## **Secure CI/CD with a Self-Hosted Runner**

### **1. Goal**

The goal is to establish a secure and automated Continuous Integration/Continuous Delivery (CI/CD) pipeline using a **self-hosted GitHub Actions runner**. This pipeline will automatically build and deploy code changes when they are pushed to the main branch of the GitHub repository.

This method is inherently more secure because it **does not require storing server SSH keys in GitHub**, eliminating the risk associated with compromised credentials. The runner operates directly on the server, executing deployment commands locally.

### **2. Server Preparation**

#### **2.1. Administrator SSH Access**

First, you need a secure way to access and manage your server as an administrator. This SSH key is for **your use only** and will **not** be used by the GitHub Actions pipeline.

1. **Log in to the server** using the provided username and password.
2. **Generate an SSH key pair** on the server. This key will allow you to log in without a password.  
   Bash

Bash  
ssh-keygen -m PEM -t rsa -b 4096 -f ~/.ssh/admin\_key.pem



1. **Authorize the public key** for passwordless login.

Bash  
cat ~/.ssh/admin\_key.pem.pub >> ~/.ssh/authorized\_keys



1. **Set correct permissions** for the SSH directory and files.

  
chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized\_keys ~/.ssh/admin\_key.pem



1. **Copy the private key** to your local machine to use for logging in. From your local terminal,run:

  
scp server-username@<server-ip>:~/.ssh/admin\_key.pem ~/admin\_key.pem



1. **Log in using the key** to confirm it works.

  
ssh -i ~/admin\_key.pem server-username@<server-ip>



#### **2.2 Automated Server Setup Script**

#### This Bash script will update the server and install all necessary software: **Git, Docker,** and **Docker Compose.**

#### On your server, create and open a new script file.

#### **** sudo apt install nano -y

#### nano setup\_server.sh

#### **Copy and paste the following script** into the file.

#### **** #!/bin/bash

#### 

#### # Exit on any error

#### set -e

#### 

#### echo "[1/4] Updating and upgrading packages..."

#### sudo apt-get update -y

#### sudo apt-get upgrade -y

#### 

#### echo "[2/4] Installing dependencies and Docker GPG key..."

#### sudo apt-get install -y ca-certificates curl gnupg lsb-release

#### sudo install -m 0755 -d /etc/apt/keyrings

#### curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

#### 

#### echo "[3/4] Setting up and installing Docker Engine & Compose..."

#### echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

#### sudo apt-get update -y

#### sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin git

#### 

#### echo "[4/4] Enabling Docker service and adding user to Docker group..."

#### sudo systemctl start docker

#### sudo systemctl enable docker

#### sudo usermod -aG docker $USER

#### 

#### echo "Setup complete! Git and Docker are installed."

#### echo " IMPORTANT: Please log out and log back in for Docker group changes to take effect."

#### 

#### ****

#### **Save the file (Ctrl+O, Enter, Ctrl+X),** make it executable, and run it.

#### **** chmod +x setup\_server.sh

#### ./setup\_server.sh

#### ****

#### **Log out and log back in** to ensure you can run Docker commands without **sudo.**

### **3. Setting Up the Self-Hosted Runner**

#### Now, install the GitHub Actions runner on your server. This runner will listen for jobs from your repository and execute them locally.

#### **Navigate to your GitHub repository** settings: **Settings > Actions > Runners > New self-hosted runner.**

#### **Select Linux** as the operating system to see the specific commands for downloading and configuring the runner.

#### **Follow the commands provided by GitHub** on your server. They will look similar to this:

####  # Create a folder for the runner

#### mkdir actions-runner && cd actions-runner

#### 

#### # Download the latest runner package (check GitHub for the latest version)

#### curl -o actions-runner-linux-x64-2.317.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-linux-x64-2.317.0.tar.gz

#### 

#### # Unpack the installer

#### tar xzf ./actions-runner-linux-x64-2.317.0.tar.gz

#### 

#### # Configure the runner (use the command from your repo's settings page)

#### ./config.sh --url https://github.com/YOUR\_USERNAME/YOUR\_REPO --token YOUR\_GENERATED\_TOKEN

#### ****

#### **Install and start the runner as a background service.** This ensures it runs automatically and persists after you log out.

#### **** sudo ./svc.sh install

#### sudo ./svc.sh start

#### **** Your runner should now show as "Idle" in your GitHub repository's runner settings.

### **4. Creating the Secure Deployment Workflow**

#### This workflow file defines the CI/CD process. It uses the self-hosted runner to check out the code and run Docker commands directly.

#### In your repository, create the workflow file at .github/workflows/deploy.yml.

#### **Add the following content**. Notice it has no SSH actions or secrets. YAML

####  name: Build and Deploy to Server

#### 

#### on:

#### push:

#### branches:

#### - main

#### 

#### jobs:

#### deploy:

#### name: Deploy Application

#### # This is the crucial line that targets your server

#### runs-on: self-hosted

#### 

#### steps:

#### - name: Checkout Repository Code

#### uses: actions/checkout@v4

#### 

#### - name: Build and Deploy Docker Container

#### run: |

#### # These commands run directly on your server's shell

#### echo "Starting deployment..."

#### docker compose down || true

#### docker compose up --build -d

#### echo "Deployment complete!"

#### 

#### 

#### **Commit and push this file**. The action will trigger and be executed by your newly configured self-hosted runner.

### **5. Expose Application via Nginx**

#### To make your application accessible to users, you can use Nginx as a reverse proxy.

#### **Install Nginx**.

####  sudo apt install nginx -y

#### 

#### **Create an Nginx configuration file** for your app.

####  sudo nano /etc/nginx/sites-available/fastapi

#### 

#### **Add the following server block**, replacing YOUR\_SERVER\_IP\_OR\_DOMAIN with your server's IP address or domain name.

####  server {

#### listen 80;

#### server\_name YOUR\_SERVER\_IP\_OR\_DOMAIN;

#### 

#### location / {

#### proxy\_pass http://localhost:8000; # Assumes your app runs on port 8000

#### proxy\_set\_header Host $host;

#### proxy\_set\_header X-Real-IP $remote\_addr;

#### }

#### }

#### 

#### **Enable the configuration** and reload Nginx

####  sudo ln -s /etc/nginx/sites-available/fastapi /etc/nginx/sites-enabled/

#### sudo nginx -t # Test configuration

#### sudo systemctl reload nginx

#### 

#### 