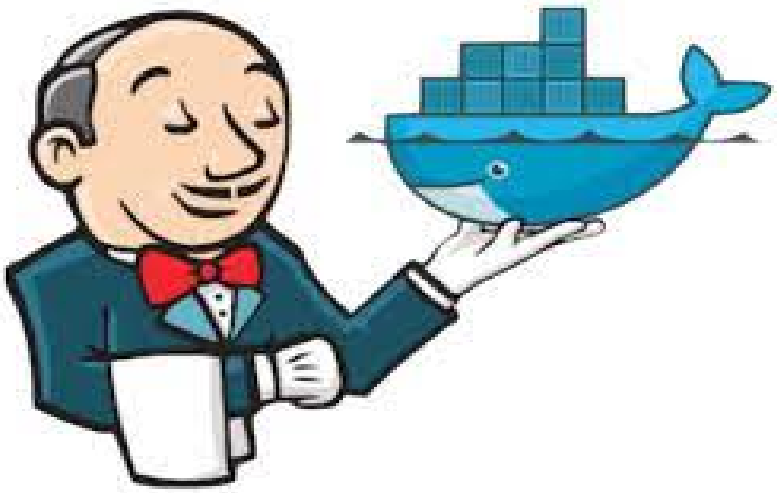
cprime

Jenkins Pipeline for Docker Deployment



NAME : T MANOJ

EMPID : LYAKE2KHS

Introduction

This documentation outlines a continuous integration and continuous deployment (CI/CD) solution implemented using Jenkins, Docker, and GitHub on an Amazon Linux EC2 instance. The system automates the process of building, testing, and deploying a web application whenever changes are pushed to the GitHub repository. This automation streamlines the development workflow, ensures consistency between environments, and enables rapid deployment of new features or bug fixes.

Overview

The solution consists of three main components:

1. **Amazon EC2 with Amazon Linux**: Provides the hosting environment
2. **Jenkins**: Orchestrates the CI/CD pipeline
3. **Docker**: Containerizes the application for consistent deployment
4. **GitHub**: Stores the source code and triggers the pipeline via webhooks

These components work together to create a fully automated deployment pipeline. When a developer pushes code to the GitHub repository, a webhook triggers Jenkins to start the pipeline. Jenkins clones the repository, builds a Docker image, removes any previous container, deploys a new container, and verifies the deployment.

Service Descriptions

Amazon Linux EC2

Description: Amazon Linux EC2 is a virtual server in Amazon's Elastic Compute Cloud that runs Amazon's Linux distribution. It serves as the hosting environment for both Jenkins and Docker, providing the necessary infrastructure to build and deploy applications.

Key Features:

- Linux-based operating system optimized for cloud environments
- Pre-configured for AWS integration
- Regular security updates
- Cost-effective computing resources
- Flexible instance sizing options

Docker

Description: Docker is a platform that uses containerization technology to package applications with their dependencies into standardized units called containers. These containers ensure consistent behavior across different environments.

Key Features:

- Application isolation through containers
- Consistent environments from development to production
- Efficient resource utilization
- Fast deployment and scaling
- Version control for application images

Jenkins

Description: Jenkins is an open-source automation server that enables developers to build, test, and deploy their applications. It supports the creation of pipelines using Jenkinsfile, which defines the entire deployment process as code.

Key Features:

- Extensible with plugins
- Pipeline-as-code support
- Integration with version control systems
- Automated build, test, and deployment
- Distributed builds across multiple machines
- Reporting and notification capabilities

GitHub

Description: GitHub is a web-based Git repository hosting service that provides source code management, version control, and collaboration features. In this setup, it stores the application code, Dockerfile, and Jenkinsfile.

Key Features:

- Version control using Git
- Collaborative development
- Issue tracking
- Webhooks for integration with other services
- Pull request workflow
- Code review capabilities

Nginx (in Docker container)

Description: Nginx is a high-performance web server and reverse proxy server used to serve the web application. It's installed in the Docker container as specified in the Dockerfile.

Key Features:

- High-performance HTTP server
- Load balancing
- Caching
- Reverse proxy functionality
- Low memory footprint
- Handling of static content

Pipeline Process Details

The Jenkins pipeline defined in the Jenkinsfile consists of several stages that automate the application deployment:

1. **Clone Repo**: Jenkins clones the GitHub repository containing the application code, Dockerfile, and Jenkinsfile.

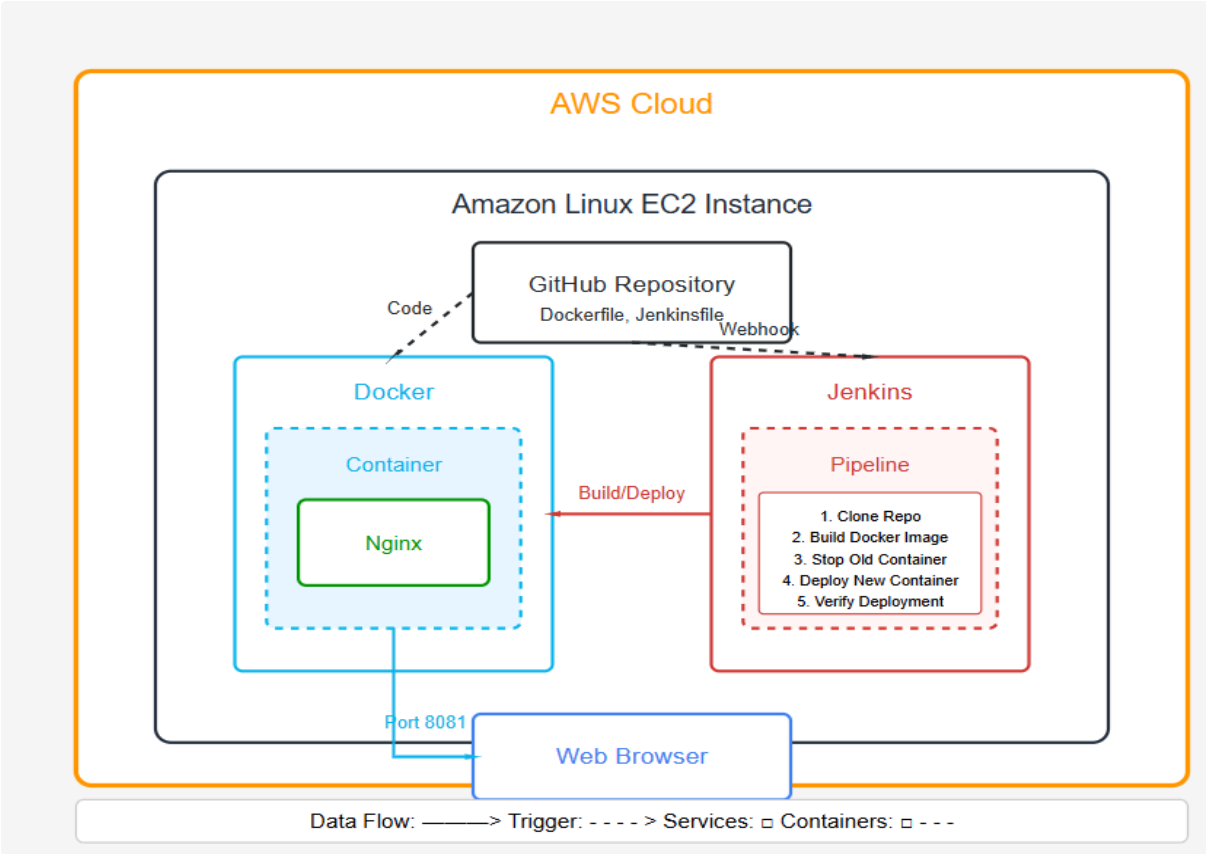
2. **Build Docker Image**: Jenkins uses the Dockerfile to build a Docker image containing the application and its dependencies, including the Nginx web server.

3. **Stop Old Container**: If a previous version of the container is running, Jenkins stops and removes it to prepare for the new deployment.

4. **Deploy New Container**: Jenkins creates a new Docker container using the newly built image, mapping port 8081 on the host to port 80 in the container.

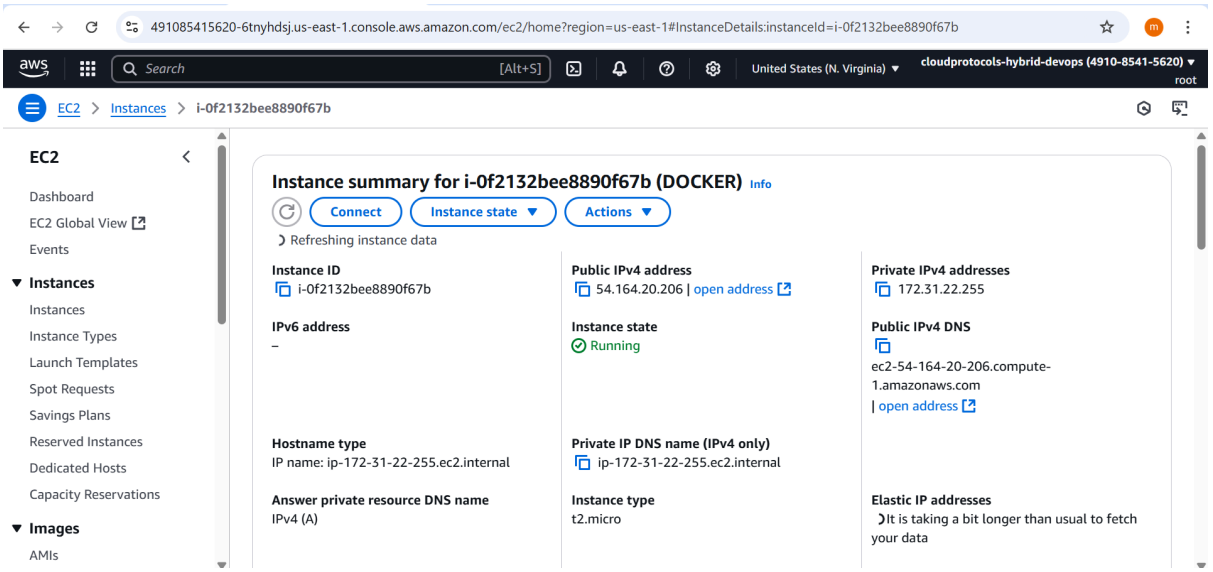
5. **Verify Deployment**: Jenkins verifies that the container is running correctly by checking its status and logs.

Architecture

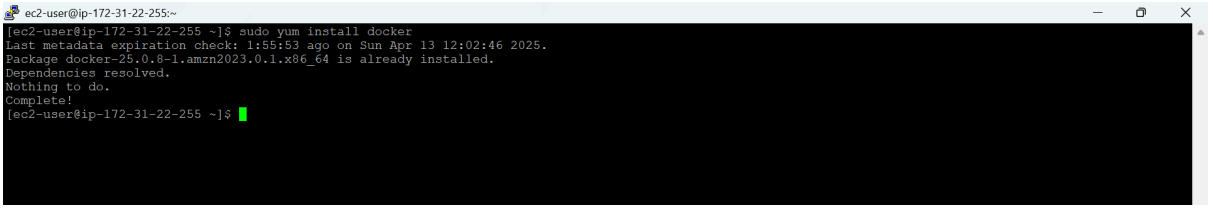


Infrastructure Setup

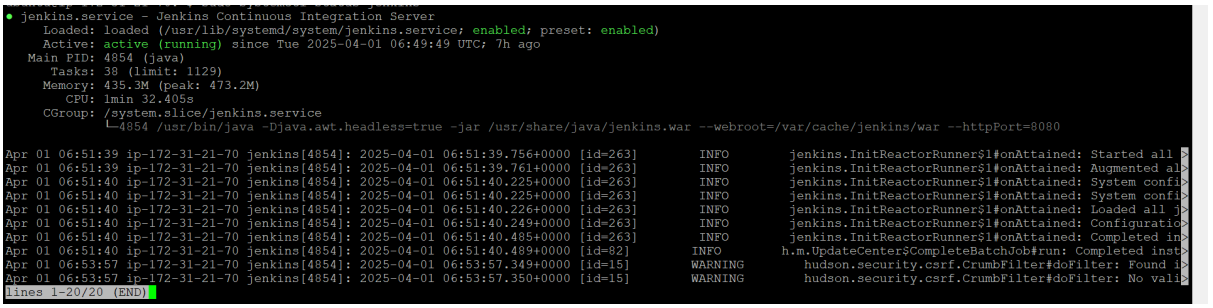
1. Created an Amazon Linux EC2 instance named "docker"



2. Installed Docker on the EC2 instance

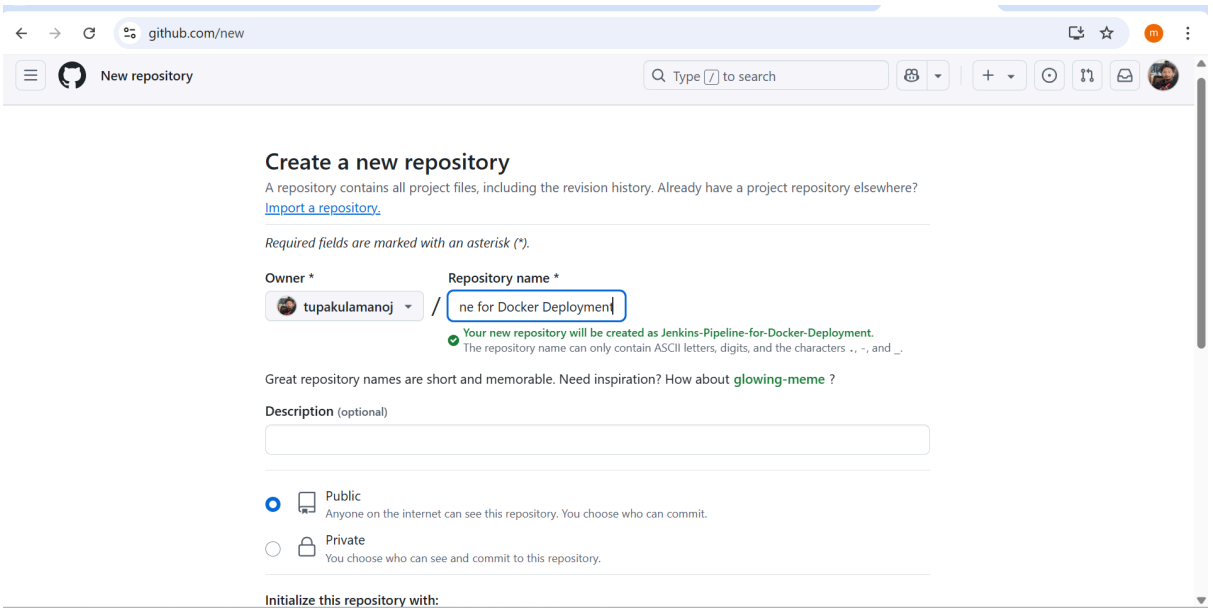


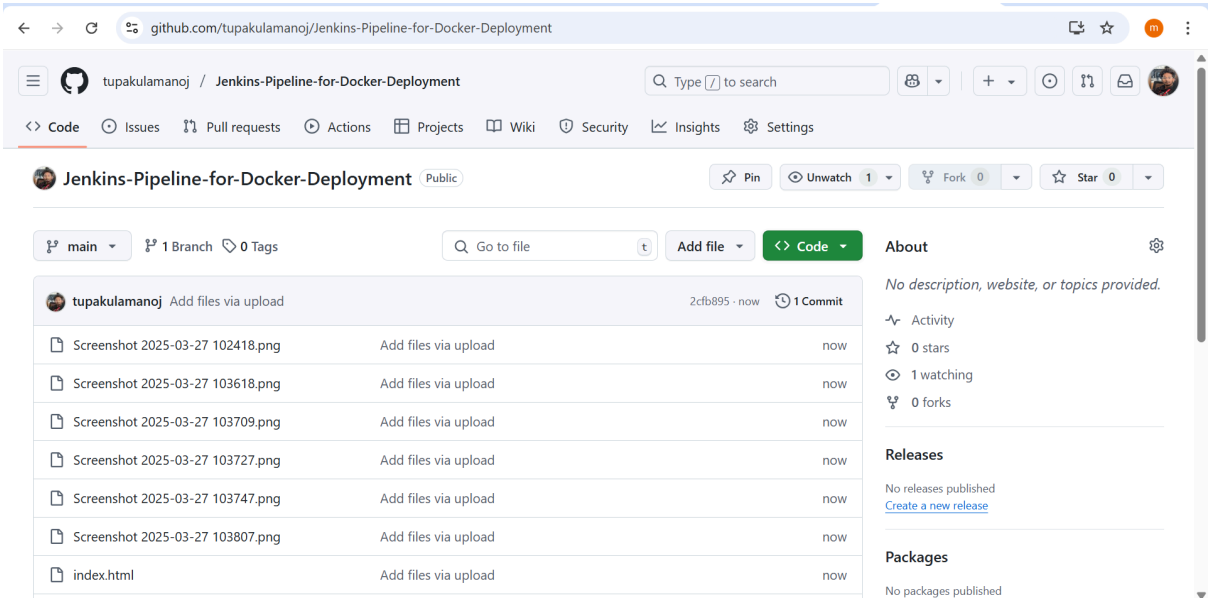
3. Installed Jenkins on the EC2 instance



Repository Creation

1. Created a GitHub repository: https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git





2. Added the following key files to the repository:
    - Dockerfile

github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment/blob/main/Dockerfile

main Jenkins-Pipeline-for-Docker-Deployment / Dockerfile

Go to file

tupakulamanoj Update Dockerfile 9feff28 · 3 minutes ago History

Code Blame 17 lines (12 loc) · 391 Bytes Code 55% faster with GitHub Copilot Raw

```
# Use a base image
FROM ubuntu:20.04

# Install dependencies
RUN apt-get update && apt-get install -y nginx

# Copy the application files from your cloned repository to the nginx serving directory
COPY . /var/www/html/

# Ensure proper permissions
RUN chown -R www-data:www-data /var/www/html/

# Expose port
EXPOSE 80

# Start nginx when container starts
CMD ["nginx", "-g", "daemon off;"]
```

- Jenkinsfile

github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment/blob/main/Jenkinsfile

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

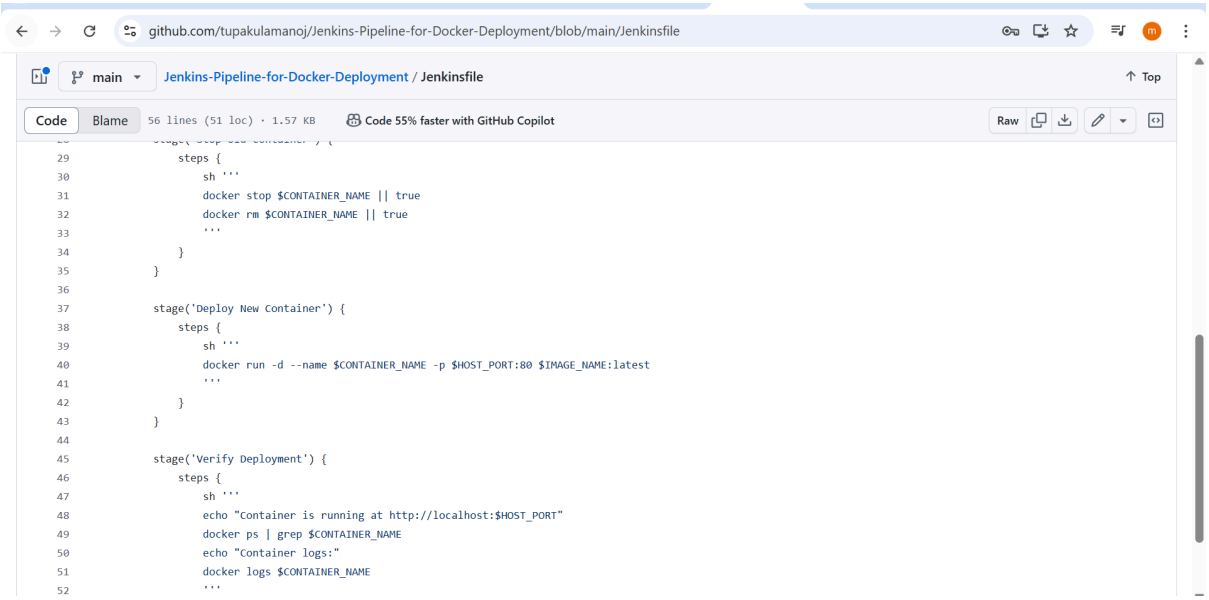
main Jenkins-Pipeline-for-Docker-Deployment / Jenkinsfile

Go to file

tupakulamanoj Update Jenkinsfile be1703b · now History

Code Blame 56 lines (51 loc) · 1.57 KB Code 55% faster with GitHub Copilot Raw

```
pipeline {
    agent any
    environment {
        IMAGE_NAME = 'Dockerimage'
        CONTAINER_NAME = 'DockerContainer'
        HOST_PORT = '8081'
    }
    stages {
        stage('Clone Repo') {
            steps {
                git branch: 'main', url: 'https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh '''
                # List files to verify we have the repository content
```



Dockerfile Details

The Dockerfile creates a container based on Ubuntu 20.04 with Nginx to serve web content:

```
FROM ubuntu:20.04

# Install dependencies
RUN apt-get update && apt-get install -y nginx

# Copy the application files from your cloned repository to the nginx
serving directory
COPY . /var/www/html/

# Ensure proper permissions
RUN chown -R www-data:www-data /var/www/html/

# Expose port
EXPOSE 80

# Start nginx when container starts
CMD ["nginx", "-g", "daemon off;"]
```

Jenkinsfile Details

The Jenkinsfile defines a pipeline with the following stages:

1. **Clone Repo**: Clones the GitHub repository
2. **Build Docker Image**: Builds a Docker image named `your-image-name`
3. **Stop Old Container**: Stops and removes any existing container named `your-container-name`
4. **Deploy New Container**: Creates a new container that maps port 8081 on the host to port 80 in the container
5. **Verify Deployment**: Confirms the container is running and displays logs

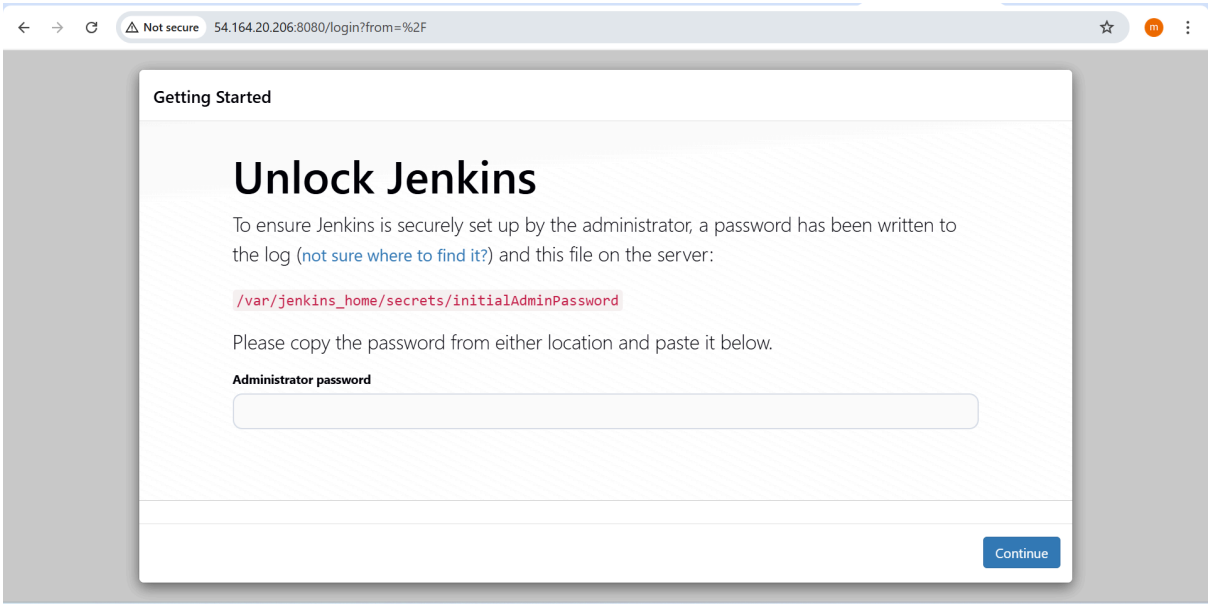
Script :

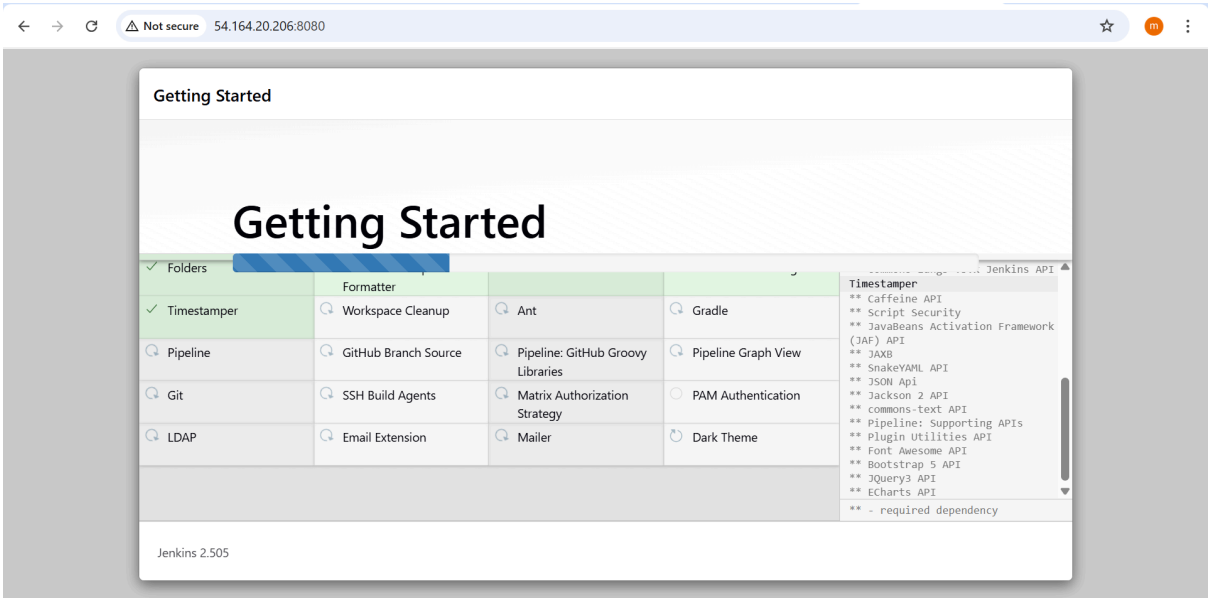
```
pipeline {
    agent any
    environment {
        IMAGE_NAME = 'Dockerimage'
        CONTAINER_NAME = 'Dockercontainername'
        HOST_PORT = '8081'
    }
    stages {
        stage('Clone Repo') {
            steps {
                git branch: 'main', url:
'https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment
.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh '''
                # List files to verify we have the repository content
                echo "Files in the build directory:"
                ls -la

                # Build the Docker image using the current directory
(with the cloned repo)
                docker build -t $IMAGE_NAME:latest .
                '''
            }
        }

        stage('Stop Old Container') {
            steps {
                sh '''
                docker stop $CONTAINER_NAME || true
                docker rm $CONTAINER_NAME || true
                '''
```

```
            }
        }

        stage('Deploy New Container') {
            steps {
                sh '''
                docker run -d --name $CONTAINER_NAME -p $HOST_PORT:80
$IMAGE_NAME:latest
                '''
            }
        }

        stage('Verify Deployment') {
            steps {
                sh '''
                echo "Container is running at
http://localhost:$HOST_PORT"
                docker ps | grep $CONTAINER_NAME
                echo "Container logs:"
                docker logs $CONTAINER_NAME
                '''
            }
        }
    }
}
```

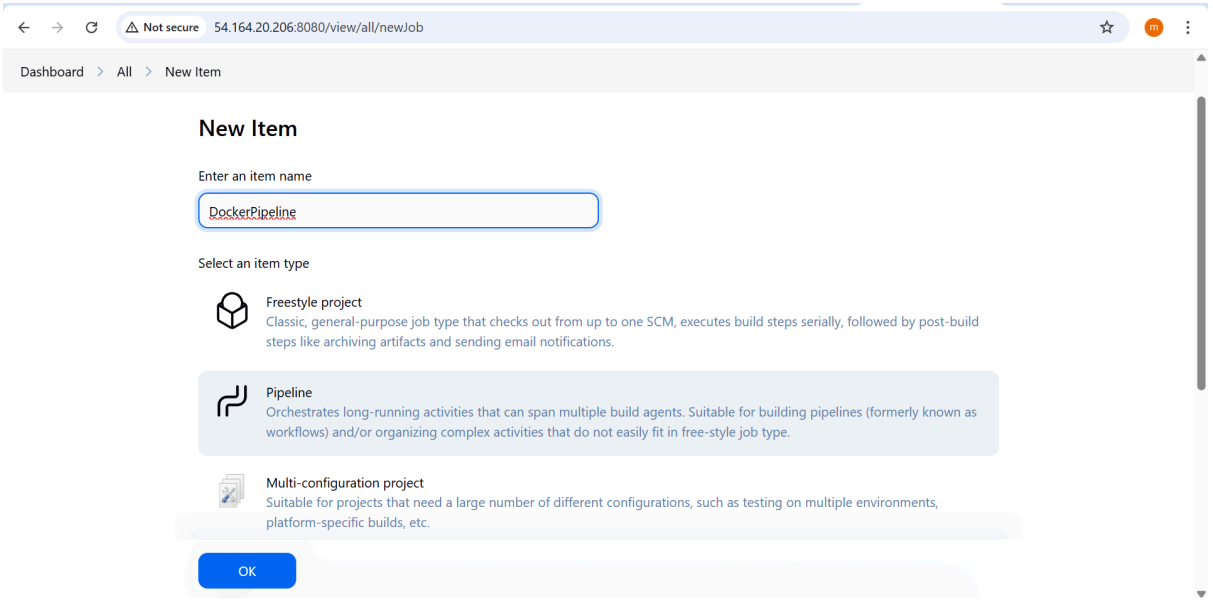
Jenkins Pipeline Setup

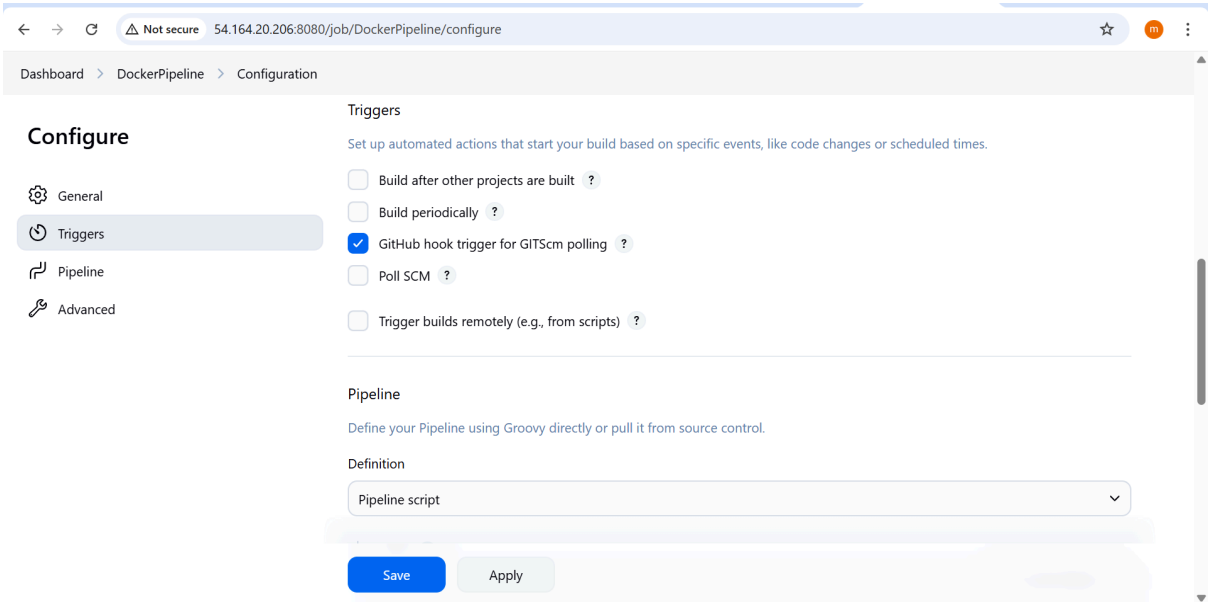
1. Logged into Jenkins



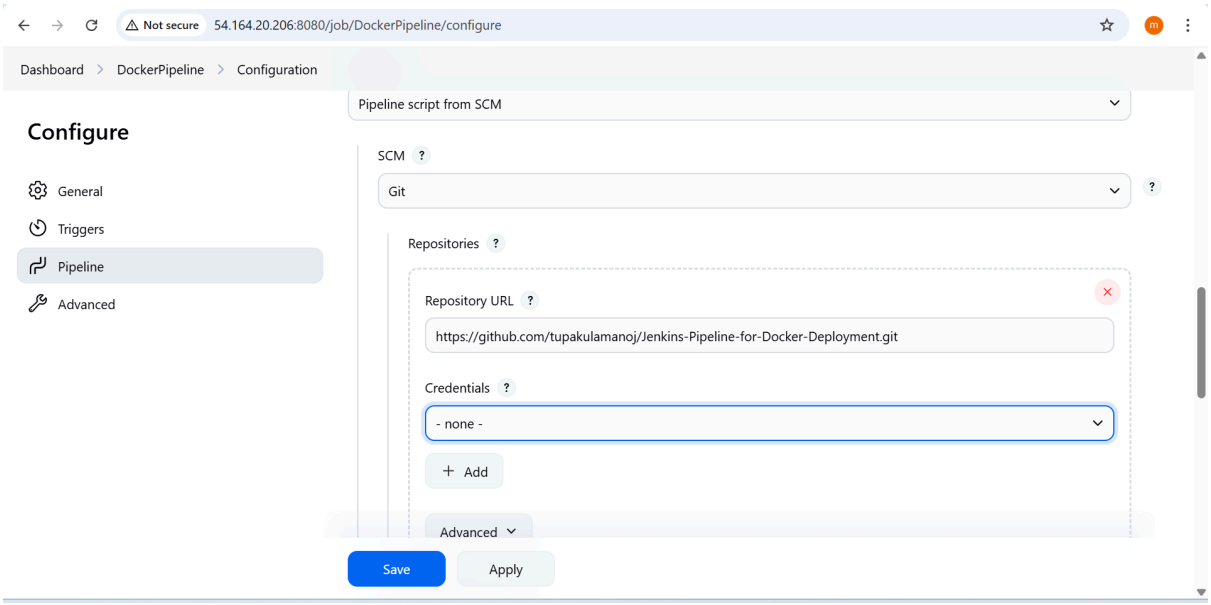


2. Created a new Pipeline job

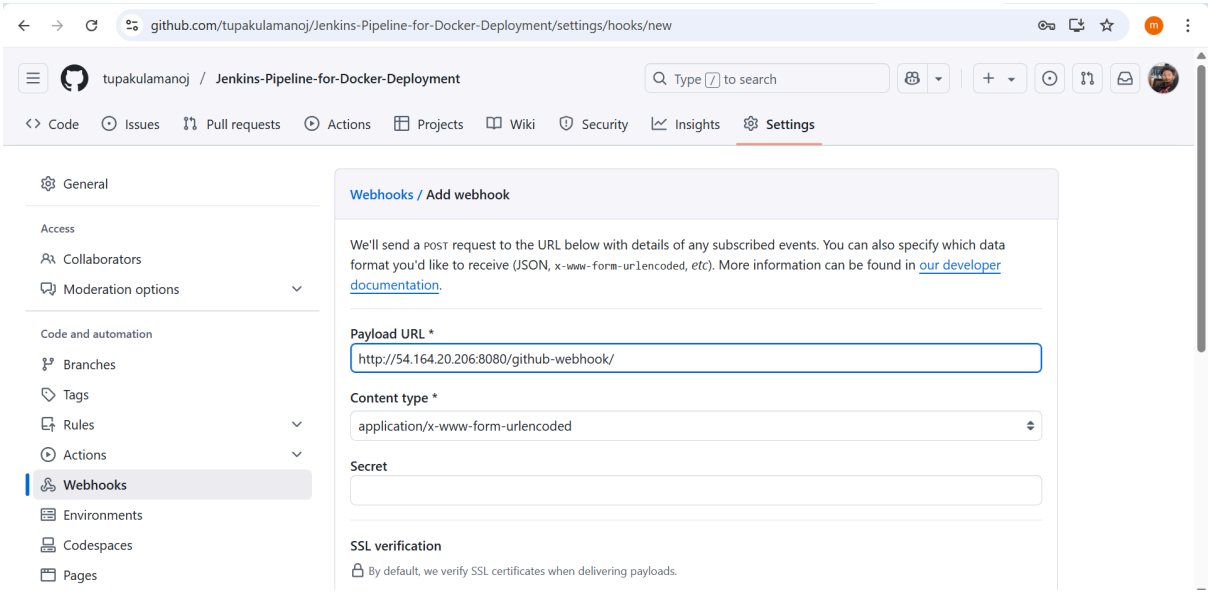




3. Configured "Pipeline script from SCM" option
4. Selected Git as the SCM

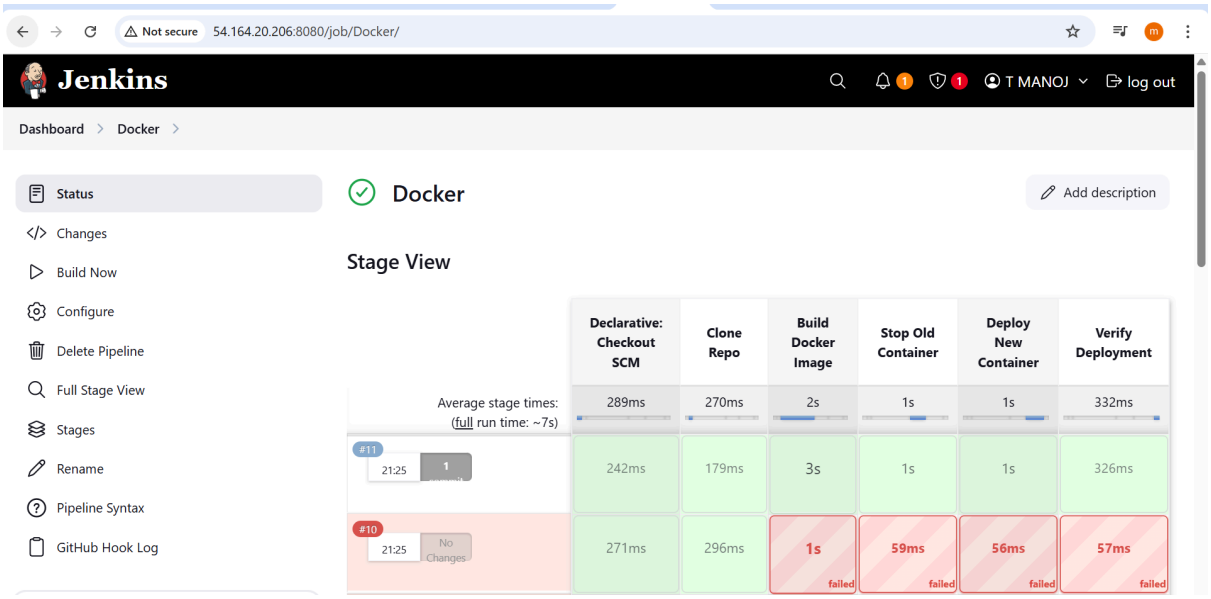


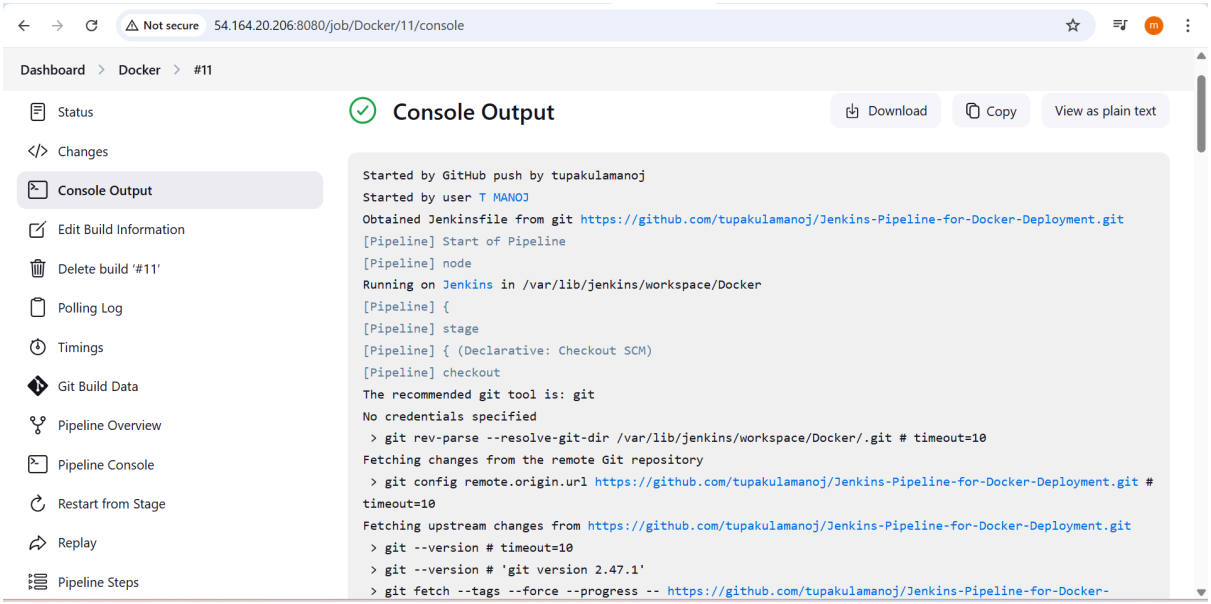
5. Added the GitHub repository URL:
   https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git
6. Set up a GitHub webhook for automatic triggering



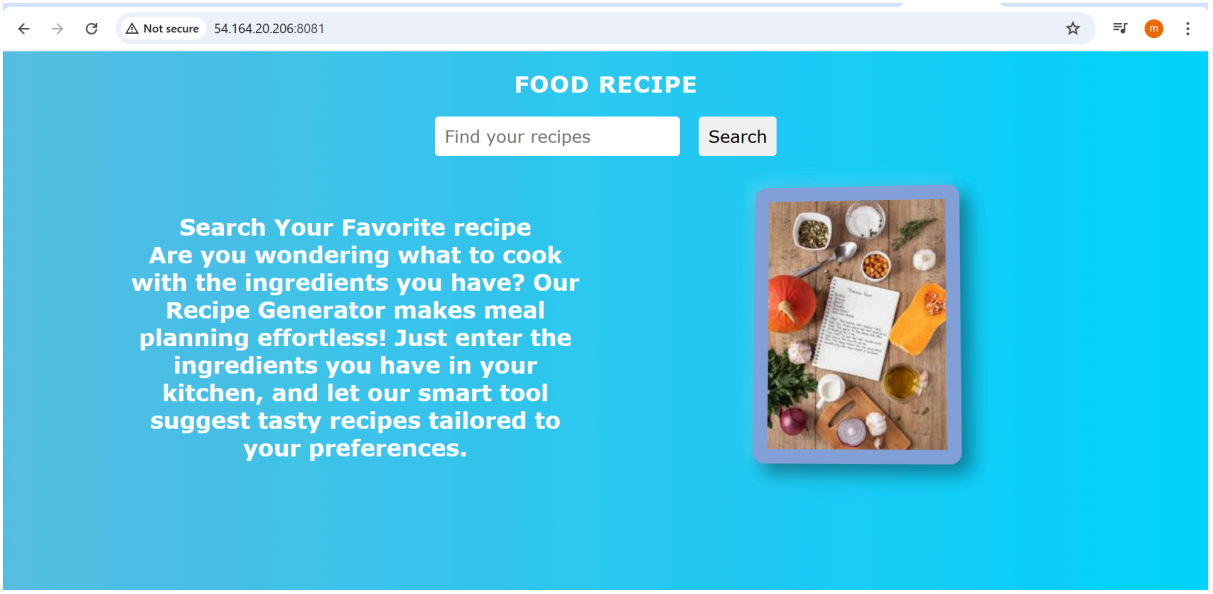
Pipeline Execution

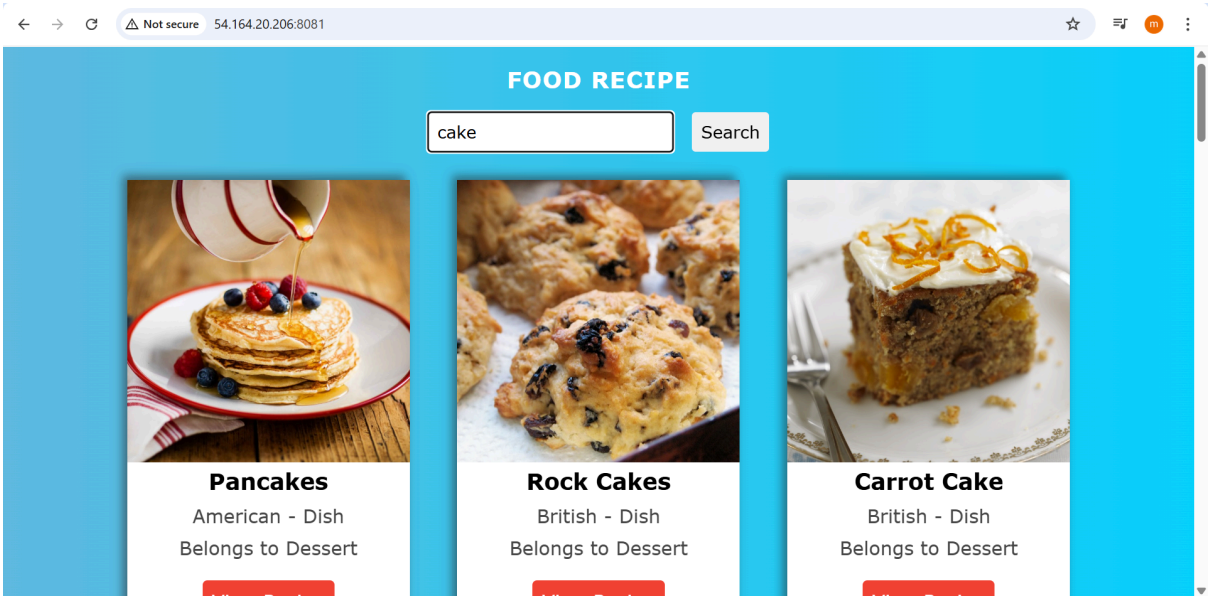
1. Jenkins monitors the GitHub repository for changes
2. When a commit is detected, Jenkins automatically:
    - Clones the repository
    - Builds a Docker image from the Dockerfile
    - Stops any existing container
    - Deploys a new container with the latest image
    - Verifies the deployment




Not secure
54.164.20.206:8080/job/Docker/11/console
Dashboard
Docker
#11
Status
Changes
Console Output
Edit Build Information
Delete build '#11'
Polling Log
Timings
Git Build Data
Pipeline Overview
Pipeline Console
Restart from Stage
Replay
Pipeline Steps
Console Output
Download
Copy
View as plain text
Started by GitHub push by tupakulamanoj
Started by user T MANOJ
Obtained Jenkinsfile from git https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Docker
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Docker/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git # timeout=10
Fetching upstream changes from https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git
> git --version # timeout=10
> git --version # 'git version 2.47.1'
> git fetch --tags --force --progress -- https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-

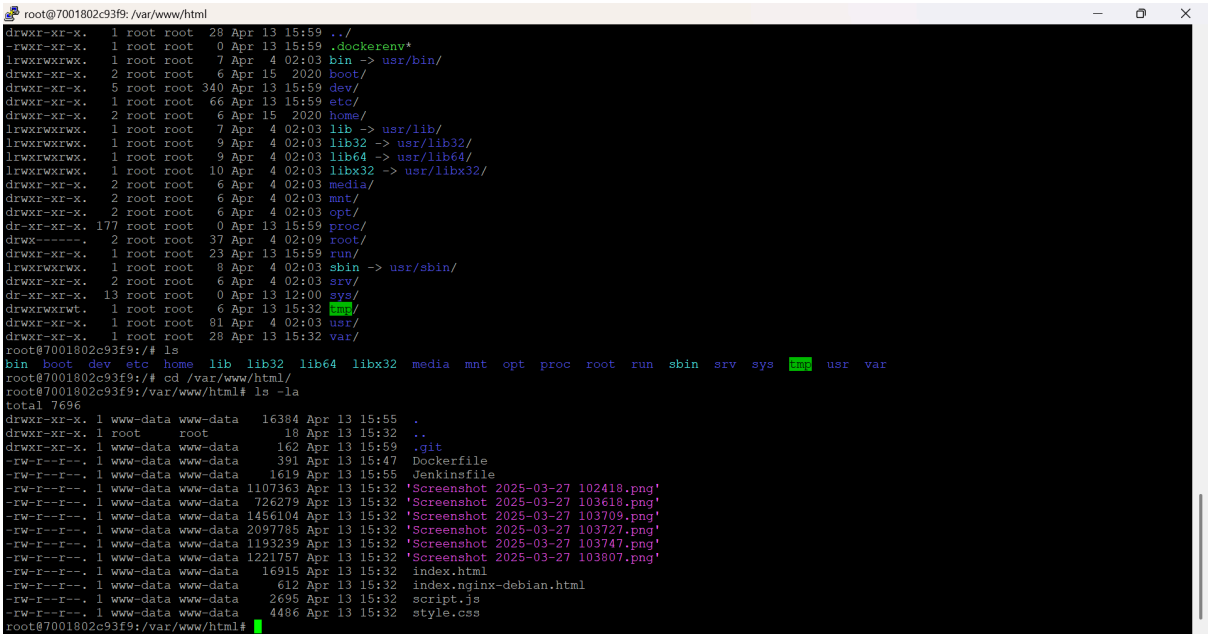

Not secure
54.164.20.206:8080/job/Docker/11/console
Dashboard
Docker
#11
Container is running at http://localhost:8081
+ grep your-container-name
+ docker ps
a7d394668e2c your-image-name:latest "nginx -g 'daemon of…" 2 seconds ago Up Less than a second
0.0.0.0:8081->80/tcp, :::8081->80/tcp your-container-name
+ echo 'Container logs:'
Container logs:
+ docker logs your-container-name
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
REST API
Jenkins 2.492.3

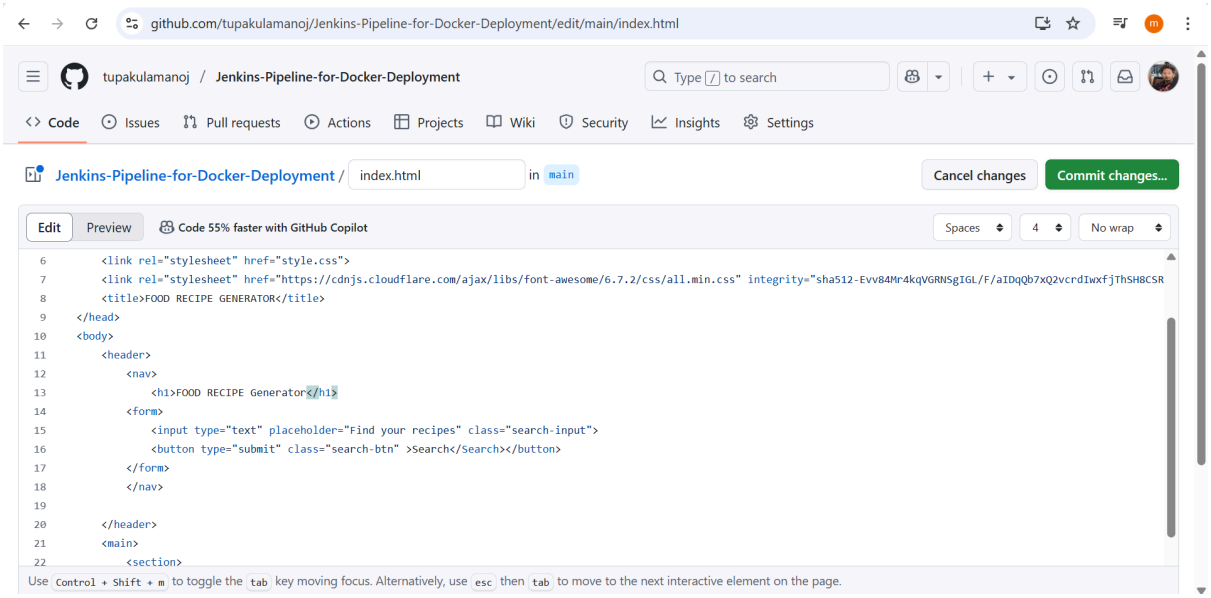




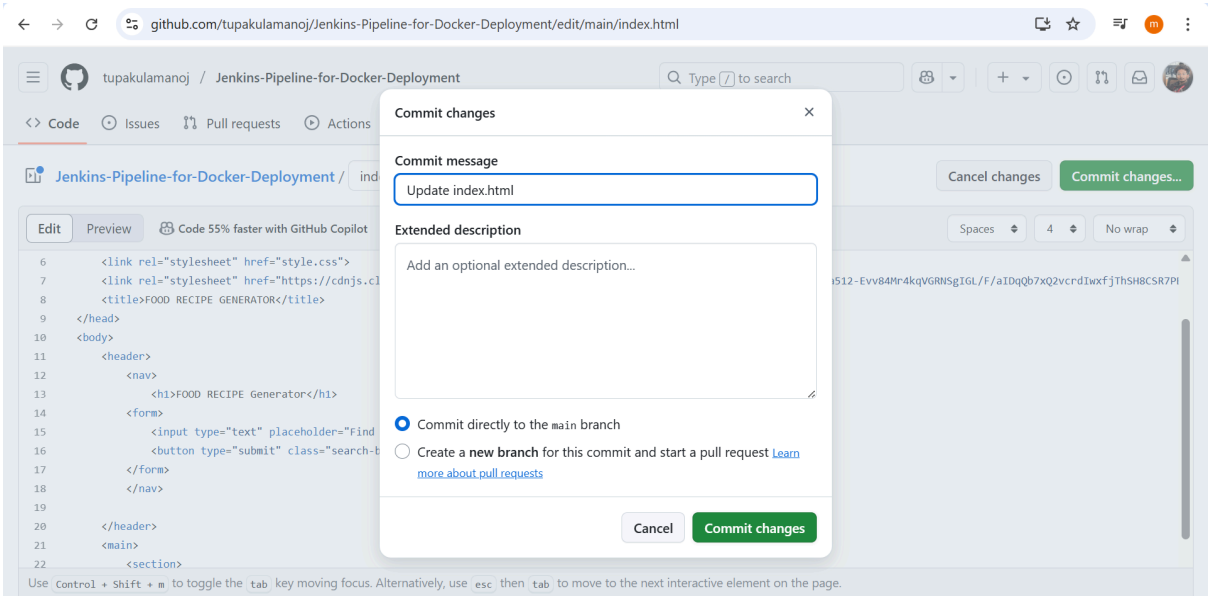
Verification

commit changes happened in GitHub along with Jenkins Pipeline.

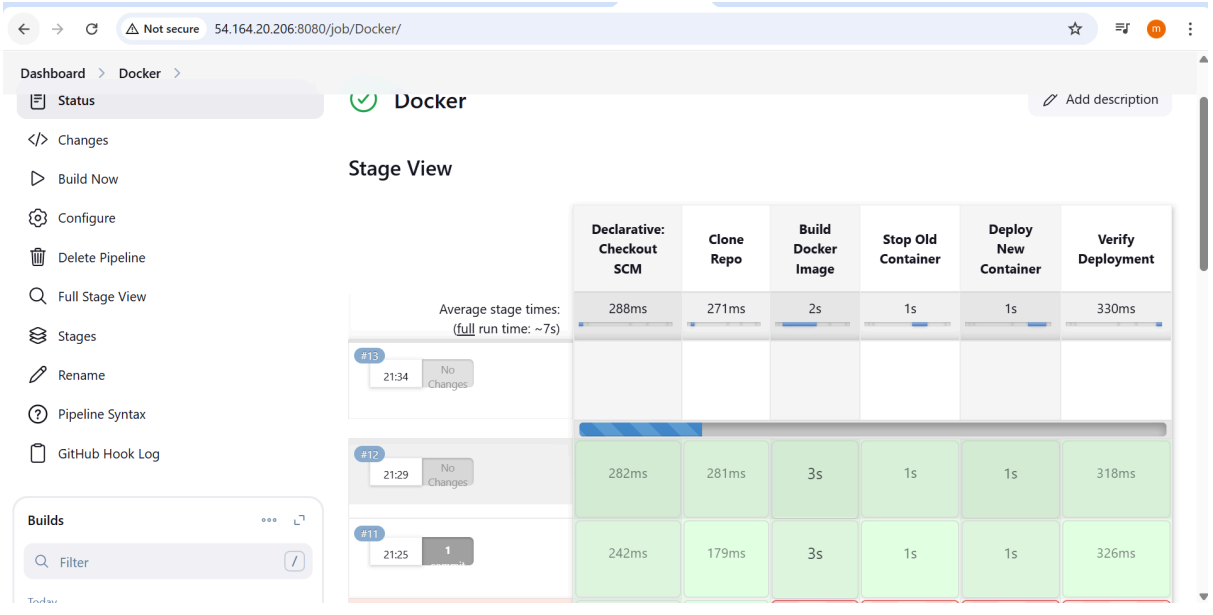


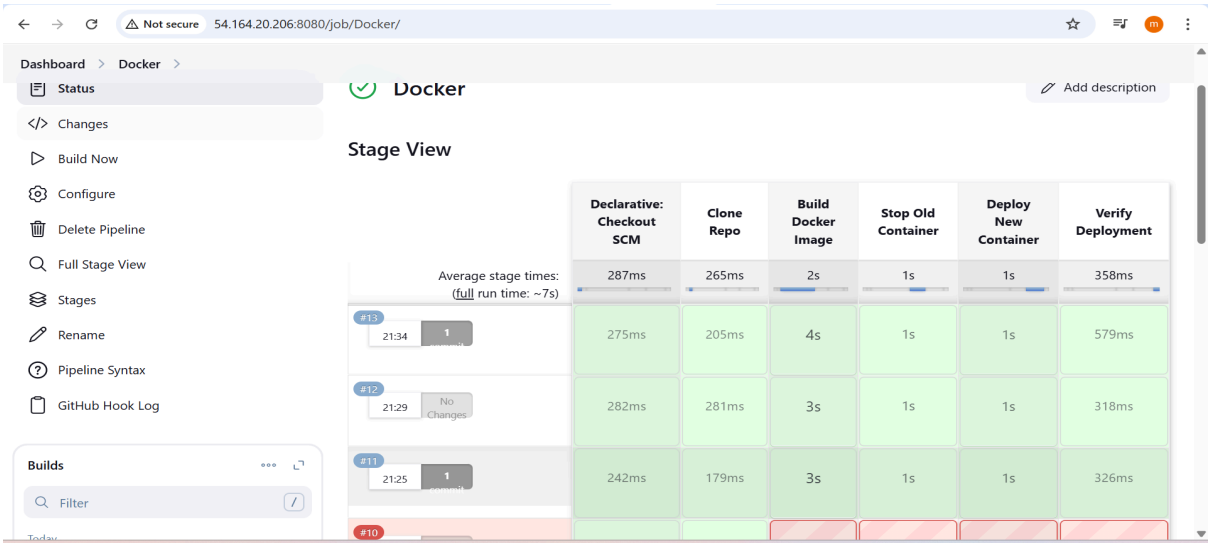


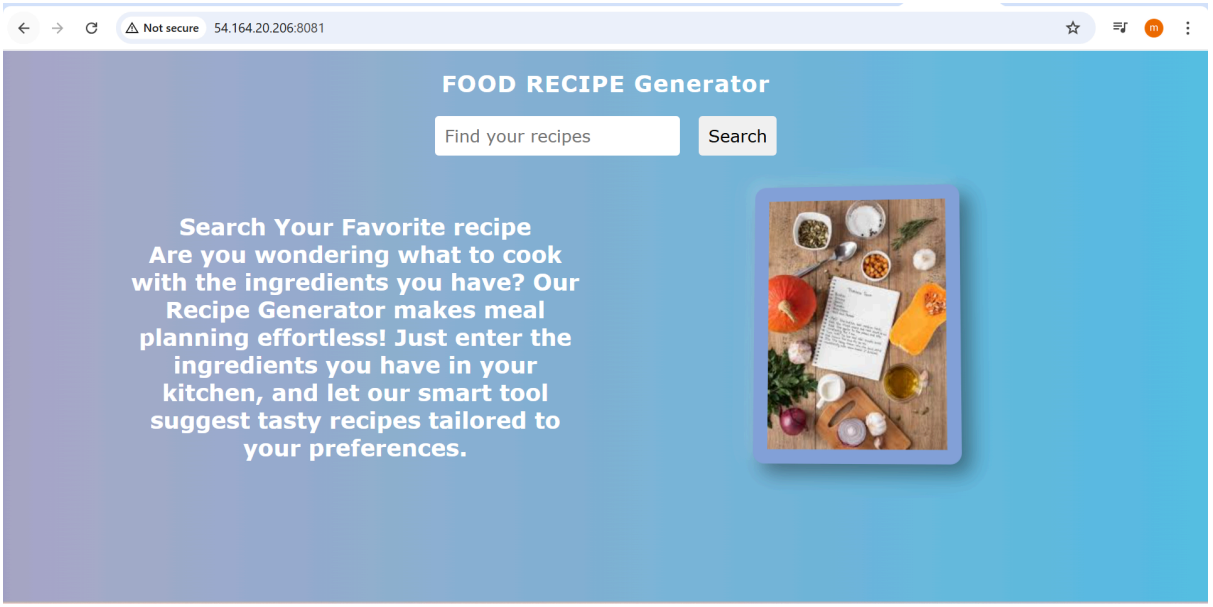
Here we are updating the h1 tag from FOOD RECIPE TO FOOD RECIPE Generator.



Pipeline Triggered







Conclusion

The Jenkins Pipeline for Docker Deployment solution represents a robust, modern approach to continuous integration and continuous deployment. By combining Amazon Linux EC2, Docker, Jenkins, and GitHub into a cohesive system.This Jenkins Pipeline for Docker Deployment architecture delivers a streamlined, automated solution for continuous integration and deployment by leveraging Amazon Linux EC2, Docker, Jenkins, and GitHub in a cohesive system. This implementation eliminates manual deployment processes, ensures environment consistency through containerization, and enables rapid deployment of code changes through webhook-triggered pipelines. The architecture provides significant benefits including reduced human error, improved developer productivity, consistent testing environments, and faster time-to-market for new features. By treating infrastructure and deployment processes as code, the solution offers transparency, auditability, and reproducibility while establishing a foundation that can be extended with additional capabilities such as container orchestration, multi-environment deployments, and comprehensive monitoring as business needs evolve.