📖 Step-by-Step CI/CD Deployment Guide for AWS EC2 (First Deployment)

📋 Required Items Checklist:

1. **AWS EC2 Instance**: Ubuntu Server (t2.micro)
2. **GitHub Repositories** (Frontend: React+Vite, Backend: FastAPI)
3. **GitHub Secrets** (`EC2_HOST`, `SSH_PRIVATE_KEY`)
4. **Local Machine Setup** (SSH key for EC2 access)

🛠️ Step-by-Step Implementation:

☑️ Step 1: Setting up EC2 Server

1. **Log in** to AWS EC2 Console.
2. **Launch a new EC2 instance**:
    - Select Ubuntu Server (22.04 LTS).
    - Instance type: `t2.micro`.
    - Allow Security Group ports:
        - SSH (`22`)
        - HTTP (`80`)
        - HTTPS (`443`)
        - TCP (8000)
    - Download and save the SSH key (`.pem` file).

☑️ Step 2: Connect to EC2 from Windows

1. **Download and Install Git Bash** from: Git for Windows
2. **Move your `` file** to a known location (e.g., `Documents` folder).
3. **Open Git Bash**, navigate to `.pem` file:

```
cd ~/Documents
chmod 400 your-key.pem
ssh -i your-key.pem ubuntu@your-ec2-ip
```

☑️ Step 3: Install Dependencies on EC2

Once connected via SSH:

```
sudo apt update
sudo apt install -y git nginx python3-pip python3-venv nodejs npm
```

☑ Step 4: Create Production Branches on GitHub

Frontend Repository:

```
git checkout frontendaarrush
git checkout -b prod_release_front
git push origin prod_release_front
```

Backend Repository:

```
git checkout backendanmol
git checkout -b prod_release_back
git push origin prod_release_back
```

☑ Step 5: Set Up GitHub Actions CI/CD Workflows

Frontend Workflow (`.github/workflows/frontend-ci-cd.yml`)

Create in your **main branch** (build automatically created and deployed):

```
name: Frontend CI/CD



on:

  push:

    branches:

      - prod_release_frontend



jobs:

  deploy:

    runs-on: ubuntu-latest

    steps:

    - uses: actions/checkout@v4

    - name: Setup Node.js

      uses: actions/setup-node@v4

      with:

        node-version: '18'
```

```
    - name: Install dependencies
      run: npm install
    - name: Install indvidual dependencies
      run: npm install jspdf
    - name: Build application
      run: npm run build
    - name: Deploy to EC2
      uses: appleboy/scp-action@master
      with:
        host: ${{ secrets.EC2_HOST }}
        username: ubuntu
        key: ${{ secrets.SSH_PRIVATE_KEY }}
        source: dist/*
        target: /home/ubuntu/frontend
    - name: Update frontend files and restart Nginx
      uses: appleboy/ssh-action@master
      with:
        host: ${{ secrets.EC2_HOST }}
        username: ubuntu
        key: ${{ secrets.SSH_PRIVATE_KEY }}
        script: |
          sudo cp -r /home/ubuntu/frontend/dist/* /var/www/html/
          sudo systemctl restart nginx
```

Backend Workflow (`.github/workflows/backend-ci-cd.yml`)

```
name: Backend CI/CD

on:
  push:
    branches:
```

```
      - prod_release_backend  # make sure your branch is named this

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'

      - name: Install dependencies (locally just for validation)
        run: |
          python -m pip install --upgrade pip

      - name: Upload backend files to EC2
        uses: appleboy/scp-action@master
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          source: "*"
          target: "/home/ubuntu/backend"

      - name: Restart backend service on EC2
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /home/ubuntu/backend
            python3 -m venv env
            source env/bin/activate
            pip install -r requirements.txt
            pkill gunicorn || true
            nohup gunicorn -w 2 -k uvicorn.workers.UvicornWorker main:app \
              --bind 0.0.0.0:8000 > gunicorn.log 2>&1 &
```

☑ Step 6: Set Up GitHub Secrets

- Go to GitHub Repository → Settings → Secrets and Variables → Actions.
- Add two new secrets:
  - `EC2_HOST`: IP or Public DNS of your EC2 instance (e.g., `65.1.130.177` or `ec2-65-1-130-177.ap-south-1.compute.amazonaws.com`)
  - `SSH_PRIVATE_KEY`: Content of your SSH `.pem` key

☑ Step 7: Code-Level Adjustments Before Deployment

Backend (main.py):

- Update CORS origins:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "https://bharatayush.ai",
        "https://chat.bharatayush.ai",
        "https://bpms.bharatayush.ai"
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
```

- Verify environment variables for database and API keys.

Frontend (App.jsx or utils/api.js):

- Ensure API endpoint points to production backend URL (`https://backend.bharatayush.ai`).

Example:

```
export const fetchFromAPI = async (endpoint, data) => {
  const response = await fetch(`https://backend.bharatayush.ai${endpoint}`,
{
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
  });

  return response.json();
};
```

☑ Step 8: Initial Deployment Trigger

Push to trigger the deployment separately:

Frontend:

```
git checkout prod_release_front
git push origin prod_release_front
```

Backend:

```
git checkout prod_release_back
git push origin prod_release_back
```

☑ Step 9: Validate Deployment (Live Domain)

Once you eventually point your domains to EC2, use:

- Frontend: https://bharatayush.ai
- Backend: https://backend.bharatayush.ai

☑ Step 10: Validate Deployment via Public IP or DNS (Safe Pre-Test)

If you're not ready to update DNS records (because they're in use on Lightsail), use the **EC2 public IP or DNS** directly:

- **Frontend Check (browser):**
- `http://<EC2-PUBLIC-IP>/`
- `or`
- `http://<EC2-PUBLIC-DNS>/`

  Example: http://ec2-65-1-130-177.ap-south-1.compute.amazonaws.com/

- **Backend Check (browser or Postman):**
- `http://<EC2-PUBLIC-IP>:8000/`
- `or`
- `http://<EC2-PUBLIC-DNS>:8000/`

🔎 If IP doesn't work but DNS does

- Nginx or frontend may be configured with absolute URLs.
- Public DNS is the better option for testing.
- It's normal for React apps to fail if built with domain-specific paths.

🚀 CI/CD is now successfully set up for your AWS EC2 deployments!

Use Public DNS for pre-release testing and migrate DNS records once you're ready to go live.

Frontend URL: http://ec2-15-206-174-149.ap-south-1.compute.amazonaws.com

Backend URL: http://ec2-15-206-174-149.ap-south-1.compute.amazonaws.com:8000/