📓 Self-Hosted CI/CD Guide using Github actions

Stack: React (Vite) + Spring Boot (Maven) + Nginx + Ubuntu VPS.

🛠 Prerequisites

1. **A VPS** (Ubuntu 20.04/22.04 recommended).
2. **SSH Access** to the VPS.
3. **A Non-Root User** (We will call this `<YOUR_USER>`).
    - *Do not run runners as root.*
4. **Java 17 Installed** on the VPS (for running the JAR).

    ```
    sudo apt update
    sudo apt install openjdk-17-jre-headless -y
    ```

5. **Nginx Installed**.

    ```
    sudo apt install nginx -y
    ```

---

1 Phase 1: VPS Directory & Permission Setup

Before installing runners, prepare the destination folders where the code will live.

1. Create Backend Folder

```
# Create folder
sudo mkdir -p /var/www/<API_FOLDER_NAME>

# Give ownership to your user (So the runner can copy files without sudo)
sudo chown -R <YOUR_USER>:<YOUR_USER> /var/www/<API_FOLDER_NAME>
```

2. Create Frontend Folder

```
# Create folder
sudo mkdir -p /var/www/<UI_FOLDER_NAME>

# Give ownership to your user
sudo chown -R <YOUR_USER>:<YOUR_USER> /var/www/<UI_FOLDER_NAME>
```

---

2 Phase 2: Install GitHub Runner

Do this for each repository (one for API, one for UI).

1. Go to **GitHub Repo** -> **Settings** -> **Actions** -> **Runners** -> **New self-hosted runner**.
2. Select **Linux**.
3. SSH into your VPS and run the commands provided by GitHub (Download & Config).
   - *Tip: Create a specific folder for the runner, e.g., `~/runner-api`.*
4. **CRITICAL STEP: Install as a Service** Once configured, run these commands inside the runner folder to make it survive server reboots:

   ```
   # Install the service to run as your specific user
   sudo ./svc.sh install <YOUR_USER>

   # Start the service
   sudo ./svc.sh start
   ```

---

3 Phase 3: Allow Sudo Commands (Visudo)

The runner runs as `<YOUR_USER>`. It needs permission to restart the API service without typing a password.

1. Find the exact path of `systemctl`:

   ```
   which systemctl
   # Usually returns: /usr/bin/systemctl
   ```

2. Edit the sudoers file:

   ```
   sudo visudo
   ```

3. Scroll to the **absolute bottom** of the file and add this line:

   ```
   <YOUR_USER> ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart <SERVICE_NAME>
   ```

   (Replace `<SERVICE_NAME>` with the name we will create in Phase 4).

---

4 Phase 4: Spring Boot Setup (Backend)

1. Create Systemd Service

This keeps your API running in the background.

```
sudo nano /etc/systemd/system/<SERVICE_NAME>.service
```

Paste this configuration:

```
[Unit]
Description=Spring Boot API Service
After=network.target

[Service]
User=<YOUR_USER>
# Pointing to the specific JAR file
ExecStart=/usr/bin/java -jar /var/www/<API_FOLDER_NAME>/app.jar --
spring.profiles.active=staging
SuccessExitStatus=143
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

2. Enable the Service

```
sudo systemctl daemon-reload
sudo systemctl enable <SERVICE_NAME>
```

3. Create GitHub Action Workflow

In your Java project, create `.github/workflows/deploy.yml`:

```
name: Deploy API

on:
  push:
    branches: [ "staging" ] # Match your branch

jobs:
  build:
    runs-on: self-hosted

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven

      - name: Build with Maven
        # -DskipTests avoids database connection errors during build
        run: mvn -B package -DskipTests --file pom.xml

      - name: Deploy to Server
        run: |
          # 1. Copy JAR (No sudo needed because we did chown)
          cp target/*.jar /var/www/<API_FOLDER_NAME>/app.jar

          # 2. Restart Service (Full path required for sudoers match)
          sudo /usr/bin/systemctl restart <SERVICE_NAME>
```

---

5 Phase 5: React/Vite Setup (Frontend)

1. GitHub Action Workflow

In your React project, create `.github/workflows/deploy.yml`:

```
name: Deploy UI

on:
  push:
    branches: [ "staging" ]

jobs:
  build:
    runs-on: self-hosted

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Use Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18.x
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Build project
        run: npm run build

      - name: Deploy to Nginx
        run: |
          # 1. Clean old files
          rm -rf /var/www/<UI_FOLDER_NAME>/*
          # 2. Copy new dist files
          cp -r dist/* /var/www/<UI_FOLDER_NAME>/

          # Note: No Nginx reload is needed for static file updates!
```

---

6 Phase 6: Application Configuration (Swagger & Paths)

To make Swagger work behind Nginx, we move it to reside inside the `/api` path.

1. Update `application.properties` (Staging/Prod)

Add this to your properties file on the server (or in the repo if not ignored):

```
# Move Swagger UI and Docs inside /api
springdoc.api-docs.path=/api/v3/api-docs
springdoc.swagger-ui.path=/api/swagger-ui.html
```

2. Update Spring Security (`SecurityConfig.java`)

Update your `configure(HttpSecurity http)` method to allow these new paths:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    // ... CSRF and CORS settings ...

    http.authorizeRequests()
        // Allow the new Swagger paths
        .antMatchers("/api/swagger-ui/**").permitAll()
        .antMatchers("/api/swagger-ui.html").permitAll()
        .antMatchers("/api/v3/api-docs/**").permitAll()

        // Your other logic
        .antMatchers("/api/**").authenticated();

    // ... filters ...
}
```

---

7 Phase 7: Nginx Reverse Proxy Configuration

Configure Nginx to serve the React files and Proxy the API.

```
sudo nano /etc/nginx/sites-available/default
```

```
server {
    listen 80 default_server;
    server_name _;

    root /var/www/<UI_FOLDER_NAME>;
    index index.html;

    # 1. Frontend: Serve React App
    location / {
        try_files $uri $uri/ /index.html;
    }

    # 2. Backend: Proxy API calls (including Swagger)
    location /api {
        # This regex ensures /api stays in the URL when passed to Spring
        rewrite ^\/api\/(.*)$ /api/$1 break;

        proxy_pass http://localhost:8080;

        # Standard Proxy Headers
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Final Reload:

```
sudo systemctl reload nginx
```

✅ Summary of Success Check

1. **Push Code:** Commit to `staging`.
2. **Check GitHub:** Actions tab should turn Green.
3. **Check UI:** `http://<VPS_IP>/` should load your React App.
4. **Check API:** `http://<VPS_IP>/api/swagger-ui.html` should load Swagger.
5. **Check Logs:** `sudo journalctl -u <SERVICE_NAME> -f` to see the backend running.