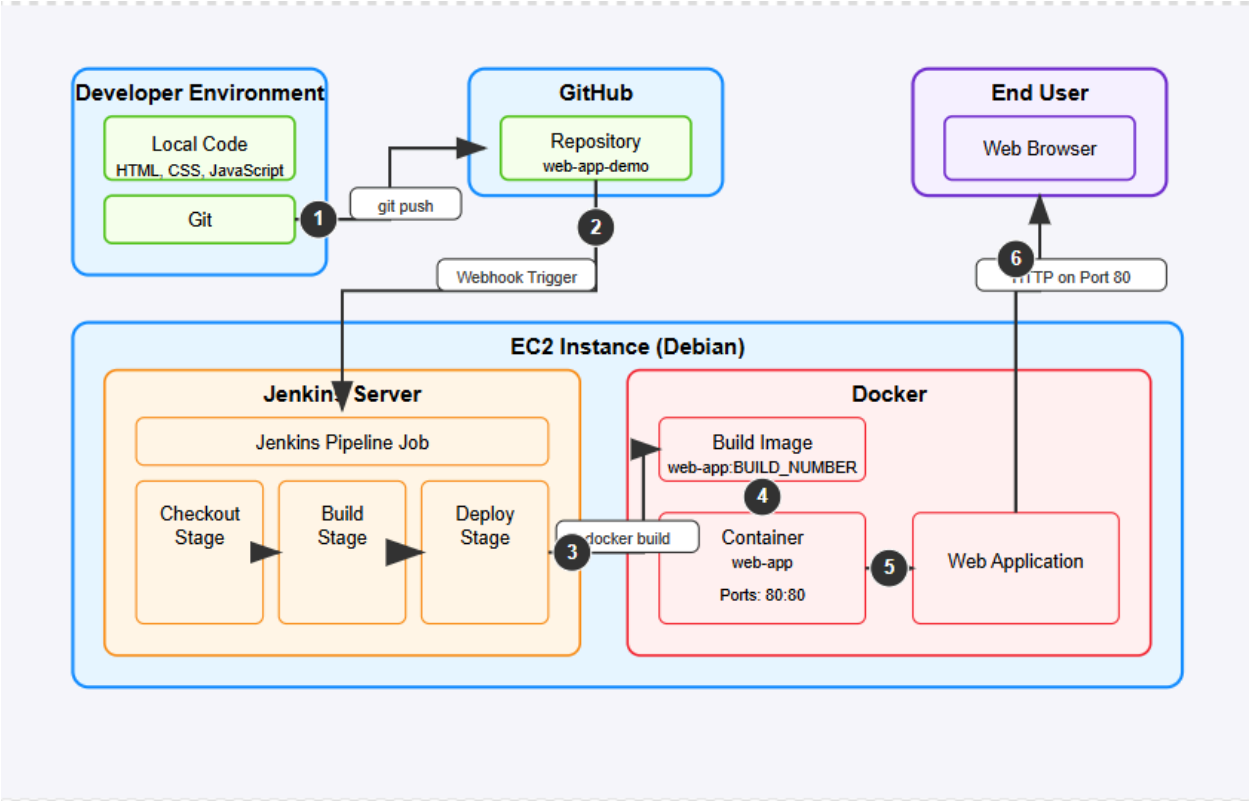
Vadakathi Muhammed Suhaib
Technical Apprentice
Emp ID: X48GRSTML
muhammed.suhaib@cprime.com

Docker, Jenkins, GitHub CI/CD Pipeline



Docker, Jenkins, GitHub CI/CD Pipeline

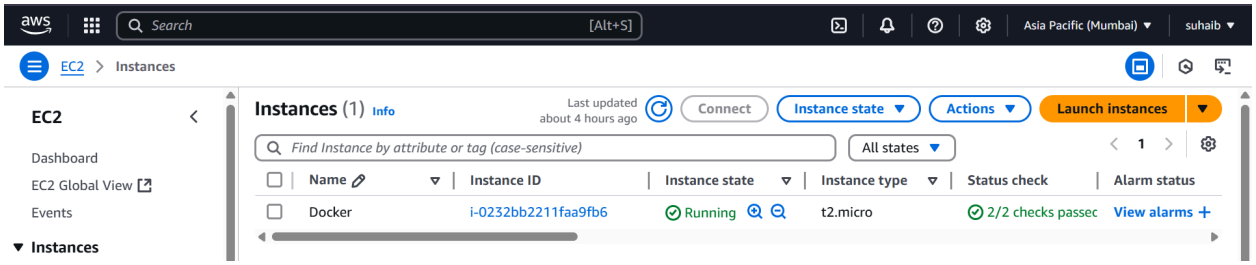
Overview

This guide provides step-by-step instructions for creating a CI/CD pipeline that:

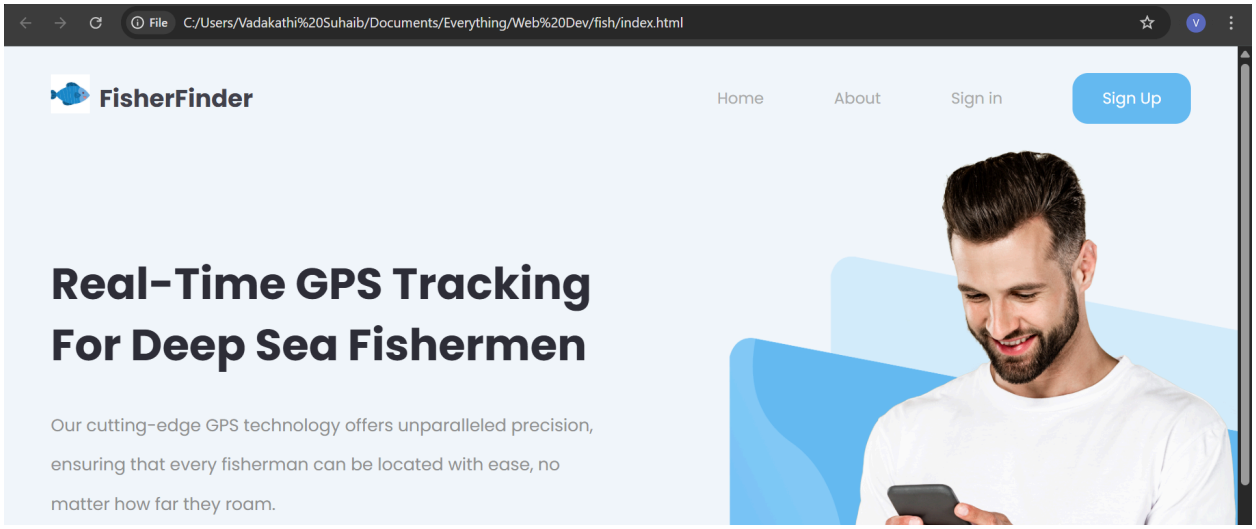
1. Pushes code from your local machine to GitHub
2. Sets up Jenkins to monitor the GitHub repository
3. Automatically deploys code changes to a Docker container
4. Demonstrates the end-to-end workflow with code modifications

Prerequisites

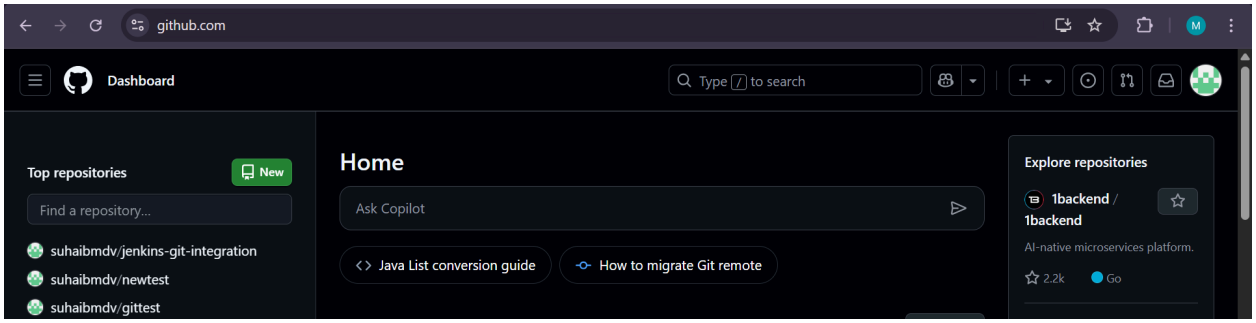
- Debian EC2 instance running



- SSH access to your EC2 instance
- Local HTML/CSS/JavaScript project ready



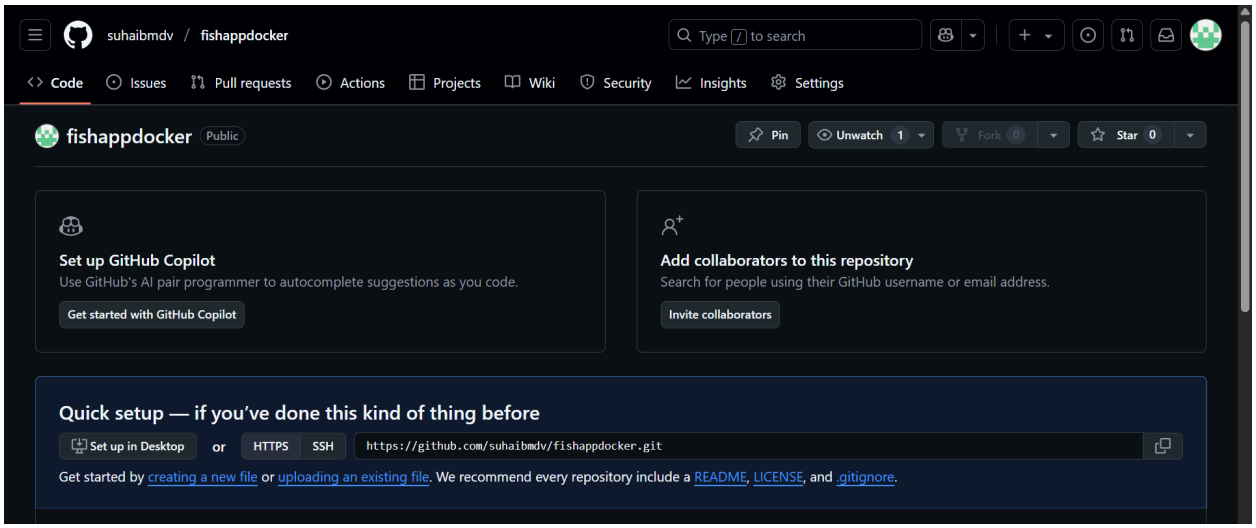
- GitHub account



Step 1: Push Local Code to GitHub

1.1 Create a New GitHub Repository

1. Go to GitHub and log in
2. Click on the "+" icon in the top-right corner
3. Select "New repository"
4. Name the repository (e.g., "web-app-demo")
5. Make it public or private according to your preference
6. Do not initialize with README, .gitignore, or license
7. Click "Create repository"



1.2 Initialize and Push Your Local Project

```
# Navigate to your project directory
cd /path/to/your/project
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    powershell
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Initialize git repository
git init
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git init
Initialized empty Git repository in C:/Users/Vadakathi Suhaib/Documents/Everything/Training @ Cprime/DevSecOps/Docker/Docker pro
ject/.git/
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Create a .gitignore file
echo "node_modules/" > .gitignore
echo ".DS_Store" >> .gitignore
echo "*.log" >> .gitignore
```

```
.gitignore U ×
.gitignore
1   node_modules/
2   .DS_Store
3   *.log
4
```

```
# Add files to git
git add .
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'app.js', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'signup.css', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'signup.html', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'signupform.js', LF will be replaced by CRLF the next time Git touches it
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Commit changes
git commit -m "Initial commit of web application"
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git commit -m "Initial comm
it of web application"
[master (root-commit) f6a2573] Initial commit of web application
 14 files changed, 899 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 app.js
 create mode 100644 img/boatman.jpg
```

```
# Add remote repository
git remote add origin https://github.com/suhaibmdv/fishappdocker.git
```

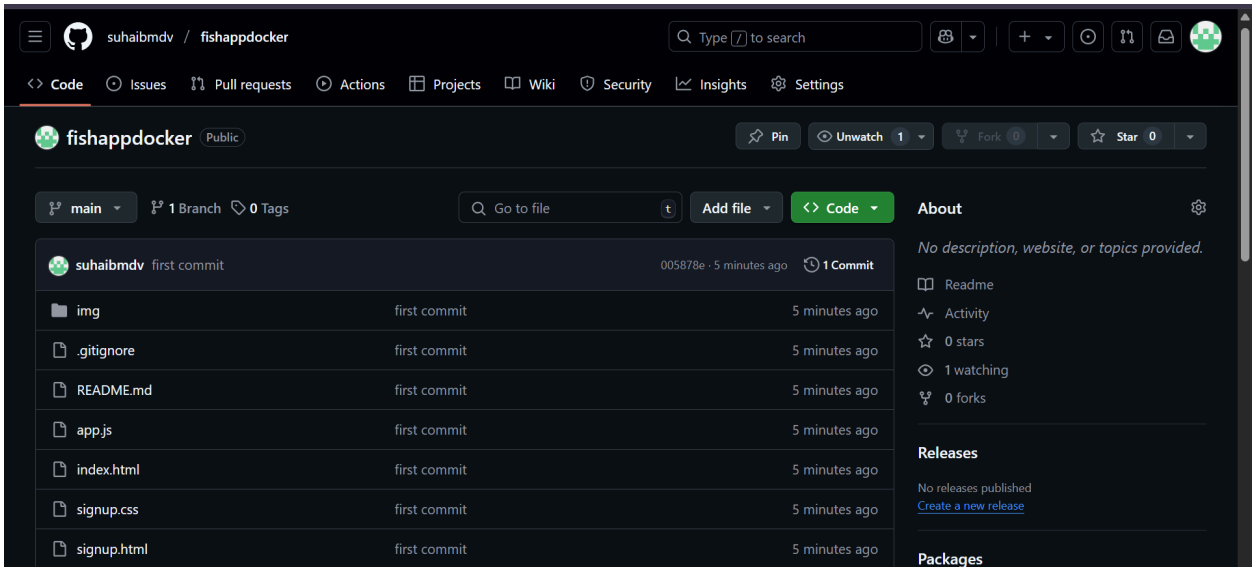
```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git remote add origin https
://github.com/suhaibmdv/fishappdocker.git
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Push to GitHub
git push -u origin main
# If you're using the default branch as master instead of main:
# git push -u origin master
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git push -u origin main
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 12 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (17/17), 2.99 MiB | 999.00 KiB/s, done.
Total 17 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/suhaibmdv/fishappdocker.git
 * [new branch]      main -> main
```

1.3 Verify Your Code on GitHub

Visit your repository URL (https://github.com/suhaibmdv/fishappdocker) to confirm your code was pushed successfully.



Step 2: Set Up Your Debian EC2 Instance

2.1 Install Docker

```
# Update package lists
sudo apt update

# Install prerequisites
sudo apt install -y apt-transport-https ca-certificates curl gnupg
lsb-release

# Add Docker's official GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg
--dearmor -o /etc/apt/keyrings/docker.gpg

# Set up the Docker repository
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update packages again
sudo apt update

# Install Docker Engine
sudo apt install -y docker-ce docker-ce-cli containerd.io
docker-compose-plugin

# Add your user to the docker group
sudo usermod -aG docker $USER

# Verify Docker installation
docker --version

# Start Docker service
sudo systemctl enable docker
sudo systemctl start docker
```

admin@ip-172-31-3-189: ~

```
root@ip-172-31-3-189:~# docker --version
Docker version 20.10.24+dfsg1, build 297e128
root@ip-172-31-3-189:~#
```

2.2 Install Jenkins

```
sudo apt update

# Install Java (required for Jenkins)
sudo apt install fontconfig openjdk-17-jre
java -version

# Add Jenkins repository key
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# Add Jenkins repository
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

# Update packages
sudo apt update

# Install Jenkins
sudo apt-get install jenkins

# Start Jenkins
sudo systemctl enable jenkins
sudo systemctl start jenkins

# Check Jenkins status
sudo systemctl status jenkins

# Get the initial admin password (you'll need this later)
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-3-189:~# java -version
openjdk version "17.0.14" 2025-01-21
OpenJDK Runtime Environment (build 17.0.14+7-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.14+7-Debian-1deb12u1, mixed mode, sharing)
root@ip-172-31-3-189:~#
```

```
root@ip-172-31-3-189:~# sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
--2025-04-11 13:09:40--  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
Resolving pkg.jenkins.io (pkg.jenkins.io)... 151.101.154.133, 2a04:4e42:24::645
Connecting to pkg.jenkins.io (pkg.jenkins.io)|151.101.154.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3175 (3.1K) [application/pgp-keys]
Saving to: '/usr/share/keyrings/jenkins-keyring.asc'

/usr/share/keyrings/j 100%[=========================>]   3.10K  --.-KB/s    in 0s

2025-04-11 13:09:40 (59.9 MB/s) - '/usr/share/keyrings/jenkins-keyring.asc' saved [3175
/3175]

root@ip-172-31-3-189:~#
```

```
root@ip-172-31-3-189:~# echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]"
\
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
root@ip-172-31-3-189:~#
```

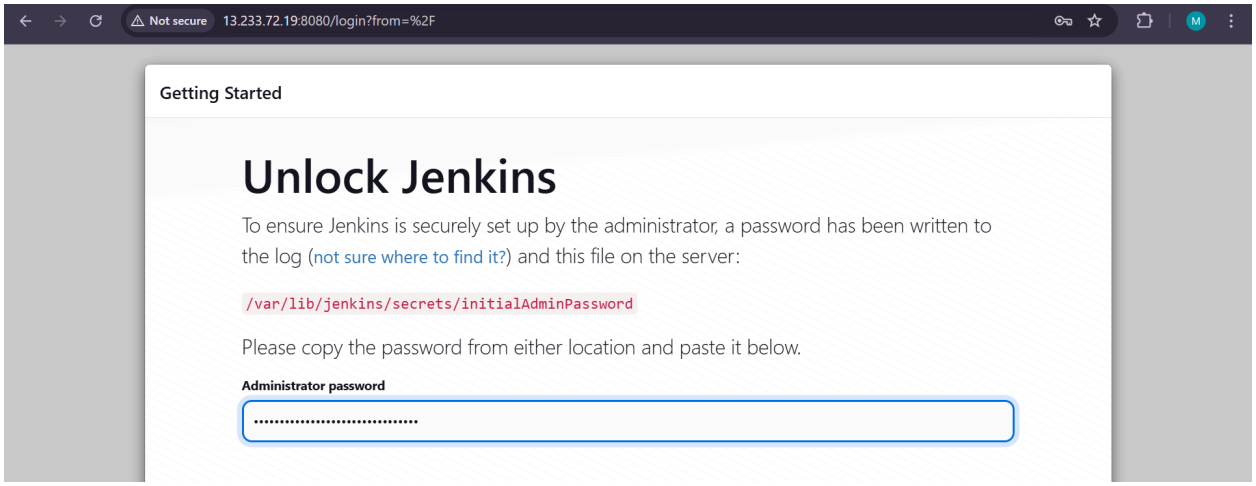
```
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /lib/syst
emd/system/jenkins.service.
Processing triggers for man-db (2.11.2-2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-3-189:~#
```

```
root@ip-172-31-3-189:~# sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-04-11 13:11:31 UTC; 30s ago
   Main PID: 6818 (java)
      Tasks: 44 (limit: 1137)
     Memory: 312.1M
```

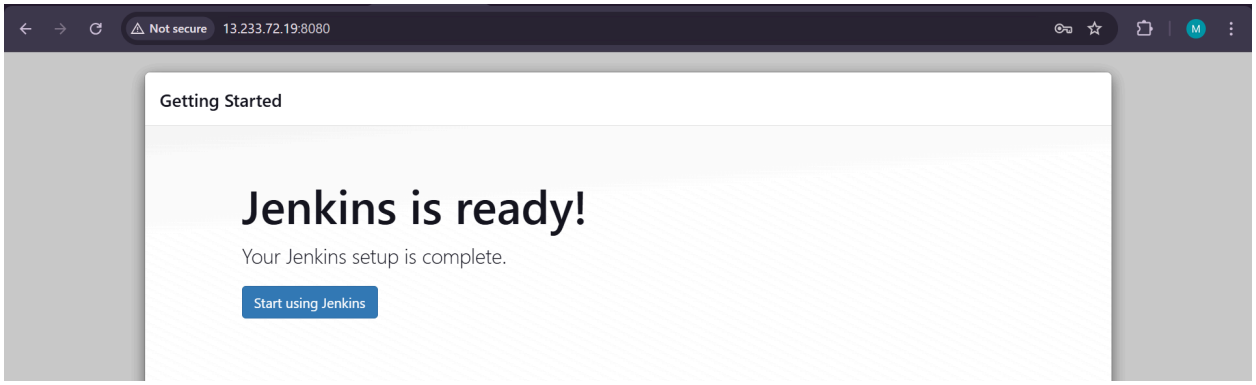
```
root@ip-172-31-3-189:~# sudo cat /var/lib/jenkins/secrets/initialAdminPassword
52bf64e8bb714db7b3345f1d70aa47ce
root@ip-172-31-3-189:~#
```

2.3 Configure Jenkins

1. Open your web browser and navigate to http://your-ec2-public-ip:8080
2. Enter the initial admin password from the previous step

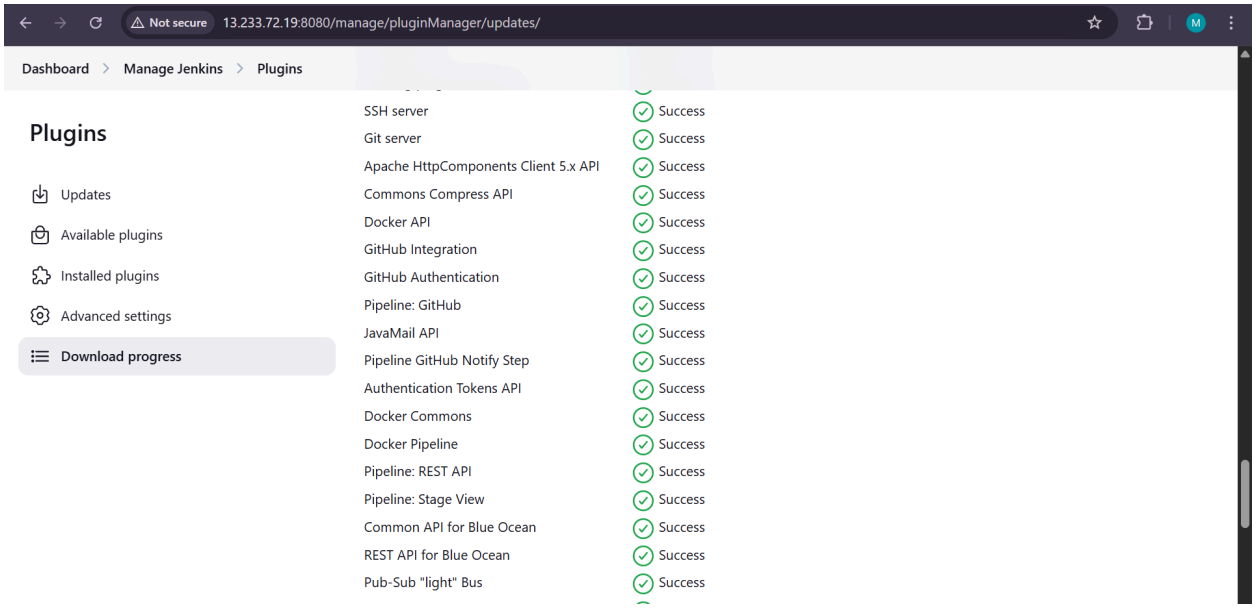


3. Choose "Install suggested plugins"
4. Create your admin user when prompted
5. Click "Save and Finish" and then "Start using Jenkins"



2.4 Install Required Jenkins Plugins

1. Go to "Manage Jenkins" > "Manage Plugins"
2. Go to the "Available" tab
3. Search and install the following plugins:
    - Git Integration
    - GitHub Integration
    - Docker Pipeline
    - Pipeline
    - Blue Ocean (optional but recommended for better UI)
4. Check "Download now and install after restart"
5. Restart Jenkins after installation is complete



2.5 Allow Jenkins to Use Docker

```
# Add Jenkins user to docker group
sudo usermod -aG docker jenkins

# Restart Jenkins to apply changes
sudo systemctl restart jenkins
```

```
root@ip-172-31-3-189:~# sudo usermod -aG docker jenkins
root@ip-172-31-3-189:~# sudo systemctl restart jenkins
root@ip-172-31-3-189:~#
```

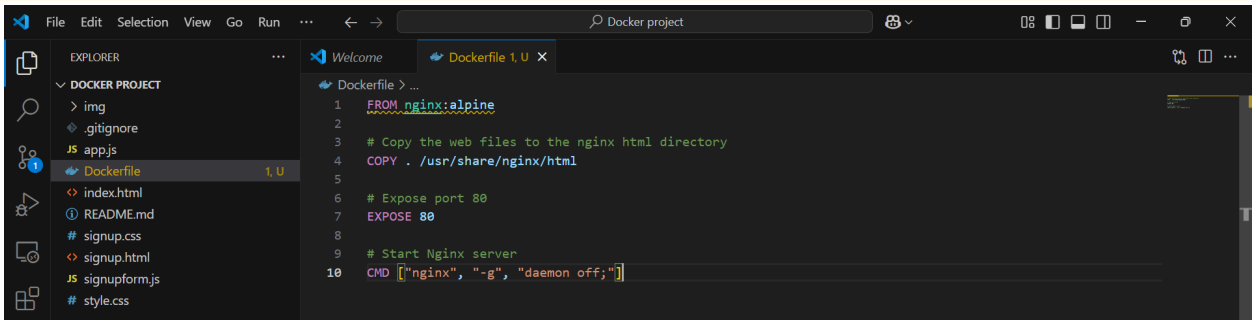
Step 3: Create a Dockerfile for Your Web Application

3.1 Create Dockerfile in Your Local Project

Create a file named `Dockerfile` in your project root:

```
FROM nginx:alpine

# Copy the web files to the nginx html directory
COPY . /usr/share/nginx/html

# Expose port 80
EXPOSE 80
```

```
# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```



3.2 Update Your Repository

```
# Add Dockerfile to git
git add Dockerfile
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git add Dockerfile
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Commit the change
git commit -m "Add Dockerfile for containerization"
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git commit -m "Add Dockerfi
le for containerization"
[main c025e1d] Add Dockerfile for containerization
 1 file changed, 10 insertions(+)
 create mode 100644 Dockerfile
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Push to GitHub
git push
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 426 bytes | 106.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/suhaibmdv/fishappdocker.git
```

Step 4: Create Jenkins Pipeline

4.1 Create a Jenkinsfile

Create a file named `Jenkinsfile` in your project root:

```
pipeline {
    agent any
```

```
    stages {
        stage('Checkout') {
            steps {
                // Get code from GitHub repository
                checkout scm
                echo 'Checkout completed'
            }
        }

        stage('Build') {
            steps {
                // Build the Docker image
                sh 'docker build -t web-app:${BUILD_NUMBER} .'
                echo 'Build completed'
            }
        }

        stage('Deploy') {
            steps {
                // Stop existing container if it exists
                sh '''
                    docker ps -f name=web-app -q | xargs --no-run-if-empty
docker stop
                    docker ps -a -f name=web-app -q | xargs
--no-run-if-empty docker rm
                '''

                // Run the new container
                sh 'docker run -d -p 80:80 --name web-app
web-app:${BUILD_NUMBER}'
                echo 'Deployment completed'
            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed!'
        }
```

```
    }
}
```

4.2 Update Your Repository

```
# Add Jenkinsfile to git
git add Jenkinsfile
```

```
# Commit the change
git commit -m "Add Jenkinsfile for CI/CD pipeline"
```

```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git add Jenkinsfile
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git commit -m "Add Jenkinsf
ile for CI/CD pipeline"
[main a28e5d1] Add Jenkinsfile for CI/CD pipeline
 1 file changed, 44 insertions(+)
 create mode 100644 Jenkinsfile
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project>
```

```
# Push to GitHub
git push
```

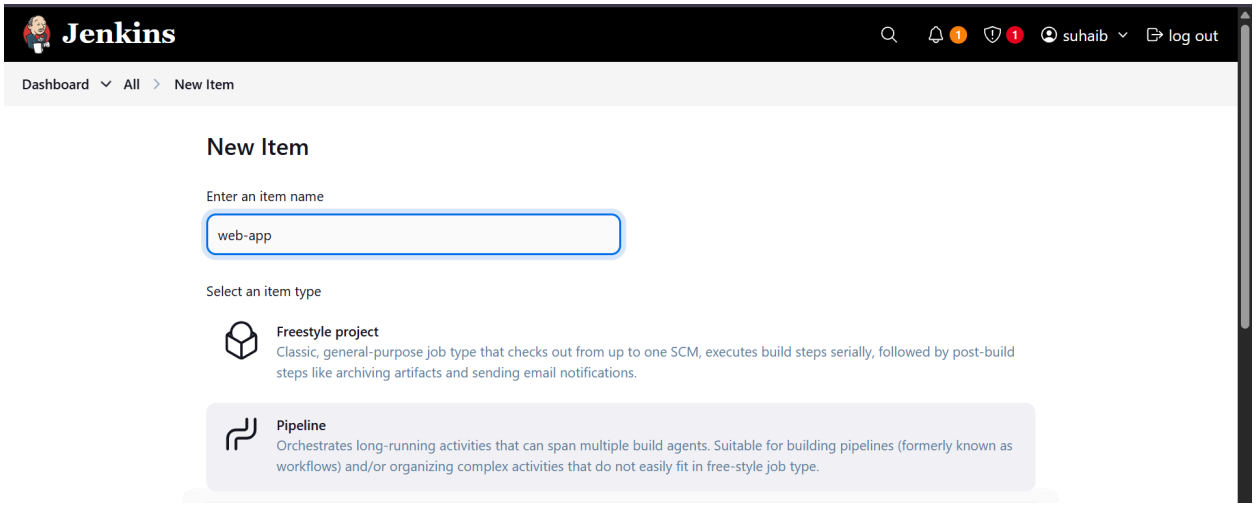
```
PS C:\Users\Vadakathi Suhaib\Documents\Everything\Training @ Cprime\DevSecOps\Docker\Docker project> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 656 bytes | 218.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/suhaibmdv/fishappdocker.git
```

Step 5: Set Up Jenkins Pipeline

5.1 Create a New Pipeline Job in Jenkins

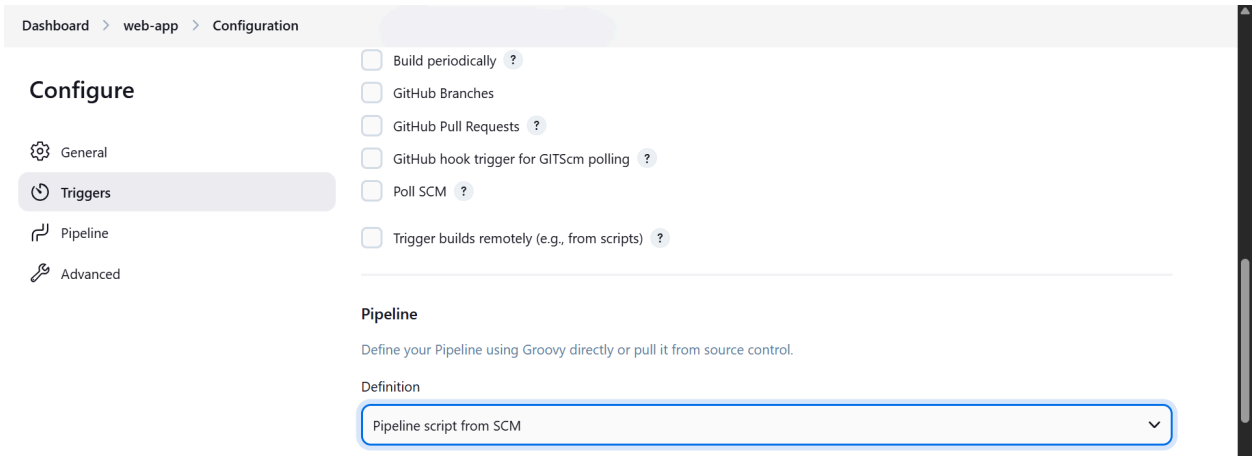
1. Log in to Jenkins
2. Click "New Item"
3. Enter a name (e.g., "web-app-pipeline")

4. Select "Pipeline" and click "OK"



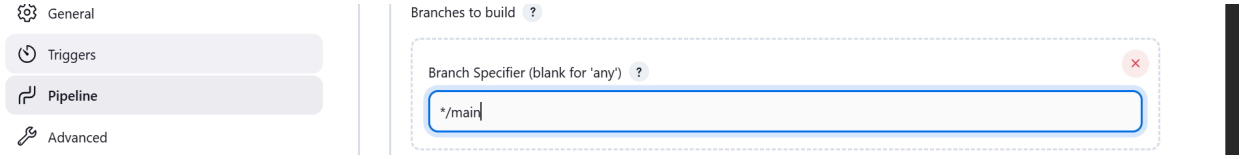
5.2 Configure the Pipeline

1. Scroll down to the "Pipeline" section
2. Select "Pipeline script from SCM" for Definition



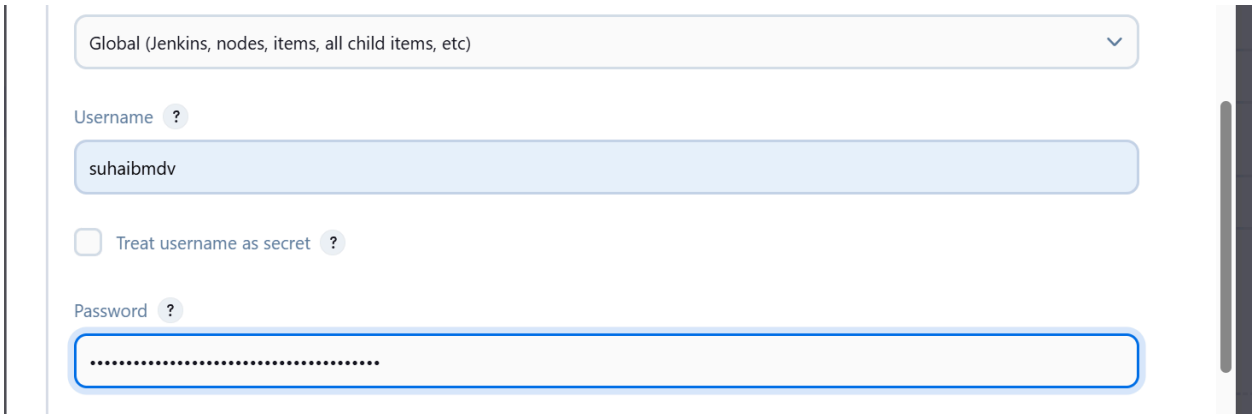
3. Select "Git" for SCM
4. Enter your repository URL: https://github.com/suhaibmdv/fishappdocker.git

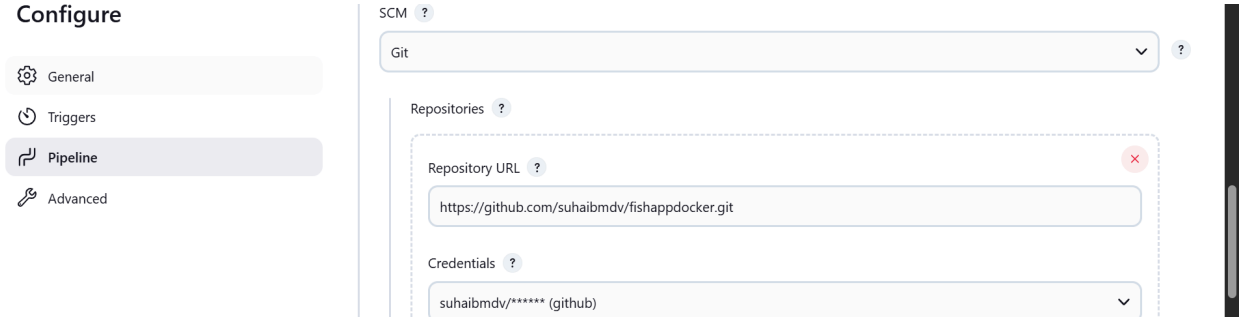
5. Specify the branch to build (e.g., */main or */master)



6. Set "Script Path" to "Jenkinsfile"

7. Add the username and personal access token to credentials to access github

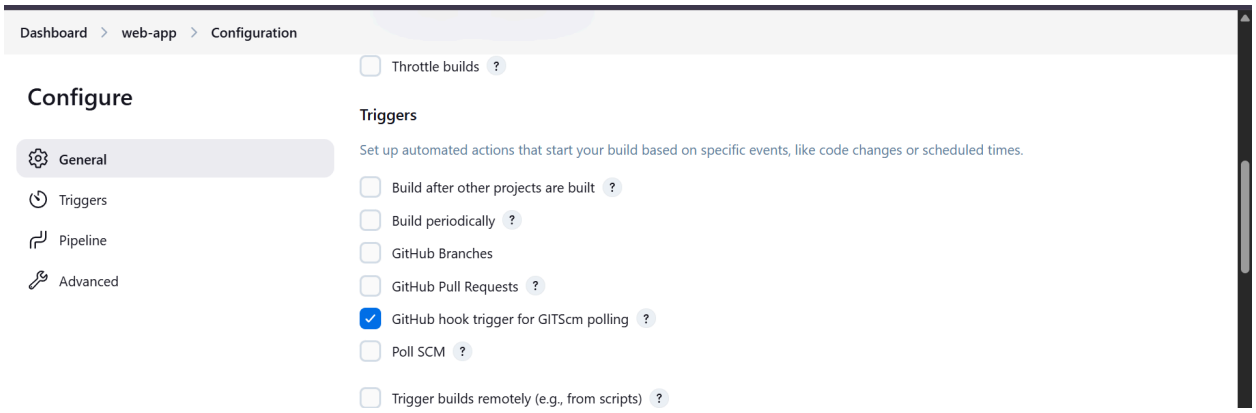




8. Click "Save"

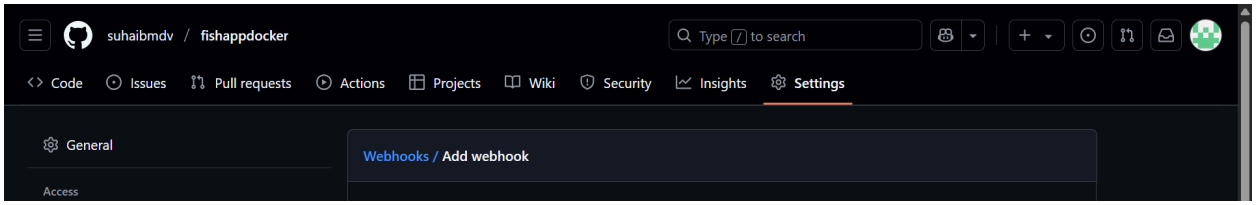
5.3 Set Up Webhook for Automatic Builds

1. In Jenkins, go to the project
2. Click "Configure"
3. Under "Build Triggers", check "GitHub hook trigger for GITScm polling"
4. Save the configuration

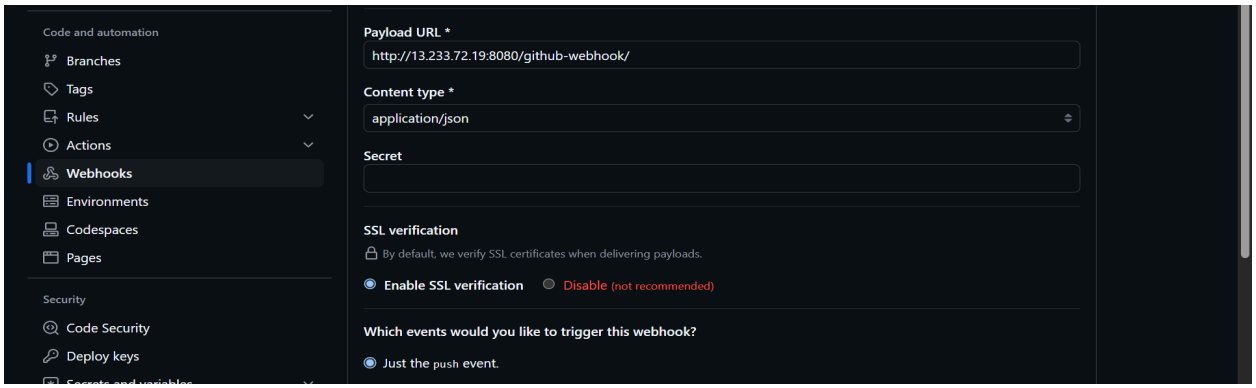


5.4 Set Up GitHub Webhook

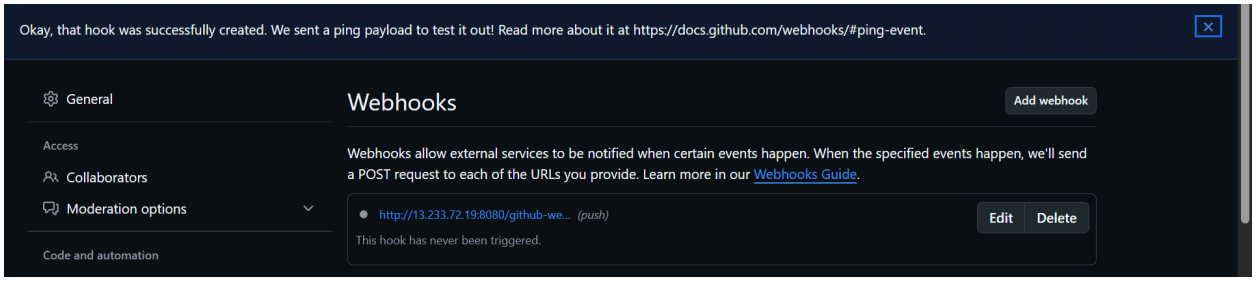
1. Go to your GitHub repository
2. Click "Settings" > "Webhooks" > "Add webhook"



3. For Payload URL, enter: http://13.233.72.19:8080/github-webhook/
4. Choose "Content type" as "application/json"
5. Select "Just the push event"



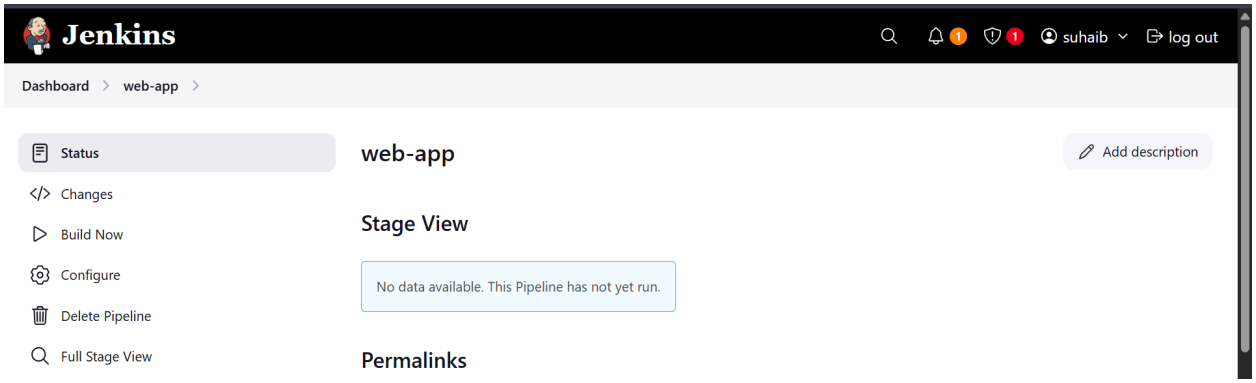
6. Click "Add webhook"



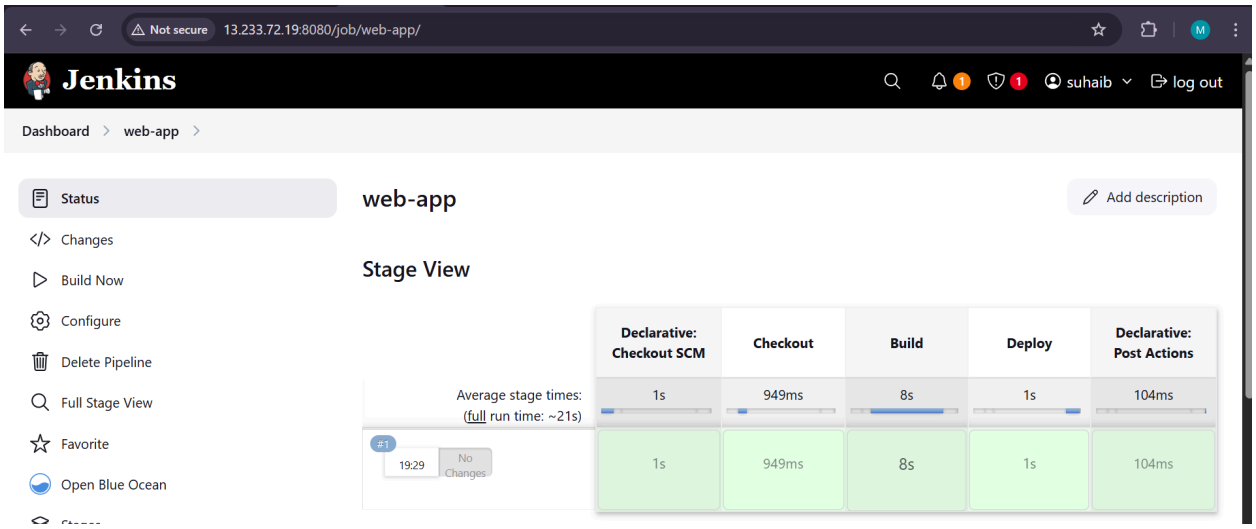
Step 6: Test the Pipeline

6.1 Manually Trigger the First Build

1. In Jenkins, navigate to your pipeline
2. Click "Build Now"



3. Monitor the build progress



Jenkins

suhaib log out

Dashboard > web-app > #1

- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#1'
- Timings
- Git Build Data
- Open Blue Ocean
- Pipeline Overview
- Pipeline Console
- Restart from Stage
- Replay
- Pipeline Steps
- Workspaces

Console Output

Download Copy View as plain text

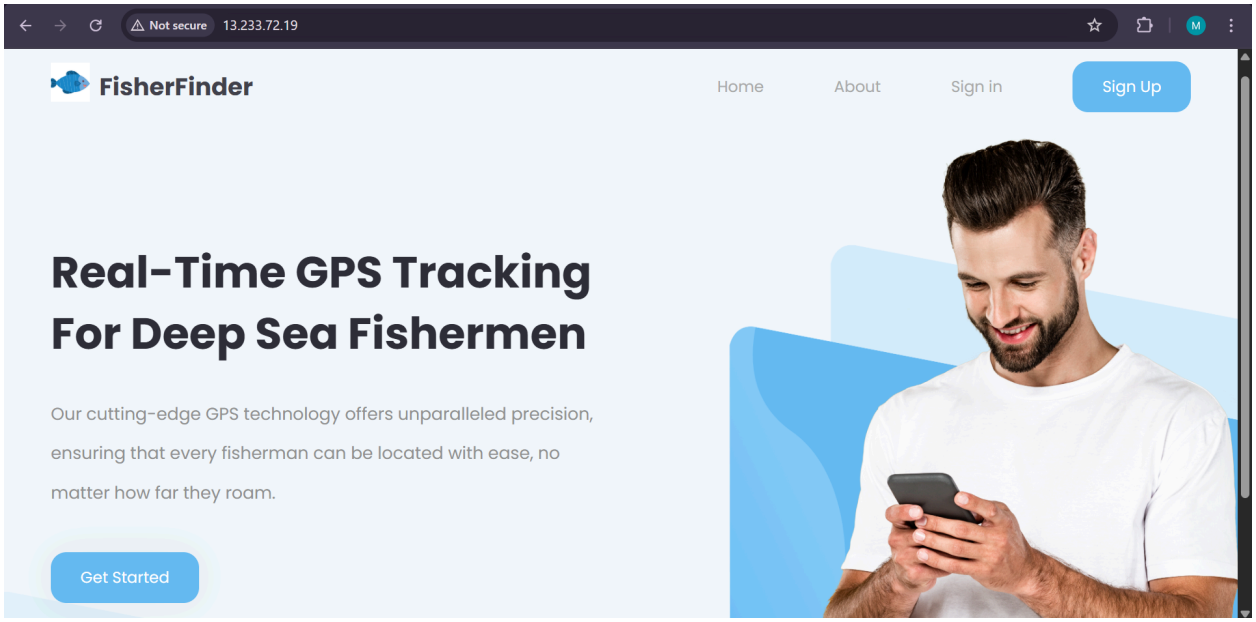
```
Started by user suhaib
Obtained Jenkinsfile from git https://github.com/suhaibmdv/fishappdocker.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/web-app
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
using credential 635ff2ca-05d2-4b1b-b9b6-80ebe64c2d79
Cloning the remote Git repository
Cloning repository https://github.com/suhaibmdv/fishappdocker.git
 > git init /var/lib/jenkins/workspace/web-app # timeout=10
Fetching upstream changes from https://github.com/suhaibmdv/fishappdocker.git
```

```
 > git --version # 'git version 2.39.5'
using GIT_ASKPASS to set credentials github
 > git fetch --tags --force --progress -- https://github.com/suhaibmdv/fishappdocker.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/suhaibmdv/fishappdocker.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision a28e5d1f4345b0f327955aa9f43cf65631034a65 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f a28e5d1f4345b0f327955aa9f43cf65631034a65 # timeout=10
Commit message: "Add Jenkinsfile for CI/CD pipeline"
First time build. Skipping changelog.
```

Dashboard > web-app > #1

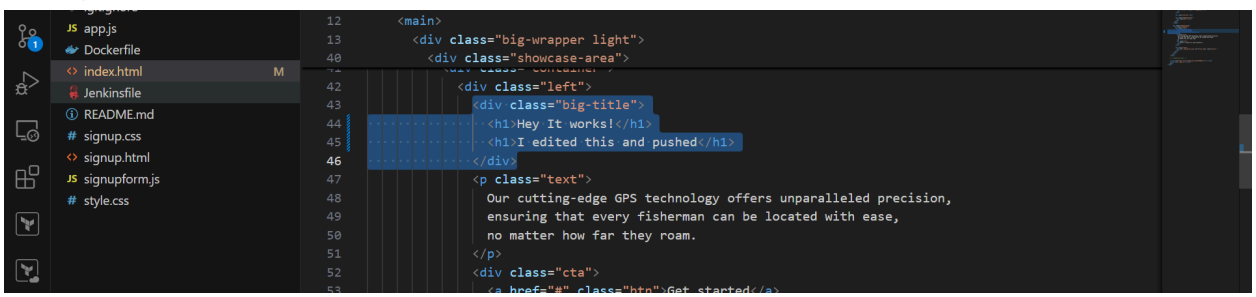
```
+ docker run -d -p 80:80 --name web-app web-app:1
d7fee4084dec159a29462427a5bc974ae0f6c63e3603efe1bfaf206c055b77c9
[Pipeline] echo
Deployment completed
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline completed successfully!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

4. Check if the website is deployed at http://your-ec2-public-ip



6.2 Make a Change to Your Code Locally

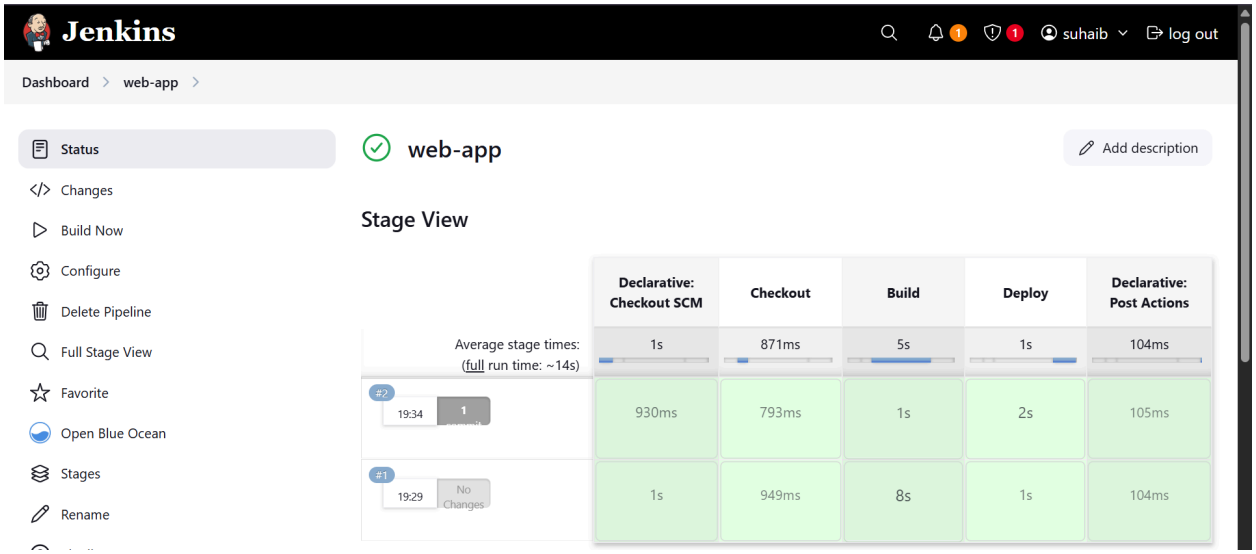
Modify an HTML, CSS, or JavaScript file



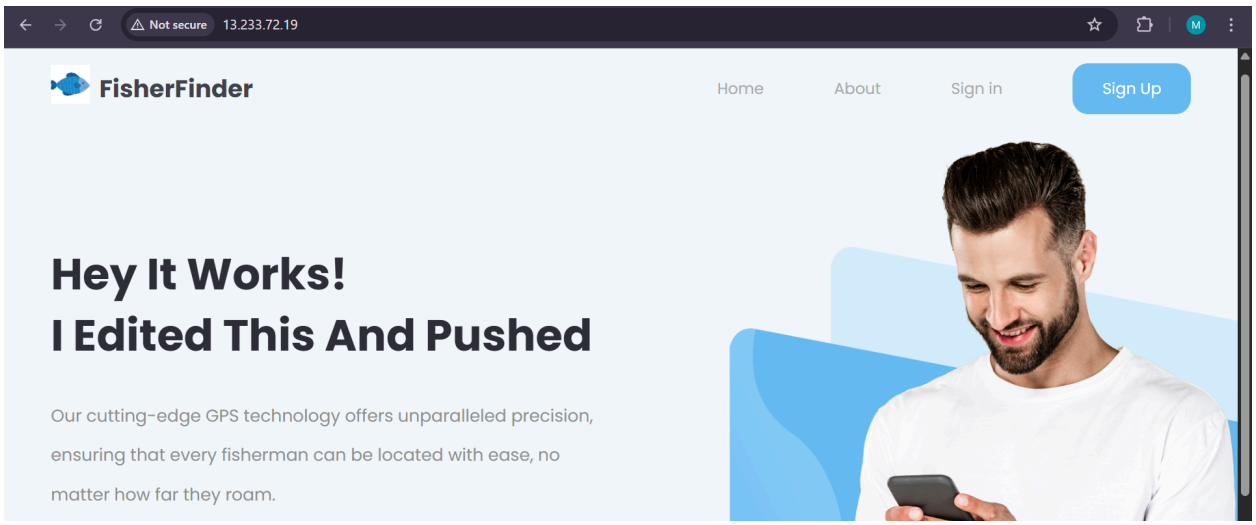

```
git add .
git commit -m "Update website title"
git push
```

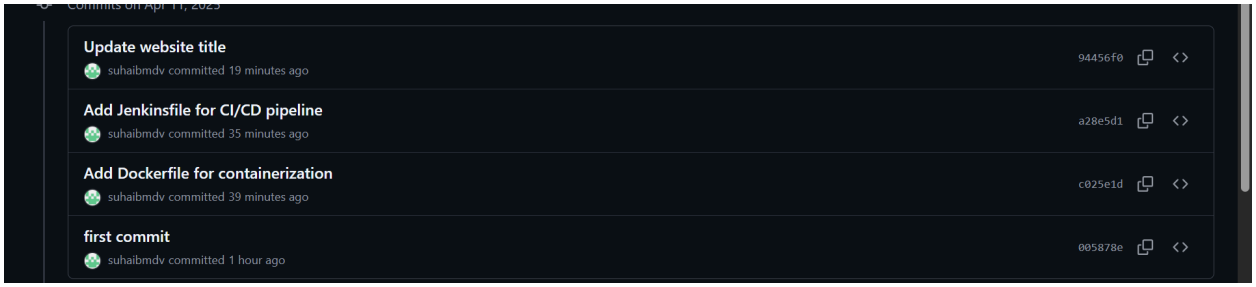
6.3 Verify the CI/CD Pipeline

1. Observe the Jenkins pipeline automatically triggering
2. Watch it progress through the Checkout, Build, and Deploy stages



3. Once completed, verify the changes on your website at http://your-ec2-public-ip





4. Confirm the Docker container is running:

```
docker ps
```

```
root@ip-172-31-3-189:~# docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORT
S                                   NAMES
b2d879f2c968   web-app:2    "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes   0.0
0.0:80->80/tcp, :::80->80/tcp   web-app
root@ip-172-31-3-189:~#
```

Step 7: Security Considerations (Optional but Recommended)

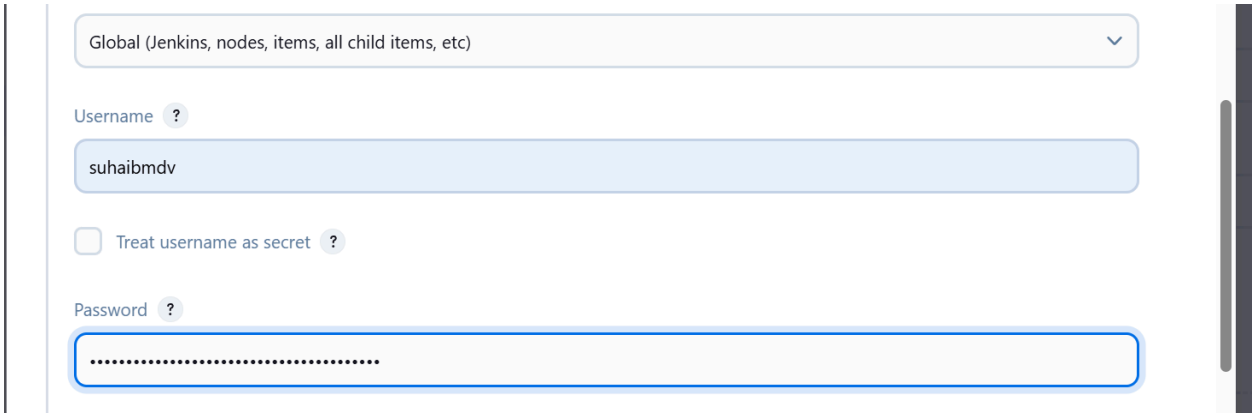
7.1 Secure Jenkins

```
# Set up a reverse proxy with NGINX
sudo apt install -y nginx

# Create an NGINX configuration for Jenkins
sudo nano /etc/nginx/sites-available/jenkins

# Add the following configuration

server {
    listen 80;
    server_name jenkins.yourdomain.com;  # Replace with your domain or IP

    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

# Create a symbolic link
sudo ln -s /etc/nginx/sites-available/jenkins /etc/nginx/sites-enabled/

# Test and restart nginx
sudo nginx -t
sudo systemctl restart nginx
```

```
# Configure firewall (optional)
sudo apt install -y ufw
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw enable
```

7.2 Use GitHub Credentials Securely

1. In Jenkins, go to "Manage Jenkins" > "Manage Credentials"
2. Click on "Jenkins" under "Stores scoped to Jenkins"
3. Click on "Global credentials"
4. Click "Add Credentials"
5. Choose "Username with password"
6. Enter your GitHub username and password/token
7. Set ID (e.g., "github-credentials")
8. Click "OK"
9. Update your Pipeline to use these credentials



Troubleshooting

Common Issues

1. **Jenkins cannot connect to GitHub**
   - Check your GitHub webhook configuration
   - Ensure your EC2 security group allows inbound traffic on port 8080
2. **Docker permission issues**
   - Ensure Jenkins user is in the docker group

- Restart Jenkins after adding to group

```
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

3. **Pipeline fails during Docker build**
   - Check if Docker is running: `sudo systemctl status docker`
   - Review Jenkins logs for specific errors

4. **Cannot access the deployed website**
   - Verify Docker container is running: `docker ps`
   - Check EC2 security group allows inbound traffic on port 80
   - Inspect container logs: `docker logs web-app`

Summary

You have now set up a complete CI/CD pipeline that:

1. Stores your code in GitHub
2. Automatically triggers Jenkins when code changes are pushed
3. Builds a Docker image from your code
4. Deploys the Docker container with your web application
5. Makes the application accessible via your EC2 instance's public IP

This automation allows you to focus on development while ensuring changes are quickly and consistently deployed to your production environment.