**Goal**:

The goal is to establish an automated **Continuous Integration/Continuous Delivery (CI/CD)** pipeline using **GitHub Actions**. This pipeline will automatically build, test, and deploy code changes whenever they are pushed to the GitHub repository, ensuring faster, more reliable, and consistent software delivery.

**Disclaimer**: this will act as a generic documentation for the simplest and most basic application.

**Steps to Create and Use SSH Keys After Server Creation**

Why are we doing this:

We create and authorize **SSH keys** on the server to boost security and enable automation. The **private key** is essential for services like **GitHub Actions runners**, allowing them to securely connect to your server for tasks like deployments without needing to store sensitive credentials directly.

#### **1. Log in to the server**

Use the username and password provided:

ssh server-username@<server-ip>

#### **2. Generate SSH key pair on the server**

****ssh-keygen -m PEM -t rsa -b 4096 -f ~/.ssh/key.pem

*   
  -m PEM: Ensures the private key is in PEM format.
* -t rsa -b 4096: Creates a 4096-bit RSA(Rivest–Shamir–Adleman) key. A widely used and well-established public-key cryptosystem
* -f ~/.ssh/key.pem: Saves the key to the specified path.

#### **3. Authorize the generated public key**

****cat ~/.ssh/key.pem.pub >> ~/.ssh/authorized\_keys

chmod 600 ~/.ssh/authorized\_keys

chmod 700 ~/.ssh

chmod 600 ~/.ssh/key.pem

#### **4. Exit the server and cat your private key**

cat ~/key.pem

Copy the contents of the key and store them safe from

-----BEGIN RSA PRIVATE KEY-----

MIIJKQIBAAK…

PR0YeIVMTr

-----END RSA PRIVATE KEY-----

#### 

#### **5. Copy the private key to your local machine**

From **your local terminal**, run:

scp server-username@<server-ip>:~/.ssh/key.pem ~/key.pem

Replace <server-ip> with your actual server IP.

#### **6. SSH into the server using the PEM key**

****ssh -i ~/key.pem server-username@<server-ip>



### **Step-by-Step: Install Git on Ubuntu 24.04**

1. **Update your package index** Always good to update before installing:

sudo apt update

1. **Install Git** Use the following command:

sudo apt install git -y

1.   
   **Verify installation** After installation, check the Git version:

git --version

1.   
   You should see something like:

git version 2.43.0



### **(Optional) Set Your Git Identity**

If you're using Git for version control (e.g., GitHub), configure your name and email:

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

To verify:

git config --global --list



Git Clone Repo:

git clone https://github.com/nando-bingani/dev-ops.git



Install **nano**, run:

sudo apt install nano



Certainly! Below is a **Bash script** you can run on a **new Ubuntu instance** to:

1. Install Docker & Docker Compose
2. Clone a Git repository (you can set your own repo URL)
3. Run docker-compose up --build

### **Script: setup\_docker\_app.sh**

### **Instructions:**

1. Save the script to a file:

nano setup\_docker\_app.sh

1. Paste the script and save (Ctrl+O, Enter, then Ctrl+X)

#!/bin/bash

# Exit on any error

set -e

# Variables (replace with your values)

REPO\_URL="https://github.com/your-username/your-repo.git"

APP\_DIR="your-repo" # This should match the folder name created after git clone

echo "[1/6] Updating packages..."

sudo apt-get update -y

sudo apt-get upgrade -y

echo "[2/6] Installing Docker..."

sudo apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# Add Docker GPG key

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

# Setup Docker repository

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

# Install Docker Engine

sudo apt-get update -y

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

echo "[3/6] Enabling Docker..."

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker $USER

echo "[4/6] Installing Git..."

sudo apt-get install -y git

echo "[5/6] Cloning repository..."

git clone $REPO\_URL

cd $APP\_DIR

echo "[6/6] Starting application using docker-compose..."

docker compose up --build -d

echo " Setup complete! Your app should now be running."

echo " Please logout and log back in to use Docker without 'sudo'."



1. Make it executable:

chmod +x setup\_docker\_app.sh

1.   
   Run it:

./setup\_docker\_app.sh



### **Notes:**

* Replace REPO\_URL with your actual Git repo URL.
* Ensure your docker-compose.yml is present in the repo.
* After the script completes, logout and login again (or run newgrp docker) to avoid using sudo with Docker.

### **Goal:**

Want a **GitHub Action** to:

* Connect to a remote **server over VPN**
* Run git commit and git push from the server **to GitHub**.

### **1. Ensure Server Has GitHub Access**

* Access GitHub through SSH (use curl or ping to test),
* Has proper credentials to push to GitHub (use one of these options):  
  + SSH key added to GitHub

### **2. Set Up GitHub Action to SSH into the Server**

Navigate to actions, new workflow -> setup workflow yourself.

Paste the below work into your yml file.

In .github/workflows/deploy.yml:

name: Deploy and Commit on Remote Server

on:

push:

branches:

- main

jobs:

remote-commit:

runs-on: self-hosted #ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v4

- name: SSH and Run Git Commands

uses: appleboy/ssh-action@v1.0.3

with:

host: ${{ secrets.SERVER\_HOST }}

username: ${{ secrets.SERVER\_USER }}

key: ${{ secrets.SERVER\_SSH\_KEY }}

script: |

cd /path/to/repo

git config --global user.name "github-actions[bot]"

git config --global user.email "41898282+github-actions[bot]@users.noreply.github.com"

# Run your update or deployment scripts

echo "Deployment finished on $(date)" >> deploy.log

git add deploy.log

git commit -m "Deployment log update"

git push origin main

### Then commit.

### **Go to Secrets Settings:**

1. Go to your **GitHub repository**.
2. Click on **Settings**.
3. In the sidebar, go to **Secrets and variables > Actions**.
4. Click **"New repository secret"**.

### **Add These Secrets: – Needs to be replaced \*\*\*\*\*\*\*\*\*\*\*\*\* not use ssh**

| **Secret Name** | **Description** |
| --- | --- |
| SERVER\_HOST | The IP or hostname of your server (VPN IP) |
| SERVER\_USER | Username to SSH into the server |
| SERVER\_SSH\_KEY | The **private SSH key** (not public!) |

### **Expose via Nginx**

Nginx acts as a **reverse proxy** to forward external traffic (port 80 or 443) to your FastAPI app inside Docker on port 8000.

#### **Step 1: Install Nginx on the server**

****sudo apt update && sudo apt install nginx -y

#### **Step 2: Create Nginx config**

Create file /etc/nginx/sites-available/fastapi with:

server {

listen 80;

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

location / {

proxy\_pass http://localhost:8000;

proxy\_set\_header Host $host;

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

}

}

Replace YOUR\_SERVER\_IP\_OR\_DOMAIN with the VPN-accessible IP address or hostname.

Enable it:

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

sudo nginx -t

sudo systemctl reload nginx

#### **Step 3: Make sure Docker still runs**

****docker-compose up -d

Now you can hit:

http://YOUR\_VPN\_IP/

If Nginx is correctly routing, you’ll see {"message": "Hello World"}

## **Finally: Deploy Code via GitHub Actions Using a Self-Hosted Runner Behind VPN**

### **On Your Server (VPN-Connected)**

#### **1. Create actions-runner directory and install the runner**

****mkdir actions-runner && cd actions-runner

# Download GitHub Actions runner (latest version as of now)

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

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



#### **2. Register the self-hosted runner on your GitHub repository**

1. Go to your **GitHub repository** → **Settings** → **Actions** → **Runners** → **New self-hosted runner**
2. Select **Linux** and copy the generated command. It looks like this:

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

Run this in the actions-runner folder on your server.

#### **3. Start the self-hosted runner as a service**

****./svc.sh install

./svc.sh start

**When to run what run.sh vs** [**svc.sh**](http://svc.sh)

You run run.sh to test your runner interactively and svc.sh to install it as a background service for long-term use.

### **run.sh (Interactive Runner)**

Use ./run.sh to start the runner in the **foreground** of your current terminal session.

* **When to use it**:
  + For **initial setup** and testing to make sure the runner connects and works correctly.
  + When you need to actively **watch the logs** for a specific workflow run.
* **Key characteristic**: The runner process stops as soon as you close the terminal or press Ctrl+C.

**svc.sh (Service Runner)**

Use sudo ./svc.sh to manage the runner as a **background service**. This is the recommended method for production.

* **When to use it**:
  + For any **production** or long-term setup.
  + When you need the runner to **start automatically** when the server reboots.
* **Key commands**:
  + sudo ./svc.sh install: Installs the runner as a systemd service.
  + sudo ./svc.sh start: Starts the service.
  + sudo ./svc.sh stop: Stops the service.
  + sudo ./svc.sh uninstall: Removes the service.

### **Summary**

| Feature | ./run.sh | sudo ./svc.sh |
| --- | --- | --- |
| **Mode** | Interactive (Foreground) | Service (Background) |
| **Best For** | Testing & Debugging | Production & Long-term use |
| **Persistence** | Stops when you close the terminal | Survives terminal closure & system reboot |
| **Typical First Step** | Run once to confirm configuration works | Run install and start for final setup |