Complete Guide: VPS Deployment with CI/CD

Secure, Scalable, Production-Ready Deployment
for Node.js Applications

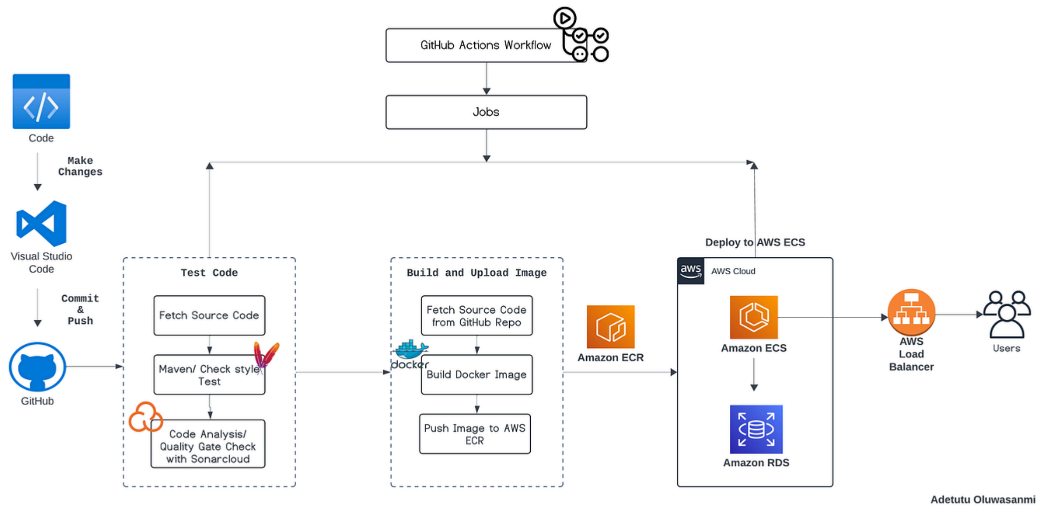
Ubuntu VPS

GitHub Actions

Docker + Nginx

Beginner-Friendly Documentation

Architecture Overview



What We're Building

Push to GitHub
Trigger CI/CD workflow on main branch

Build & Deploy
GitHub Actions builds Docker image & deploys via SSH

Run on VPS
Docker container runs Node.js application

Nginx Reverse Proxy
Forwards traffic to container with SSL

Phase 1: Server Setup & SSH Access

01 SSH into Server

Login using root credentials provided by VPS provider

02 Update System

Install latest security patches and updates

03 Create Deployment User

Set up a dedicated user for deployment tasks with sudo privileges

04 Switch to Deployer User

Log in as the new deployment user for all subsequent operations

<> COMMANDS

Login to server

```
ssh user@ip
```

Update packages

```
sudo apt update &&
sudo apt upgrade -y
```

Create deployer user

```
adduser deployer
```

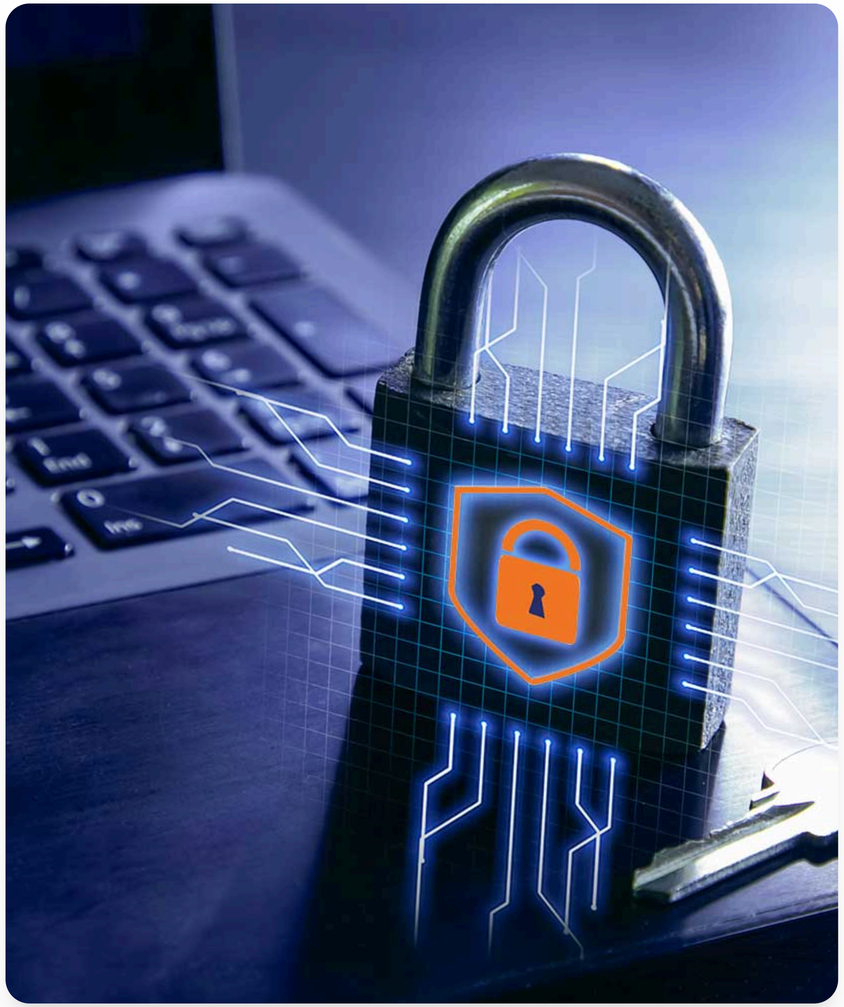
Grant sudo access

```
usermod -aG sudo deployer
```

Switch user

```
su - deployer
```

Phase 2: Security Hardening



SSH Key Authentication

Generate secure ED25519 key pair on local machine

```
ssh-keygen -t ed25519
```

Disable Password & Root Login

Edit `/etc/ssh/sshd_config` and remove comments

```
PasswordAuthentication no
```

```
PermitRootLogin no
```

Restart SSH: sudo systemctl restart ssh

Firewall Configuration

Configure UFW to allow only necessary traffic

```
sudo ufw allow OpenSSH
```

```
sudo ufw allow 80/tcp
```

```
sudo ufw allow 443/tcp
```

```
sudo ufw enable
```

Fail2Ban Installation

Protect against brute force attacks

```
sudo apt install fail2ban -y
```

Phase 3: Docker Installation

Install Docker

Install Docker Engine on Ubuntu server

```
sudo apt install docker.io -y
```

Add Deployer to Docker Group

Grant deployer user Docker privileges

```
sudo usermod -aG docker deployer
```

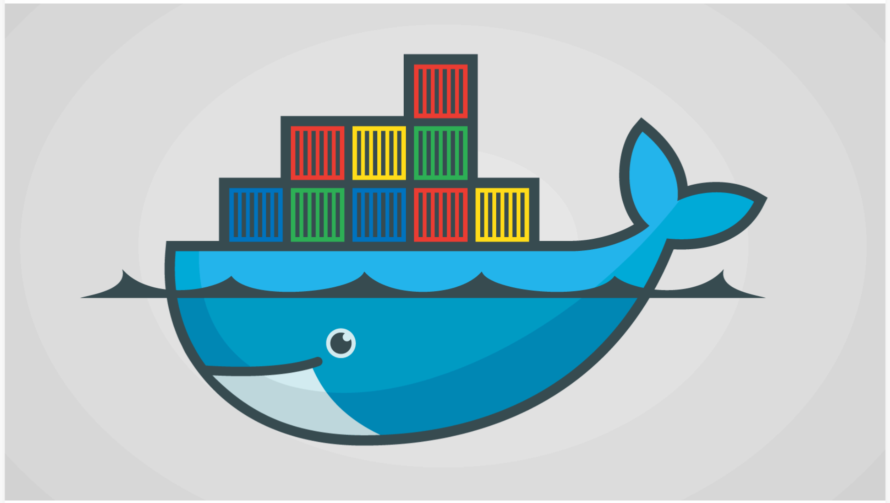
Logout and login again for changes to take effect

Create Dockerfile

Define your application container configuration

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["npm", "start"]
```

✓ Consistent Environment ✓ Easy Scaling



Containerization Made Simple

Phase 4: Nginx Reverse Proxy

Install Nginx

High-performance web server and reverse proxy

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

Configure Reverse Proxy

Create site config: `/etc/nginx/sites-available/myapp`

```
server {
  server_name api.yourdomain.com;
  location / {
    proxy_pass http://localhost:3000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
  }
}
```

Enable & Test

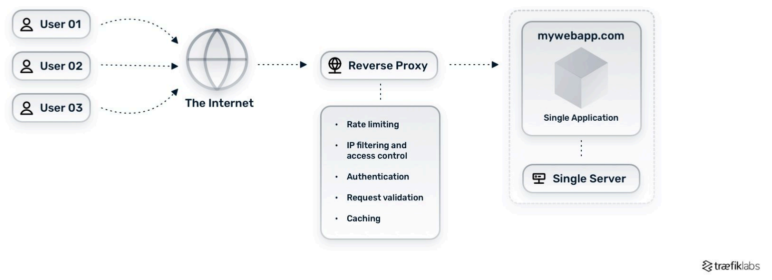
Activate configuration and verify syntax

```
sudo ln -s ...
```

```
sudo nginx -t
```

```
sudo systemctl restart nginx
```

Load Balancing   SSL Termination



Traffic Flow: Users → Nginx → Docker Container

Phase 5: SSL/TLS with Let's Encrypt

Install Certbot

Certbot tool with Nginx plugin for automatic SSL configuration

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

Issue SSL Certificate

Request free certificate from Let's Encrypt

```
sudo certbot --nginx -d api.yourdomain.com
```

Replace `api.yourdomain.com` with your actual domain

Auto-Renewal Setup

Certificates expire in 90 days, auto-renewal is configured by default

Free Forever | Auto-Renew

Encrypted Connection | Browser Trust

HTTPS Enabled for Secure Traffic

Phase 6: GitHub Actions Workflow

Create Workflow File

Location: `.github/workflows/deploy.yml`

```yaml
name: Deploy to VPS

on:
  push:
    branches: ["main"]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build & Push Docker Image
        uses: docker/build-push-action@v4
        with:
          push: true
          tags: youruser/myapp:latest

      - name: Deploy via SSH
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.VPS_IP }}
          username: deployer
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            docker pull youruser/myapp:latest
            docker stop myapp || true
            docker rm myapp || true
            docker run -d --name myapp -p 3000:3000 \
              --restart unless-stopped \
              --env-file .env youruser/myapp:latest
```

Automatic Deployment

Triggers on every push to main branch

Docker Build & Push

Builds Docker image and pushes to registry

SSH Connection

Securely connects to VPS using SSH keys

Seamless Updates

Stops old container, starts new one

Required Secrets

- `VPS_IP`
- `SSH_PRIVATE_KEY`

Phase 7: GitHub Secrets Configuration

1 **Open Repository Settings**
Navigate to your GitHub repository

2 **Access Secrets Section**
Settings → Security → Secrets and variables

3 **Add VPS_IP Secret**
Click "New repository secret", name it VPS_IP

4 **Add SSH_PRIVATE_KEY Secret**
Paste your private SSH key content

5 **Verify & Deploy**
Check GitHub Actions tab for workflow status

Required Secrets

VPS_IP

Your VPS public IP address

`Example: 123.45.67.89`

SSH_PRIVATE_KEY

Content of your private SSH key

`Usually from ~/.ssh/id_ed25519`

Why Secrets?

Securely store sensitive data. Secrets are encrypted and only accessible to workflows. Never commit them to git!

Summary & Best Practices

7 Phases Completed

1. Server Setup & SSH Access
2. Security Hardening
3. Docker Installation
4. Nginx Reverse Proxy
5. SSL/TLS with Let's Encrypt
6. GitHub Actions Workflow
7. GitHub Secrets Configuration

✓ Production-Ready Deployment

Security Best Practices

- Never share SSH keys
- Keep system updated
- Monitor logs
- Use strong passwords
- Regular backups
- Disable root login

Troubleshooting Tips

- Check Docker logs
- Test firewall rules
- Verify nginx config
- Review GitHub Actions

Next Steps

- Add health checks
- Set up monitoring
- Implement CI improvements
- Deploy more apps