# 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

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

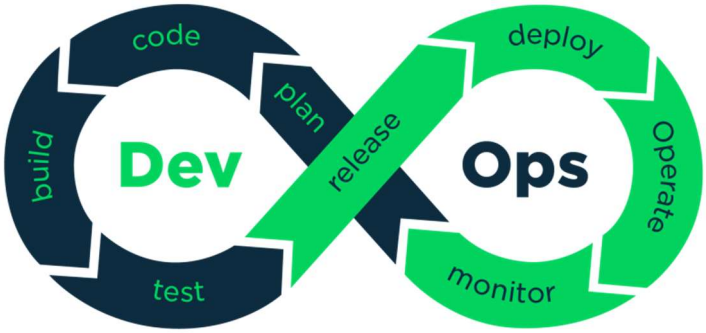
4.2 Software Requirement

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

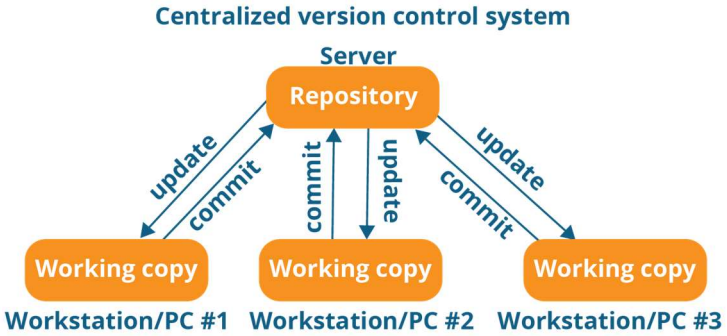
The CI/CD pipeline architecture is the backbone of a streamlined DevOps workflow, enabling continuous integration, automated testing, and seamless deployment. It consists of a series of interconnected stages, from code commits to production releases, each designed to ensure code quality, stability, and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

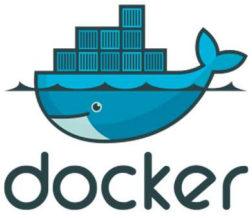
Maven streamlines the management of project dependencies, ensuring a standardized build process across the team. It automatically downloads and configures project dependencies, plugins, and libraries, eliminating compatibility issues and enhancing the overall development experience.


- **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.

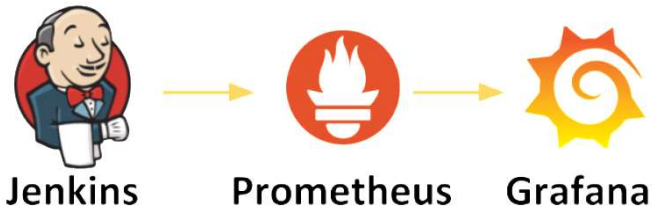


- **Ansible for Configuration Management**

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

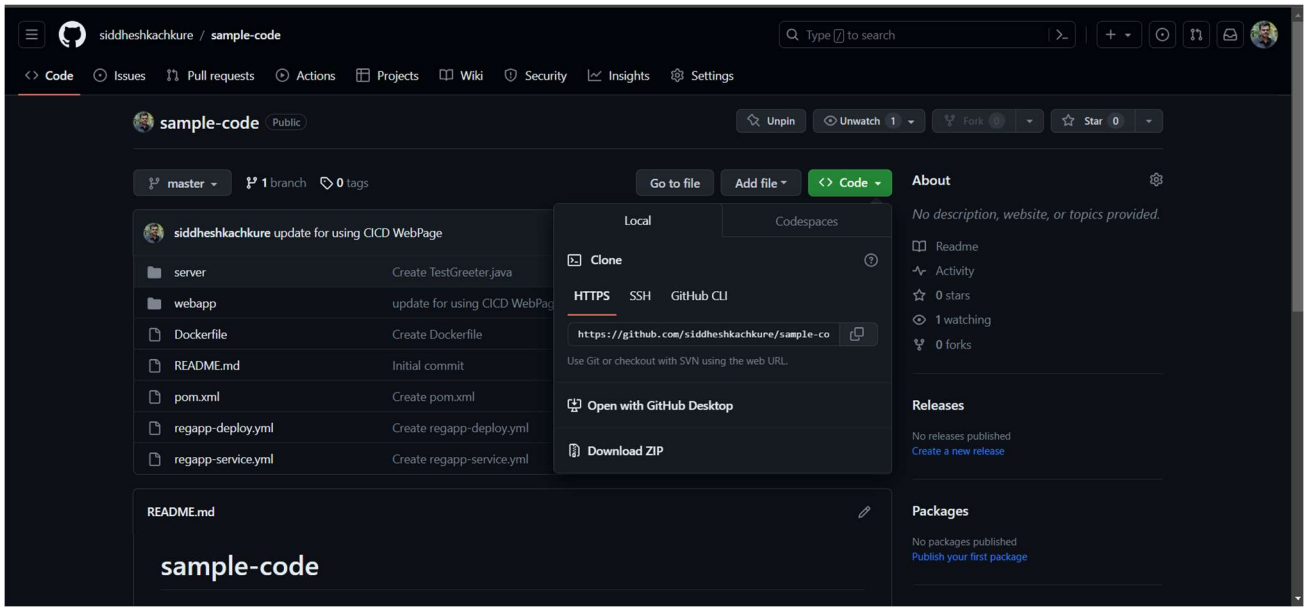
- Integrating Prometheus for Metrics Collection

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

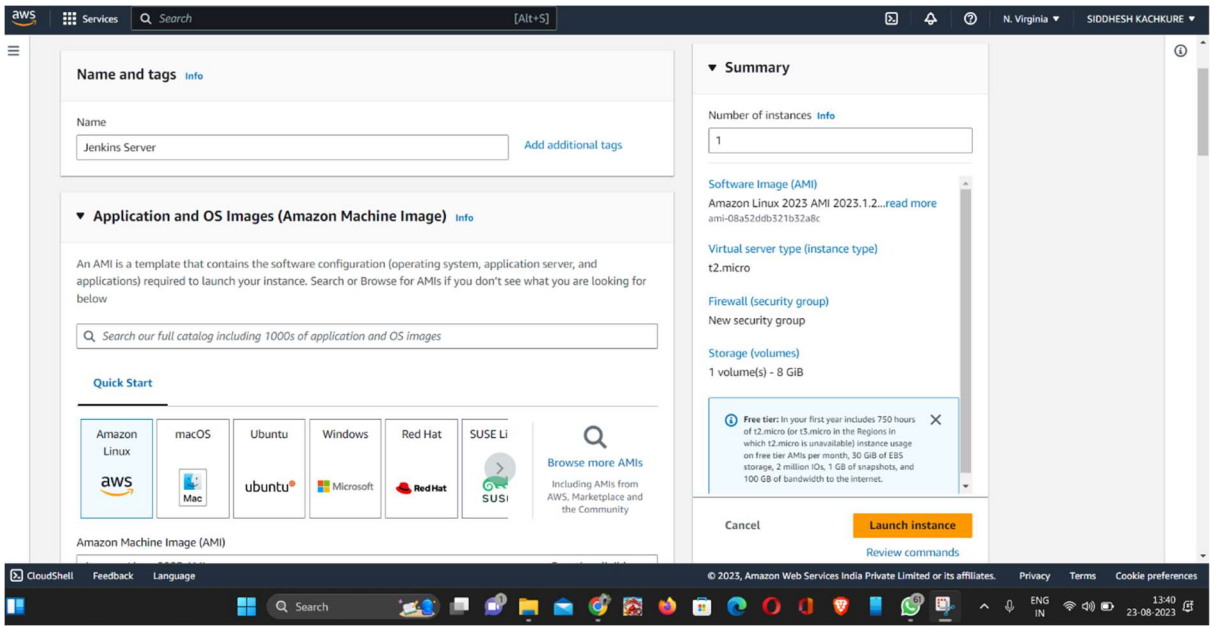
Grafana is employed to visualize metrics collected by Prometheus. Interactive dashboards are designed to display real-time insights into application performance, resource utilization, and health.

>> Setup GitHub Code Repository



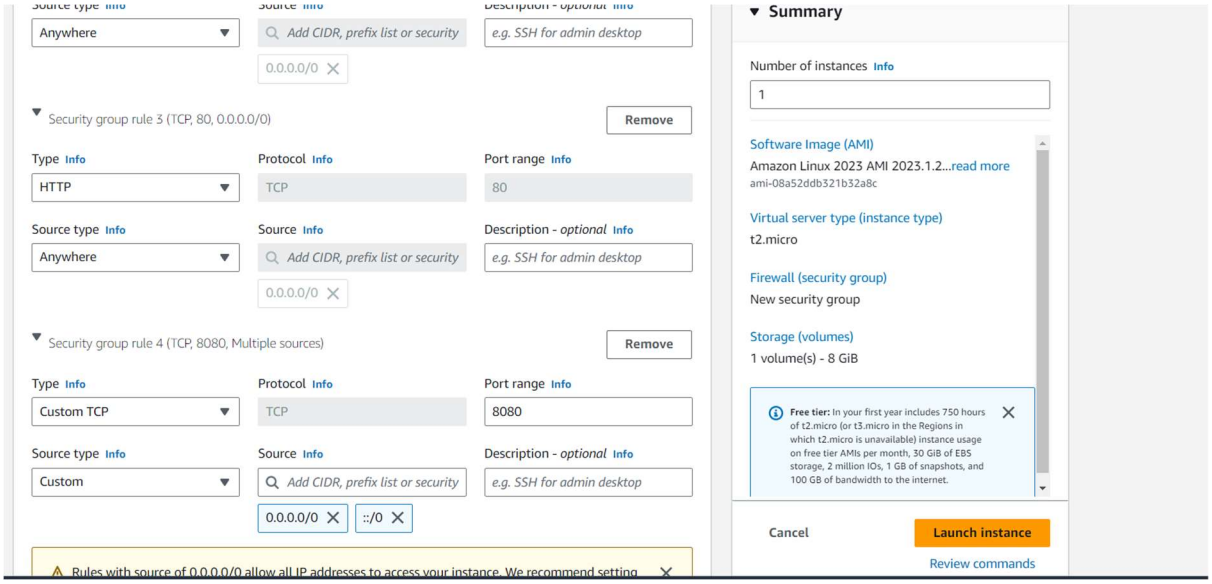

https://github.com/siddheshkachkure/sample-code.git

**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



## >>Install Jenkins on AWS EC2

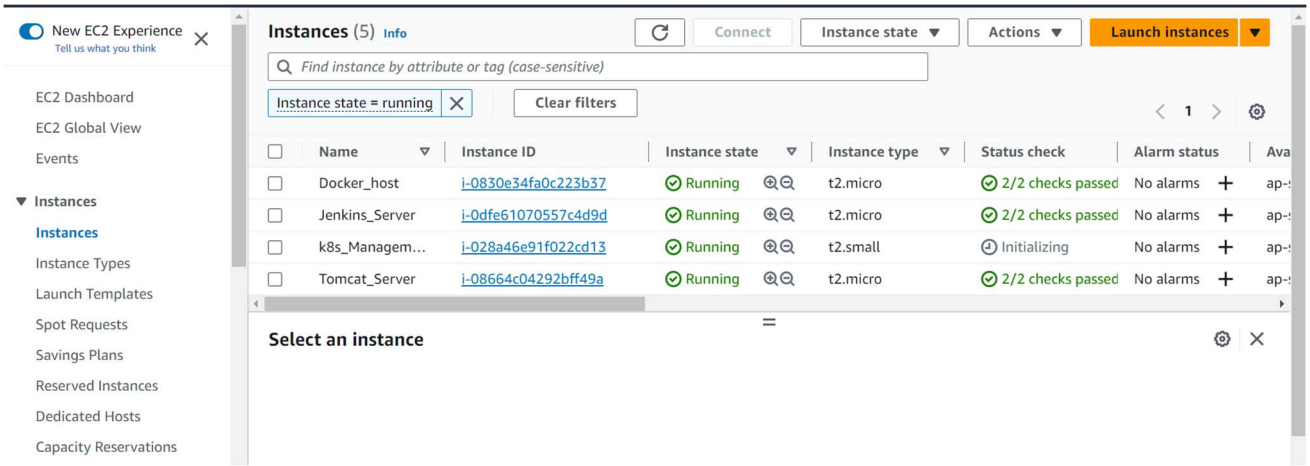
Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems. As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.




## ##Pre-requisites

1. EC2 Instance
   - With Internet Access
   - Security Group with Port `8080` open for internet

Total EC2 Instance Used –



2. Java 11 should be installed

```
/usr/lib/jvm/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 ~]# echo $JAVA_HOME
/usr/lib/jvm/java-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, 2a04:4e42:78::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                                                                                                  252 kB/s |  27 kB     00:00
Dependencies resolved.
================================================================================================================================================
 Package                         Architecture                    Version                         Repository                              Size
================================================================================================================================================
Installing:
 jenkins                         noarch                          2.401.3-1.1                     jenkins                                 94 M

Transaction Summary
================================================================================================================================================
Install  1 Package

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

```
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
PATH=$PATH:$HOME/bin:$JAVA_HOME

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

**>>Install Jenkins**

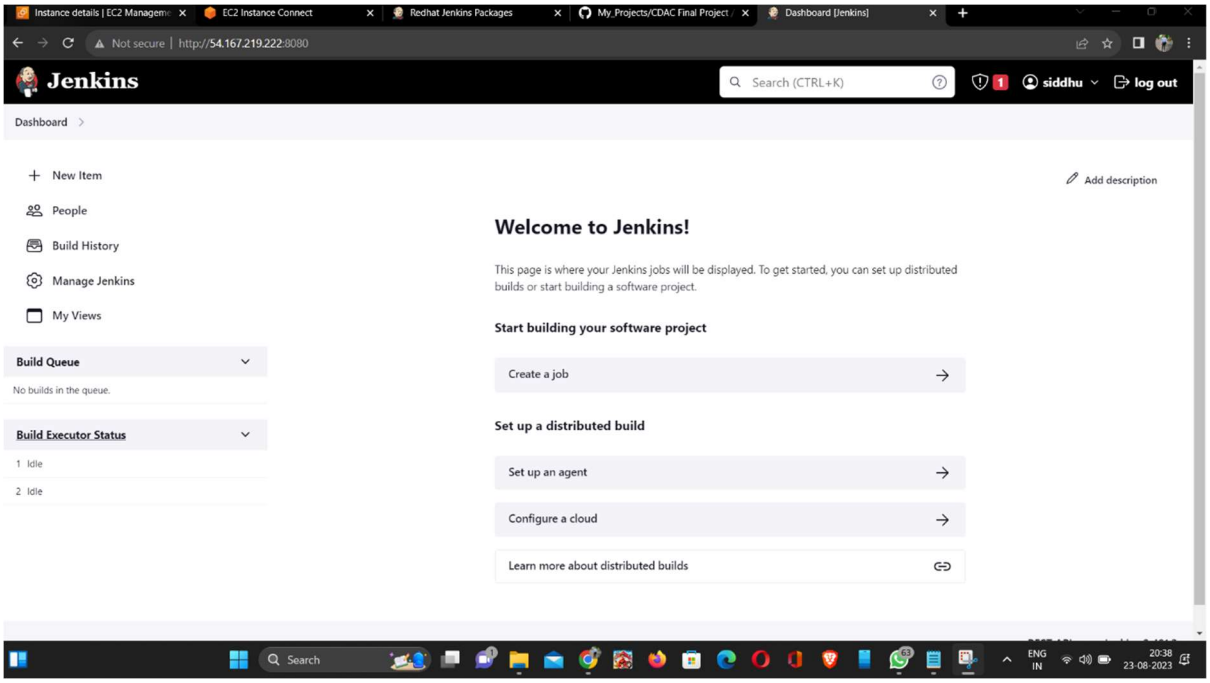
You can install jenkins using the rpm or by setting up the repo. We will set up the repo so that we can update it easily in the future.

*#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*

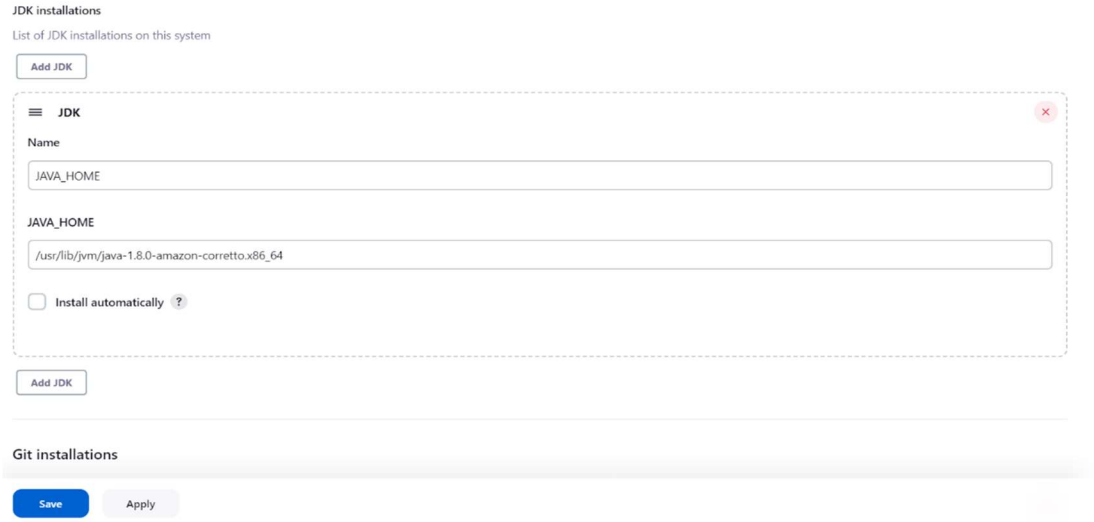
>>Setup Jenkins to start at boot,
Accessing Jenkins
By defaultjenkins runs at port `8080`, You can access jenkins at
>>http://YOUR-SERVER-PUBLIC-IP:8080
* Configure Jenkins
- The default Username is `admin`
- Grab the default password
- Password Location:`/var/lib/jenkins/secrets/initialAdminPassword`
- `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"




**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*

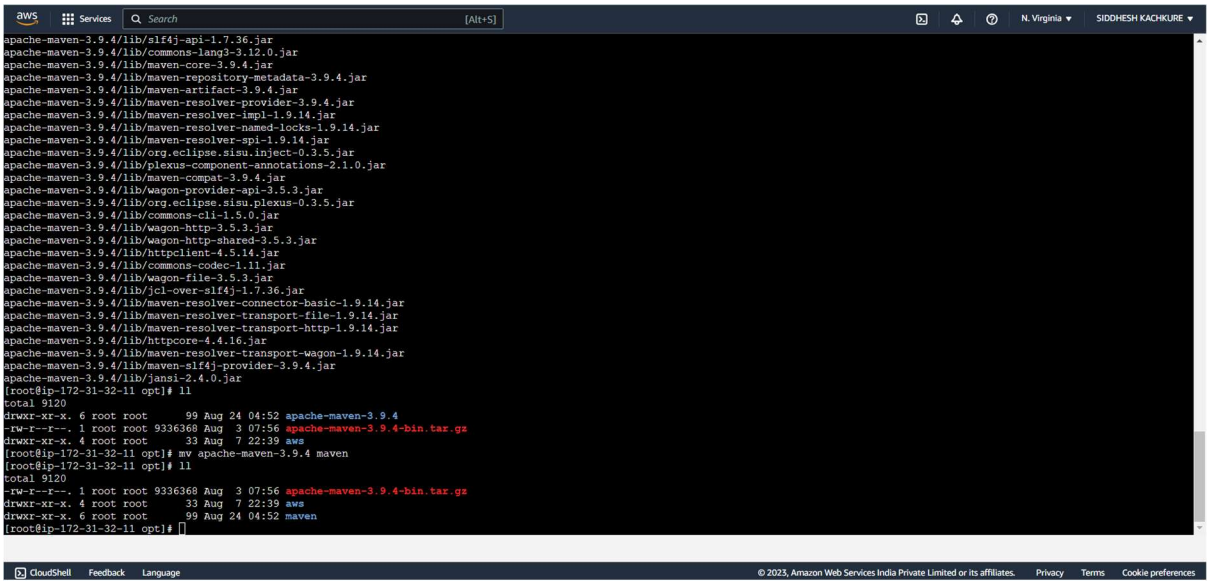
2. Setup M2_HOME and M2 paths in .bash_profile of the user and add these to the path variable

*# vi ~/.bash_profile*

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

M2=$M2_HOME/bin

PATH=<Existing_PATH>:$M2_HOME:$M2

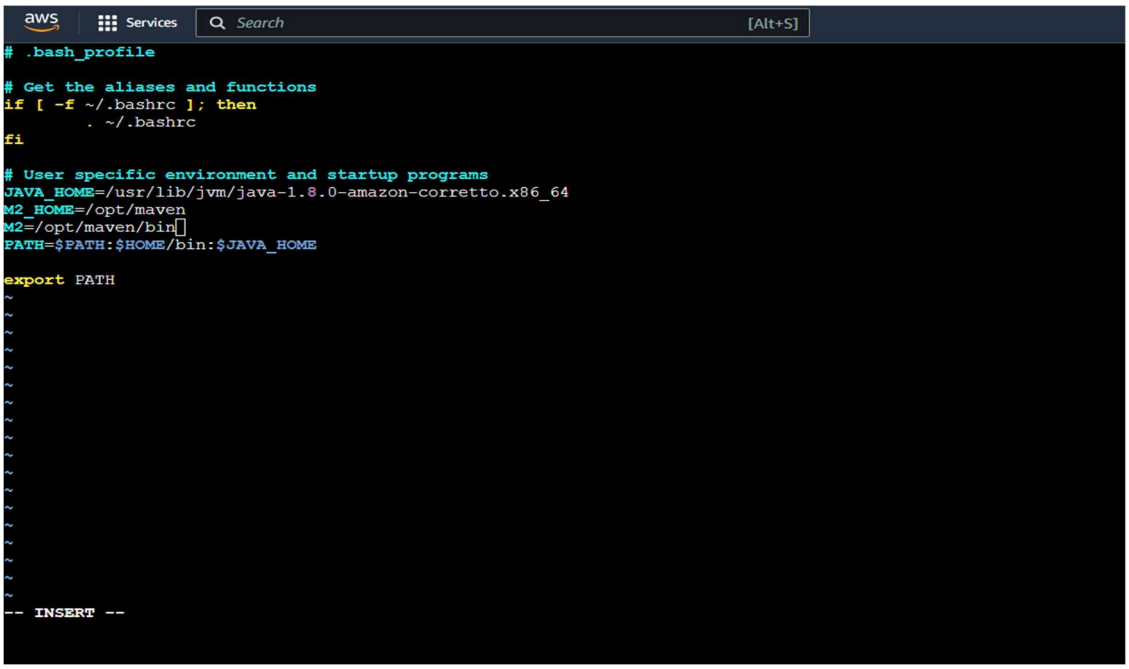



* Checkpoint

1. logout and login to check maven version

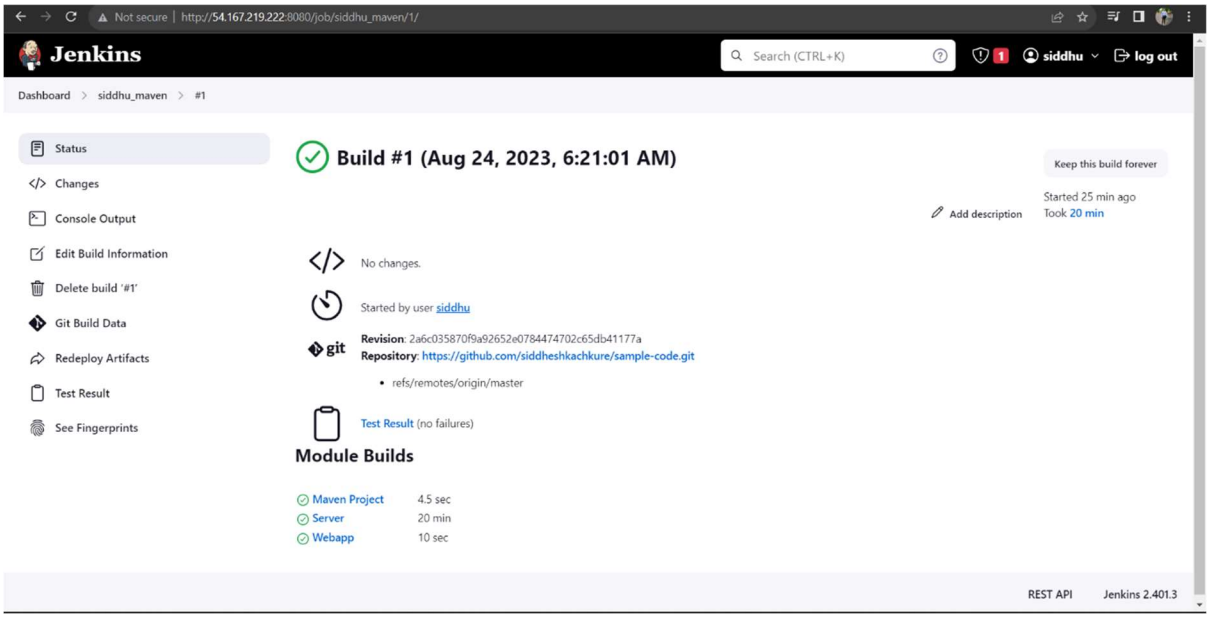
# mvn --version

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.




* 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`




## 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

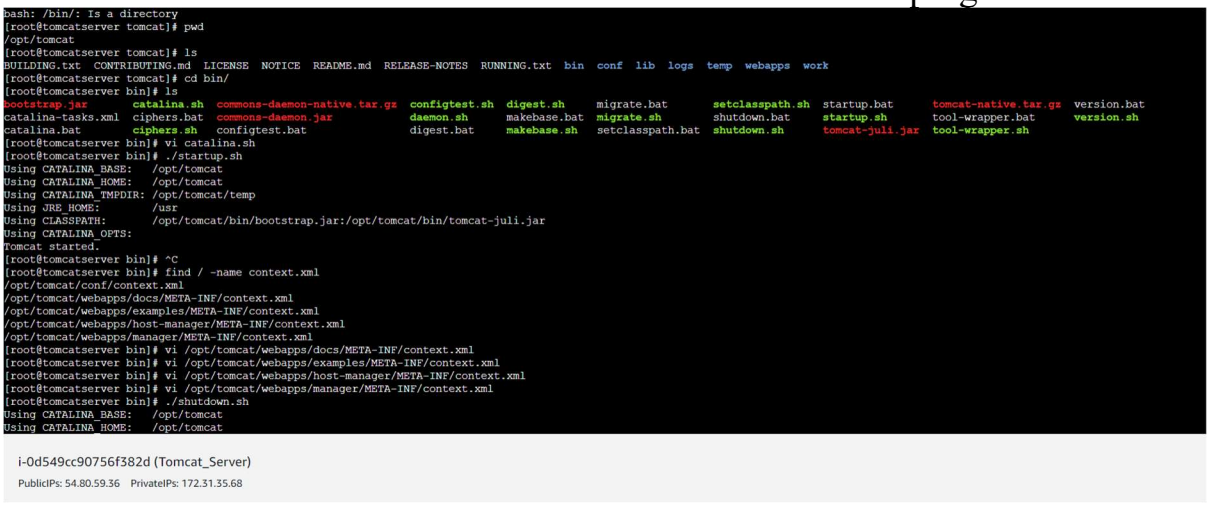
*# cd /opt*

*# 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`




**>>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*

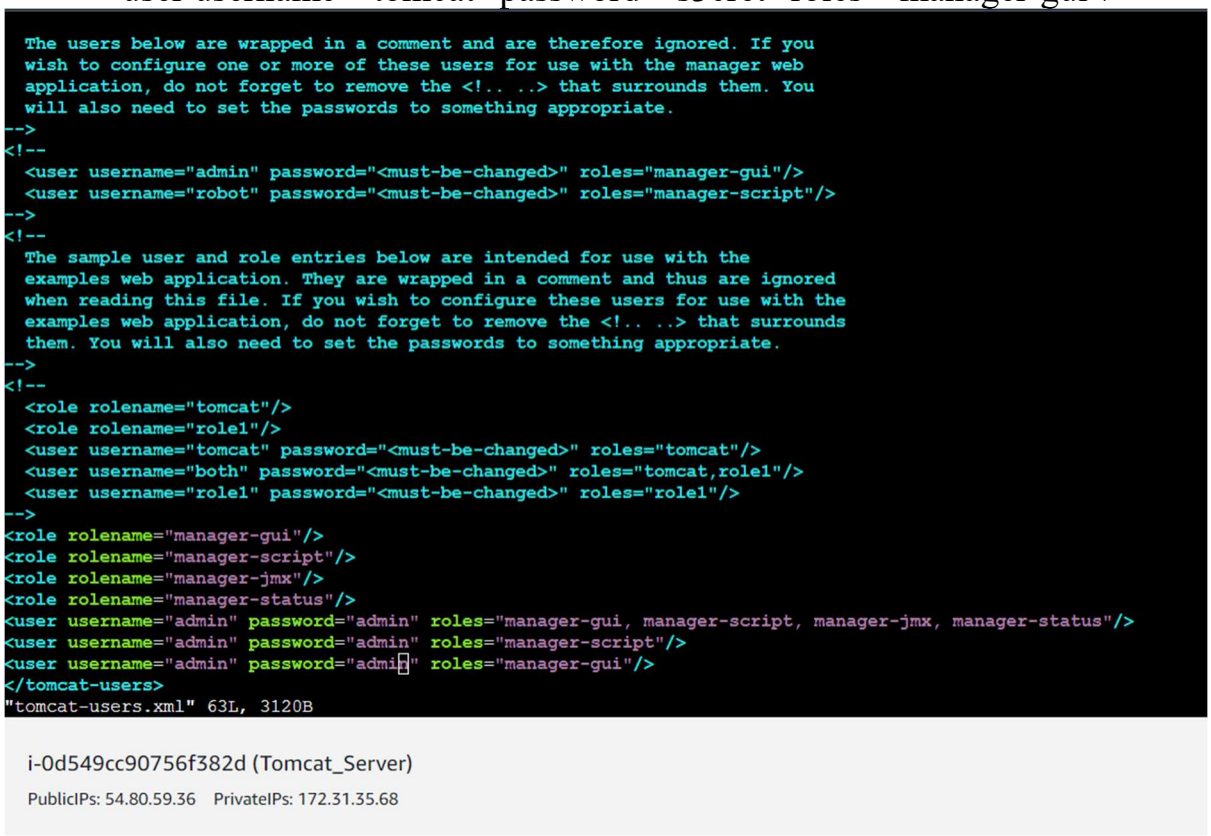
*# find / -name context.xml*

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

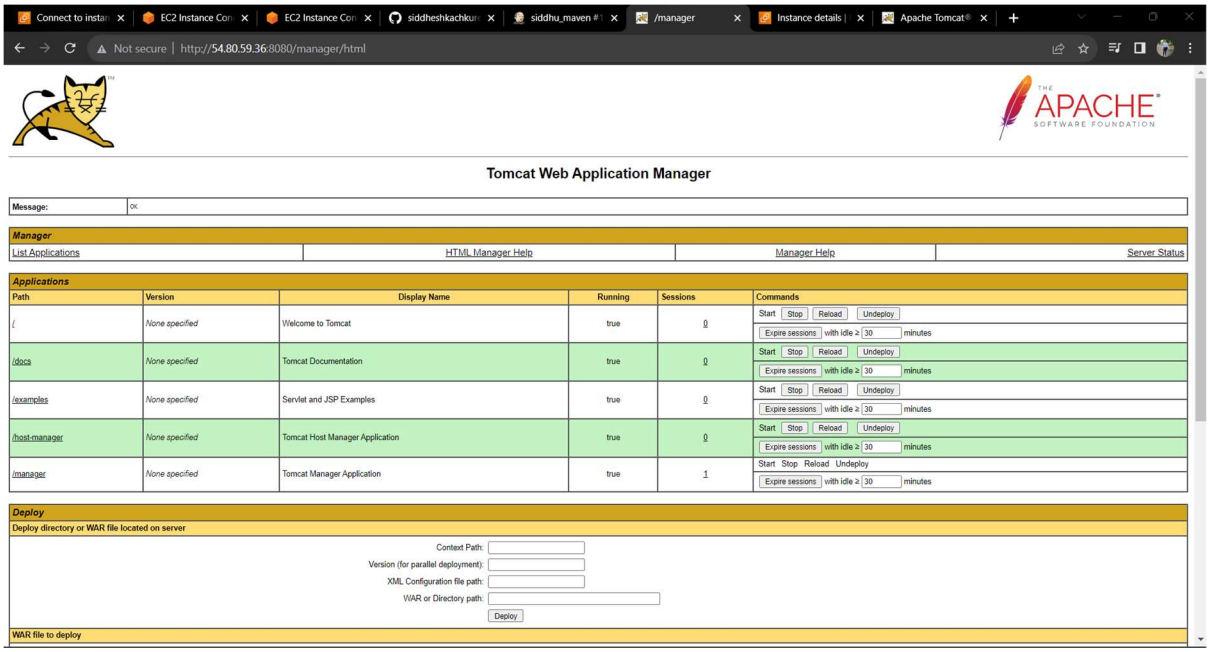
After that restart tomcat services to effect these changes.
At the time of writing this lecture below 2 files are updated.
# /opt/tomcat/webapps/host-manager/META-INF/context.xml
# /opt/tomcat/webapps/manager/META-INF/context.xml
  * Restart tomcat services

3. Update users information in the tomcat-users.xml file
goto tomcat home directory and Add below users to conf/tomcat-users.xml file

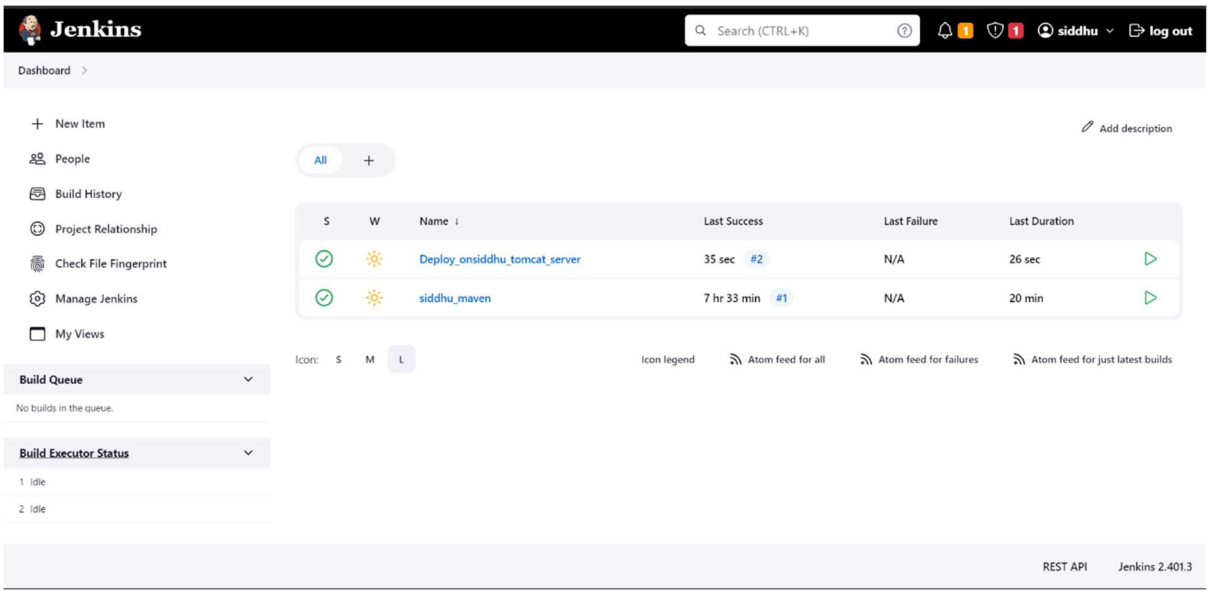
```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>
<user username="deployer" password="deployer" roles="manager-script"/>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

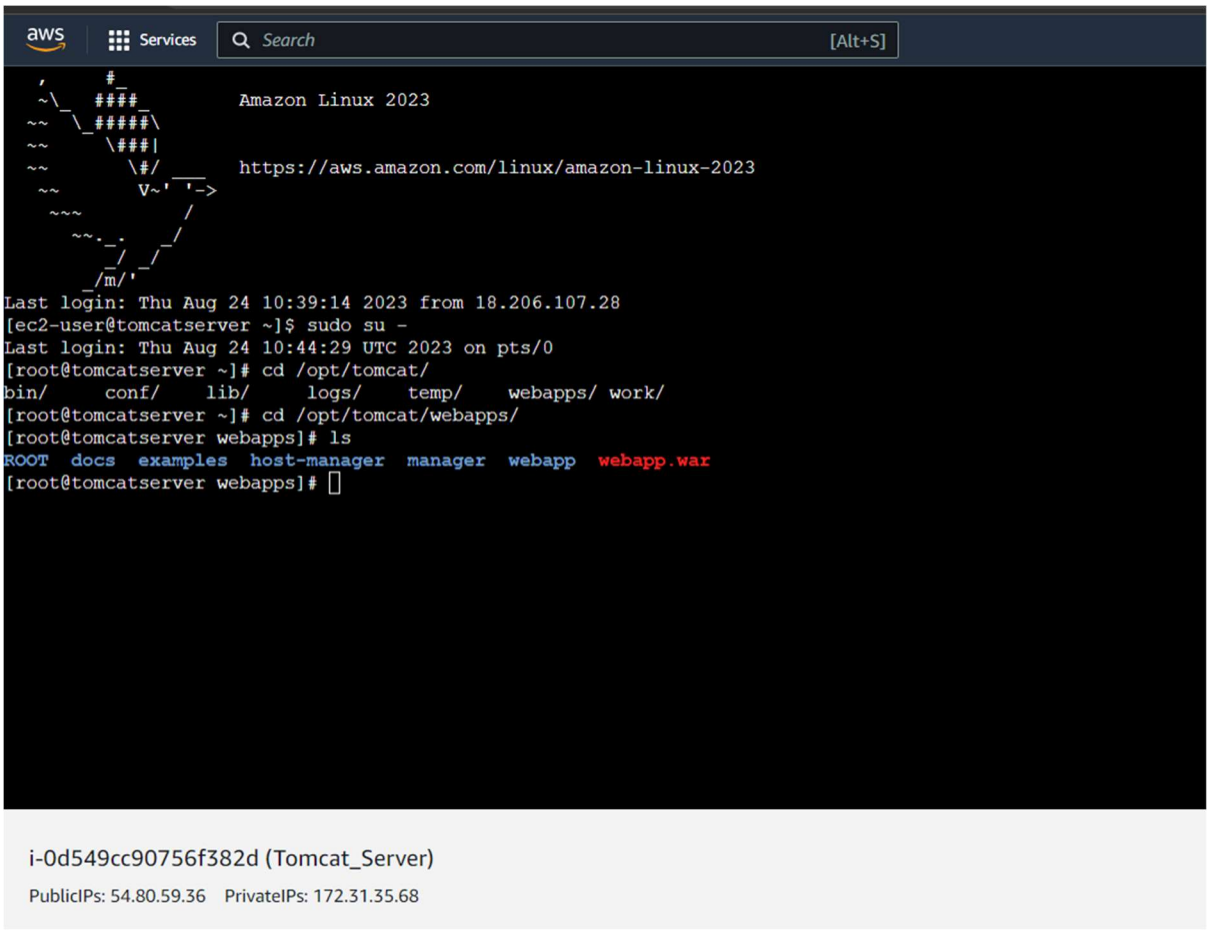



Restart serivce and try to login to tomcat application from the browser. This time it should be Successful




# Deploy a war file on Tomcat VM using Jenkins






**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
*# useradddockeradmin*
*# passwd dockeradmin*

3. add a user to docker group to manage docker
# usermod -aG docker dockeradmin

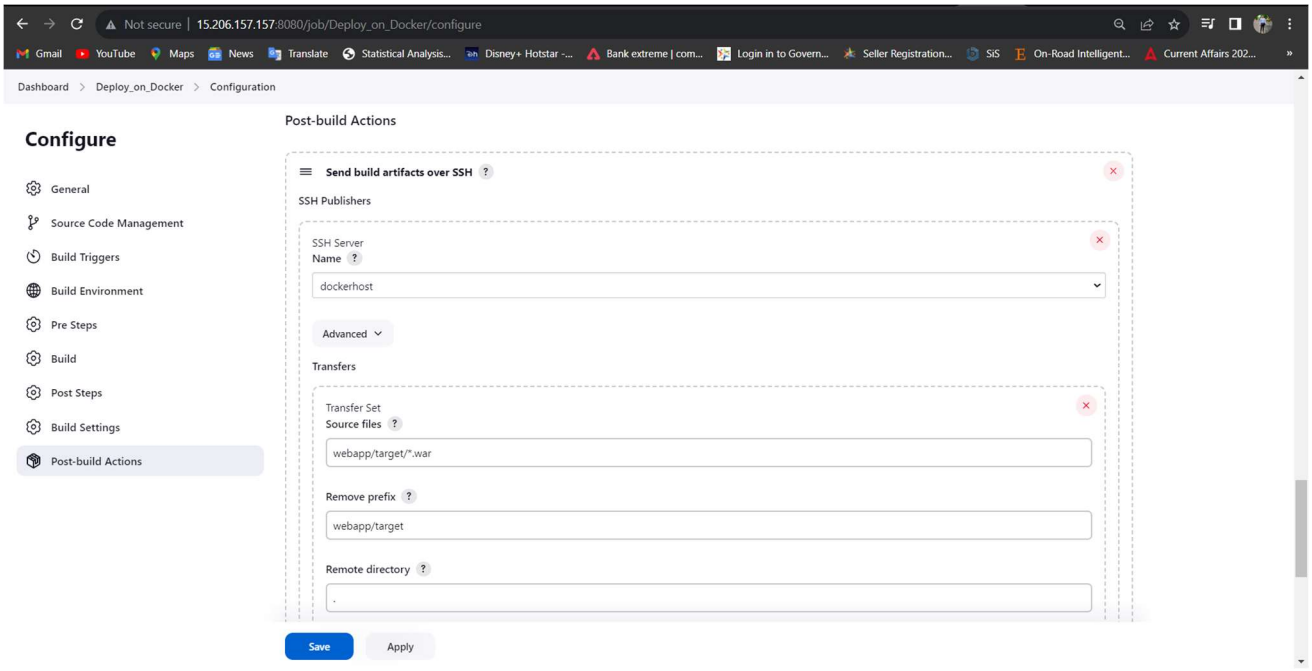
**>>Validation test**

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*
*# useradddockeradmin*
*# 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*
*# useraddansadmin*
*# 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

# 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`

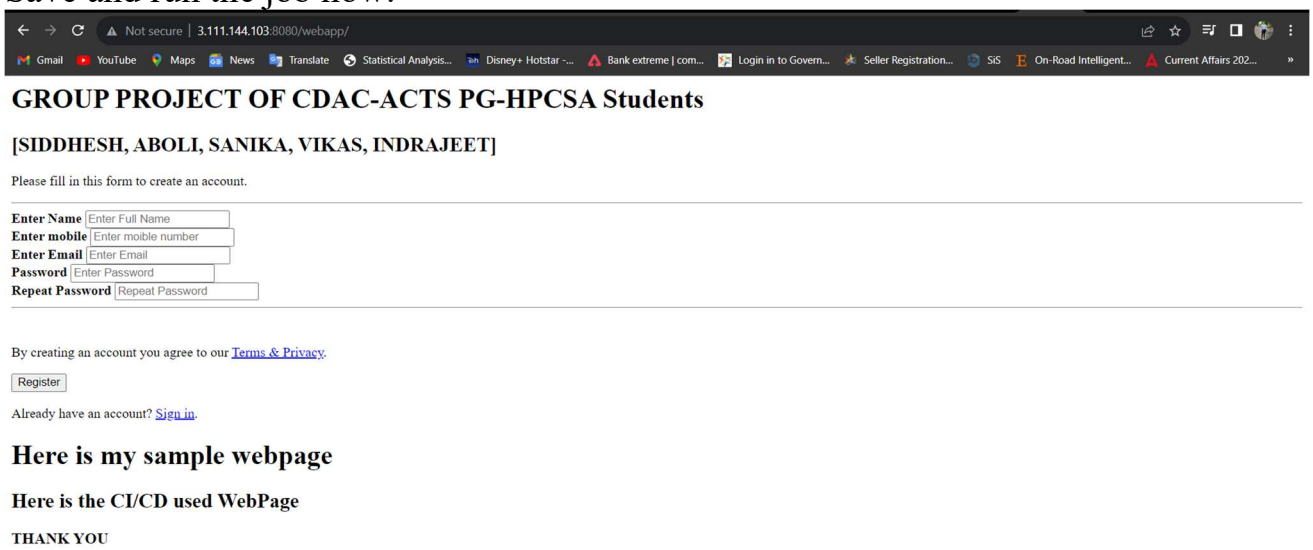
  - *Source Code Management:*
    - Repository: `https://github.com/yankils/hello-world.git`
    - Branches to build : `*/master`
  - *Poll SCM* :      - `* * * * *`

  - *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.




## >> 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)

*useraddansadmin*

*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*

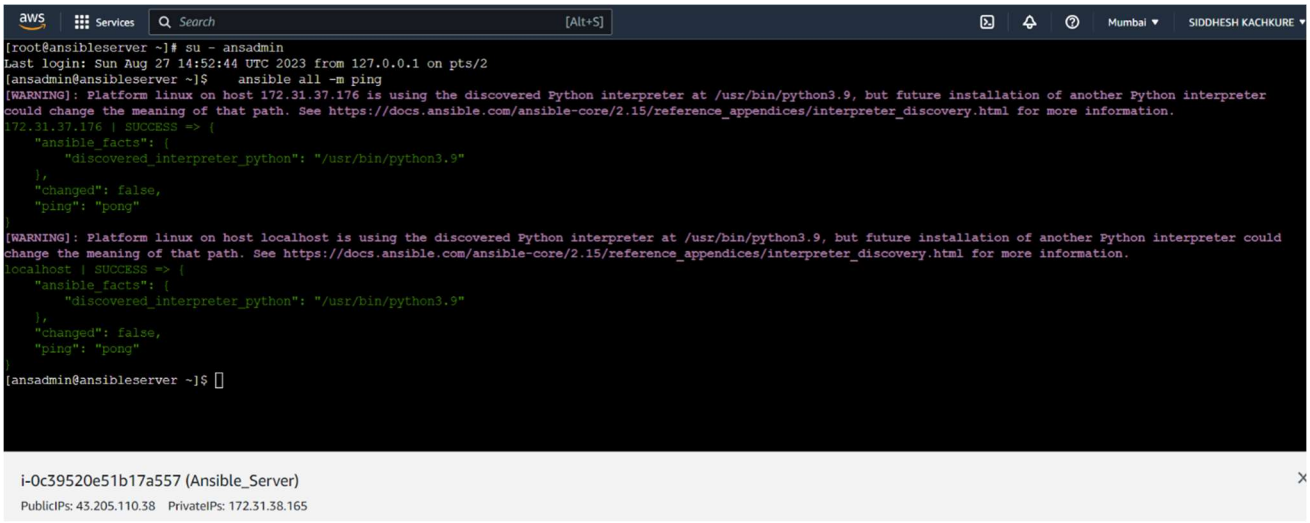
*usermod -aG docker ansadmin*

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*




>> 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

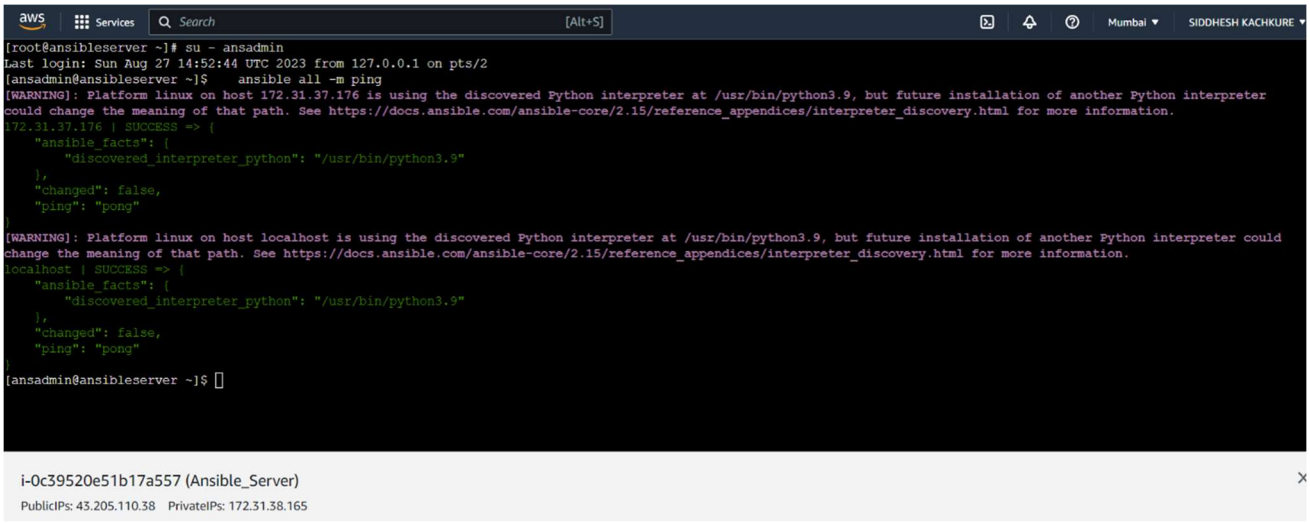
>>Part-01 Integration Setps

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`

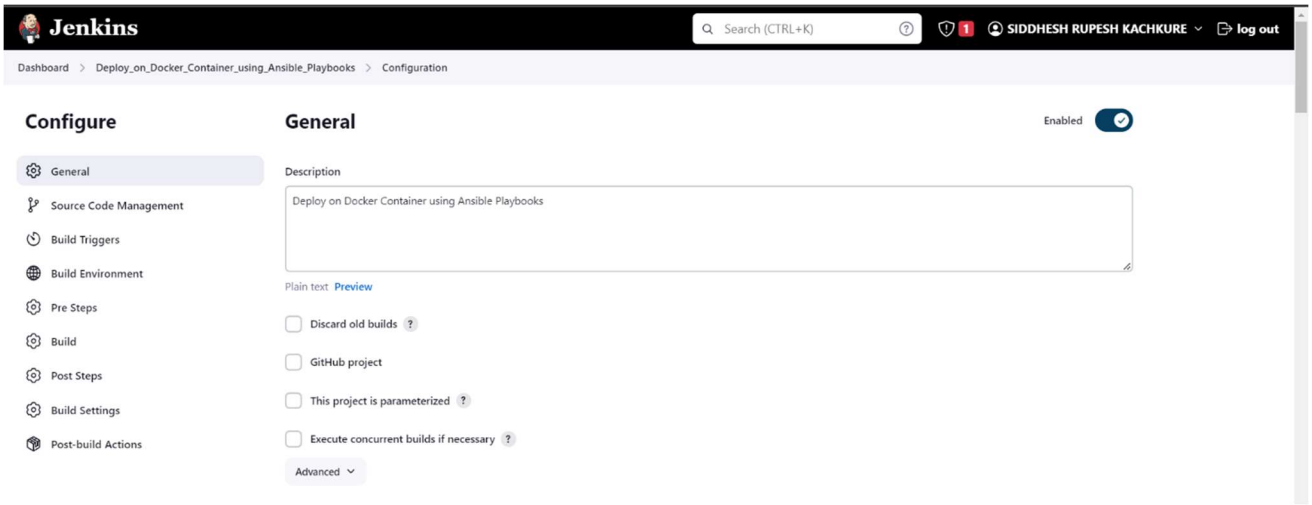
- SSH Servers:
  - Hostname:`<ServerIP>`
  - username: `admin`
  - password: `admin`

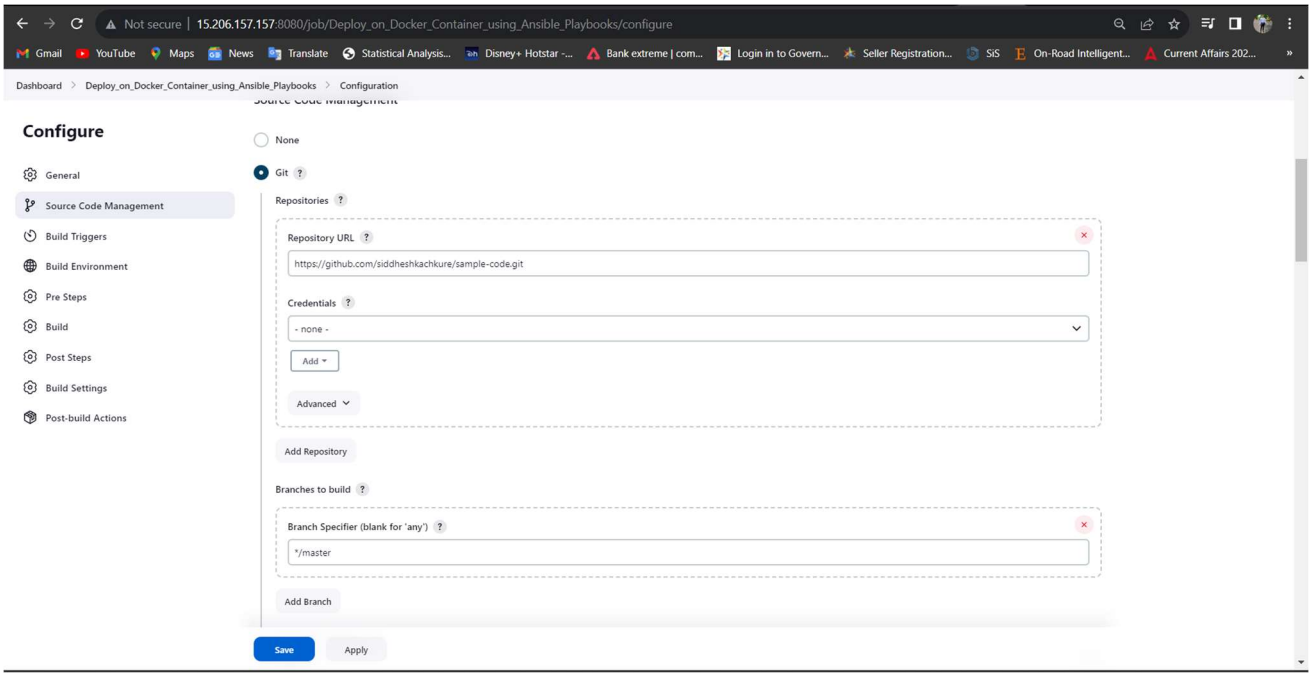
Test the connection "Test Connection"

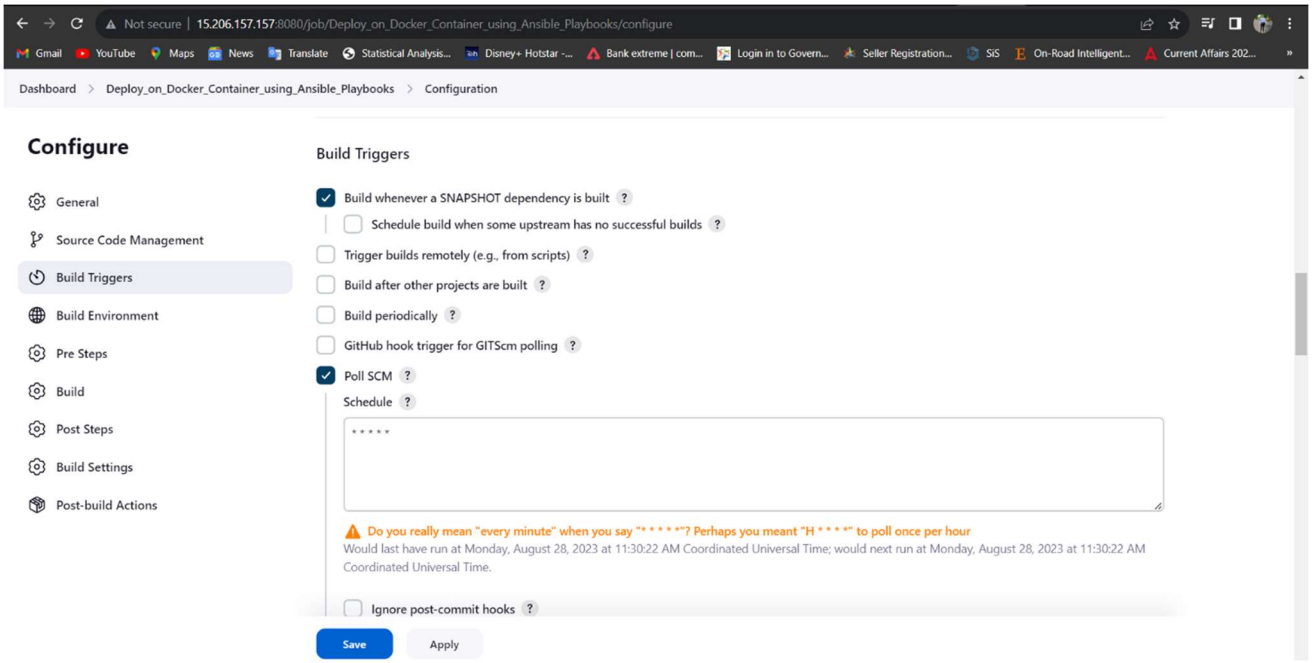


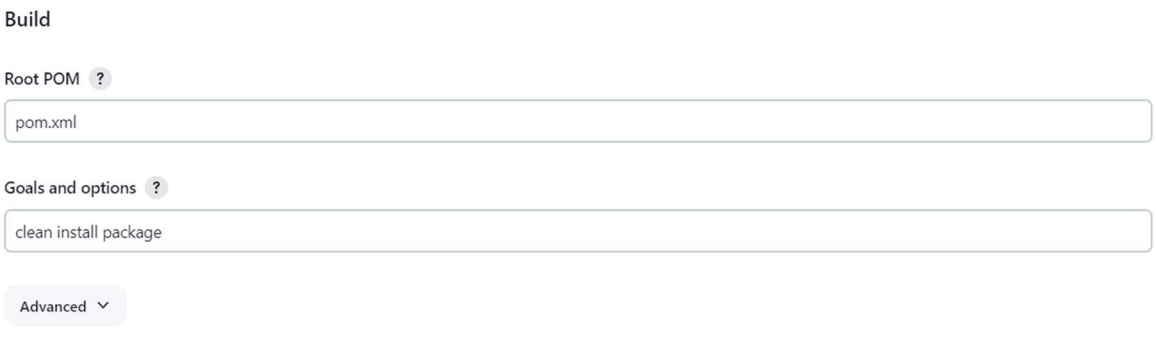

>> Deploy on a docker container using Ansible
>>*Jenkins Job name:* `Deploy_on_Container_using_ansible_playbooks`

## **Pre-requisites**
1. Jenkins server
2. Docker-host server
3. Ansible server




Not secure | 15.206.157.157:8080/job/Deploy_on_Docker_Container_using_Ansible_Playbooks/configure
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
None
Git
Repositories
Repository URL
https://github.com/siddheshkachkure/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
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

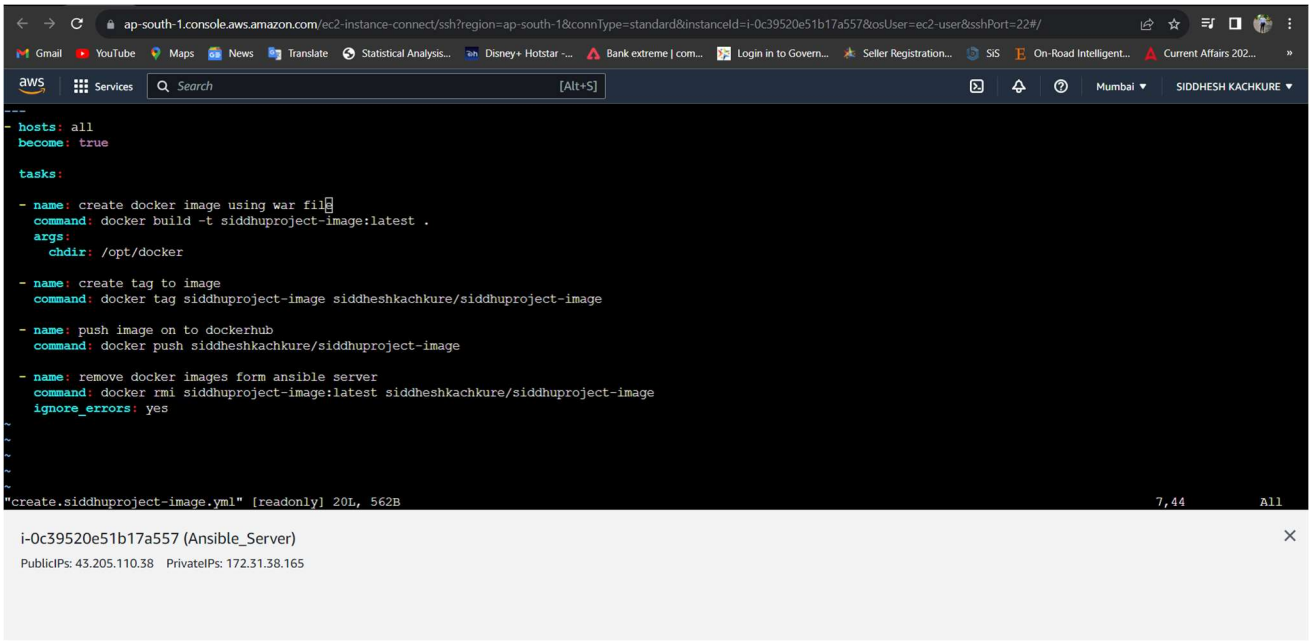

Build
Root POM
pom.xml
Goals and options
clean install package
Advanced

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

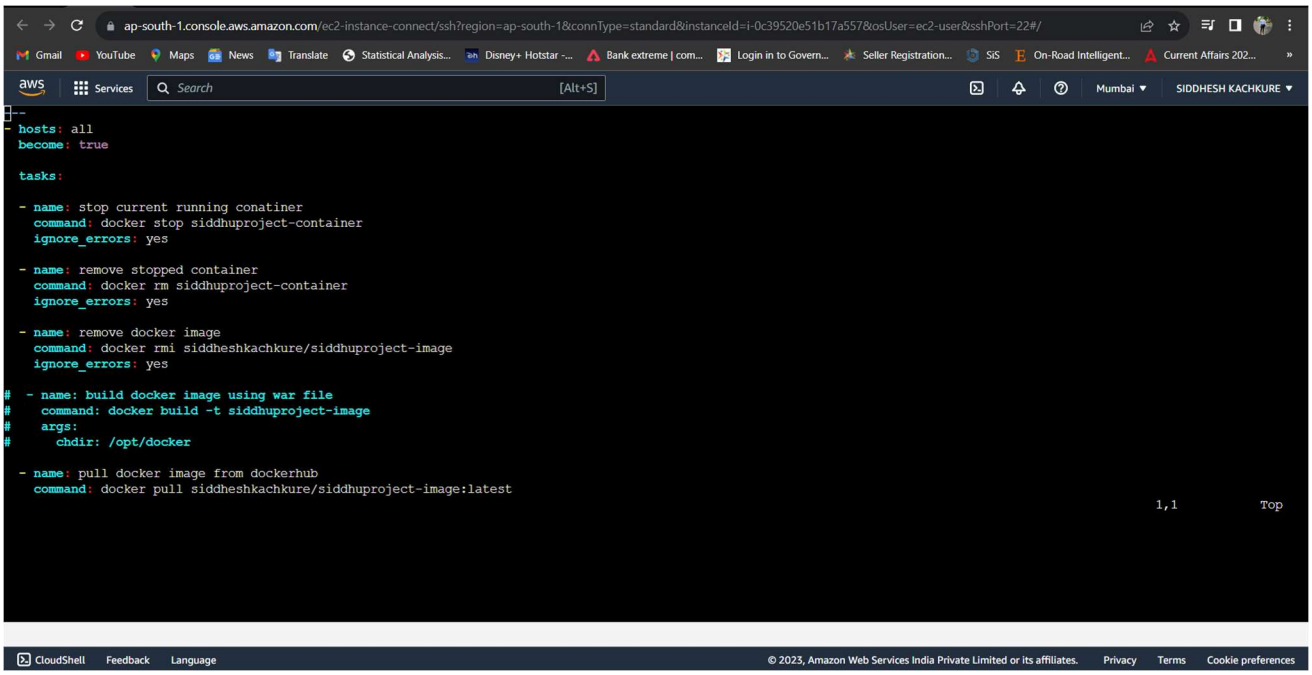
```
cat create.siddhuproject.yml
---
- hosts: all
  become: true

  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 rmisiddheshkachkure/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
```




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 rmisiddhuproject-image:latestsiddheshkachkure/siddhuproject-
image
ignore_errors: yes ### Integration between Ansible-control-node and Jenkins
```




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: `*******`

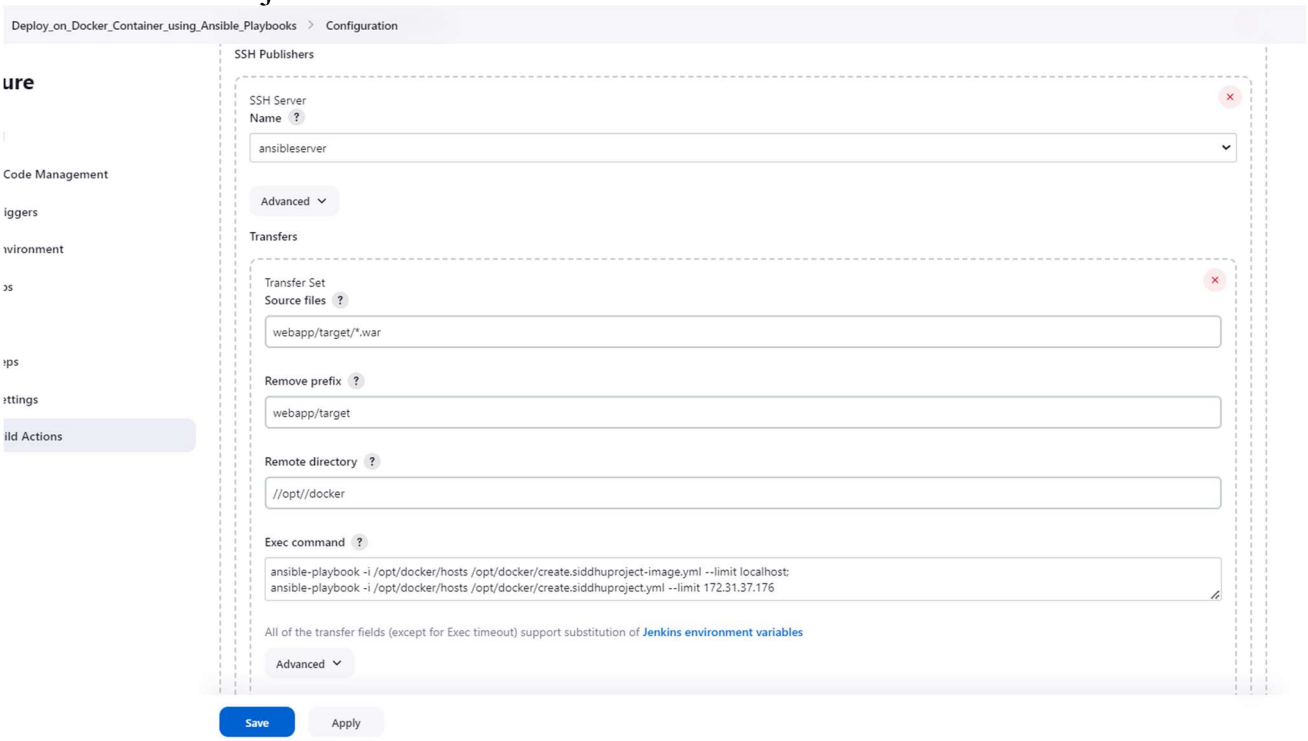
## Steps to create "Deploy_on_Container_using_ansible" Jenkin job
## From Jenkins home page select "New Item"
- Enter an item name: `Deploy_on_Container_using_ansible`
- Copy from: `Deploy_on_Container`

- *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`

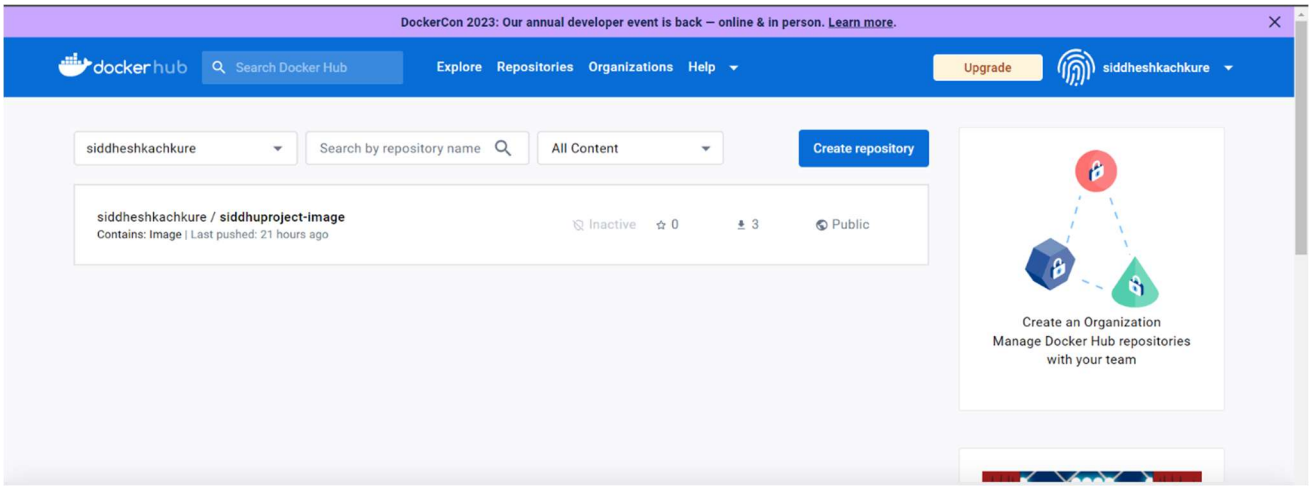
- *Post-build Actions*
  - Send build artifacts over SSH
    - *SSH Publishers*
     - SSH Server Name: `ansible-server`
      - `Transfers` > `Transfer set`
          - Source files: `webapp/target/*.war`
              - Remove prefix: `webapp/target`
              - Remote directory: `//opt//docker`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

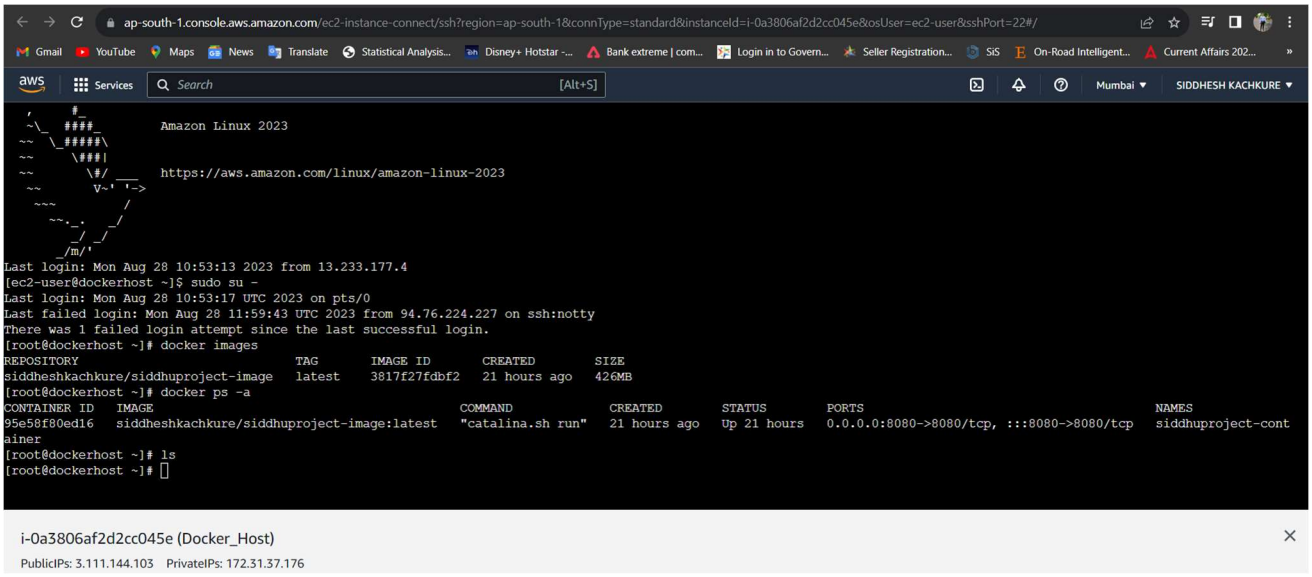



So now on Jenkins server we have installed git run following commands on the Jenkins_Server

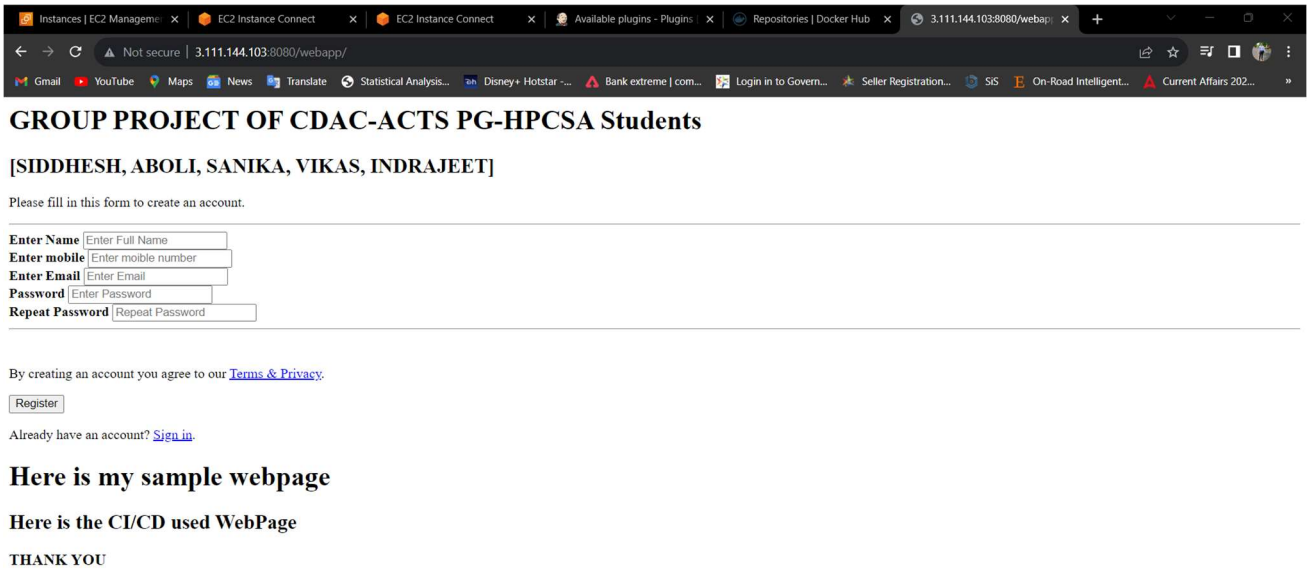
Now you can see here are only 3 time the image is pushed on docker hub




We can see here in Docker-Host the previous images



First see our ready web page



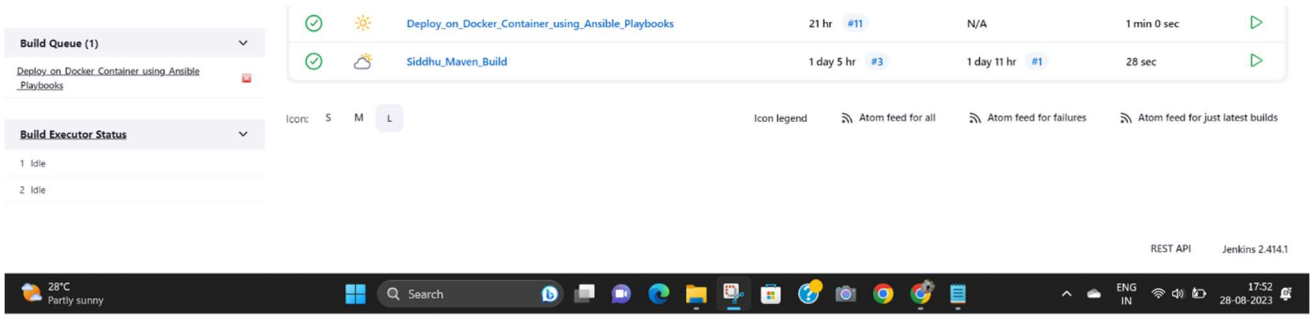

Now we will change our code by commit in GitHub in Jenkins Server

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

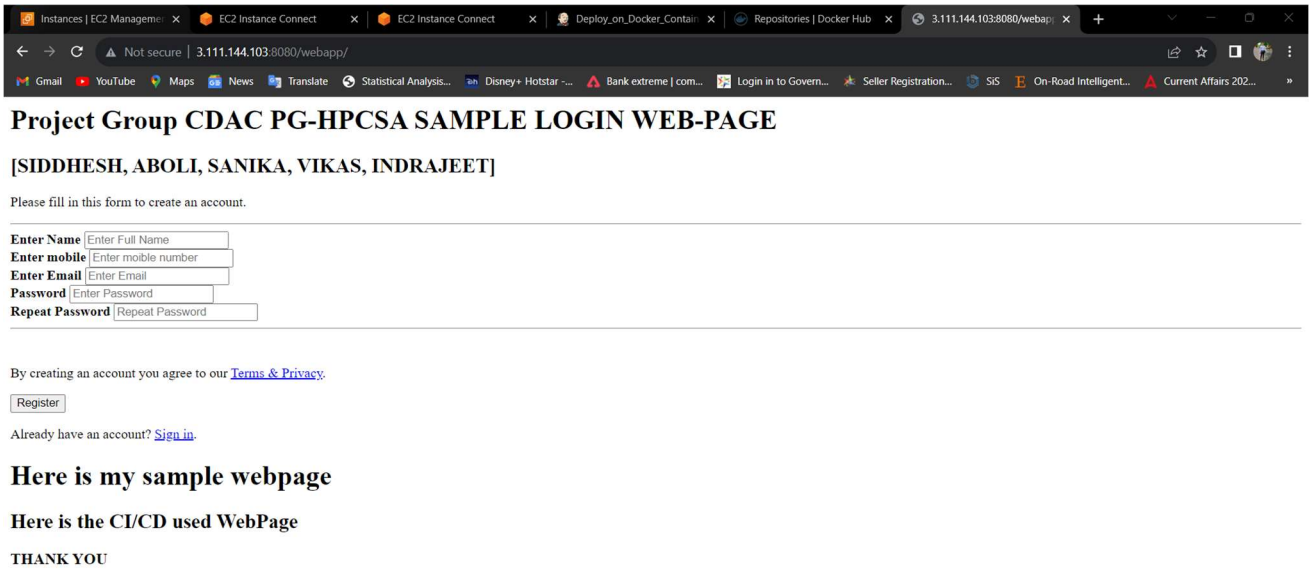
[root@jenkinsserver~]# ls
sample-code
[root@jenkinsserver~]# cd sample-code/
[root@jenkinsserver sample-code]# ls
Dockerfile README.md pom.xml regapp-deploy.ymlregapp-service.yml
server webapp
[root@jenkinsserver sample-code]# cd webapp/src/main/webapp/WEB-INF/
[root@jenkinsserver WEB-INF]# ls
web.xml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

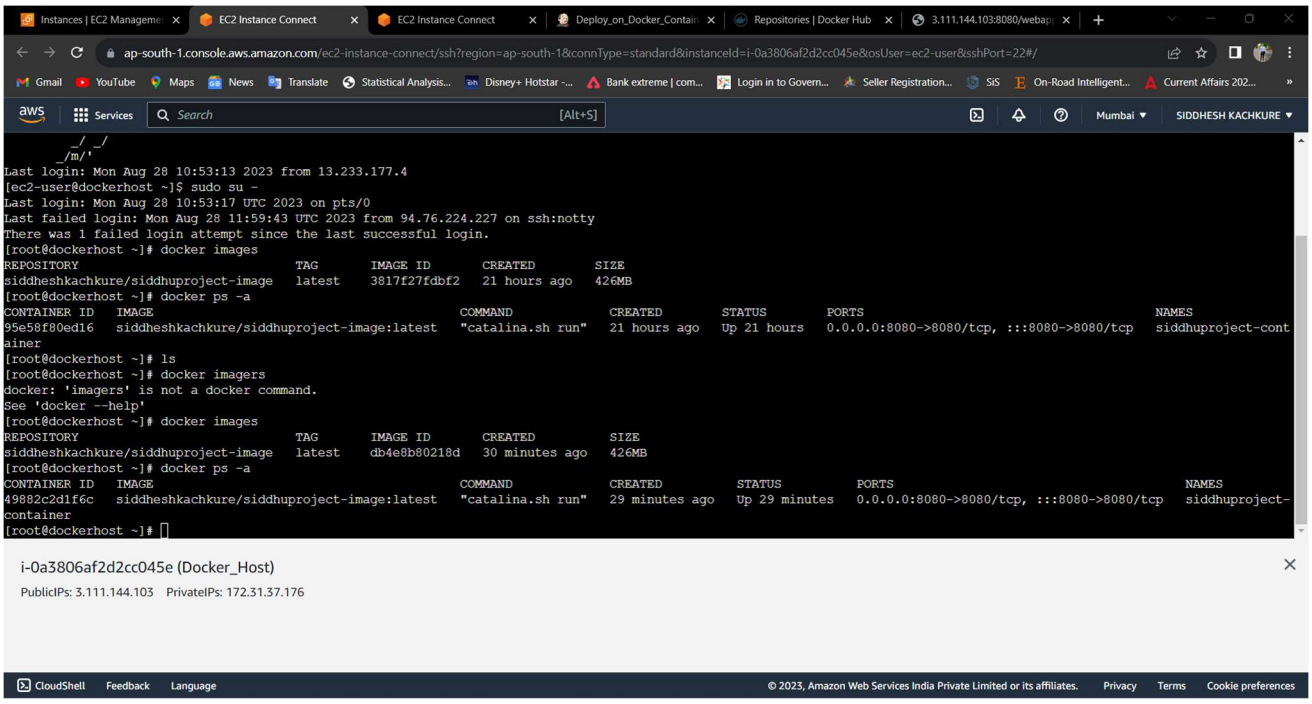


Then we will see our change in webpage.:

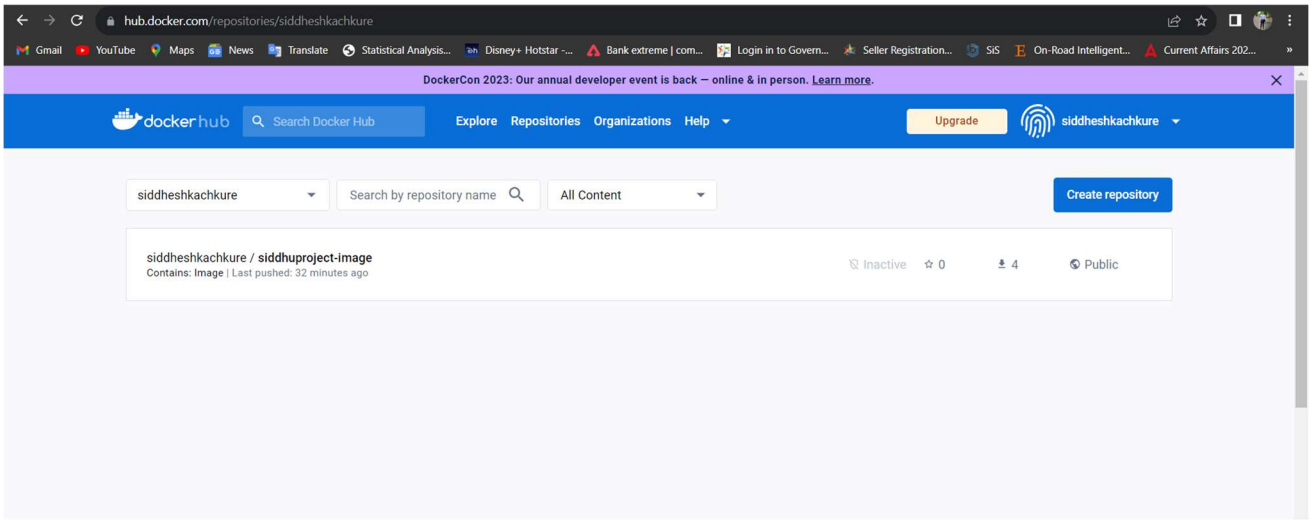



---

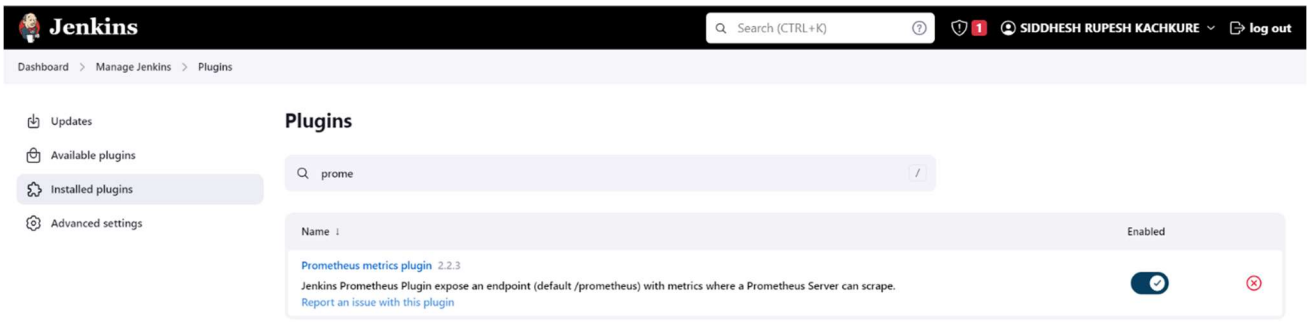
Here you can see previous Image & Container created 21 hours ago and then we can see the latest Image & Container created 29 minutes ago and the Status is “Up”



Then we will see our change in DockerHub :

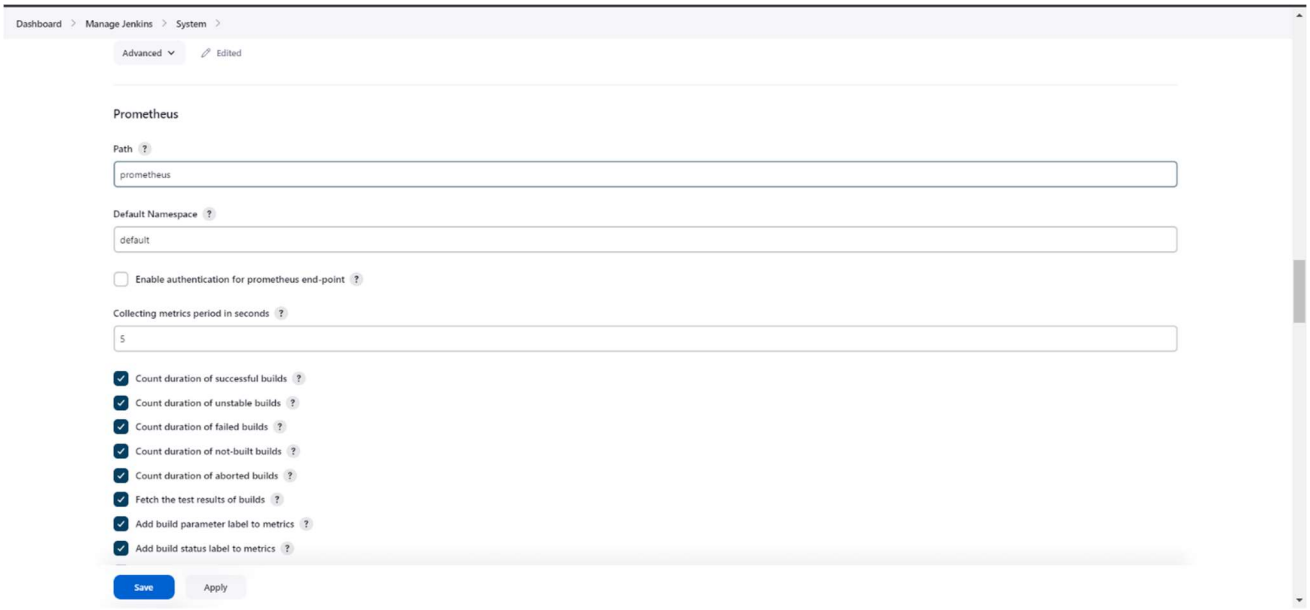



>> Now we will integrate Prometheus and Grafana in our Jenkins




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




# 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 promethus we need to make few changes in prometheus config file >

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

/promethues

# cd /etc/prometheus

# vi prometheus.yml

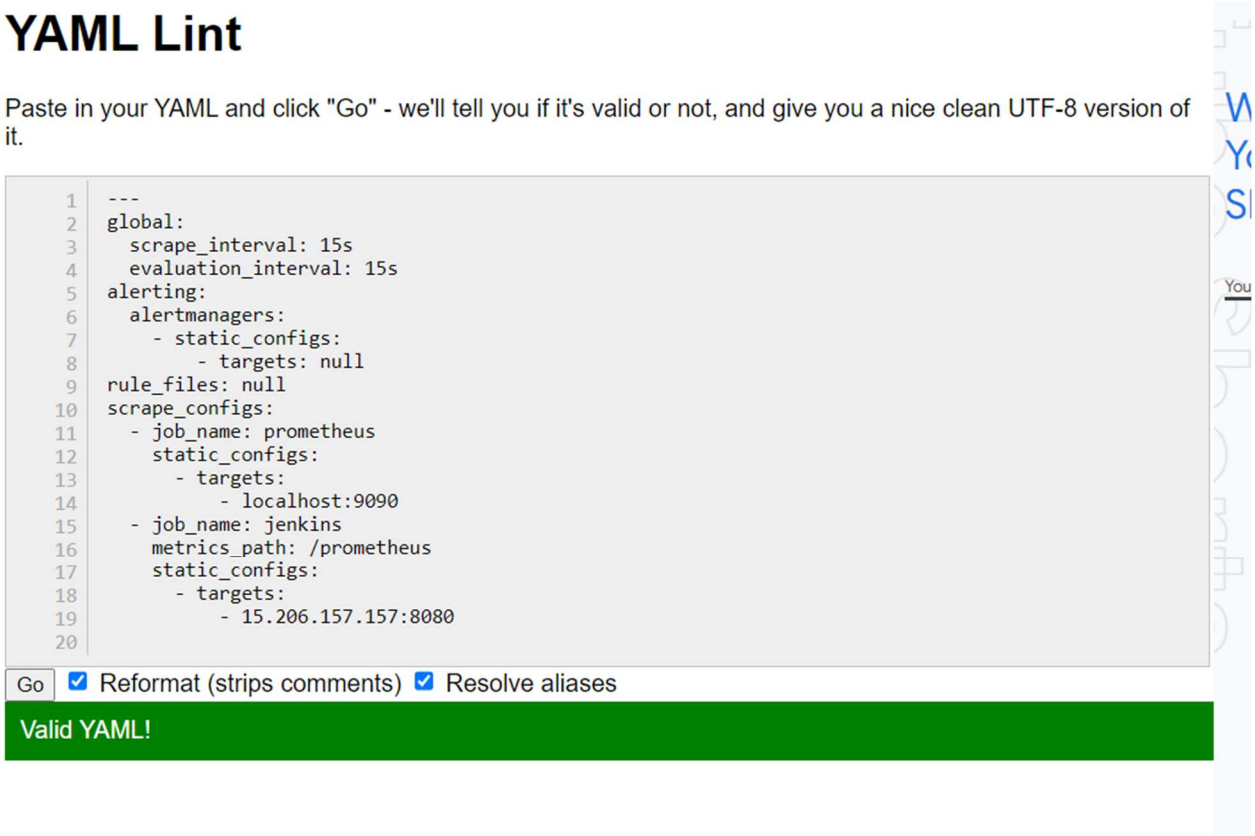
- job_name: "jenkins"

metrics_path: /prometheus

static_configs:

- targets: ['15.206.157.157:8080']

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




---

# 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']
```

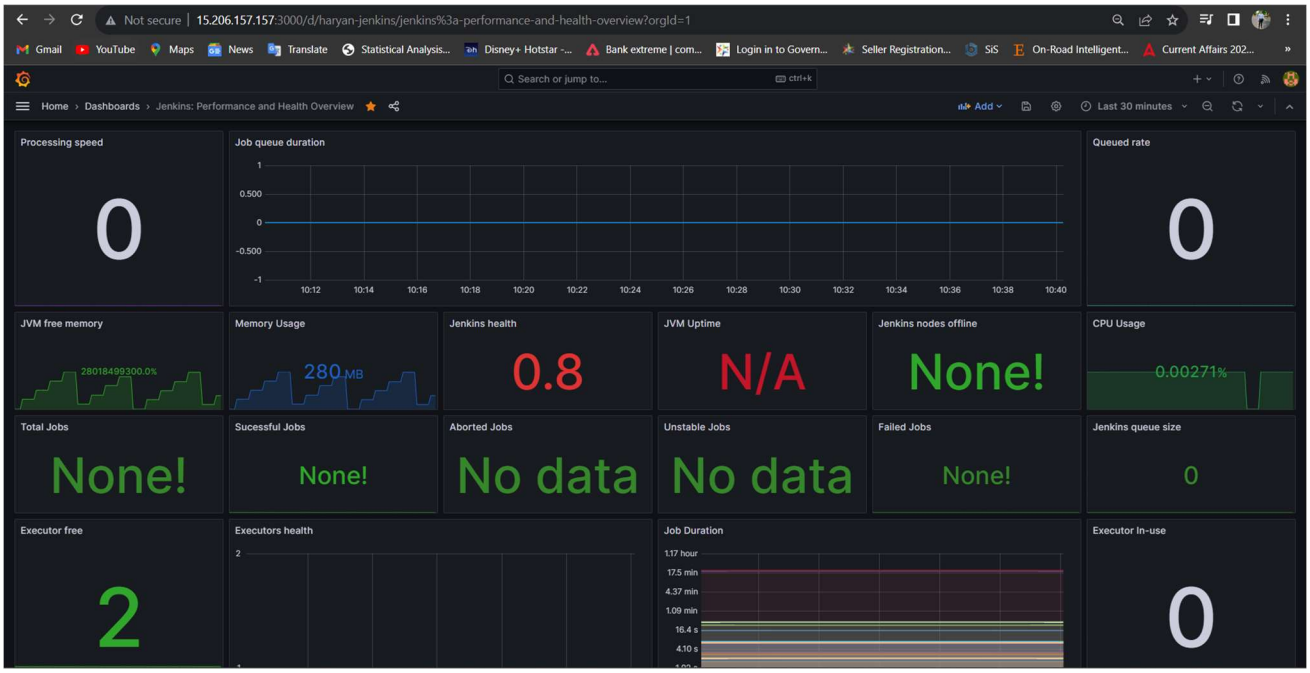
---

> after edit in yaml restart the prometheusconatiner>

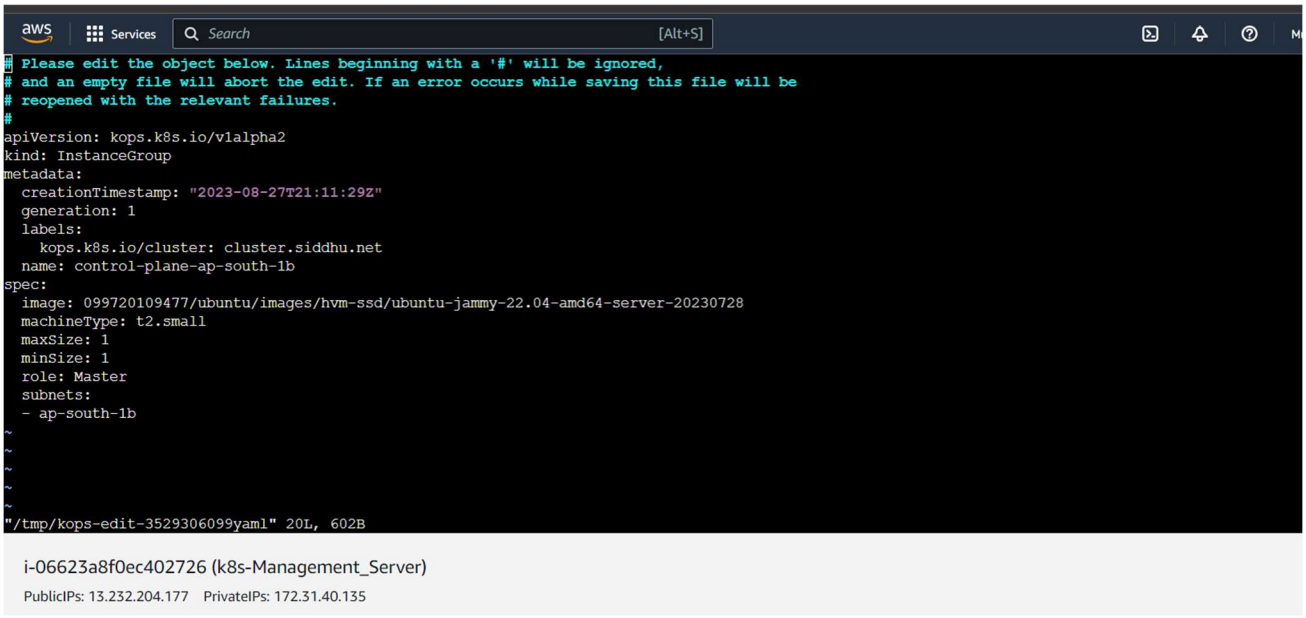
```
        # docker restart <proemtheus-container-id>
```

>> 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




**>>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*
*sudomv ./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)

  Routeh53 --> hosted zones --> created hosted zone
  Domain Name: siddhu.net
  Type: Private hosted zone for Amzon 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 kubernetescluser

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

14. Validate your cluster

*kops validate cluster*

15. To list nodes

*kubectl get nodes*

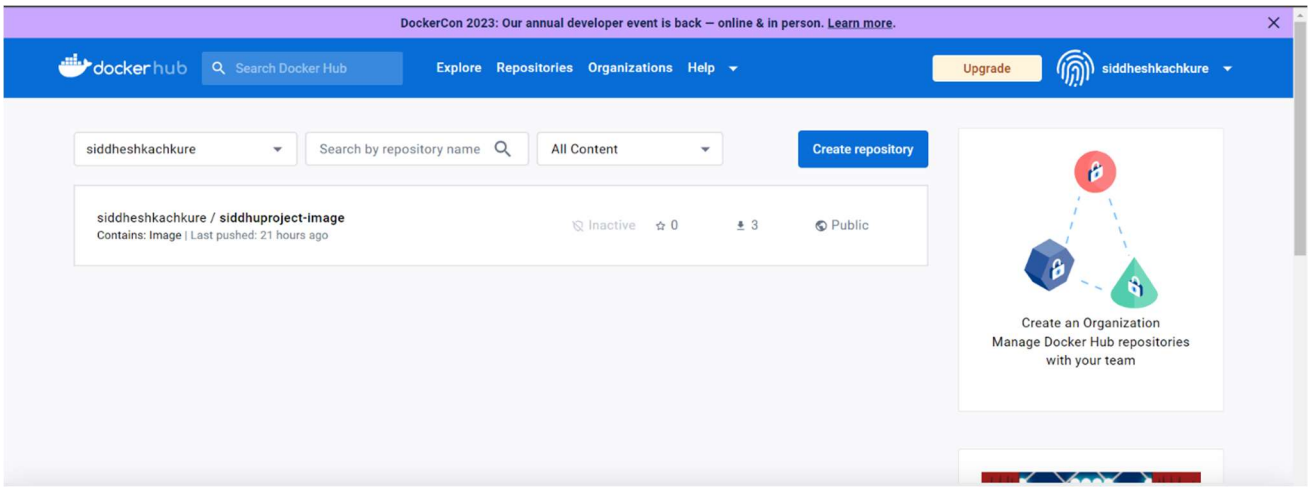
16. To delete cluster

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

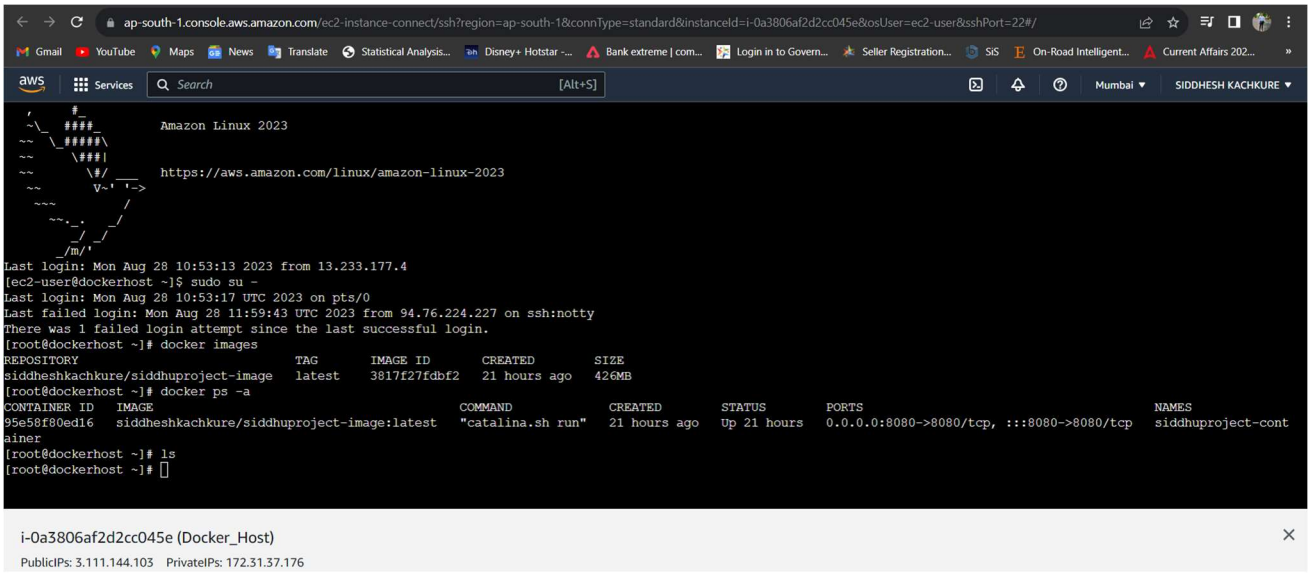
7. Results

```
[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]$
```

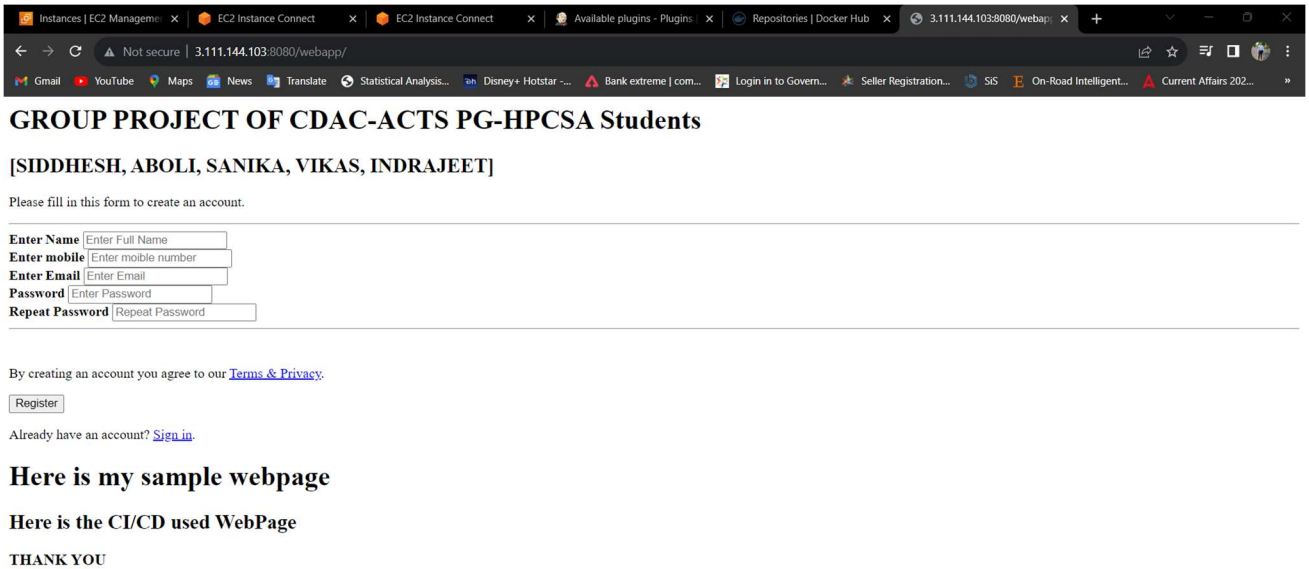
Here you can see .war file on <ansible server> it will change after the change in code



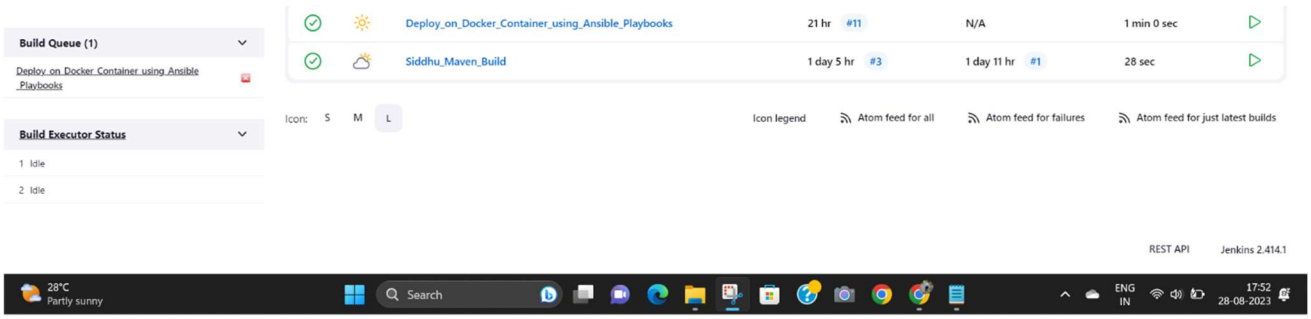

Here You can see 3 push in docker-hub repository



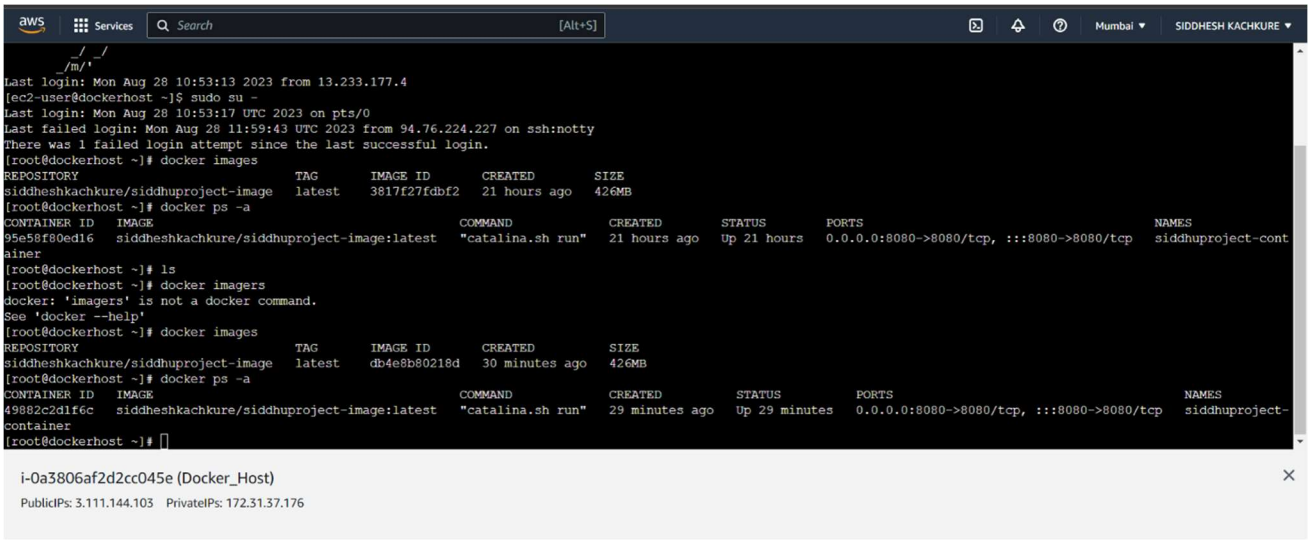

Here in <docker_host> we have previous created image and container running after the change in code it will delete and new image and container will come instead of it .



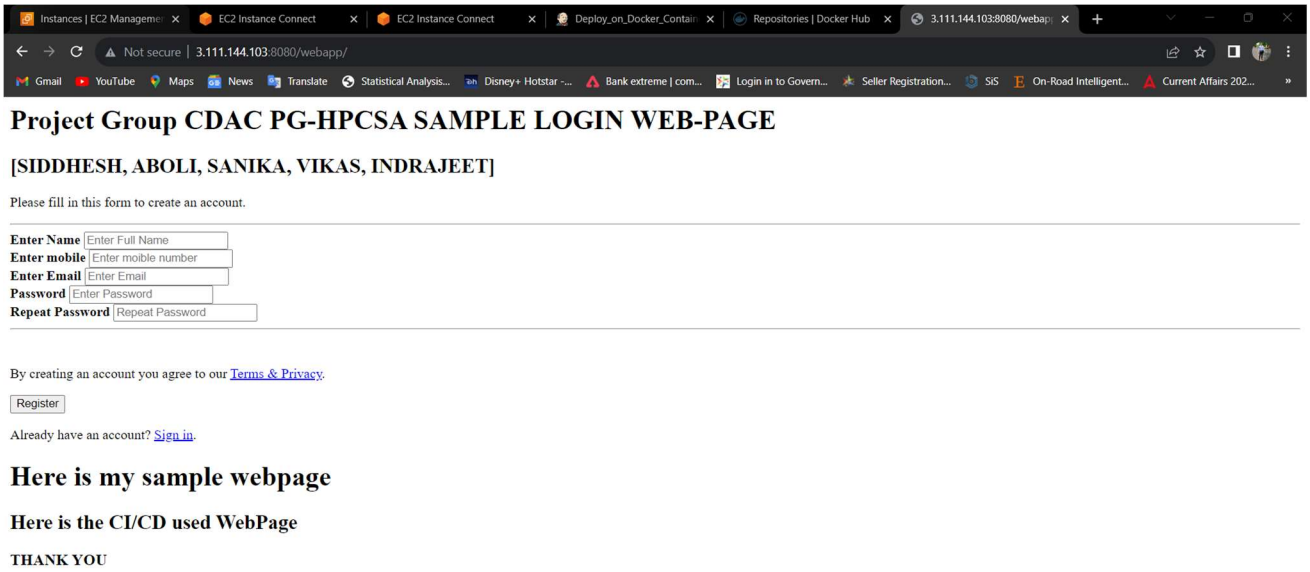
This is Web Page before changing the code



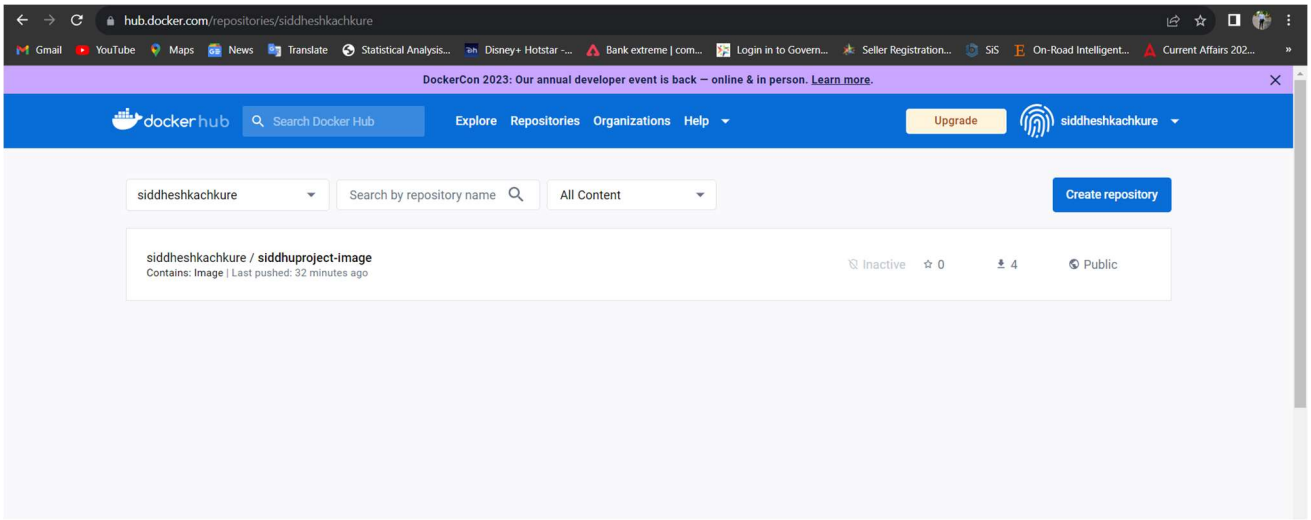

After changing and pushing the code in github the ansible will run a job on Jenkins automatically you can see the Deloy in the Build Queue:




See here is the change in the image and container in the <docker_host>




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/Pharak123/My_Projects.git
   https://github.com/indra7011/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/