest 3817f27fdbf2 21 hours ago 426MB
[root@dockerhost ~]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
95e58f80ed16 siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 21 hours ago Up 21 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
[root@dockerhost ~]# ls
[root@dockerhost ~]#
```

Below the terminal, the CloudShell summary shows the instance ID "i-0a3806af2d2cc045e (Docker\_Host)" and its public and private IP addresses: "Public IPs: 3.111.144.103" and "Private IPs: 172.31.37.176".

## First see our ready web page



### Here is my sample webpage

Here is the CI/CD used WebPage

THANK YOU

Now we will change our code by commit in GitHub in Jenkins Server

, #  
~\ #####\_ Amazon Linux 2023  
~~ \#####\ |  
~~ \###|  
~~ \#/ https://aws.amazon.com/linux/amazon-linux-2023  
~~ V~'-'>  
~~~ /  
~~\_. /  
~/ /  
~/m/'

```
[root@jenkinsserver~]# ls
sample-code
[root@jenkinsserver~]# cd sample-code/
[root@jenkinsserver sample-code]# ls
Dockerfile README.md pom.xml regapp-deploy.ymlregapp-service.yml
server webapp
[root@jenkinsserver sample-code]# cd webapp/src/main/webapp/WEB-INF/
[root@jenkinsserver WEB-INF]# ls
web.xml
[root@jenkinsserver WEB-INF]# cd ..
[root@jenkinsserver webapp]# ls
```

WEB-INF index.jsp

```
[root@jenkinsserverwebapp]# vi index.jsp
```

```
[root@jenkinsserverwebapp]# git status
```

On branch master

Your branch is up to date with 'origin/master'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: index.jsp

no changes added to commit (use "git add" and/or "git commit -a")

```
[root@jenkinsserverwebapp]# git add .
```

```
[root@jenkinsserverwebapp]# git commit -m "updated webpage for the
Continous Deployment "
```

```
[master 0bed1bd] updated webpage for the Continous Deployment
```

1 file changed, 2 insertions(+), 1 deletion(-)

```
[root@jenkinsserverwebapp]# git push origin master
```

Username for 'https://github.com': siddheshkachkure

Password for 'https://siddheshkachkure@github.com':

Enumerating objects: 13, done.

Counting objects: 100% (13/13), done.

Compressing objects: 100% (5/5), done.

Writing objects: 100% (7/7), 605 bytes | 605.00 KiB/s, done.

Total 7 (delta 2), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

To https://github.com/siddheshkachkure/sample-code.git

5eb4bc8..0bed1bd master -> master

```
[root@jenkinsserverwebapp]#
```

Now in Jenkins we see automated CI-CD of the Job just after pushing to GitHub. :

The screenshot shows the Jenkins dashboard. At the top, there is a 'Build Queue (1)' section with two items: 'Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks' (status: green, duration: 21 hr, build number: #11) and 'Siddhu\_Maven\_Build' (status: green, duration: 1 day 5 hr, build number: #3). Below this is a 'Build Executor Status' section showing 1 idle and 2 idle executors. At the bottom, a Windows taskbar is visible with various icons and system status information.

Then we will see our change in webpage.:

Project Group CDAC PG-HPCSA SAMPLE LOGIN WEB-PAGE

[SIDDHESH, ABOLI, SANIKA, VIKAS, INDRAJEET]

Please fill in this form to create an account.

Enter Name [Enter Full Name]  
Enter mobile [Enter mobile number]  
Enter Email [Enter Email]  
Password [Enter Password]  
Repeat Password [Repeat Password]

By creating an account you agree to our [Terms & Privacy](#).

[Register](#)

Already have an account? [Sign in](#).

### Here is my sample webpage

Here is the CI/CD used WebPage

THANK YOU

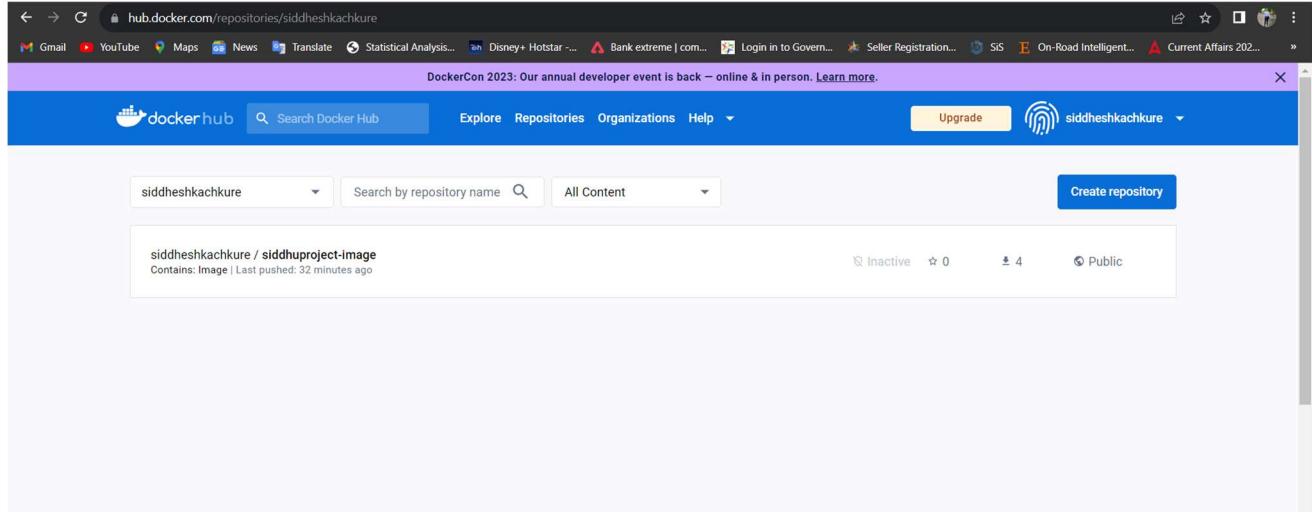
Here you can see previous Image & Container created 21 hours ago and then we can see the latest Image & Container created 29 minutes ago and the Status is “Up”

```
Last login: Mon Aug 28 10:53:13 2023 from 13.233.177.4
[ec2-user@dockerhost ~]$ sudo su -
Last login: Mon Aug 28 10:53:17 UTC 2023 on pts/0
Last failed login: Mon Aug 28 11:59:43 UTC 2023 from 94.76.224.227 on ssh:notty
There was 1 failed login attempt since the last successful login.
[root@dockerhost ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest 3817f27fdbf2 21 hours ago 426MB
[root@dockerhost ~]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
95e58f80e0d16 siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 21 hours ago Up 21 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
[root@dockerhost ~]# ls
[root@dockerhost ~]# docker images
docker: 'imagers' is not a docker command.
See 'docker --help'.
[root@dockerhost ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest db4e8b80218d 30 minutes ago 426MB
[root@dockerhost ~]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
49882c2df16c siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 29 minutes ago Up 29 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
[root@dockerhost ~]#
```

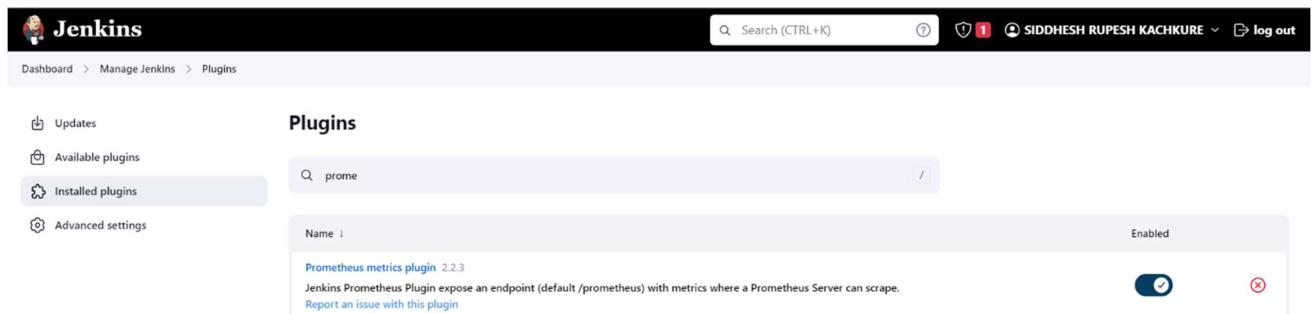
i-0a3806af2d2cc045e (Docker\_Host)

PublicIPs: 3.111.144.103 PrivateIPs: 172.31.37.176

Then we will see our change in DockerHub :



>> Now we will integrate Prometheus and Grafana in our Jenkins



you should have Jenkins First For Monitor

In Jenkins > install plugin (Prometheus metrics) > it will create endpoint  
`http://<Jenkins:ip>:8080/prometheus` > now install expose 9090 port for Prometheus and install it

Dashboard > Manage Jenkins > System >

Advanced  Edited

**Prometheus**

Path

Default Namespace

Enable authentication for prometheus end-point

Collecting metrics period in seconds

Count duration of successful builds

Count duration of unstable builds

Count duration of failed builds

Count duration of not-built builds

Count duration of aborted builds

Fetch the test results of builds

Add build parameter label to metrics

Add build status label to metrics

```
docker run -d --name prometheus -p 9090:9090 prom/prometheus
```

>> we will use docker to install grafana on JENKINS\_SERVER >

```
docker run -d --name grafana -p 3000:3000 grafana/grafana
```

```
docker ps -a
```

>> we will get 2 containers >prometheus&grafana> to access jenkins data in prometheus we need to make few changes in prometheus config file >

```
docker exec -it <conatiner id> /bin/sh
```

```
/promethues
```

```
cd /etc/prometheus
```

```
vi prometheus.yml
```

```
- job_name: "jenkins"
```

```
metrics_path: /prometheus
```

static\_configs:

```
- targets: ['15.206.157.157:8080']
```

> now check the yml file by inserting the ymlscript yamllint.com >

# YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or not, and give you a nice clean UTF-8 version of it.

```
1 ---
2 global:
3 scrape_interval: 15s
4 evaluation_interval: 15s
5 alerting:
6 alertmanagers:
7 - static_configs:
8 - targets: null
9 rule_files: null
10 scrape_configs:
11 - job_name: prometheus
12 static_configs:
13 - targets:
14 - localhost:9090
15 - job_name: jenkins
16 metrics_path: /prometheus
17 static_configs:
18 - targets:
19 - 15.206.157.157:8080
20
```

Reformat (strips comments)  Resolve aliases

Valid YAML!

---

```
my global config
global:
 scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1
 minute.
 evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
 minute.
 # scrape_timeout is set to the global default (10s).

Alertmanager configuration
alerting:
 alertmanagers:
 - static_configs:
 - targets:
 # - alertmanager:9093

Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
 # - "first_rules.yml"
 # - "second_rules.yml"

A scrape configuration containing exactly one endpoint to scrape:
```

```

Here it's Prometheus itself.
scrape_configs:
 # The job name is added as a label `job=<job_name>` to any timeseries scraped from
 # this config.
 - job_name: "prometheus"

 # metrics_path defaults to '/metrics'
 # scheme defaults to 'http'.

static_configs:
 - targets: ["localhost:9090"]
 - job_name: "jenkins"
metrics_path: /prometheus
static_configs:
 - targets: ['15.206.157.157:8080']

```

---



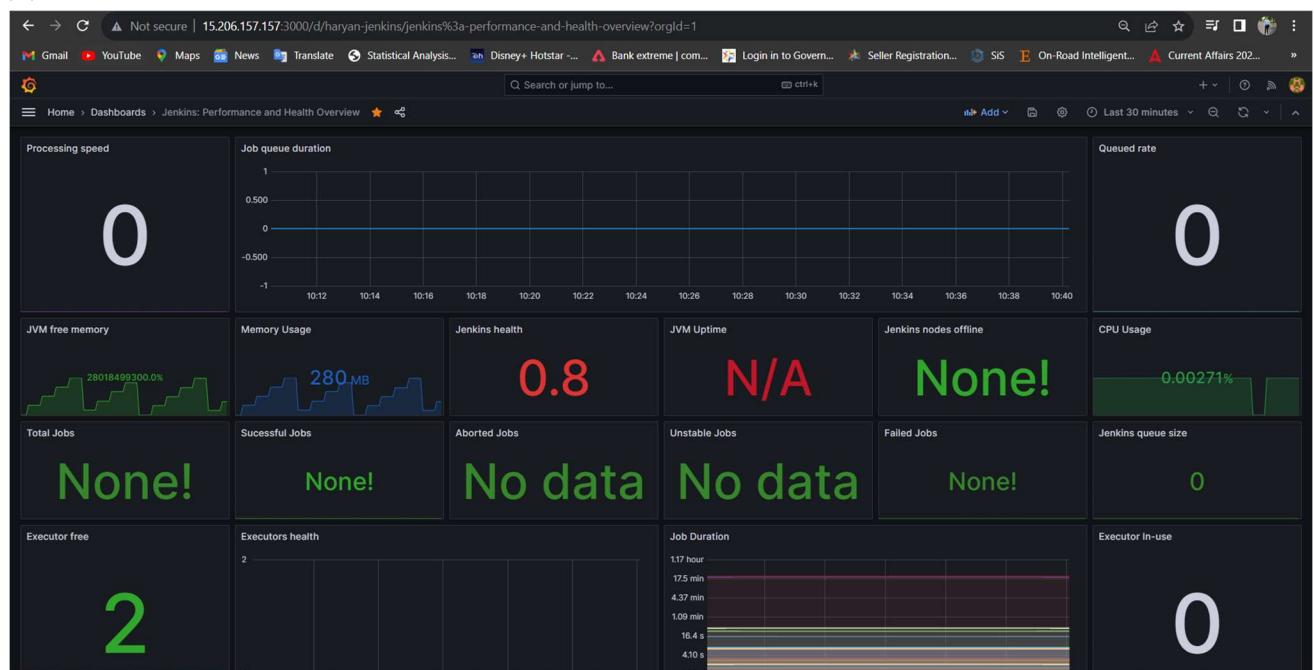
---

> after edit in yaml restart the prometheusconatiner>

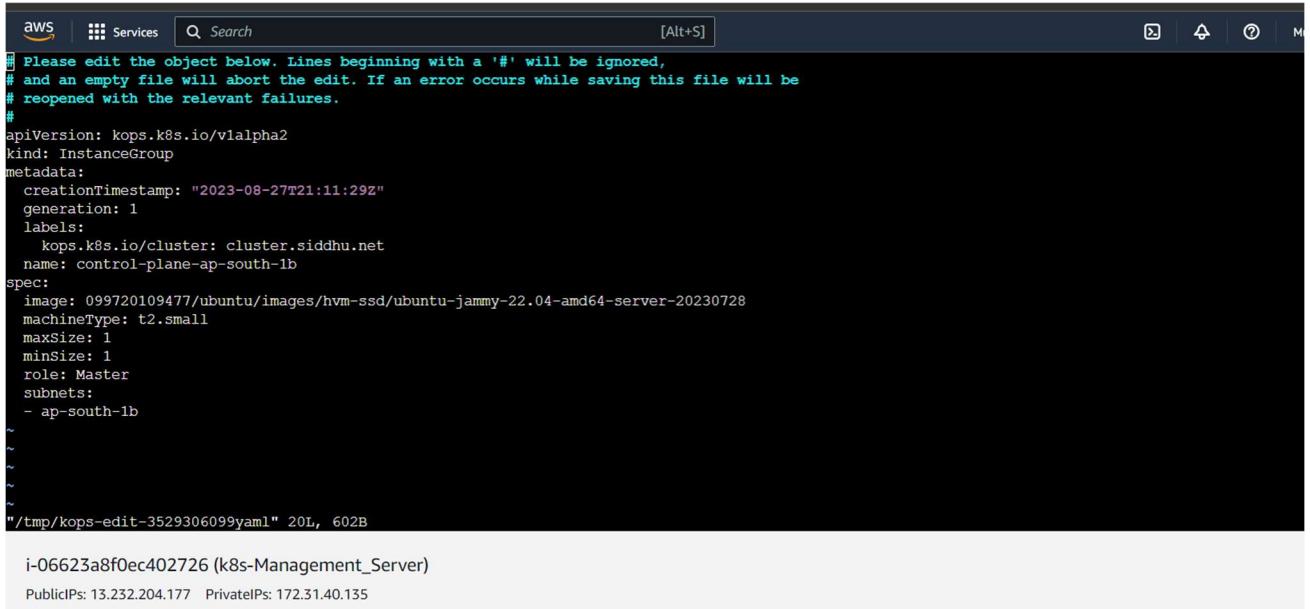
```
docker restart <proemtheus-container-id>
```

>> After that go for the web pages of Grafana and Prometheus

And integrate them >> and make dashboard for the Jenkins >> by importing json file  
 >>



Also For further Development we have made Kubernetes Cluster On AWS



```
Please edit the object below. Lines beginning with a '#' will be ignored,
and an empty file will abort the edit. If an error occurs while saving this file will be
reopened with the relevant failures.
#
apiVersion: kops.k8s.io/v1alpha2
kind: InstanceGroup
metadata:
 creationTimestamp: "2023-08-27T21:11:29Z"
 generation: 1
 labels:
 kops.k8s.io/cluster: cluster.siddhu.net
 name: control-plane-ap-south-1b
spec:
 image: 099720109477/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230728
 machineType: t2.small
 maxSize: 1
 minSize: 1
 role: Master
 subnets:
 - ap-south-1b
~/tmp/kops-edit-3529306099yaml" 20L, 602B

i-06623a8f0ec402726 (k8s-Management_Server)
PublicIPs: 13.232.204.177 PrivateIPs: 172.31.40.135
```

## >>Setup Kubernetes (K8s) Cluster on AWS

1. Create Ubuntu EC2 instance

2. install AWSCLI

```
curl https://s3.amazonaws.com/aws-cli/awscli-bundle.zip -o awscli-bundle.zip
sudo apt update
sudo apt install unzip python
unzip awscli-bundle.zip
#sudo apt-get install unzip - if you dont have unzip in your system
./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

3. Install kubectl on ubuntu instance

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

4. Install kops on ubuntu instance

```
curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s
https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d
"" -f 4)/kops-linux-amd64
```

```
chmod +x kops-linux-amd64
sudo mv kops-linux-amd64 /usr/local/bin/kops
```

5. Create an IAM user/role with Route53, EC2, IAM and S3 full access

6. Attach IAM role to ubuntu instance

# Note: If you create IAM user with programmatic access then provide Access keys.  
Otherwise region information is enough  
*aws configure*

7. Create a Route53 private hosted zone (you can create Public hosted zone if you have a domain)

Route53 --> hosted zones --> created hosted zone  
Domain Name: siddhu.net  
Type: Private hosted zone for Amazon VPC

8. Create an S3 bucket

```
aws s3 mb s3://siddhubucket.net
```

9. Expose environment variable:

```
export KOPS_STATE_STORE=s3://siddhubucket.net
```

10. Create sshkeys before creating cluster

```
ssh-keygen
```

11. Create kubernetes cluster definitions on S3 bucket

```
kops create cluster --cloud=aws --zones=ap-south-1b --name=cluster.siddhu.net --dns-zone=siddhu.net --dns private
```

12. If you wish to update the cluster worker node sizes use below command

```
kops edit ig --name=CHANGE_TO_CLUSTER_NAME nodes
```

13. Create kubernetes cluster

```
kops update cluster demo.k8s.valaxy.net --yes
```

## 14. Validate your cluster

```
kops validate cluster
```

## 15. To list nodes

```
kubectl get nodes
```

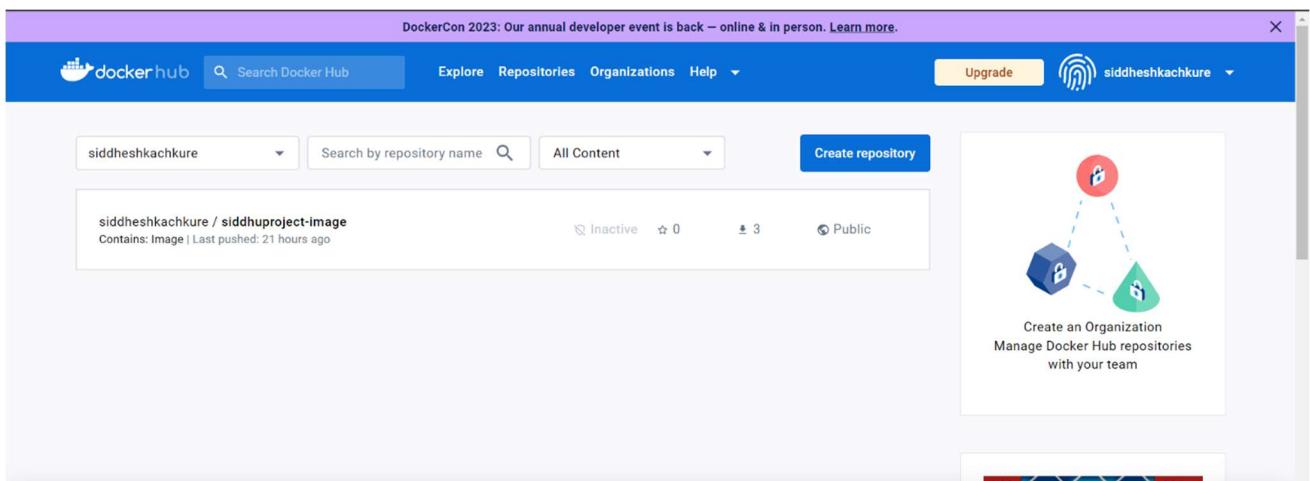
## 16. To delete cluster

```
kops delete cluster demo.k8s.valaxy.net --yes
```

## 7. Results

```
[ansadmin@ansibleserver docker]$ ll
total 20
-rw-r--r--. 1 ansadmin ansadmin 101 Aug 27 12:38 Dockerfile
-rw-r--r--. 1 ansadmin ansadmin 562 Aug 27 12:43 create.siddhuproject-image.yml
-rw-r--r--. 1 ansadmin ansadmin 792 Aug 27 10:32 create.siddhuproject.yml
-rw-r--r--. 1 ansadmin ansadmin 24 Aug 27 12:28 hosts
-rw-rw-r--. 1 ansadmin ansadmin 2464 Aug 27 14:52 webapp.war
[ansadmin@ansibleserver docker]$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
tomcat latest b0a197eea51c 2 days ago 426MB
[ansadmin@ansibleserver docker]$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[ansadmin@ansibleserver docker]$ []
```

Here you can see .war file on <ansible server> it will change after the change in code



Here You can see 3 push in docker-hub repository

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Mon Aug 28 10:53:13 2023 from 13.233.177.4
[ec2-user@dockerhost ~]$ sudo su -
Last login: Mon Aug 28 10:53:17 UTC 2023 on pts/0
Last failed login: Mon Aug 28 11:59:43 UTC 2023 from 94.76.224.227 on ssh:notty
There was 1 failed login attempt since the last successful login.
[root@dockerhost ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest 3817f27fdbf2 21 hours ago 426MB
[root@dockerhost ~]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
95e58f80ed16 siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 21 hours ago Up 21 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
[root@dockerhost ~]# ls
[root@dockerhost ~]#

```

i-0a3806af2d2cc045e (Docker\_Host)  
PublicIPs: 3.111.144.103 PrivateIPs: 172.31.37.176

Here in <docker\_host> we have previous created image and container running after the change in code it will delete and new image and container will come instead of it .

**GROUP PROJECT OF CDAC-ACTS PG-HPCSA Students**

[SIDDHESH, ABOLI, SANIKA, VIKAS, INDRAJEET]

Please fill in this form to create an account.

|                 |                          |
|-----------------|--------------------------|
| Enter Name      | <input type="text"/>     |
| Enter mobile    | <input type="text"/>     |
| Enter Email     | <input type="text"/>     |
| Password        | <input type="password"/> |
| Repeat Password | <input type="password"/> |

By creating an account you agree to our [Terms & Privacy](#).

Already have an account? [Sign in](#).

### Here is my sample webpage

Here is the CI/CD used WebPage

THANK YOU

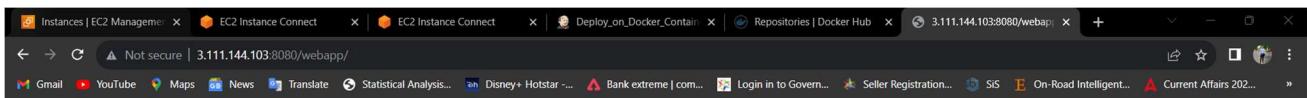
This is Web Page before changing the code

Build Queue (1) Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks 21 hr #11 N/A 1 min 0 sec >  
Deploy\_on\_Docker\_Container\_using\_Ansible\_Playbooks  
Siddhu\_Maven\_Build 1 day 5 hr #3 1 day 11 hr #1 28 sec >  
Build Executor Status 1 Idle 2 Idle  
Icon: S M L Icon legend Atom feed for all Atom feed for failures Atom feed for just latest builds  
REST API Jenkins 2.414.1

After changing and pushing the code in github the ansible will run a job on Jenkins automatically you can see the Delay in the Build Queue:

```
aws | Services | Search [Alt+S] | Mumbai | SIDDHESH KACHKURE | /m/ | /
Last login: Mon Aug 28 10:53:13 2023 from 13.233.177.4
[ec2-user@dockerhost ~]$ sudo su -
Last login: Mon Aug 28 10:53:17 UTC 2023 on pts/0
Last failed login: Mon Aug 28 11:59:43 UTC 2023 from 94.76.224.227 on ssh:notty
There was 1 failed login attempt since the last successful login.
(root@dockerhost ~)$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest 3817f27fdbf2 21 hours ago 426MB
(root@dockerhost ~)$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
95e58fb0ed16 siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 21 hours ago Up 21 hours 0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
(root@dockerhost ~)$ ls
(root@dockerhost ~)$ docker imagers
docker: 'imagers' is not a docker command.
See 'docker --help'
(root@dockerhost ~)$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
siddheshkachkure/siddhuproject-image latest db4eb00218d 30 minutes ago 426MB
(root@dockerhost ~)$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
49802c2d1f6c siddheshkachkure/siddhuproject-image:latest "catalina.sh run" 29 minutes ago Up 29 minutes 0.0.0:8080->8080/tcp, :::8080->8080/tcp siddhuproject-container
(root@dockerhost ~)$
i-0a3806af2d2cc045e (Docker_Host)
PublicIPs: 3.111.144.103 PrivateIPs: 172.31.37.176
```

See here is the change in the image and container in the <docker\_host>



## Project Group CDAC PG-HPCSA SAMPLE LOGIN WEB-PAGE

[SIDDHESH, ABOLI, SANIKA, VIKAS, INDRAJEET]

Please fill in this form to create an account.

|                 |                     |
|-----------------|---------------------|
| Enter Name      | Enter Full Name     |
| Enter mobile    | Enter mobile number |
| Enter Email     | Enter Email         |
| Password        | Enter Password      |
| Repeat Password | Repeat Password     |

By creating an account you agree to our [Terms & Privacy](#).

[Register](#)

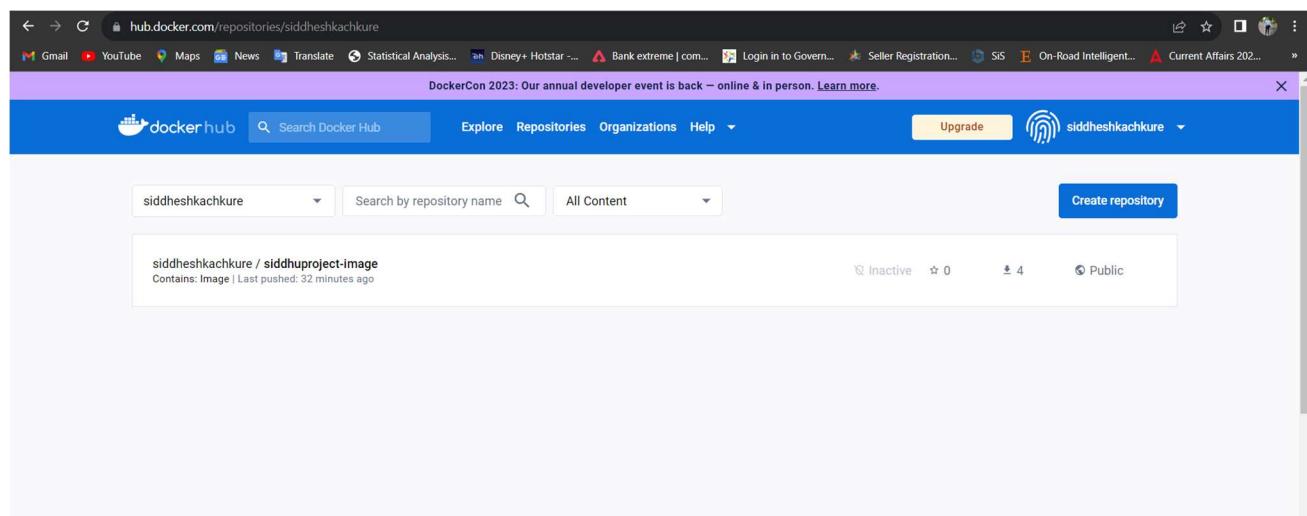
Already have an account? [Sign in](#).

### Here is my sample webpage

Here is the CI/CD used WebPage

THANK YOU

Here is our updated web page.....



And Here you can see the Push count has been changed to 4

## 8. Conclusion

### 8.1 Achievements of the Project

Throughout the course of this project, several notable achievements were realized. The successful integration of a variety of DevOps tools resulted in the creation of a cohesive and streamlined environment for software development and deployment.

#### **Key accomplishments include:**

**Establishment of a Comprehensive DevOps Ecosystem:** The project successfully brought together tools such as Git, Jenkins, Maven, Tomcat, Docker, Ansible, and Grafana to create an integrated and efficient DevOps workflow.

#### **Automation of Key Processes:**

Automation was a cornerstone of our approach, leading to the elimination of manual and error-prone tasks, resulting in improved efficiency and consistency.

#### **Enhanced Monitoring and Insights:**

The integration of Grafana, although an alternative to Kubernetes, provided real-time monitoring capabilities, offering valuable insights into application performance and system metrics.

### 8.2 Benefits of the Implemented DevOps Environment

The implemented DevOps environment has yielded a range of benefits, impacting both development and operations aspects of the project:

#### **Faster Time-to-Market:**

Automation and streamlined processes reduced development cycles, allowing for quicker delivery of features and updates.

#### **Improved Collaboration:**

The integrated tools facilitated seamless collaboration among development and operations teams, enhancing communication and reducing bottlenecks.

#### **Reduced Manual Intervention:**

Automated testing, deployment, and configuration management minimized manual errors, leading to higher quality code and increased reliability.

#### **Scalability and Consistency:**

Docker and Ansible played pivotal roles in ensuring consistent application behavior across various environments, enhancing portability and scalability.

### **8.3 Future Potential and Expansion of the Environment**

As this project concludes, it opens the door to future enhancements and potential expansions:

#### **Further Integration:**

The environment's potential for expansion lies in the integration of additional tools or technologies that align with the project's objectives.

#### **Advanced Orchestration:**

Exploring advanced orchestration techniques and tools beyond Kubernetes could provide new dimensions for scaling and managing containerized applications.

#### **Security Enhancements:**

Ongoing efforts to enhance security measures within the pipeline can fortify the environment against potential cyber threats.

#### **Continuous Learning and Improvement:**

Regularly assessing the DevOps environment, adopting best practices, and incorporating feedback will be essential for sustained success.

## **9. Recommendations**

### **9.1 Enhancing Security in the Pipeline**

To ensure the security of the DevOps pipeline:

Employ robust authentication and authorization mechanisms for accessing tools and repositories. Regularly update and patch tools and dependencies to address vulnerabilities.

### **9.2 Extending Automated Testing Coverage**

To enhance code quality:

Expand automated testing coverage to encompass various testing scenarios, including regression, integration, and performance testing.

Implement continuous testing practices to catch issues early in the development cycle.

### **9.3 Deployment on, Backup and Recovery Strategies for Kubernetes**

For projects venturing into Kubernetes:

Develop comprehensive backup and recovery strategies to ensure data integrity and application availability.

Explore Kubernetes deployment options, such as Helm charts, to simplify application deployment and management.

### **9.4 Documentation and Knowledge Sharing**

To ensure sustainability and knowledge transfer:

Maintain up-to-date documentation detailing the DevOps environment setup, configuration, and best practices.

Foster a culture of knowledge sharing through regular team meetings, presentations, and collaborative platforms.

## 10. References

1. Jenkins Documentation for pipeline  
<https://www.jenkins.io/doc/tutorials/#pipeline>
2. Maven Documentation  
<https://maven.apache.org/download.cgi>
3. Tomcat Server Documentation  
<https://tomcat.apache.org/>
4. References from Git  
<https://github.com/siddheshkachkure/sample-code.git>  
<https://github.com/siddheshkachkure/My-CDAC-Project.git>  
[https://github.com/vikasrai8423/My\\_Projects.git](https://github.com/vikasrai8423/My_Projects.git)  
[https://github.com/Pharak123/My\\_Projects.git](https://github.com/Pharak123/My_Projects.git)  
[https://github.com/indra7011/My\\_Projects.git](https://github.com/indra7011/My_Projects.git)  
[https://github.com/bhanage/My\\_Projects.git](https://github.com/bhanage/My_Projects.git)
5. AWS Documentation  
<https://docs.aws.amazon.com/>
6. Ansible Documentation  
<https://docs.ansible.com/>