Placement Empowerment Program

Cloud Computing and DevOps Centre

Automate Docker Image Builds Using GitHub Actions: Set up a GitHub Actions workflow to build and push a Docker image to a Docker Hub repository whenever code is pushed to the repository.

Name: Hema S

Department: ECE

Introduction

In modern software development, automation plays a crucial role in ensuring efficiency and reliability. This Proof of Concept (PoC) demonstrates how to automate Docker image builds using **GitHub Actions** and push them to **Docker Hub**. By integrating CI/CD practices, developers can streamline the containerization process and ensure that every change to the source code triggers an automated build and deployment.

Overview

This PoC covers the following key steps:

1. **Setting up a Dockerfile** – Creating a containerized environment using a simple Nginx-based Docker image.
2. **Configuring GitHub Actions** – Writing a GitHub Actions workflow to automate Docker builds.
3. **Authenticating with Docker Hub** – Using GitHub Secrets for secure login to Docker Hub.
4. **Building and Pushing the Image** – Automating the build and push process upon code commits.
5. **Verifying the Image** – Pulling and running the pushed image locally to confirm success.

Objective

The main objective of this PoC is to:

1. **Automate Docker image builds** using GitHub Actions.
2. **Eliminate manual Docker build and push steps**, reducing deployment overhead.
3. **Ensure consistency in containerized environments** with version-controlled builds.
4. **Enhance CI/CD practices** by integrating Docker with GitHub.

Importance

1. **Increases Developer Productivity:** Automating builds removes repetitive manual tasks.
2. **Ensures Deployment Consistency:** Every build is reproducible and follows a version-controlled process.
3. **Improves Security:** Secrets management in GitHub Actions ensures safe authentication with Docker Hub.
4. **Accelerates CI/CD Pipelines:** Streamlining image builds allows for faster deployments and testing.
5. **Facilitates Collaboration:** Any team member pushing code to the repository automatically triggers a new Docker image build.

Step-by-Step Overview

Step 1:

1. Install Git

Download Git from Git's official website.

Verify installation by opening **Command Prompt (cmd)** and running:

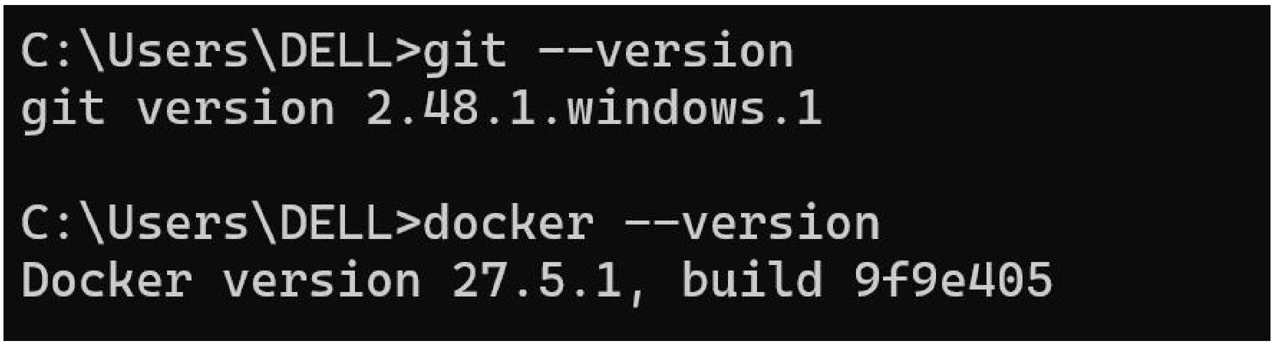
git --version

2. Install Docker Desktop

Download and install Docker Desktop from Docker's official website.

Verify by running:

docker –version



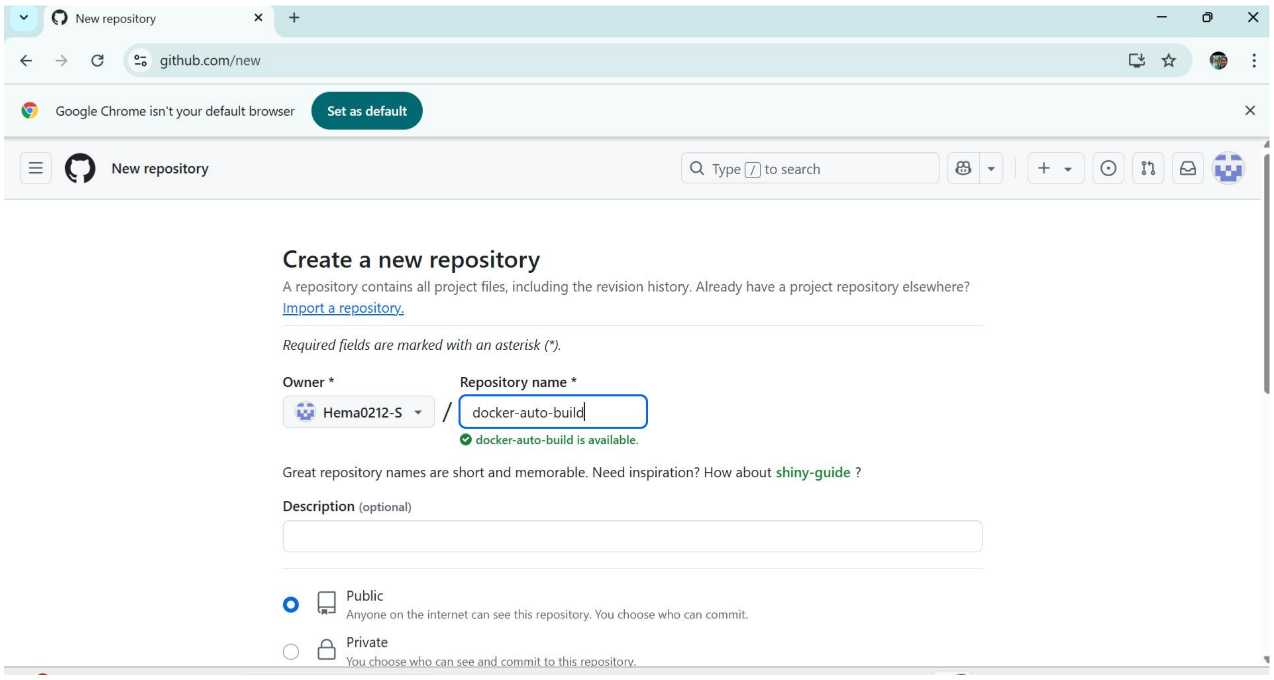
```
C:\Users\DELL>git --version
git version 2.48.1.windows.1

C:\Users\DELL>docker --version
Docker version 27.5.1, build 9f9e405
```

Step 2:

1. Go to GitHub and log in.

2. Click **New Repository** → **Give it a name (e.g., docker-auto-build)**.

3. Choose **Public** or **Private** and click **Create Repository**.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * Hema0212-S / Repository name * docker-auto-build

docker-auto-build is available.

Great repository names are short and memorable. Need inspiration? How about shiny-guide ?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Step 3:

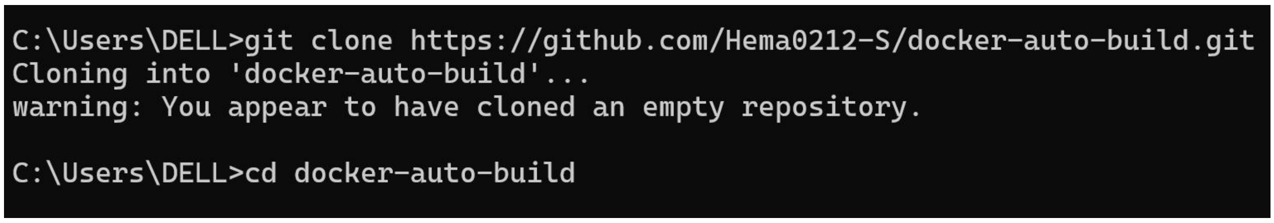
1. Open **Command Prompt (cmd)** and run:

git clone https://github.com/YOUR_GITHUB_USERNAME/docker-auto-build.git

(Replace YOUR_GITHUB_USERNAME with your actual GitHub username.)

2. Navigate into the cloned folder:

cd docker-auto-build

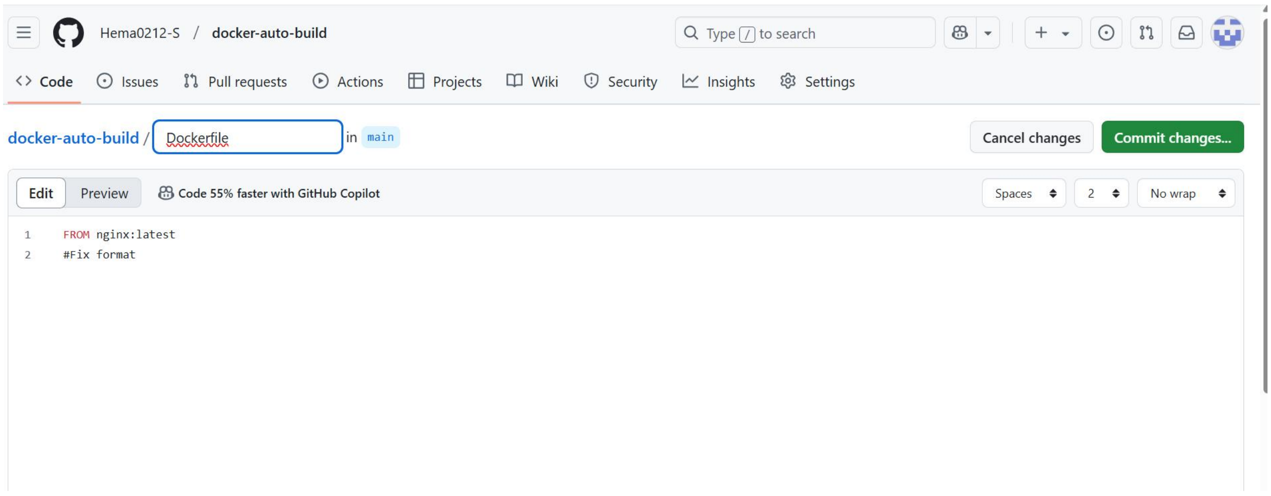


```
C:\Users\DELL>git clone https://github.com/Hema0212-S/docker-auto-build.git
Cloning into 'docker-auto-build'...
warning: You appear to have cloned an empty repository.

C:\Users\DELL>cd docker-auto-build
```

Step 4:

A **Dockerfile** defines how your application should be containerized.

1. Inside the repository folder, create a new file named **Dockerfile**.
2. Open it in **Notepad** .
3. Add the following content (example for an Nginx web server):
4. Save the file.



```
FROM nginx:latest
#Fix format
```

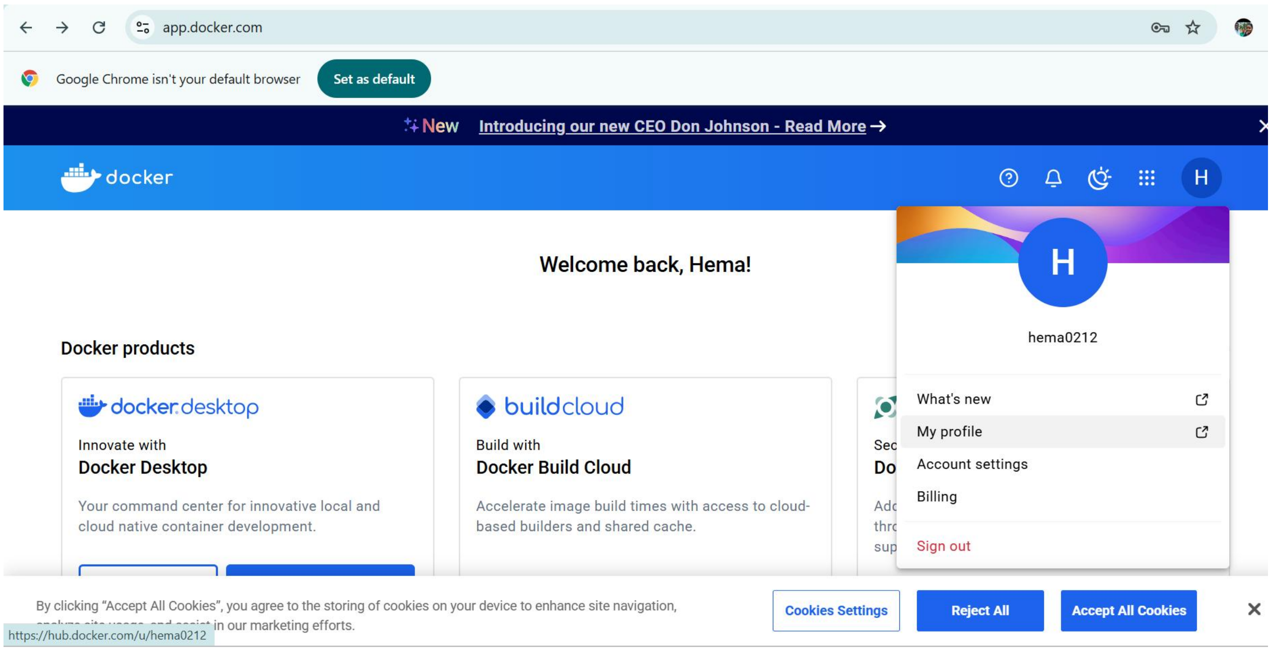
Step 5:

Since we need to push the Docker image to **Docker Hub**, we must store our **Docker Hub username and password** securely in GitHub.

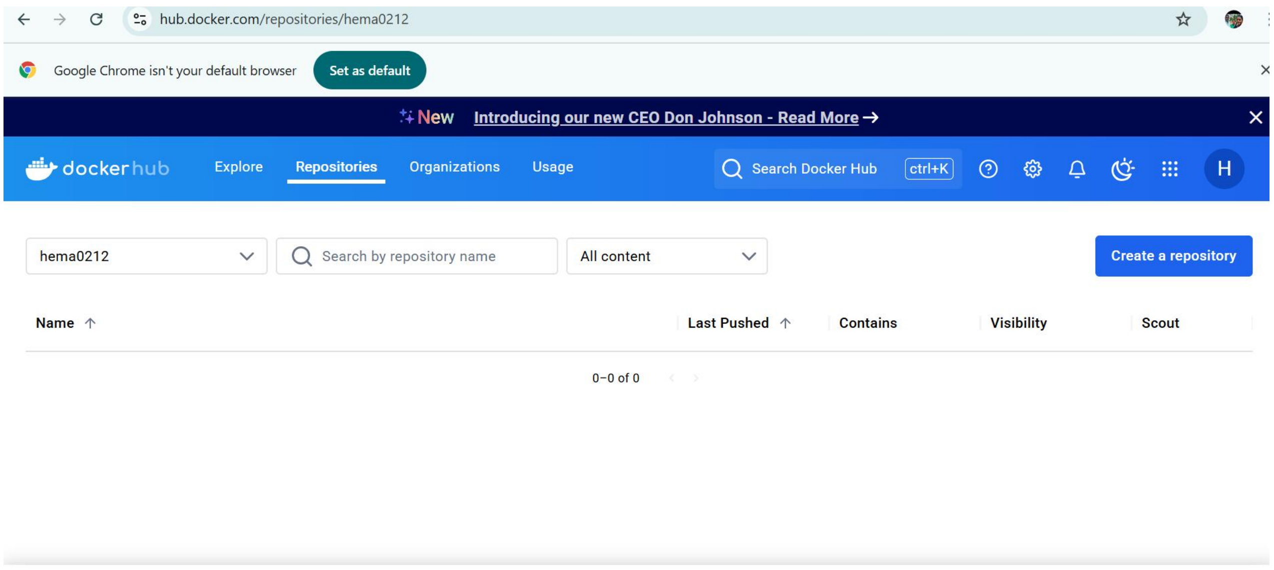
Get a Docker Hub Account

Go to Docker Hub and sign up (if you don’t have an account).

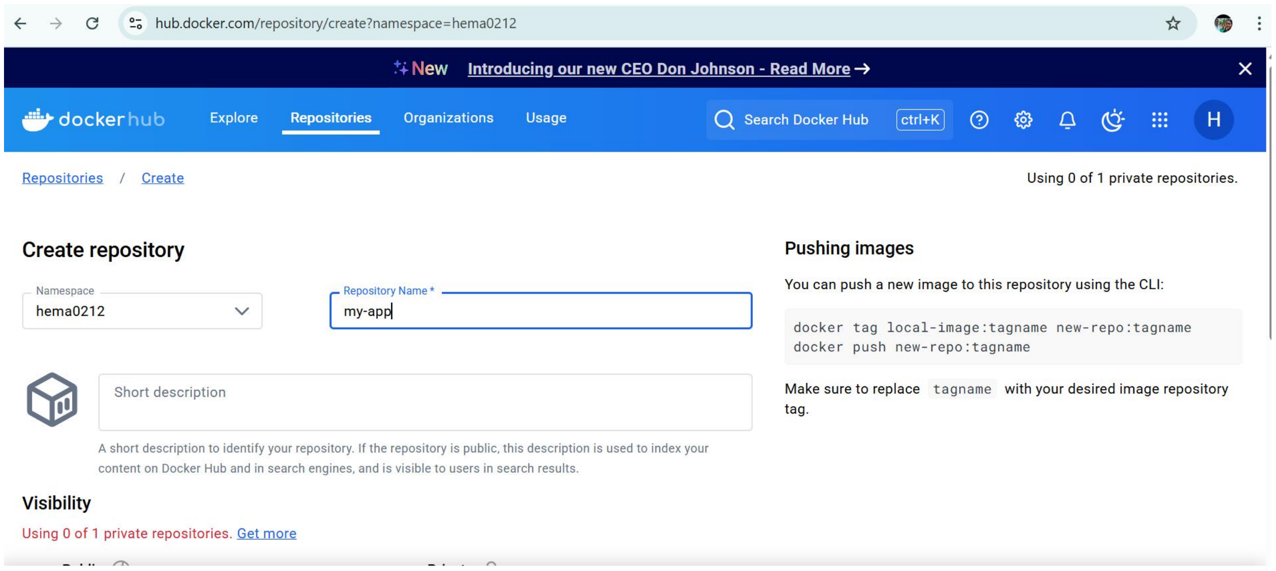
Click **Create Repository** → Name it **my-app** → Set it to **Public** or **Private**.



app.docker.com
Google Chrome isn't your default browser
Set as default
New Introducing our new CEO Don Johnson - Read More →
docker
H
Welcome back, Hema!
hema0212
What's new
My profile
Account settings
Billing
Sign out
Docker products
docker desktop
Innovate with
Docker Desktop
Your command center for innovative local and cloud native container development.
build cloud
Build with
Docker Build Cloud
Accelerate image build times with access to cloud-based builders and shared cache.
By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.
Cookies Settings
Reject All
Accept All Cookies
https://hub.docker.com/u/hema0212



,



hub.docker.com/repository/create?namespace=hema0212
New Introducing our new CEO Don Johnson - Read More →
docker hub
Explore
Repositories
Organizations
Usage
Search Docker Hub
ctrl+K
H
Repositories / Create
Using 0 of 1 private repositories.
Create repository
Namespace
hema0212
Repository Name *
my-app
Short description
A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.
Visibility
Using 0 of 1 private repositories. Get more
Pushing images
You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

Step 6:

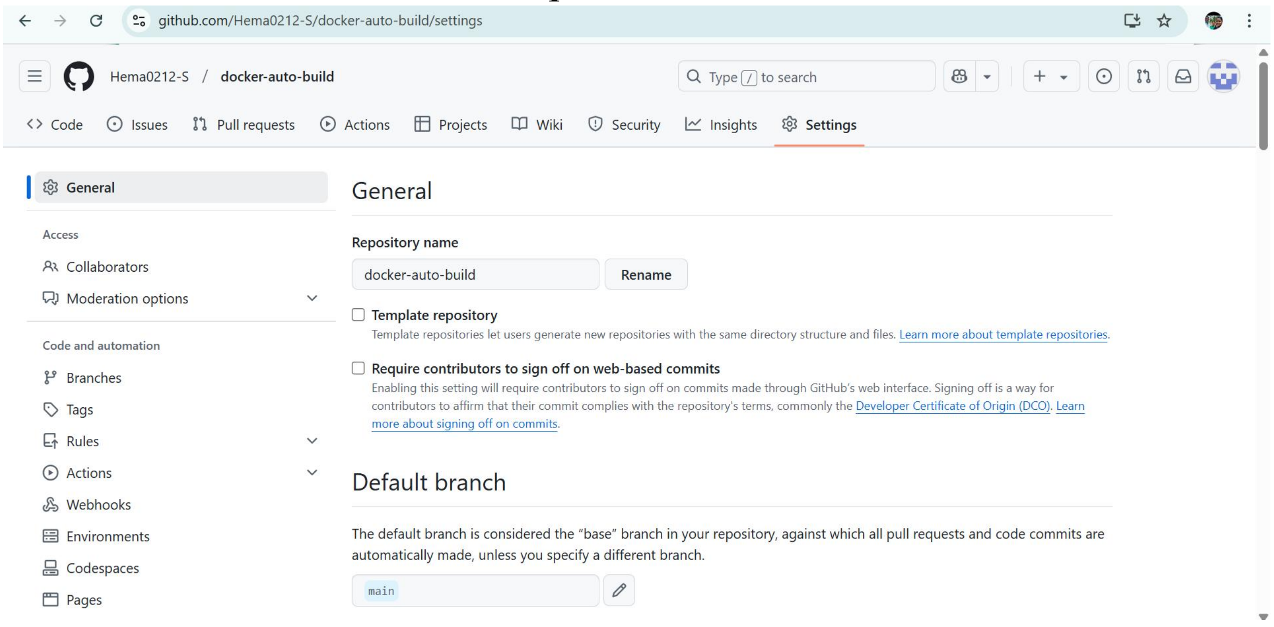
1. Go to your **GitHub repository** → **Settings** → **Secrets and variables** → **Actions**.

2. Click **New Repository Secret** and add:

 - **Name:** DOCKER_HUB_USERNAME
 - **Value:** Your Docker Hub username

3. Click **New Repository Secret** again and add:

 - **Name:** DOCKER_HUB_PASSWORD
 - **Value:** Your Docker Hub password


github.com/Hema0212-S/docker-auto-build/settings

Hema0212-S / docker-auto-build

General

Repository name

docker-auto-build

Rename

Template repository

Template repositories let users generate new repositories with the same directory structure and files. Learn more about template repositories.

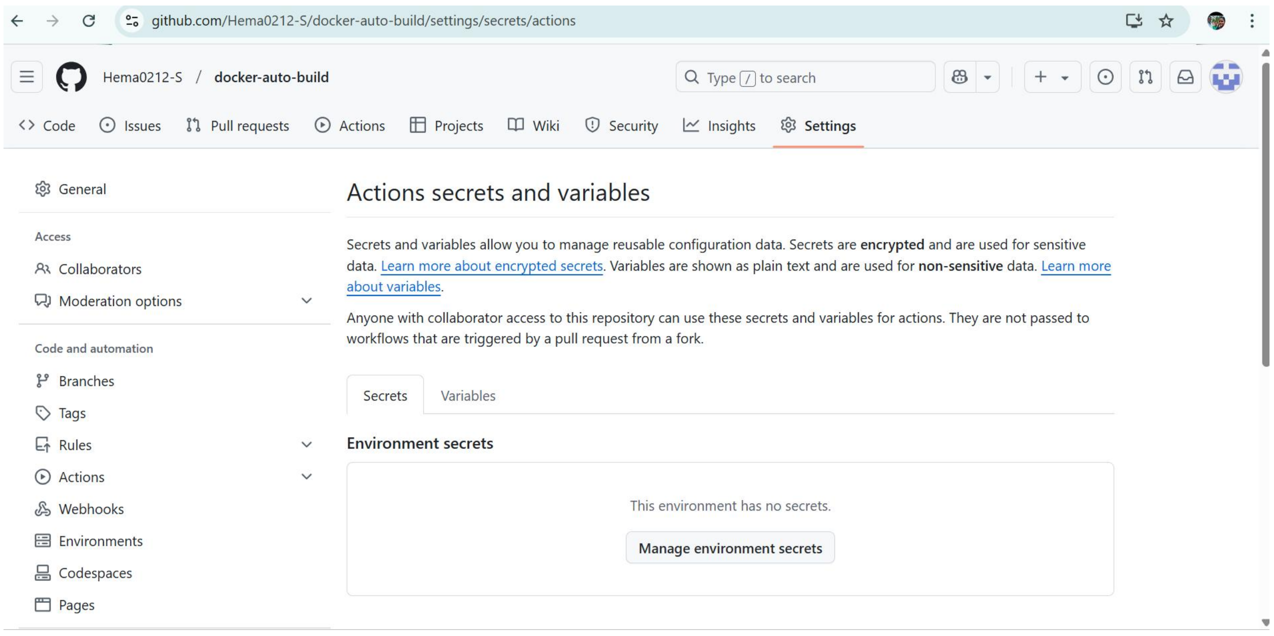
Require contributors to sign off on web-based commits

Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

main


github.com/Hema0212-S/docker-auto-build/settings/secrets/actions

Hema0212-S / docker-auto-build

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for **non-sensitive** data. Learn more about variables.

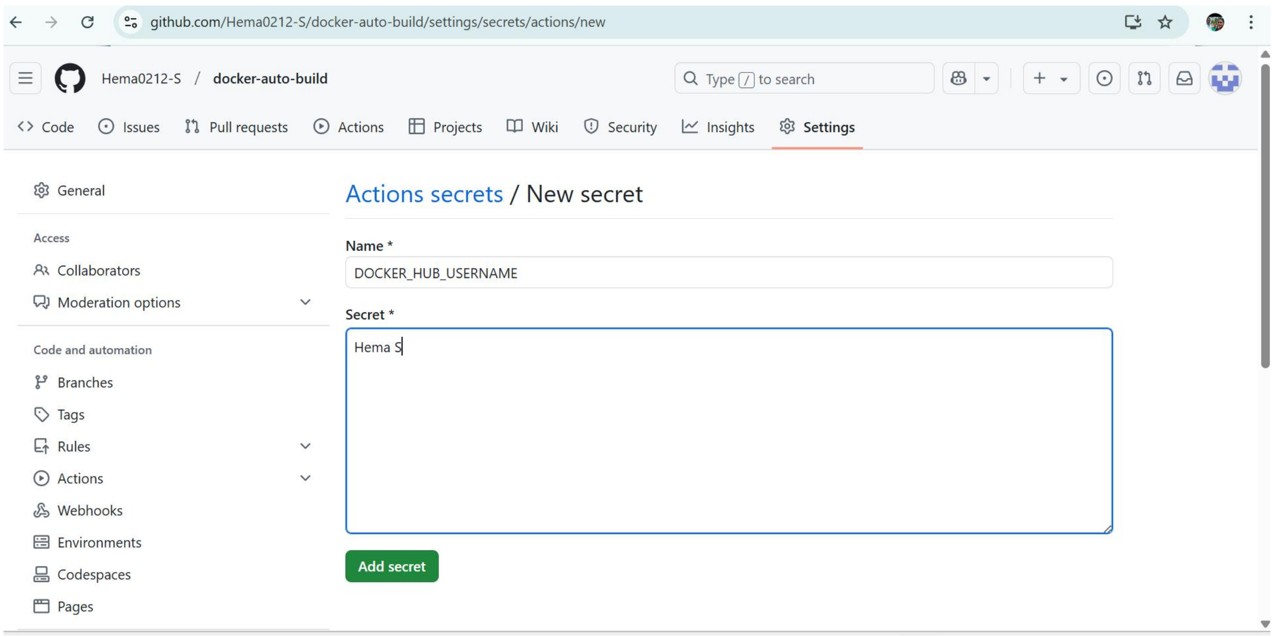
Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets | Variables

Environment secrets

This environment has no secrets.

Manage environment secrets


github.com/Hema0212-S/docker-auto-build/settings/secrets/actions/new

Hema0212-S / docker-auto-build

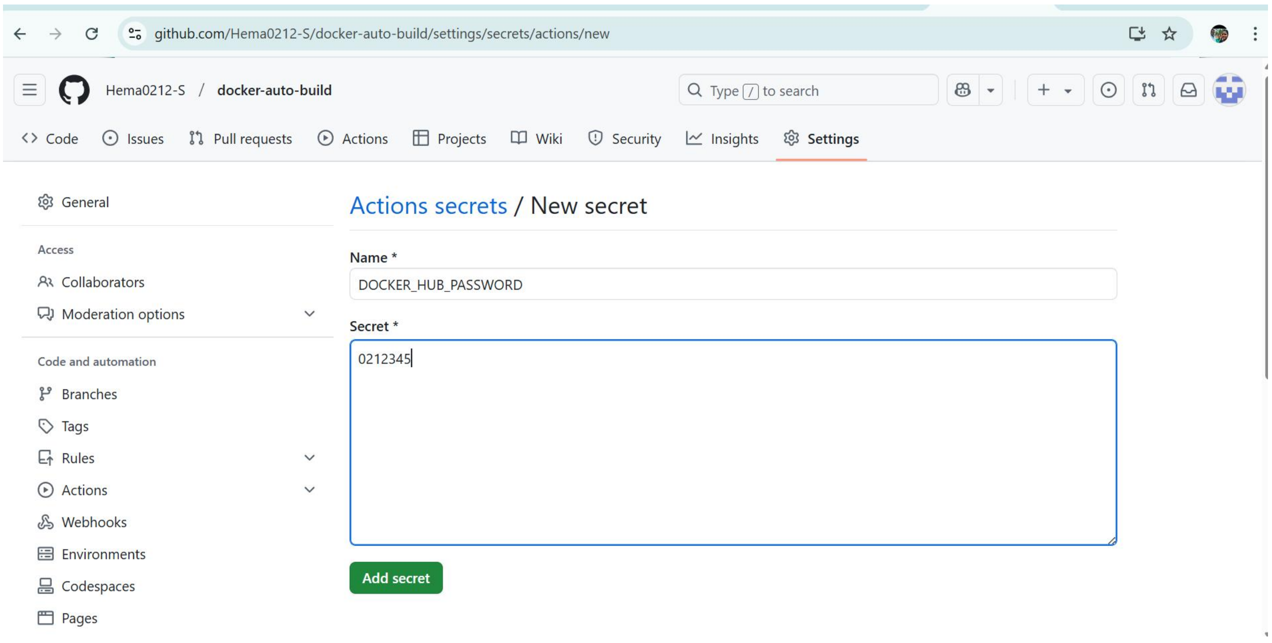
Actions secrets / New secret

Name *

DOCKER_HUB_USERNAME

Secret *

Hema S


github.com/Hema0212-S/docker-auto-build/settings/secrets/actions/new

Hema0212-S / docker-auto-build

Actions secrets / New secret

Name *

DOCKER_HUB_PASSWORD

Secret *

0212345

Step 7:

Create the GitHub Actions Directory

Run the following in **Command Prompt**:

mkdir .github\workflows

This creates a folder for GitHub Actions workflows.

```
C:\Users\DELL\docker-auto-build>mkdir .github-workflows
```

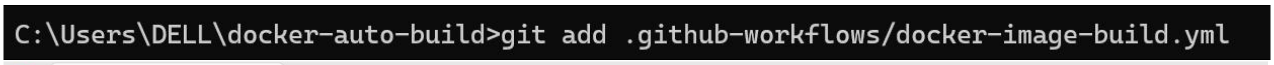
Step 8:

1. Inside .github/workflows, create a new file named **docker-image-build.yml**.

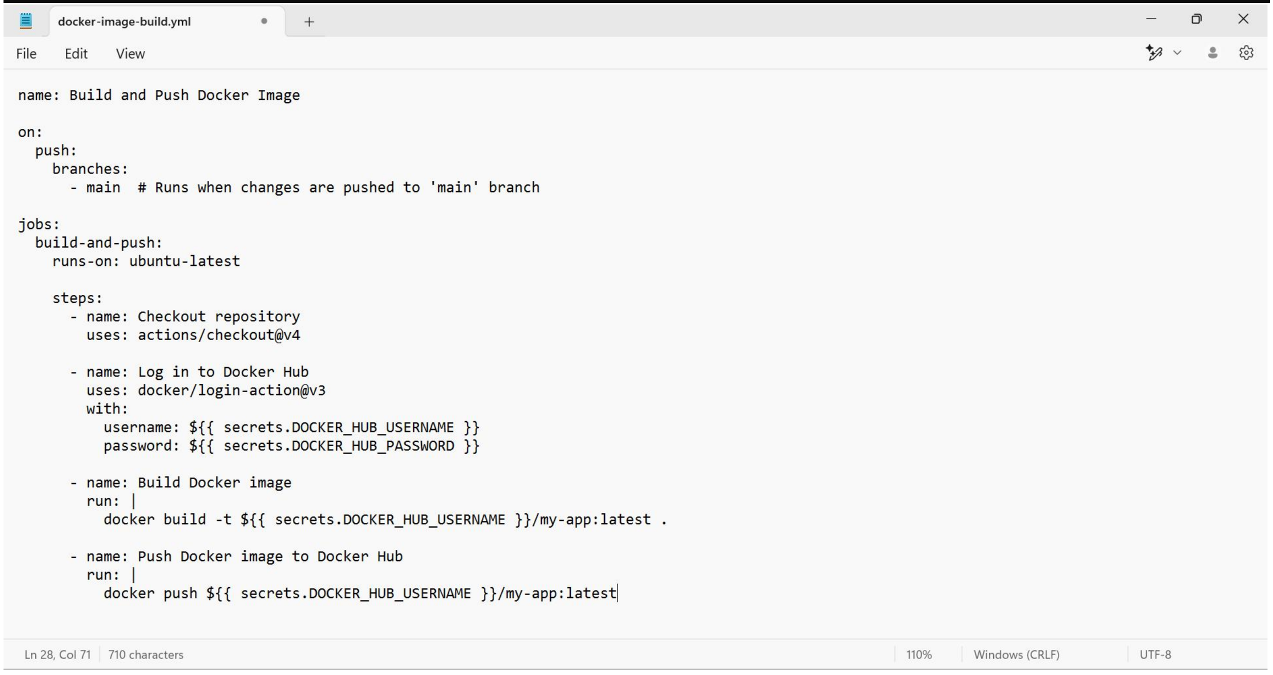
2. Open it in **Notepad** .

3. Add the following code

4. Save the file.



```
C:\Users\DELL\docker-auto-build>git add .github-workflows/docker-image-build.yml
```



```
name: Build and Push Docker Image

on:
  push:
    branches:
      - main  # Runs when changes are pushed to 'main' branch

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_PASSWORD }}

      - name: Build Docker image
        run: |
          docker build -t ${{ secrets.DOCKER_HUB_USERNAME }}/my-app:latest .

      - name: Push Docker image to Docker Hub
        run: |
          docker push ${{ secrets.DOCKER_HUB_USERNAME }}/my-app:latest
```

Step 9:

Now, we need to push our changes to GitHub.

1. Add all files to Git:

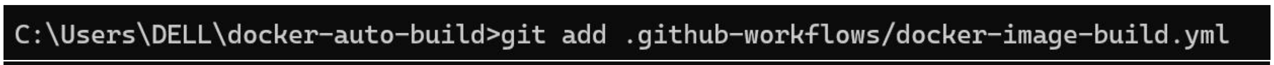
   git add .

2. Commit the changes:

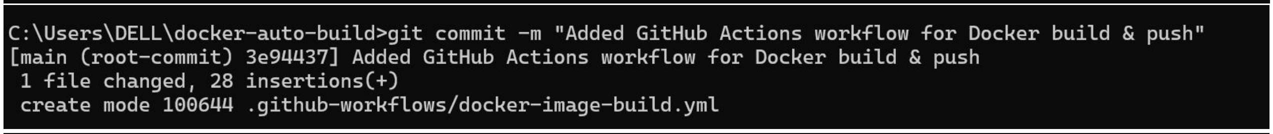
   git commit -m "Add Dockerfile and GitHub Actions workflow"

3. Push to GitHub:

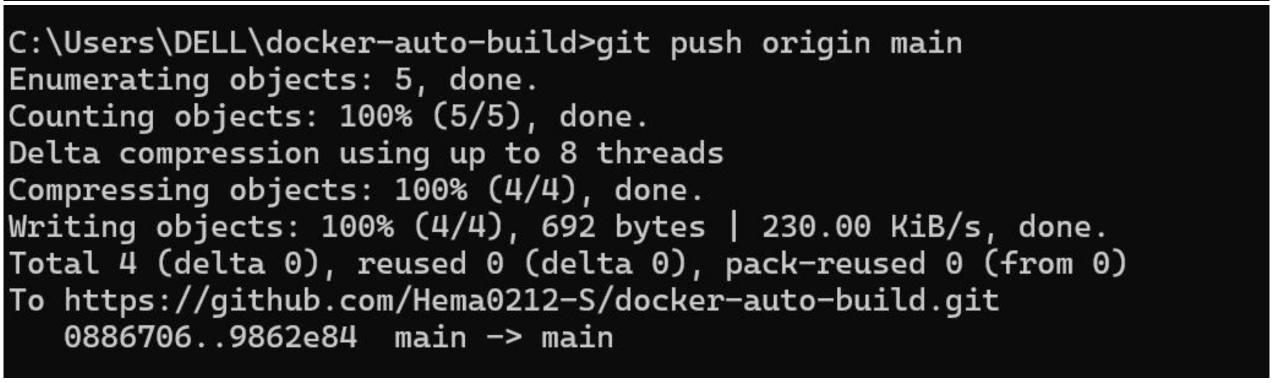
   git push origin main



```
C:\Users\DELL\docker-auto-build>git add .github-workflows/docker-image-build.yml
```



```
C:\Users\DELL\docker-auto-build>git commit -m "Added GitHub Actions workflow for Docker build & push"
[main (root-commit) 3e94437] Added GitHub Actions workflow for Docker build & push
 1 file changed, 28 insertions(+)
 create mode 100644 .github-workflows/docker-image-build.yml
```



```
C:\Users\DELL\docker-auto-build>git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 692 bytes | 230.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Hema0212-S/docker-auto-build.git
   0886706..9862e84  main -> main
```

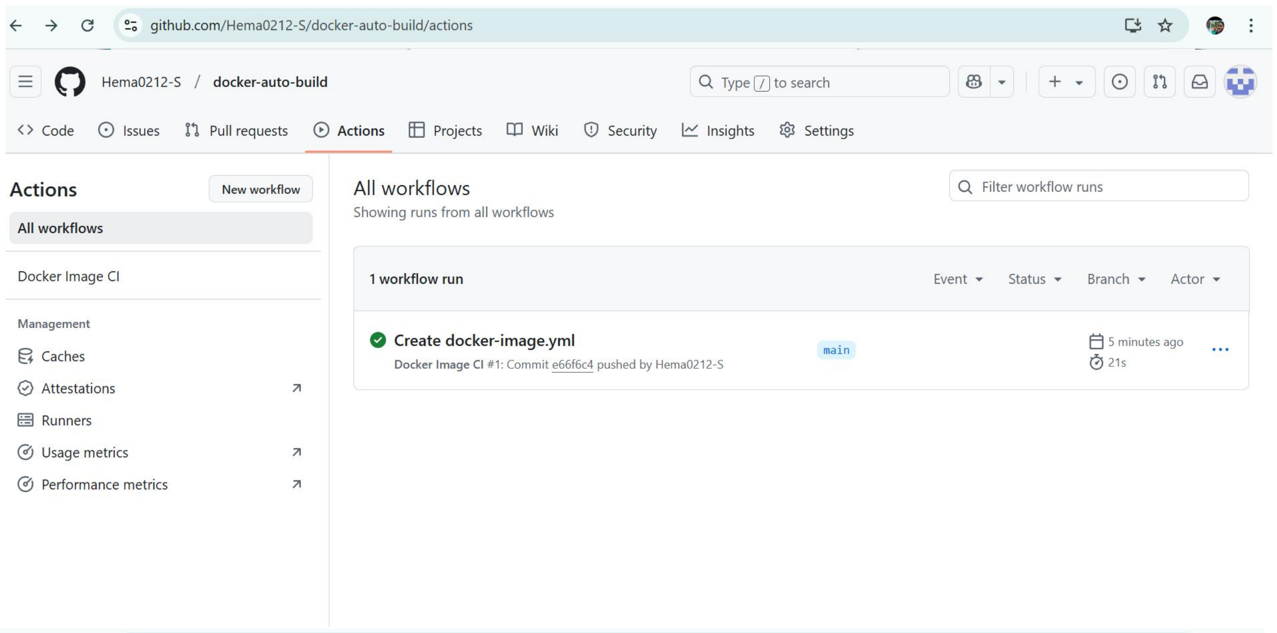
Step 10:

1. Go to your **GitHub repository → Actions** tab.

2. You should see a workflow running.

3. Wait for it to complete.

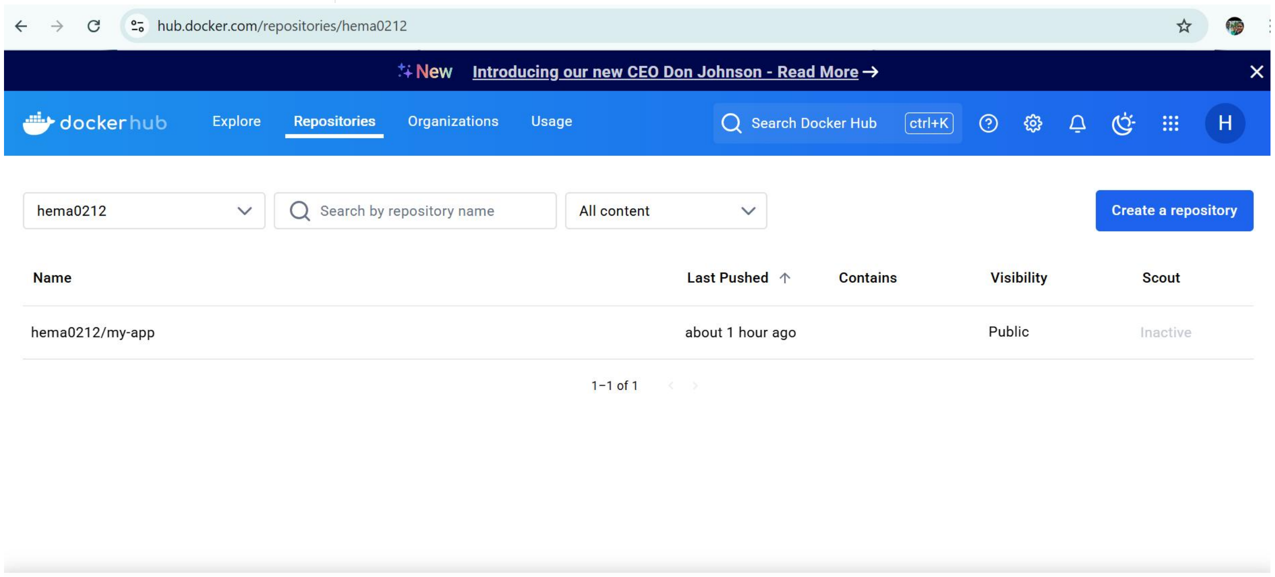
4. If successful, check **Docker Hub** to see if your image is uploaded.



All workflows
Showing runs from all workflows

1 workflow run

Create docker-image.yml
Docker Image CI #1: Commit e66f6c4 pushed by Hema0212-S — main — 5 minutes ago — 21s


hub.docker.com/repositories/hema0212
New Introducing our new CEO Don Johnson - Read More →
docker hub
Explore
Repositories
Organizations
Usage
Search Docker Hub
ctrl+K
H
hema0212
Search by repository name
All content
Create a repository

Name	Last Pushed	Contains	Visibility	Scout
hema0212/my-app	about 1 hour ago		Public	Inactive

1–1 of 1

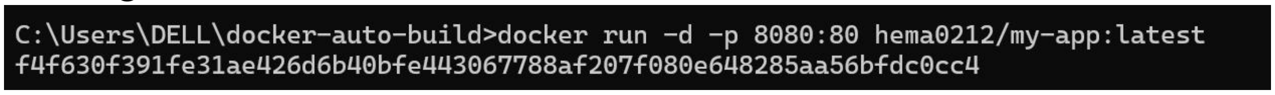
Step 11:

Test the Docker Image

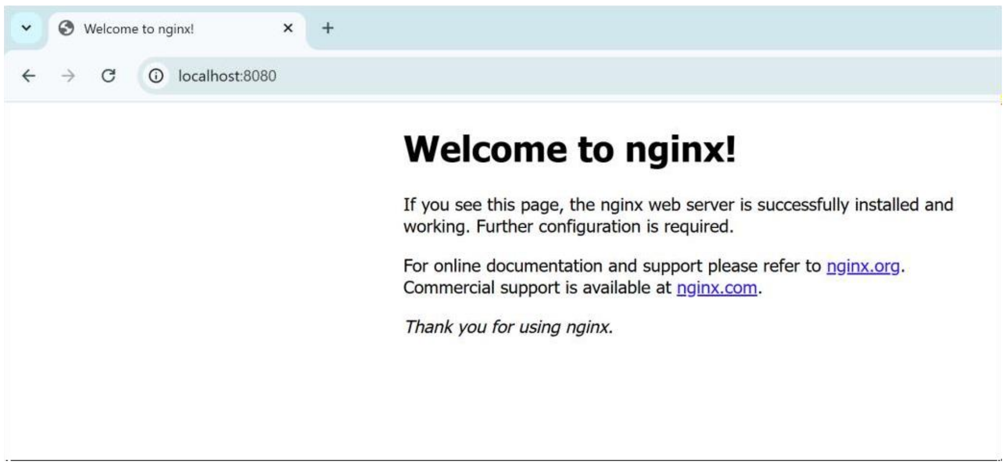
To run the image locally:

docker run -d -p 8080:80 YOUR_DOCKER_HUB_USERNAME/my-app:latest

Now, open **http://localhost:8080** in your browser to see your app running!



```
C:\Users\DELL\docker-auto-build>docker run -d -p 8080:80 hema0212/my-app:latest
f4f630f391fe31ae426d6b40bfe443067788af207f080e648285aa56bfdc0cc4
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

PoC is **successfully completed!**

Created a Dockerfile. Configured GitHub Actions to automate Docker image builds. Pushed the image to Docker Hub. Verified the image by pulling and running it locally.

Outcomes

By completing this **Automating Docker Image Builds Using GitHub Actions** PoC, you will:

1. **Understand Docker Image Automation** – Gain hands-on experience in automating Docker image builds using GitHub Actions.

2. **Implement CI/CD for Containerized Applications** – Learn how to integrate GitHub Actions with Docker Hub to streamline the build and deployment process.

3. **Configure Secure Authentication** – Use GitHub Secrets to securely authenticate with Docker Hub, ensuring secure and automated image pushes.

4. **Build and Push Docker Images Efficiently** – Automate the process of building a Docker image and pushing it to a container registry whenever there is a code change.