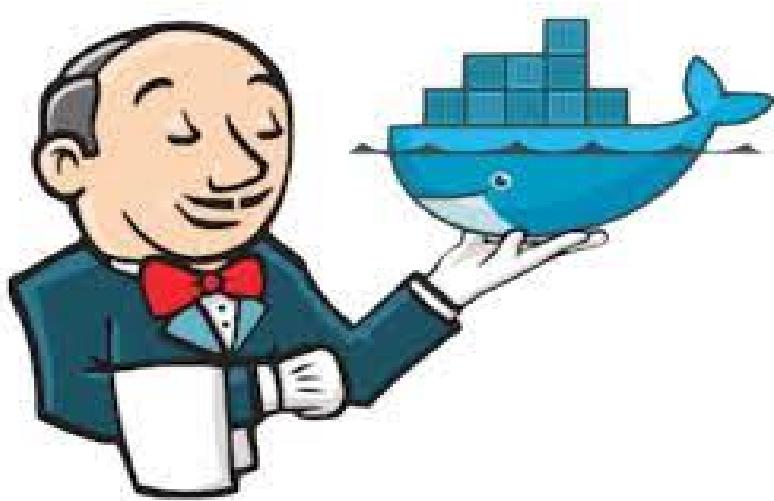


cprime

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:

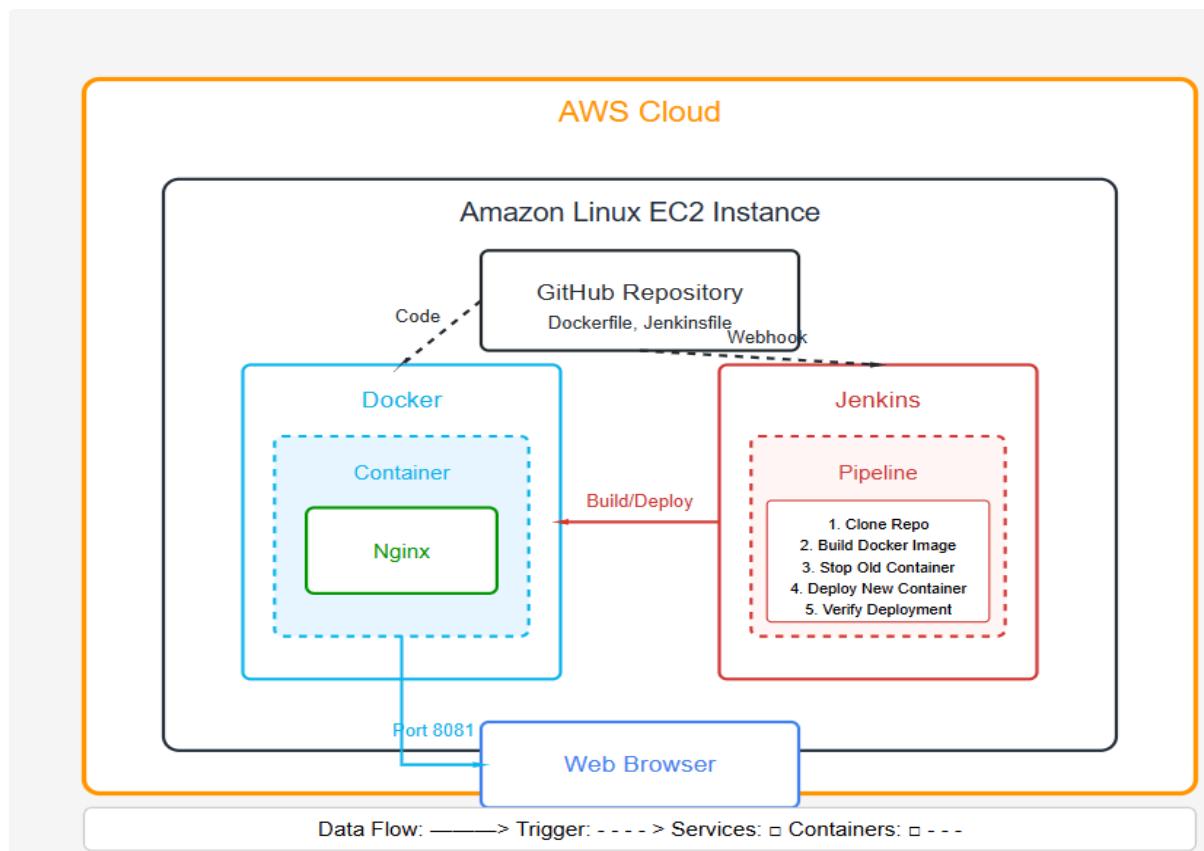
- High-performance HTTP server
- Load balancing
- Caching
- Reverse proxy functionality
- Low memory footprint
- Handling of static content

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"

The screenshot shows the AWS EC2 Instances page with the instance summary for i-0f2132bee8890f67b. The instance is labeled as 'DOCKER'. Key details include:

- Instance ID:** i-0f2132bee8890f67b
- Public IPv4 address:** 54.164.20.206
- Instance state:** Running
- Private IP address:** 172.31.22.255
- Public IPv4 DNS:** ec2-54-164-20-206.compute-1.amazonaws.com
- Hostname type:** IP name: ip-172-31-22-255.ec2.internal
- Answer private resource DNS name:** IPv4 (A)
- Instance type:** t2.micro

2. Installed Docker on the EC2 instance

```
[ec2-user@ip-172-31-22-255 ~]$ sudo yum install docker
Last metadata expiration check: 1:55:53 ago on Sun Apr 13 12:02:46 2025.
Package docker-25.0.8-1.amzn2023.0.1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-22-255 ~]$
```

3. Installed Jenkins on the EC2 instance

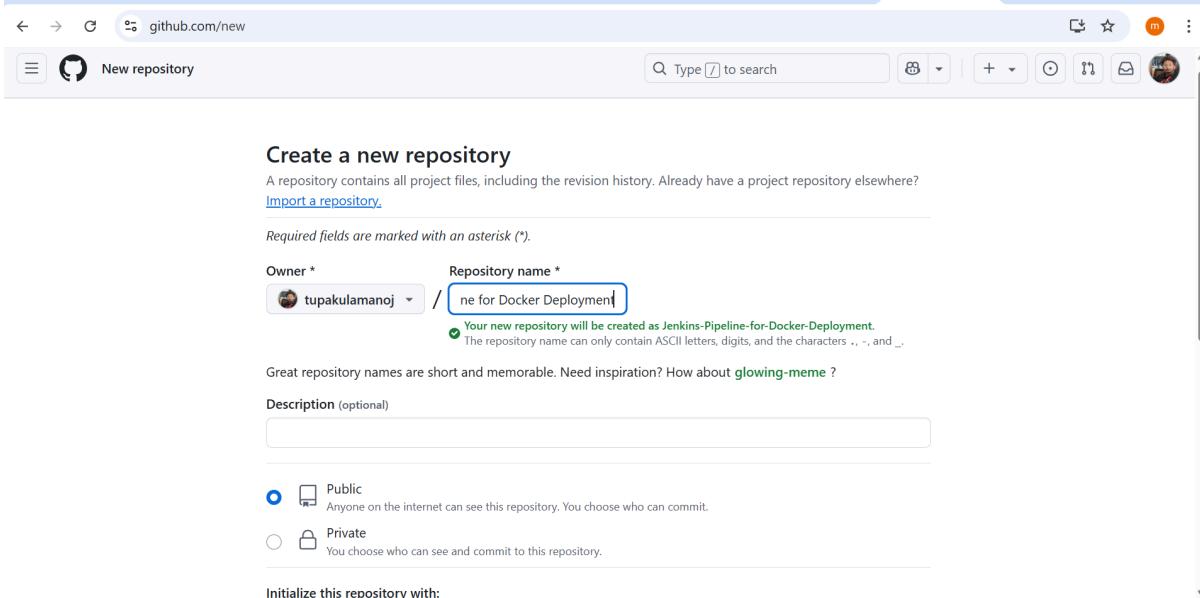
```
* jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Tue 2025-04-01 06:49:49 UTC; 7h ago
     Main PID: 4954 (java)
        Tasks: 38 (limit: 1129)
       Memory: 435.3M (peak: 473.2M)
          CPU: 32.40ss
         CGroup: /system.slice/jenkins.service
             └─4954 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Apr 01 06:51:39 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:39.756+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: Started all >
Apr 01 06:51:39 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:39.761+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: Augmented al>
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.225+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: System config>
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.225+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: System config>
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.226+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: Loaded all >
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.249+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: Configuration >
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.485+0000 [id=263]      INFO      jenkins.InitReactorRunner$1#onAttained: Completed in >
Apr 01 06:51:40 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:51:40.489+0000 [id=82]      INFO      h.m.UpdateCenter$CompleteBatchJob#run: Completed inst >
Apr 01 06:53:57 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:53:57.349+0000 [id=15]      WARNING    hudson.security.csrf.CrumbFilter#doFilter: Found >
Apr 01 06:53:57 ip-172-31-21-70 jenkins[4854]: 2025-04-01 06:53:57.350+0000 [id=15]      WARNING    hudson.security.csrf.CrumbFilter#doFilter: No val >
lines 1-20/20 (END)
```

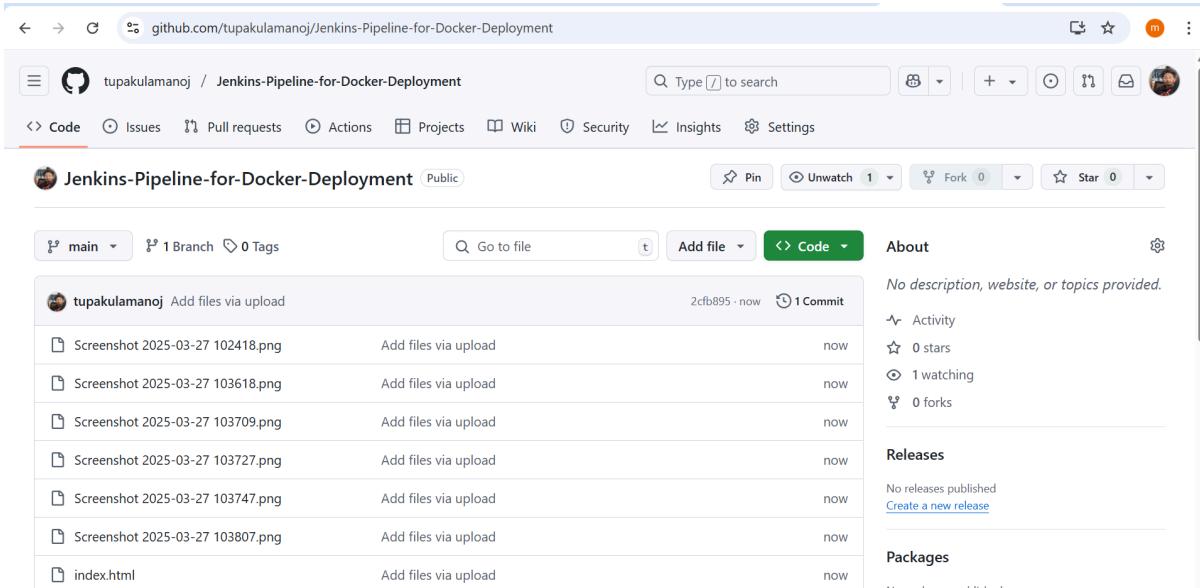
Repository Creation

1. Created a GitHub repository:

<https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git>



The screenshot shows the GitHub interface for creating a new repository. At the top, it says "Create a new repository". Below that, there's a note: "A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository." A "Required fields are marked with an asterisk (*)." message is present. The "Owner" field is set to "tupakulamanoj" and the "Repository name" field is "ne for Docker Deployment". A note below the name says "Your new repository will be created as Jenkins-Pipeline-for-Docker-Deployment." and "The repository name can only contain ASCII letters, digits, and the characters ., -, and _." There's a suggestion "Great repository names are short and memorable. Need inspiration? How about glowing-meme ?". The "Description (optional)" field is empty. Under "Visibility", "Public" is selected, with a description: "Anyone on the internet can see this repository. You choose who can commit." Below that, "Private" is also listed with its description: "You choose who can see and commit to this repository." At the bottom, there's a section for "Initialize this repository with:" which is currently empty.



The screenshot shows the GitHub repository details page for "Jenkins-Pipeline-for-Docker-Deployment". The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The repository name "Jenkins-Pipeline-for-Docker-Deployment" is shown with a Public link. The main content area shows the "main" branch with 1 Branch and 0 Tags. A list of files is displayed, all added via upload by "tupakulamanoj": Screenshot 2025-03-27 102418.png, Screenshot 2025-03-27 103618.png, Screenshot 2025-03-27 103709.png, Screenshot 2025-03-27 103727.png, Screenshot 2025-03-27 103747.png, Screenshot 2025-03-27 103807.png, and index.html. On the right side, there are sections for "About", "Activity", "Releases", and "Packages". The "About" section notes "No description, website, or topics provided.". The "Activity" section shows 0 stars, 1 watching, and 0 forks. The "Releases" section indicates "No releases published" and a link to "Create a new release". The "Packages" section shows "No packages published".

2. Added the following key files to the repository:

- o Dockerfile

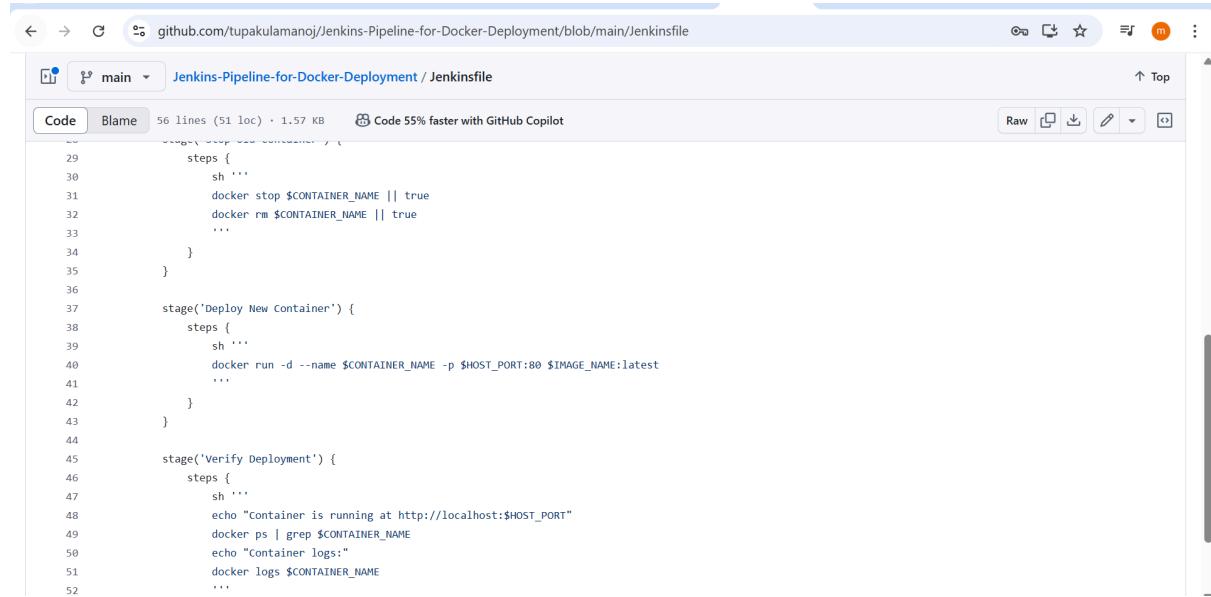
A screenshot of a GitHub repository page showing the Dockerfile. The URL is github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment/blob/main/Dockerfile. The Dockerfile contains 17 lines of code:

```
1 # Use a base image
2 FROM ubuntu:20.04
3
4 # Install dependencies
5 RUN apt-get update && apt-get install -y nginx
6
7 # Copy the application files from your cloned repository to the nginx serving directory
8 COPY . /var/www/html/
9
10 # Ensure proper permissions
11 RUN chown -R www-data:www-data /var/www/html/
12
13 # Expose port
14 EXPOSE 80
15
16 # Start nginx when container starts
17 CMD ["nginx", "-g", "daemon off;"]
```

○ Jenkinsfile

A screenshot of a GitHub repository page showing the Jenkinsfile. The URL is github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment/blob/main/Jenkinsfile. The Jenkinsfile contains 56 lines of Groovy script:

```
1 pipeline {
2     agent any
3     environment {
4         IMAGE_NAME = 'Dockerimage'
5         CONTAINER_NAME = 'DockerContainer'
6         HOST_PORT = '8081'
7     }
8     stages {
9         stage('Clone Repo') {
10             steps {
11                 git branch: 'main', url: 'https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git'
12             }
13         }
14         stage('Build Docker Image') {
15             steps {
16                 sh '''
17                     # List files to verify we have the repository content
18                 ls
19             '''
20         }
21     }
22 }
```



The screenshot shows a GitHub code editor interface for a file named 'Jenkinsfile'. The file contains Jenkins pipeline code. The code defines three stages: 'Stop Old Container', 'Deploy New Container', and 'Verify Deployment'. Each stage contains steps to stop old containers, run new ones, and verify their status.

```
29     steps {
30         sh ***
31         docker stop $CONTAINER_NAME || true
32         docker rm $CONTAINER_NAME || true
33         ...
34     }
35 }
36
37 stage('Deploy New Container') {
38     steps {
39         sh ***
40         docker run -d --name $CONTAINER_NAME -p $HOST_PORT:$IMAGE_NAME:latest
41         ...
42     }
43 }
44
45 stage('Verify Deployment') {
46     steps {
47         sh ***
48         echo "Container is running at http://localhost:$HOST_PORT"
49         docker ps | grep $CONTAINER_NAME
50         echo "Container logs:"
51         docker logs $CONTAINER_NAME
52     }
}
```

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
'''
            }
        }
    }
}

```

```

        }

    }

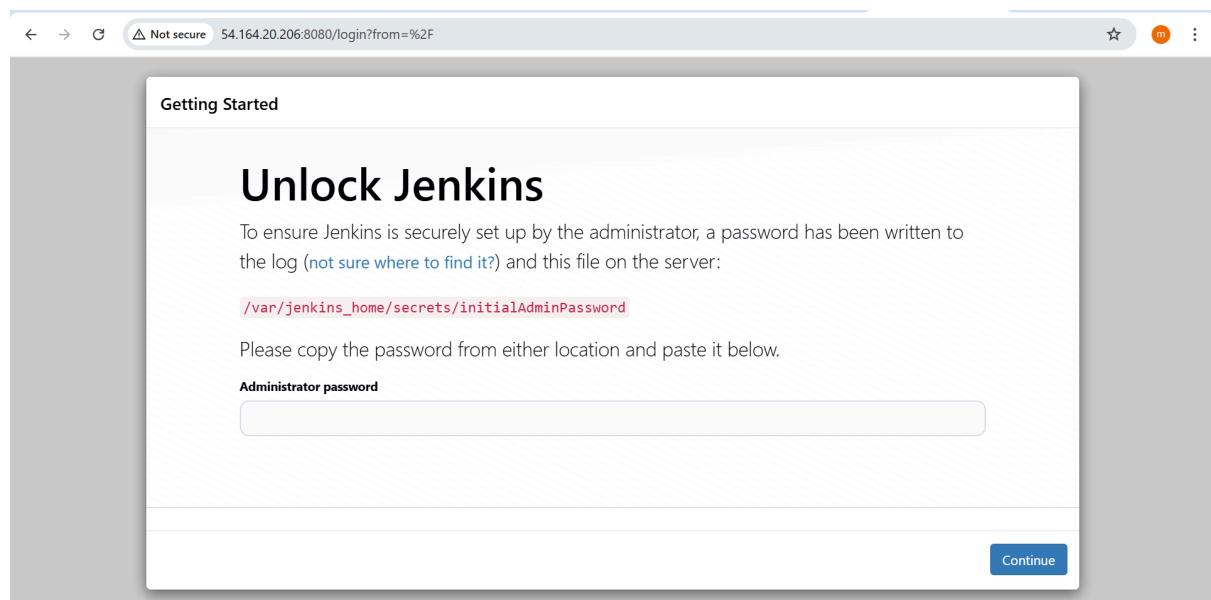
    stage('Deploy New Container') {
        steps {
            sh '''
                docker run -d --name $CONTAINER_NAME -p $HOST_PORT:80
$IMAGE_NAME:latest
            '''
        }
    }

    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



The screenshot shows the Jenkins 'Getting Started' page. At the top, there's a navigation bar with icons for back, forward, search, and refresh, followed by a status message 'Not secure' and the URL '54.164.20.206:8080'. Below the header is a 'Getting Started' section with a large 'Getting Started' heading. To the left is a sidebar titled 'Folders' containing links to 'Timestamper', 'Pipeline', 'Git', and 'LDAP'. To the right is a sidebar titled 'Formatter' containing links to 'Workspace Cleanup', 'GitHub Branch Source', 'SSH Build Agents', 'Email Extension', 'Ant', 'Pipeline: GitHub Groovy Libraries', 'Matrix Authorization Strategy', 'Mailer', 'Gradle', 'Pipeline Graph View', 'PAM Authentication', and 'Dark Theme'. A vertical sidebar on the right is titled 'Jenkins API' and lists numerous Java API endpoints, such as 'Caffeine API', 'Script Security', 'JavaBeans Activation Framework (JAF) API', 'JAXB', 'SnakeYAML API', 'JSON API', 'Jackson 2 API', 'commons-text API', 'Pipeline: Supporting APIs', 'Plugin Utilities API', 'Font Awesome API', 'Bootstrap 5 API', 'jQuery3 API', 'ECharts API', and a note about a required dependency.

2. Created a new Pipeline job

The screenshot shows the Jenkins 'New Item' creation page. The URL in the address bar is '54.164.20.206:8080/view/all/newJob'. The page title is 'New Item'. There is a text input field labeled 'Enter an item name' with the value 'DockerPipeline'. Below it is a section titled 'Select an item type' with three options: 'Freestyle project' (selected), 'Pipeline' (disabled), and 'Multi-configuration project' (disabled). The 'Freestyle project' description states: 'Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.' The 'Pipeline' description states: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.' The 'Multi-configuration project' description states: 'Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.' At the bottom is a blue 'OK' button.

The screenshot shows the Jenkins job configuration page for 'DockerPipeline'. In the 'Triggers' section, the 'GitHub hook trigger for GITScm polling' option is selected. Under the 'Pipeline' section, the 'Definition' dropdown is set to 'Pipeline script'. At the bottom, there are 'Save' and 'Apply' buttons.

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

The screenshot shows the Jenkins job configuration page for 'DockerPipeline'. In the 'Pipeline' section, the 'SCM' dropdown is set to 'Git'. The 'Repositories' section shows a single repository with the URL 'https://github.com/tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment.git' and no credentials defined. At the bottom, there are 'Save' and 'Apply' buttons.

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

The screenshot shows the GitHub repository settings page for 'Jenkins-Pipeline-for-Docker-Deployment'. The 'Webhooks' tab is selected. A new webhook is being configured with the following details:

- Payload URL ***: `http://54.164.20.206:8080/github-webhook/`
- Content type ***: `application/x-www-form-urlencoded`
- Secret**: (empty field)
- SSL verification**: (checkbox checked)

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

The screenshot shows the Jenkins Pipeline stage view for the 'Docker' pipeline. The stage is named 'Docker' and is currently in progress. The stage view table provides a breakdown of the execution times for each step:

Declarative: Checkout SCM	Clone Repo	Build Docker Image	Stop Old Container	Deploy New Container	Verify Deployment
289ms	270ms	2s	1s	1s	332ms
242ms	179ms	3s	1s	1s	326ms
271ms	296ms	1s failed	59ms failed	56ms failed	57ms failed

The last two rows show stages that failed, indicated by red backgrounds and error messages.

← → ⌂ △ 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

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-Deployment.git>

← → ⌂ △ 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 off;'  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
```

Not secure 54.164.20.206:8081

FOOD RECIPE

Find your recipes

Search Your Favorite recipe
Are you wondering what to cook with the ingredients you have? Our Recipe Generator makes meal planning effortless! Just enter the ingredients you have in your kitchen, and let our smart tool suggest tasty recipes tailored to your preferences.



Not secure 54.164.20.206:8081

FOOD RECIPE



Pancakes American - Dish Belongs to Dessert	Rock Cakes British - Dish Belongs to Dessert	Carrot Cake British - Dish Belongs to Dessert
----------------------------------------------------------	-----------------------------------------------------------	------------------------------------------------------------

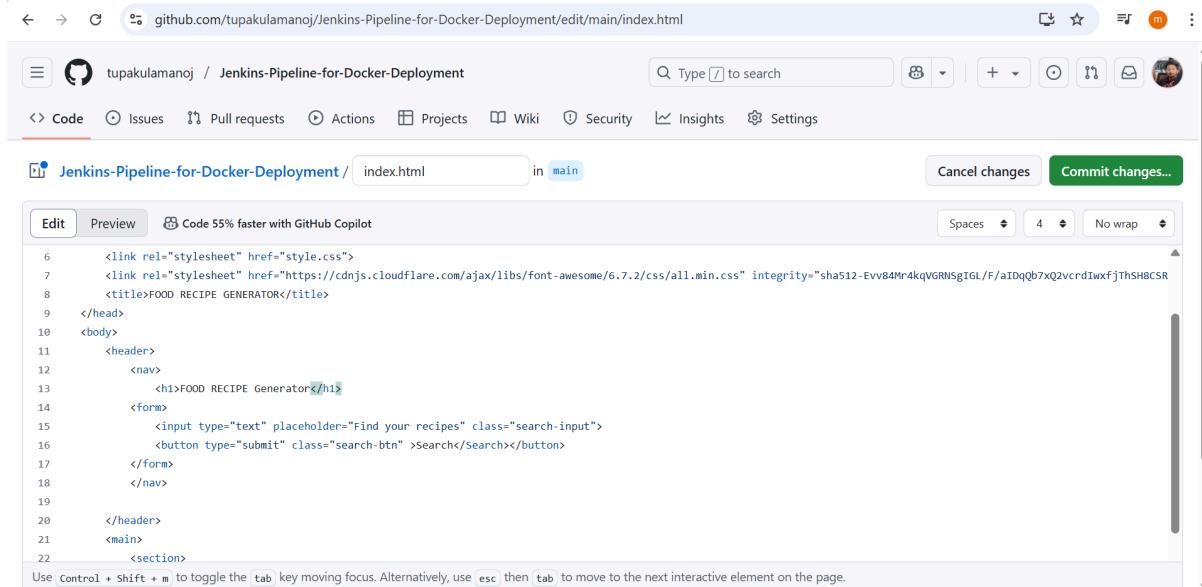
Verification

commit changes happened in GitHub along with Jenkins Pipeline.

```

root@7001802c93f9: /var/www/html
drwxr-xr-x. 1 root root 28 Apr 13 15:59 ./
drwxr-xr-x. 1 root root 0 Apr 13 15:59 .dockerenv*
drwxrwxrwx. 1 root root 7 Apr 15 02:03 bin -> usr/bin/
drwxr-xr-x. 2 root root 6 Apr 15 2020 boot/
drwxr-xr-x. 5 root root 340 Apr 13 15:59 dev/
drwxr-xr-x. 1 root root 66 Apr 13 15:59 etc/
drwxr-xr-x. 2 root root 6 Apr 15 2020 home/
lrwxrwxrwx. 1 root root 7 Apr 4 02:03 lib -> usr/lib/
lrwxrwxrwx. 1 root root 9 Apr 4 02:03 lib32 -> usr/lib32/
lrwxrwxrwx. 1 root root 10 Apr 4 02:03 libx32 -> usr/libx32/
drwxr-xr-x. 2 root root 6 Apr 4 02:03 media/
drwxr-xr-x. 2 root root 6 Apr 4 02:03 mnt/
drwxr-xr-x. 2 root root 6 Apr 4 02:03 opt/
dr-xr-xr-x. 177 root root 0 Apr 13 15:59 proc/
drwxr-xr-x. 2 root root 37 Apr 4 02:09 root/
drwxr-xr-x. 1 root root 23 Apr 13 15:59 run/
lrwxrwxrwx. 1 root root 8 Apr 4 02:03 sbin -> usr/sbin/
drwxr-xr-x. 2 root root 6 Apr 4 02:03 srv/
dr-xr-xr-x. 13 root root 0 Apr 13 12:00 sys/
drwxrwxrwt. 1 root root 6 Apr 13 15:32 tmp/
drwxr-xr-x. 1 root root 81 Apr 4 02:03 usr/
drwxr-xr-x. 1 root root 28 Apr 13 15:32 var/
root@7001802c93f9: # ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@7001802c93f9: # cd /var/www/html/
root@7001802c93f9: /var/www/html# ls -la
total 7696
drwxr-xr-x. 1 www-data www-data 16384 Apr 13 15:55 .
drwxr-xr-x. 1 root root 18 Apr 13 15:32 ..
drwxr-xr-x. 1 www-data www-data 162 Apr 13 15:59 .git
-rw-r--r--. 1 www-data www-data 391 Apr 13 15:47 Dockerfile
-rw-r--r--. 1 www-data www-data 1619 Apr 13 15:55 Jenkinsfile
-rw-r--r--. 1 www-data www-data 1107363 Apr 13 15:32 "Screenshot 2025-03-27 102418.png"
-rw-r--r--. 1 www-data www-data 726279 Apr 13 15:32 "Screenshot 2025-03-27 103618.png"
-rw-r--r--. 1 www-data www-data 1456104 Apr 13 15:32 "Screenshot 2025-03-27 103709.png"
-rw-r--r--. 1 www-data www-data 2097785 Apr 13 15:32 "Screenshot 2025-03-27 103727.png"
-rw-r--r--. 1 www-data www-data 1193239 Apr 13 15:32 "Screenshot 2025-03-27 103747.png"
-rw-r--r--. 1 www-data www-data 1221757 Apr 13 15:32 "Screenshot 2025-03-27 103807.png"
-rw-r--r--. 1 www-data www-data 16915 Apr 13 15:32 index.html
-rw-r--r--. 1 www-data www-data 612 Apr 13 15:32 index.nginx-debian.html
-rw-r--r--. 1 www-data www-data 2695 Apr 13 15:32 script.js
-rw-r--r--. 1 www-data www-data 4486 Apr 13 15:32 style.css
root@7001802c93f9: /var/www/html#

```



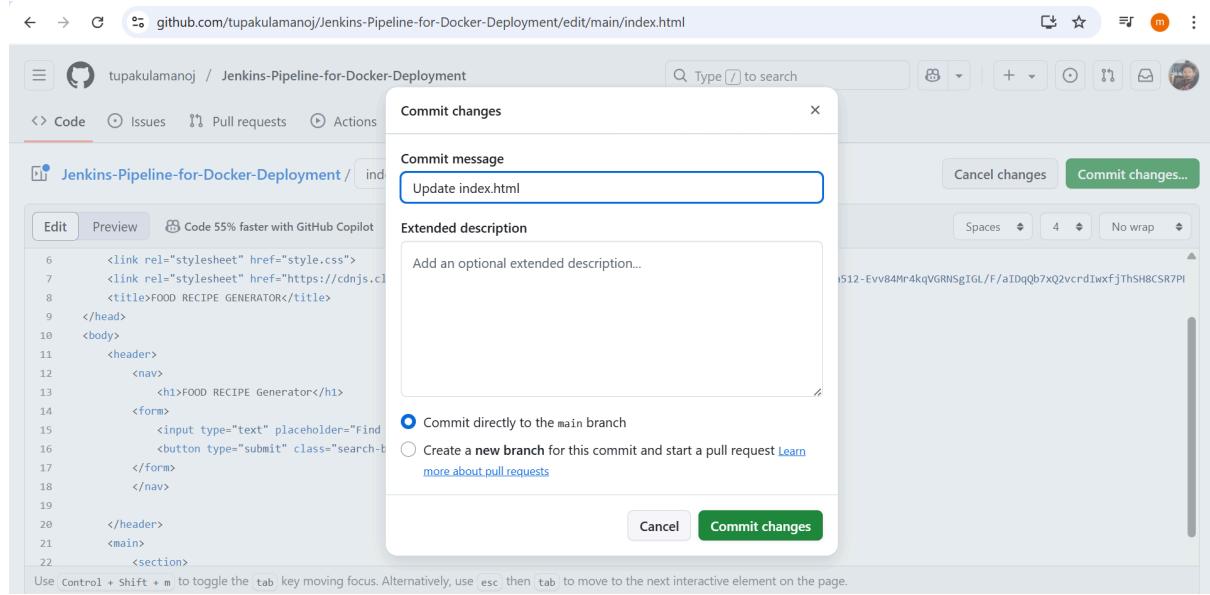
The screenshot shows a GitHub code editor interface for the file `index.html` in the `main` branch of the repository `tupakulamanoj/Jenkins-Pipeline-for-Docker-Deployment`. The code contains HTML and CSS, including a title, a search form, and some styles. The GitHub Copilot feature is active, indicated by the text "Code 55% faster with GitHub Copilot". The code editor has standard GitHub syntax highlighting and includes tabs for "Edit", "Preview", and "Copilot". There are buttons for "Cancel changes" and "Commit changes..". The GitHub interface also shows navigation icons, a search bar, and user profile information.

```

<link rel="stylesheet" href="style.css">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/css/all.min.css" integrity="sha512-Evv84Mr4kqVRNsgIGL/F/aIDqqb7xQ2vcrdIwxFjThSH8CSR
<title>FOOD RECIPE GENERATOR</title>
</head>
<body>
  <header>
    <nav>
      <h1>FOOD RECIPE Generator</h1>
    <form>
      <input type="text" placeholder="Find your recipes" class="search-input">
      <button type="submit" class="search-btn">Search</Search></button>
    </form>
  </nav>
  <main>
    <section>

```

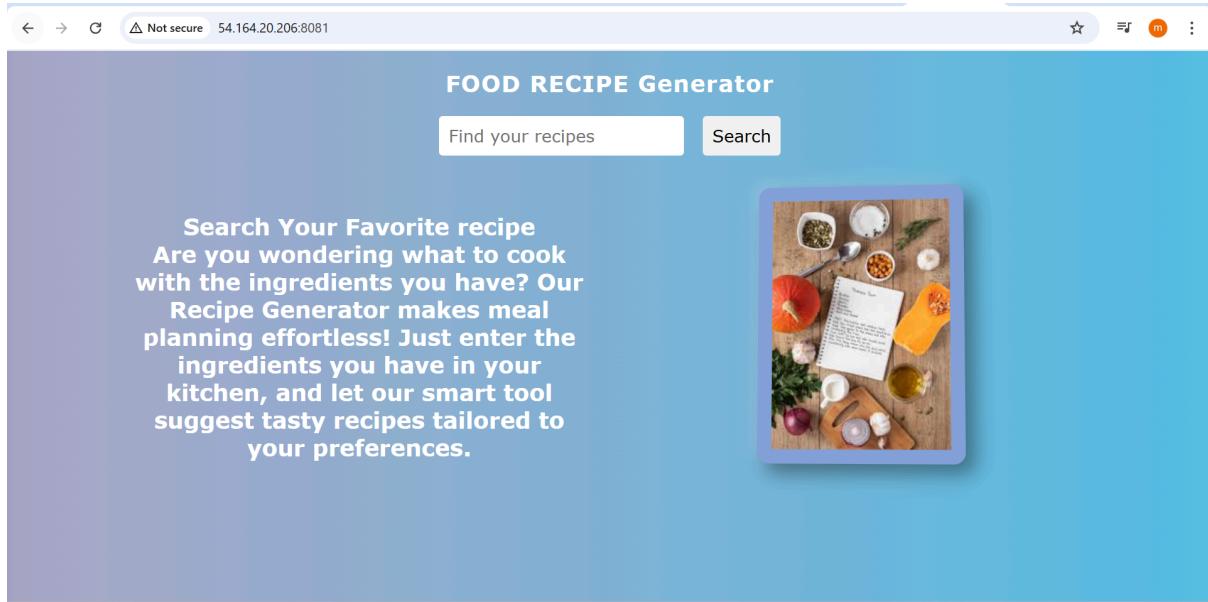
Here we are updating the h1 tag from FOOD RECIPE TO FOOD RECIPE Generator.



Pipeline Triggered

Stage	Duration
Declarative: Checkout SCM	288ms
Clone Repo	271ms
Build Docker Image	2s
Stop Old Container	1s
Deploy New Container	1s
Verify Deployment	330ms

Stage	Duration
Declarative: Checkout SCM	287ms
Clone Repo	265ms
Build Docker Image	2s
Stop Old Container	1s
Deploy New Container	1s
Verify Deployment	358ms



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.