Deployment steps :

Step 1: Set Up Your GitHub Repository

1. Create a new repository on GitHub named "deploy_guide"
2. Initialize your local project and push it to GitHub:

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/yourusername/deploy_guide.git
git push -u origin main
```

Step 2: Create Required Files for Docker and Deployment

File Structure

streamlit-app/
├──
app.py # Your Streamlit application code
├── Dockerfile # Docker container configuration
├── requirements.txt # Python dependencies
├── .github/
│ └── workflows/
│ └── deploy.yml # GitHub Actions workflow
├── scripts/

```
|   └──
deploy_container.sh          # EC2 deployment script
├── .dockerignore          # Files to exclude from Docker image
└── nginx/
└── app.conf          # Nginx configuration for the app
```

Step 3: Create Docker and Deployment Files

requirements.txt

```
Copy
streamlit==1.24.0
pandas==2.0.3
numpy==1.24.3
matplotlib==3.7.2
```

Dockerfile :

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8501

HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health || exit 1
```

```
ENTRYPOINT ["streamlit", "run", "app.py", "--server.port=8501", "--server.addres
```

.dockerignore

```
Copy
.git
.github
.gitignore
README.md
*.pyc
__pycache__/
.DS_Store
```

Nginx Configuration (nginx/app.conf)

```
server {
    listen 80;
    server_name _;  # This will match any hostname

    location / {
        proxy_pass http://localhost:8501;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_read_timeout 86400;
    }
}
```

.github/workflows/deploy.yml

```
name: Build and Deploy with Self-hosted Runner

on:
  push:
    branches: [ main ]

jobs:
  build-and-deploy:
    runs-on: self-hosted

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ap-south-1

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, tag, and push image to Amazon ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
        IMAGE_TAG: ${{ github.sha }}
```

```
      run: |
        # Build the docker image
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker tag $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG $ECR_REGIS

        # Push the docker images
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:latest

    - name: Install Nginx
      run: |
        # Install Nginx if not installed
        if ! command -v nginx &> /dev/null; then
            echo "Installing Nginx..."
            sudo apt-get update
            sudo apt-get install -y nginx
        fi

        # Create directories if they don't exist
        sudo mkdir -p /etc/nginx/sites-available
        sudo mkdir -p /etc/nginx/sites-enabled

    - name: Deploy Container
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
        IMAGE_TAG: ${{ github.sha }}
      run: |
        # Stop any existing container
        docker stop streamlit-container 2>/dev/null || true
        docker rm streamlit-container 2>/dev/null || true

        # Run the container
        docker run -d --name streamlit-container -p 8501:8501 --restart always $EC

        # Configure Nginx
```

```
        echo "Creating Nginx configuration..."
        cat > /tmp/streamlit_nginx << 'EOL'
        server {
            listen 80;
            server_name _;

            location / {
                proxy_pass http://localhost:8501;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
                proxy_read_timeout 86400;
            }
        }
        EOL
        sudo cp /tmp/streamlit_nginx /etc/nginx/sites-available/streamlit
        sudo ln -sf /etc/nginx/sites-available/streamlit /etc/nginx/sites-enabled/
        sudo rm -f /etc/nginx/sites-enabled/default 2>/dev/null || true
        sudo nginx -t
        sudo systemctl restart nginx
```

deploy_container.sh

```
name: Build and Deploy with Self-hosted Runner

on:
  push:
    branches: [ main ]

jobs:
  build-and-deploy:
```

```
runs-on: self-hosted

steps:
- name: Checkout code
  uses: actions/checkout@v2

- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ap-south-1

- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1

- name: Build, tag, and push image to Amazon ECR
  id: build-image
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
    IMAGE_TAG: ${{ github.sha }}
  run: |
    # Build the docker image
    docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
    docker tag $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG $ECR_REGIS

    # Push the docker images
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:latest

- name: Install Nginx
  run: |
    # Install Nginx if not installed
    if ! command -v nginx &> /dev/null; then
```

```
        echo "Installing Nginx..."
        sudo apt-get update
        sudo apt-get install -y nginx
      fi

      # Create directories if they don't exist
      sudo mkdir -p /etc/nginx/sites-available
      sudo mkdir -p /etc/nginx/sites-enabled

  - name: Deploy Container
    env:
      ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
      IMAGE_TAG: ${{ github.sha }}
    run: |
      # Stop any existing container
      docker stop streamlit-container 2>/dev/null || true
      docker rm streamlit-container 2>/dev/null || true

      # Run the container
      docker run -d --name streamlit-container -p 8501:8501 --restart always $EC

      # Configure Nginx
      echo "Creating Nginx configuration..."
      cat > /tmp/streamlit_nginx << 'EOL'
      server {
          listen 80;
          server_name _;

          location / {
              proxy_pass http://localhost:8501;
              proxy_http_version 1.1;
              proxy_set_header Upgrade $http_upgrade;
              proxy_set_header Connection "upgrade";
              proxy_set_header Host $host;
              proxy_cache_bypass $http_upgrade;
```

```
        proxy_read_timeout 86400;
    }
}
EOL
sudo cp /tmp/streamlit_nginx /etc/nginx/sites-available/streamlit
sudo ln -sf /etc/nginx/sites-available/streamlit /etc/nginx/sites-enabled/
sudo rm -f /etc/nginx/sites-enabled/default 2>/dev/null || true
sudo nginx -t
sudo systemctl restart nginx
```

Step 4: Set Up AWS Infrastructure in Mumbai Region (ap-south-1)

1. Create an IAM User for GitHub Actions

1. Go to AWS IAM console
2. Create a new IAM user with programmatic access
3. Attach policies:
    - AmazonECR-FullAccess
    - AmazonEC2ContainerRegistryFullAccess
4. Save the Access Key ID and Secret Access Key

2. Create an ECR Repository in Mumbai Region

1. Go to AWS ECR console (make sure you're in the Mumbai region)
2. Click "Create repository"
3. Enter a name for your repository (e.g., "streamlit-app")
4. Keep default settings and click "Create repository"
5. Note the repository URI (it should include "ap-south-1" in the path)

3. Launch an EC2 Instance in Mumbai Region

1. Go to EC2 console (make sure you're in the Mumbai region)
2. Launch a new EC2 instance:
    - Choose Ubuntu Server 20.04 LTS
    - Select t2.micro (free tier) or appropriate size
    - Configure security groups:
        - Allow SSH (port 22) from your IP
        - Allow HTTP (port 80) from anywhere
        - Allow HTTPS (port 443) from anywhere
        - Allow Custom TCP (port 8501) from anywhere
    - Launch with a new key pair (save the .pem file)

1. Attach this role to your EC2 instance:
    - Go to EC2 console
    - Select your instance
    - Actions → Security → Modify IAM role
    - Select the role you created and click "Save"

Step 5: Set Up GitHub Secrets

In your GitHub repository:

1. Go to Settings → Secrets → New repository secret
2. Add these secrets:

- `AWS_ACCESS_KEY_ID`: Your IAM user access key
- `AWS_SECRET_ACCESS_KEY`: Your IAM user secret key
- `ECR_REPOSITORY`: Your ECR repository name (just the name, not the full URI)
- `EC2_HOST`: Your EC2 instance public IP or DNS
- `EC2_SSH_KEY`: The private SSH key content (.pem file)

Step 6: Deploy Your Application

1. Push your code to GitHub:

```
bash
Copy
git add .
git commit -m "Add ECR deployment configuration"
git push
```

Step 7: Prepare Your EC2 Instance

First, make sure your EC2 instance has the necessary dependencies:

```
bash
Copy
# Update packages
sudo apt-get update

# Install Docker
sudo apt-get install -y apt-transport-https ca-certificates curl software-prope
rties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
-
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/lin
ux/ubuntu $(lsb_release -cs) stable"
```

```
sudo apt-get update
sudo apt-get install -y docker-ce
sudo usermod -aG docker ubuntu

# Install other dependencies
sudo apt-get install -y curl jq git
```

Step 8: Add the GitHub Runner

1. In your GitHub repository, go to **Settings** → **Actions** → **Runners**
2. Click on **New self-hosted runner**
3. Select **Linux** as the operating system and **x64** as the architecture
4. You'll see commands to download and configure the runner. Run these commands on your EC2 instance:

```
bash
Copy
# Create a folder for the runner
mkdir actions-runner && cd actions-runner

# Download the runner package
curl -o actions-runner-linux-x64-2.303.0.tar.gz -L https://github.com/actions/
runner/releases/download/v2.303.0/actions-runner-linux-x64-2.303.0.tar.gz

# Verify the hash
echo "e4a9fb7269c1a156eb5d5369232d0cd62e06bec2fd2b321600e85ac914
a9cc73  actions-runner-linux-x64-2.303.0.tar.gz" | shasum -a 256 -c

# Extract the installer
tar xzf ./actions-runner-linux-x64-2.303.0.tar.gz

# Configure the runner
```

```
./config.sh --url https://github.com/YOUR_USERNAME/deploy_guide --token Y
OUR_TOKEN
```

Step 3: Install the Runner as a Service

To run the GitHub runner as a service so it starts automatically:

```
bash
Copy
sudo ./svc.sh install
sudo ./svc.sh start
```

- Install the runner as a service:

  ```
  sudo ./svc.sh install
  ```

- Start the runner service:

  ```
  sudo ./svc.sh start
  ```

- Verify the service is running:

  ```
  sudo ./svc.sh status
  ```

You can also check the Docker container status on your EC2 instance:

```
bash
Copy
docker ps
```

If you need to check the logs of your container:

```
bash
Copy
docker logs streamlit-container
```

check your app Here :

Try these URLs instead:

1. For direct access to Streamlit:

   ```
   Copy
   http://3.110.32.101:8501
   ```

2. For access through Nginx (on the standard HTTP port):

   ```
   Copy
   http://3.110.32.101
   ```

Make sure to use `http://` not `https://` unless you've specifically set up SSL.

Deployment Flow :

1. Components Involved:

- **GitHub Actions**: Automates CI/CD workflows.
- **Self-Hosted Runner (EC2 Instance)**: Executes the GitHub Actions workflow.
- **Amazon ECR**: Stores Docker images.
- **EC2 Instance**: Runs the containerized application.
- **Nginx**: Acts as a reverse proxy for handling HTTP requests.

---

2. Deployment Flow

Step 1: Developer Pushes Code to GitHub

- A developer pushes changes to the `main` branch of the GitHub repository.
- This triggers the GitHub Actions workflow.

Step 2: GitHub Actions Workflow Executes on Self-Hosted EC2 Runner

- Since `runs-on: self-hosted` is used, the workflow executes on an EC2 instance where the GitHub self-hosted runner is installed.

Step 3: Checkout Code

- GitHub Actions pulls the latest code from the repository.

Step 4: Configure AWS Credentials

- The workflow uses stored AWS secrets to authenticate and interact with AWS services.

Step 5: Login to Amazon ECR

- The workflow logs in to Amazon ECR to push and pull Docker images.

Step 6: Build and Push Docker Image to Amazon ECR

- The application is built into a Docker image.
- The image is tagged using the commit SHA (`github.sha`).
- The image is pushed to **ECR**, ensuring it is available for deployment.

Step 7: Install & Configure Nginx

- If Nginx is not installed, it is installed on the EC2 instance.
- Configuration files are set up for reverse proxy.

Step 8: Stop and Remove Any Existing Container

- If a container is already running, it is stopped and removed to ensure a fresh deployment.

Step 9: Run the New Container on EC2

- The latest image is pulled from ECR.
- A new container is started using the latest image, running on **port 8501**.

Step 10: Configure Nginx as a Reverse Proxy

- Nginx is configured to forward incoming traffic on **port 80** to **port 8501** (where Streamlit runs).
- This ensures the application is accessible via the public IP or domain name.

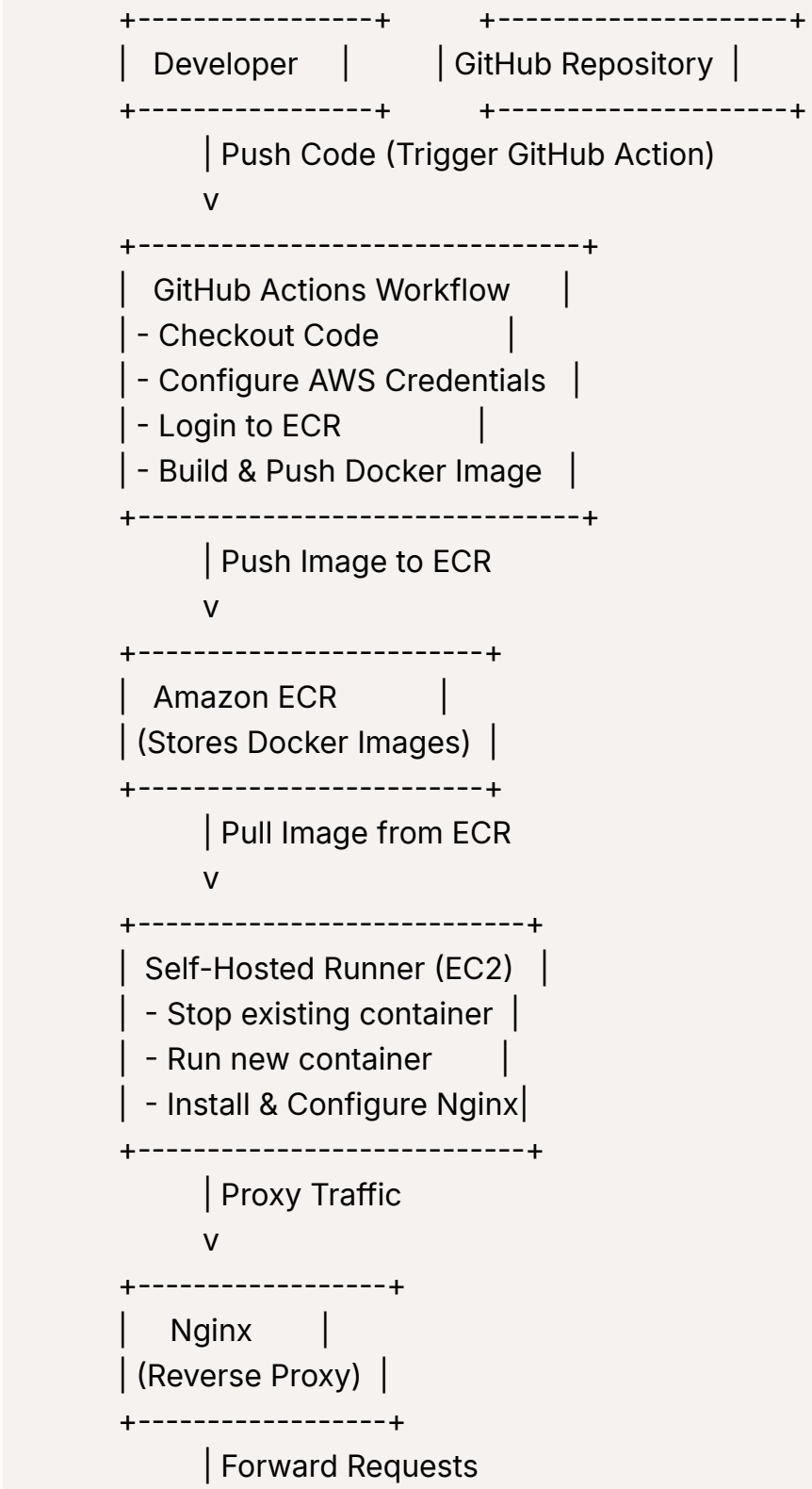
Step 11: Restart Nginx

- After updating the Nginx configuration, the service is restarted to apply the changes.

---

3. Architecture Diagram

pgsql
CopyEdit



```
          v
+---------------------------+
| Docker Container (Streamlit) |
| Running on EC2 (port 8501)  |
+---------------------------+
```

4. Explanation of Key Workflow Steps

1️⃣ Checkout Code

```
yaml
CopyEdit
- name: Checkout code
  uses: actions/checkout@v2
```

- This pulls the latest code from the GitHub repository to the self-hosted runner (EC2).

2️⃣ Configure AWS Credentials

```
yaml
CopyEdit
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ap-south-1
```

- Provides access to AWS services (ECR, EC2, etc.) using **secrets** stored in GitHub.

3️⃣ Login to Amazon ECR

```
yaml
CopyEdit
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1
```

- Authenticates with Amazon ECR so Docker images can be pushed/pulled.

4️⃣ Build and Push Docker Image to ECR

```
yaml
CopyEdit
- name: Build, tag, and push image to Amazon ECR
  id: build-image
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
    IMAGE_TAG: ${{ github.sha }}
  run: |
    docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
    docker tag $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG $ECR_REGIS
TRY/$ECR_REPOSITORY:latest
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:latest
```

- **Builds** the Docker image.
- **Tags** the image using the GitHub commit SHA.
- **Pushes** the image to Amazon ECR.

5️⃣ Install & Configure Nginx

```
yaml
CopyEdit
- name: Install Nginx
  run: |
    if ! command -v nginx &> /dev/null; then
        sudo apt-get update
        sudo apt-get install -y nginx
    fi
```

- Installs Nginx if it is not already installed.
- Nginx acts as a **reverse proxy**.

6️⃣ Stop and Remove Existing Container

```
yaml
CopyEdit
- name: Deploy Container
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY }}
    IMAGE_TAG: ${{ github.sha }}
  run: |
    docker stop streamlit-container 2>/dev/null || true
    docker rm streamlit-container 2>/dev/null || true
```

- Stops and removes any previously running container.

7️⃣ Run New Docker Container

```
yaml
CopyEdit
```

```
docker run -d --name streamlit-container -p 8501:8501 --restart always $ECR
_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
```

- Pulls the latest image from ECR.
- Runs the container with auto-restart enabled.

8️⃣ Configure Nginx Reverse Proxy

```
yaml
CopyEdit
server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://localhost:8501;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_read_timeout 86400;
    }
}
```

- Directs **incoming traffic (port 80)** to **Streamlit (port 8501)**.

9️⃣ Restart Nginx

```
yaml
CopyEdit
sudo systemctl restart nginx
```

- Applies new Nginx settings.