

Project Title:

Dockerized CI/CD Pipeline with Jenkins, GitHub, and Docker Hub.

1.Introduction

This project demonstrates the implementation of a Dockerized CI/CD pipeline using Jenkins, GitHub, Docker, and Docker Hub. The purpose was to automate the process of building, testing, and deploying containerized applications. The pipeline runs on AWS EC2 instances and integrates GitHub webhooks to trigger builds automatically.

2. Objectives

- Set up a Jenkins-based CI/CD pipeline on AWS EC2.
- Automate build and deployment processes for containerized applications.
- Ensure version control and traceability of Docker images.
- Deploy applications using Docker Compose.

3. Project Procedure

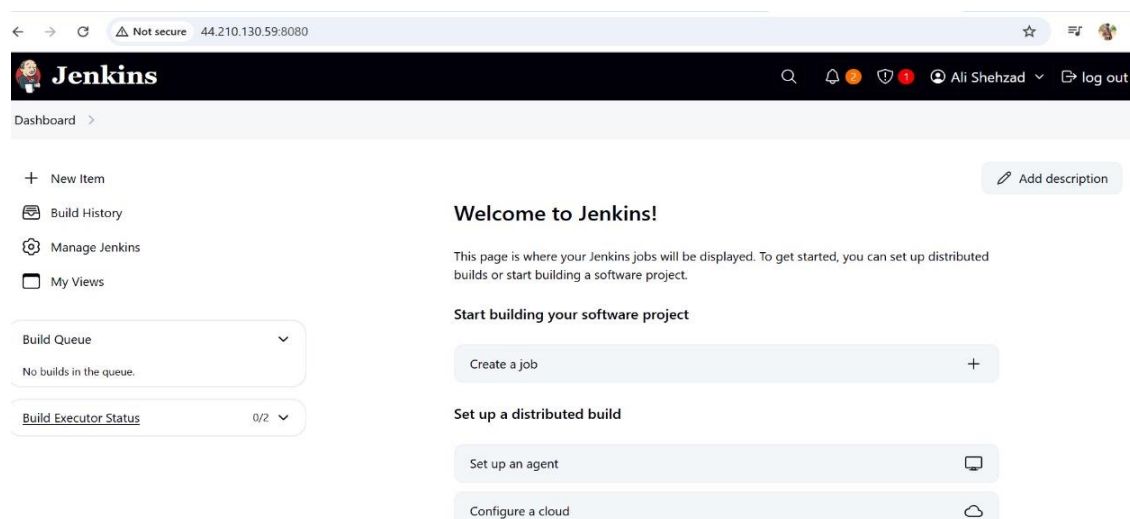
Step 1: EC2 Setup

Created two AWS EC2 instances: one master and one worker. Installed Jenkins on the master node and configured security groups and ports (8080) to access Jenkins via browser.

Step 2: Jenkins Installation and Access

Installed Jenkins ([Jenkins Script](#)) on the EC2 instance and accessed it using the public IP in a browser. Installed required plugins for Docker and GitHub integration.

Public IP:8080



Step 3: GitHub Integration

Created a GitHub repository and pushed project files, including the Jenkinsfile. Configured Jenkins to fetch pipeline code directly from GitHub using webhooks.

Step 4: Pipeline Build Process

- The application was a static login form (index.html).
- A Dockerfile was written to containerize the login form.
- The Jenkins pipeline used an image name parameter (\$img).
- Docker images were built from source code, tagged with version numbers, and pushed to Docker Hub for storage and version control.

Jenkins Pipeline Is Here: [Link](#)

Step 5: Docker Compose Deployment

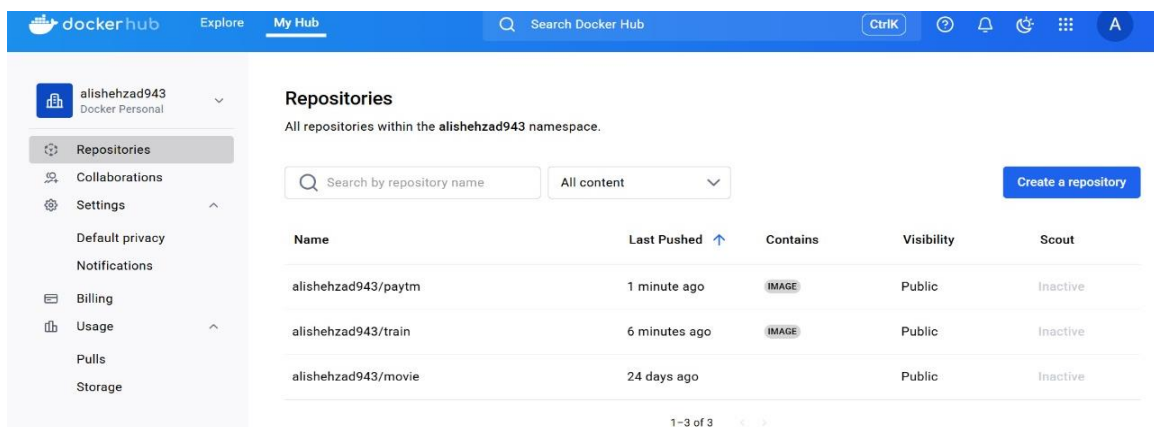
- Created a docker-compose.yml file to define the container services.
- Configured the service to run the login form application inside the container and exposed ports for browser access.
- Deployed containers using Docker Compose.

Docker Compose File is Here: [Link](#)

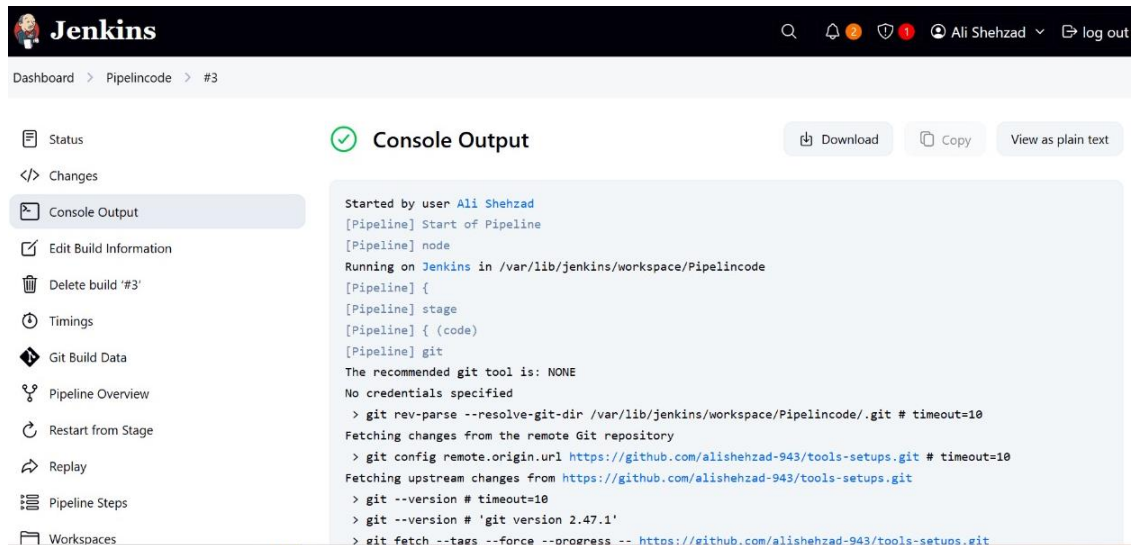
Step 6: Verification

- Opened the browser and accessed the login form (index.html) from the container.
- Verified that the containerized application was running successfully.

Verify Images in Docker Hub :



Verify By Pipeline Console Output:






The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search icon, notification icons, and the user name 'Ali Shehzad' with a 'log out' button. The breadcrumb trail is 'Dashboard > Pipelincode > #3'. On the left sidebar, the 'Console Output' tab is selected. The main area displays the console output for a pipeline run. The output starts with 'Started by user Ali Shehzad' and '[Pipeline] Start of Pipeline'. It then shows the command 'node' and 'Running on Jenkins in /var/lib/jenkins/workspace/Pipelincode'. The output continues with '[Pipeline] {', '[Pipeline] stage', '[Pipeline] { (code)', and '[Pipeline] git'. It then states 'The recommended git tool is: NONE' and 'No credentials specified'. The next line is '> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Pipelincode/.git # timeout=10', followed by 'Fetching changes from the remote Git repository'. The output then shows '> git config remote.origin.url https://github.com/alishehzad-943/tools-setups.git # timeout=10', followed by 'Fetching upstream changes from https://github.com/alishehzad-943/tools-setups.git'. The next line is '> git --version # timeout=10', followed by '> git --version # 'git version 2.47.1''. The final line is '> git fetch --tags --force --progress -- https://github.com/alishehzad-943/tools-setups.git'.

Verify Image First public Ip with container port 88:

⚠ Not secure 52.202.200.247:88




Devops By tahira

	<input type="text" value="Username"/>
	<input type="text" value="Email"/>
	<input type="password" value="Password"/>
<input type="button" value="Register"/>	

Change the Code In index.html file on GitHub and again run Pipeline. Then verify Second Image with port 84:

⚠ Not secure 52.202.200.247:84

Devops By Ali

	<input type="text" value="Username"/>
	<input type="text" value="Email"/>
	<input type="password" value="Password"/>
<input type="button" value="Register"/>	

4. Tools and Technologies

- AWS EC2: Used to host Jenkins and Docker environment.

- **Jenkins:** For automation of builds and deployments.
- **GitHub:** Source code management and pipeline integration.
- **Docker & Docker Hub:** For containerization and image registry.
- **Docker Compose:** For multi-container deployment.

5. Results

Successfully implemented CI/CD pipeline with Jenkins, Docker, and GitHub. Automated the build → tag → push process for Docker images. Images were stored in Docker Hub with proper version tags. Application deployed successfully with Docker Compose and verified in the browser.

6. Conclusion

This project demonstrated a complete DevOps pipeline workflow:

- Code pushed to GitHub.
- Jenkins automatically triggered builds.
- Docker images built, tagged, and pushed to Docker Hub.
- Application deployed using Docker Compose.
- Successful deployment verified via browser.

This pipeline provides a scalable, repeatable, and automated solution for containerized applications.