Deploy Any ReactJS App on a Linux Server via CI/CD (GitHub Actions)

Generated: 2025-08-29 06:00

This guide walks you through building a ReactJS app with routing and reusable components, hosting the code on GitHub, and setting up a fully automated CI/CD pipeline to deploy to a Linux server (Ubuntu/Debian/CentOS) using Nginx. Optional Docker instructions are included.

What you'll deliver

• GitHub repo with React app (min. 3 reusable components + basic routing).

• Working GitHub Actions pipeline: install, lint, test, build, and deploy on push to main.

• Linux server serving your app via Nginx (or Dockerized Nginx).

• Documentation (this PDF) explaining the setup.

Prerequisites

• A Linux server (Ubuntu/Debian/CentOS) with SSH access and a public IP or domain.

• A non-root sudo user on the server (e.g., 'deploy').

• GitHub account and a new repository.

• Node.js 20+ and npm locally for development (not required on the server for static hosting).

1) Create a React App with Routing & Reusable Components

Below is a quick-start using Vite + React, React Router, ESLint, and Vitest (tests).

Initialize project

```
# Using Vite
npm create vite@latest my-react-app -- --template react
cd my-react-app

# Install router, lint, test deps
npm i react-router-dom
npm i -D eslint @eslint/js eslint-plugin-react vite-plugin-eslint
npm i -D vitest @testing-library/react @testing-library/jsdom

# Optional: set Node version for CI
echo "20" > .nvmrc
```

Minimal app structure (suggested)

```
my-react-app/
  ■■ src/
  ■   ■■ components/
  ■   ■   ■■ NavBar.jsx
  ■   ■   ■■ Card.jsx
  ■   ■   ■■ Footer.jsx
  ■   ■■ pages/
  ■   ■   ■■ Home.jsx
  ■   ■   ■■ About.jsx
  ■   ■   ■■ Dashboard.jsx
  ■   ■■ App.jsx
  ■   ■■ main.jsx
  ■   ■■ App.test.jsx
  ■■ public/
  ■■ index.html
  ■■ package.json
  ■■ vite.config.js
  ■■ README.md
```

Example: basic routing (src/App.jsx)

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
import Dashboard from './pages/Dashboard';
import NavBar from './components/NavBar';
import Footer from './components/Footer';

export default function App() {
  return (
    <BrowserRouter>
      <NavBar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/dashboard" element={<Dashboard />} />
      </Routes>
      <Footer />
    </BrowserRouter>
  );
}
```

Example: reusable component (src/components/Card.jsx)

```
export default function Card({ title, children }) {
  return (
    <div style={{border:'1px solid #ddd', padding:'1rem', borderRadius:8}}>
      <h3>{title}</h3>
      <div>{children}</div>
    </div>
  );
}
```

Add scripts to package.json

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "test": "vitest run",
    "lint": "eslint ."
  }
}
```

2) Push to GitHub & Set Up CI (GitHub Actions)

Create a new GitHub repository and push the code:

```
git init
git add .
git commit -m "feat: initial app"
git branch -M main
git remote add origin git@github.com:<YOUR-USER>/<YOUR-REPO>.git
git push -u origin main
```

Add the GitHub Actions workflow to build, lint, test, and create the production build:

.github/workflows/ci-cd.yml (CI + CD via rsync over SSH)

```
name: CI/CD - React to Linux (Nginx)

on:
  push:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'
      - run: npm ci
      - run: npm run lint
      - run: npm test
      - run: npm run build
      - name: Archive production build
        uses: actions/upload-artifact@v4
        with:
          name: react-build
          path: |
            dist/**
  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: Download build artifact
        uses: actions/download-artifact@v4
        with:
          name: react-build
          path: dist

      - name: Prepare web root (first run only needs to create once)
        uses: appleboy/ssh-action@v1.2.0
        with:
          host: ${{ secrets.SSH_HOST }}
          username: ${{ secrets.SSH_USER }}
          key: ${{ secrets.SSH_KEY }}
          script: |
            set -e
            sudo mkdir -p /var/www/myapp
            sudo chown -R ${{ secrets.SSH_USER }}: /var/www/myapp

      - name: Sync files
        run: |
          rsync -avz --delete dist/ ${{ secrets.SSH_USER }}@${{ secrets.SSH_HOST }}:/var/www/myapp/
      - name: Reload Nginx
```

```
        uses: appleboy/ssh-action@v1.2.0
        with:
          host: ${{ secrets.SSH_HOST }}
          username: ${{ secrets.SSH_USER }}
          key: ${{ secrets.SSH_KEY }}
          script: |
            sudo nginx -t && sudo systemctl reload nginx
```

Add GitHub Secrets (Repo → Settings → Secrets and variables → Actions):

- SSH_HOST = your.server.ip.or.domain
- SSH_USER = deploy (non-root sudo user)
- SSH_KEY = private key contents (PEM) matching the public key placed on the server

3) Prepare the Linux Server (One-time)

Run the following steps on your Ubuntu/Debian/CentOS server. Replace values as needed.

Create a deploy user & add SSH key

```
# as root
adduser deploy
usermod -aG sudo deploy

# on your local machine:
ssh-copy-id deploy@YOUR_SERVER_IP

# or manually add your public key into:
#   /home/deploy/.ssh/authorized_keys  (permissions 600, .ssh folder 700)
```

Install Nginx

```
# Ubuntu/Debian
sudo apt update && sudo apt install -y nginx

# CentOS/RHEL (with dnf or yum)
# sudo dnf install -y nginx  && sudo systemctl enable --now nginx
# sudo systemctl enable --now nginx
```

Configure Nginx to serve the React build

Create a new site configuration (replace domain/IP as appropriate):

```
# /etc/nginx/sites-available/myapp
server {
    listen 80;
    server_name YOUR_DOMAIN_OR_IP;

    root /var/www/myapp;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    access_log /var/log/nginx/myapp_access.log;
    error_log  /var/log/nginx/myapp_error.log;
}
```

```
# Enable site (Ubuntu/Debian)
sudo ln -s /etc/nginx/sites-available/myapp /etc/nginx/sites-enabled/myapp
sudo nginx -t
sudo systemctl reload nginx
```

If you want HTTPS, use Certbot (Let's Encrypt):

```
sudo apt install -y certbot python3-certbot-nginx
sudo certbot --nginx -d YOUR_DOMAIN
```

4) How the CD Works

On every push to main:

• CI job installs deps, lints, runs tests, and builds the app into 'dist/'.

• The build artifact is downloaded in the deploy job.

• The deploy job rsyncs 'dist/' to '/var/www/myapp' on the server via SSH.

• Nginx is reloaded automatically. No manual steps.

Zero-Downtime (Optional)

For larger apps, you can deploy to versioned releases and swap a symlink:

```
# On server side (run via SSH in workflow):
sudo mkdir -p /var/www/myapp/releases
REL=/var/www/myapp/releases/$(date +%Y%m%d%H%M%S)
mkdir -p "$REL"
# rsync to $REL then update symlink:
ln -sfn "$REL" /var/www/myapp
sudo nginx -t && sudo systemctl reload nginx
```

5) Optional: Dockerized Deployment (Nginx Static Server)

Build the static assets in a Node stage, then serve via Nginx in the final stage:

Dockerfile

```
# Dockerfile (multi-stage)
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/dist /usr/share/nginx/html
```

nginx.conf

```
server {
  listen 80;
  server_name _;
  root /usr/share/nginx/html;
  index index.html;
  location / {
    try_files $uri /index.html;
  }
}
```

docker-compose.yml (on server)

```
services:
  web:
    image: ghcr.io/<YOUR-USER>/<YOUR-REPO>:latest
    ports:
      - "80:80"
    restart: unless-stopped
```

GitHub Actions (build + push image)

```
name: CI/CD - Docker

on:
  push:
    branches: [ "main" ]

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'
      - run: npm ci
      - run: npm test
      - run: npm run build
      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
```

```
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}
    - name: Build and push
      uses: docker/build-push-action@v6
      with:
        context: .
        push: true
        tags: ghcr.io/${{ github.repository_owner }}/${{ github.event.repository.name }}:latest
deploy:
  needs: build-and-push
  runs-on: ubuntu-latest
  steps:
    - name: Remote deploy
      uses: appleboy/ssh-action@v1.2.0
      with:
        host: ${{ secrets.SSH_HOST }}
        username: ${{ secrets.SSH_USER }}
        key: ${{ secrets.SSH_KEY }}
        script: |
          docker compose pull && docker compose up -d
```

6) Monitoring & Logs

• Nginx access logs: /var/log/nginx/myapp_access.log

• Nginx error logs: /var/log/nginx/myapp_error.log

• Dockerized: docker logs -f

• Uptime checks: free tools like Uptime Kuma or external services

7) End-to-End Smoke Test

• Push a commit to main.

• Watch Actions run: CI passes, deploy runs.

• Open http://YOUR_DOMAIN_OR_IP to see the app.

• Break a test and push to verify pipeline blocks bad deploys.

8) Tech Choices Explained

• Vite for fast builds and modern React tooling.

• React Router for basic routing.

• ESLint + Vitest/RTL for code quality and tests.

• Nginx serving static assets (fast, minimal server load).

• Rsync-over-SSH for simple, reliable deployments (or Docker for parity).

Appendix A: Minimal Example Tests

```
import { describe, it, expect } from 'vitest';
import { render } from '@testing-library/react';
import App from './App';

describe('App', () => {
  it('renders without crashing', () => {
    const { container } = render(<App />);
    expect(container).toBeTruthy();
  });
});
```

Appendix B: Security Tips

• Use SSH keys (no passwords) for deploys.

• Restrict sudo for the deploy user if possible.

• Keep Nginx and system packages updated.

• Use HTTPS with Let's Encrypt in production.

• Consider a WAF/CDN like Cloudflare.