# Complete CI/CD Guide for Full-Stack Applications with PM2

This guide provides step-by-step instructions for setting up a complete CI/CD pipeline for a full-stack application (React + Node.js) with automatic deployment to a VPS, including SSL configuration and PM2 process management.

## Table of Contents

1. [Prerequisites](#prerequisites)

2. [VPS Setup](#vps-setup)

3. [Domain Configuration](#domain-configuration)

4. [SSL Certificate Setup](#ssl-certificate-setup)

5. [GitHub Actions Configuration](#github-actions-configuration)

6. [Application Configuration](#application-configuration)

7. [PM2 Process Management](#pm2-process-management)

8. [Nginx Configuration](#nginx-configuration)

9. [Deployment Process](#deployment-process)

10. [Monitoring and Maintenance](#monitoring-and-maintenance)

11. [Troubleshooting](#troubleshooting)

## Prerequisites

### Required Tools

* Ubuntu VPS (20.04 or later)
* Domain name with DNS access
* GitHub repository
* SSH access to VPS
* Basic knowledge of Linux commands

### Domain Setup

* `your-domain.com` - Frontend
* `api.your-domain.com` - Backend API
* `www.your-domain.com` - Redirect to main domain

## VPS Setup

### Step 1: Connect to Your VPS

ssh root@your-vps-ip  
# or  
ssh username@your-vps-ip

### Step 2: Create a Non-Root User (Recommended)

# Create new user  
adduser deploy  
usermod -aG sudo deploy  
# Switch to new user  
su - deploy

### Step 3: Update System

sudo apt update && sudo apt upgrade -y

### Step 4: Install Required Packages

# Install essential packages  
sudo apt install -y curl wget git nginx certbot python3-certbot-nginx ufw fail2ban  
# Install Node.js 18  
curl -fsSL [https://deb.nodesource.com/setup\_18.x](https://deb.nodesource.com/setup\_18.x) | sudo -E bash -  
sudo apt-get install -y nodejs  
# Install PM2 globally  
sudo npm install -g pm2  
# Install Docker (Optional but recommended)  
curl -fsSL [https://get.docker.com](https://get.docker.com) -o get-docker.sh  
sudo sh get-docker.sh  
sudo usermod -aG docker $USER  
# Install Docker Compose  
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
sudo chmod +x /usr/local/bin/docker-compose

### Step 5: Configure Firewall

# Configure UFW firewall  
sudo ufw allow ssh  
sudo ufw allow 80  
sudo ufw allow 443  
sudo ufw --force enable  
# Configure fail2ban for additional security  
sudo systemctl enable fail2ban  
sudo systemctl start fail2ban

## Domain Configuration

### Step 1: Configure DNS Records

In your domain registrar's DNS settings, add these records:

Type: A  
Name: @  
Value: YOUR\_VPS\_IP  
TTL: 300  
Type: A  
Name: www  
Value: YOUR\_VPS\_IP  
TTL: 300  
Type: A  
Name: api  
Value: YOUR\_VPS\_IP  
TTL: 300

### Step 2: Verify DNS Propagation

# Check if DNS is propagated  
nslookup your-domain.com  
nslookup api.your-domain.com  
nslookup [www.your-domain.com](https://www.your-domain.com)

## SSL Certificate Setup

### Step 1: Install Certbot

sudo apt install certbot python3-certbot-nginx

### Step 2: Get SSL Certificates

# Get certificates for all domains  
sudo certbot --nginx -d your-domain.com -d [www.your-domain.com](https://www.your-domain.com) -d api.your-domain.com --non-interactive --agree-tos --email [your-email@example.com](mailto:your-email@example.com)

### Step 3: Set Up Auto-Renewal

# Test auto-renewal  
sudo certbot renew --dry-run  
# Add to crontab for automatic renewal  
sudo crontab -e  
# Add this line (without the #):  
0 12 \* \* \* /usr/bin/certbot renew --quiet

## GitHub Actions Configuration

### Step 1: Generate SSH Key for Deployment

# Create deploy user if it doesn't exist  
sudo adduser deploy  
sudo usermod -aG sudo deploy  
# Create SSH directory for deploy user  
sudo mkdir -p /home/deploy/.ssh  
sudo chown -R deploy:deploy /home/deploy/.ssh  
sudo chmod 700 /home/deploy/.ssh  
# Generate SSH key for deploy user  
sudo -u deploy ssh-keygen -t rsa -b 4096 -C "your-email@example.com" -f /home/deploy/.ssh/github\_actions  
# Add public key to authorized\_keys  
sudo -u deploy cat /home/deploy/.ssh/github\_actions.pub >> /home/deploy/.ssh/authorized\_keys  
sudo chmod 600 /home/deploy/.ssh/authorized\_keys  
# Get the private key for GitHub secrets  
sudo -u deploy cat /home/deploy/.ssh/github\_actions

### Step 2: Configure GitHub Secrets

In your GitHub repository, go to Settings > Secrets and variables > Actions, add:

* `VPS\_HOST`: Your VPS IP address
* `VPS\_USERNAME`: Your VPS username (e.g., `deploy`)
* `VPS\_SSH\_KEY`: Content of the private key file

### Step 3: Create GitHub Actions Workflow

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

name: Deploy to VPS  
on:  
 push:  
 branches: [ main ]  
 pull\_request:  
 branches: [ main ]  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v4  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v4  
 with:  
 node-version: '18'  
 cache: 'npm'  
  
 - name: Install frontend dependencies  
 run: npm install  
  
 - name: Build frontend  
 run: npm run build  
  
 - name: Install backend dependencies  
 run: |  
 cd backend  
 npm install  
  
 - name: Deploy to VPS  
 uses: appleboy/ssh-action@v1.0.3  
 with:  
 host: ${{ secrets.VPS\_HOST }}  
 username: ${{ secrets.VPS\_USERNAME }}  
 key: ${{ secrets.VPS\_SSH\_KEY }}  
 script: |  
 cd /var/www/your-app-name  
 sudo -u deploy git pull origin main  
  
 # Install frontend dependencies and build  
 sudo -u deploy npm install  
 sudo -u deploy npm run build  
  
 # Install backend dependencies  
 cd backend  
 sudo -u deploy npm install  
 cd ..  
  
 # Restart PM2 services  
 sudo -u deploy pm2 restart all  
 sudo systemctl reload nginx

## Application Configuration

### Step 1: Clone Repository

# Create application directory  
sudo mkdir -p /var/www/your-app-name  
sudo chown -R deploy:deploy /var/www/your-app-name  
# Clone repository  
cd /var/www/your-app-name  
sudo -u deploy git clone [https://github.com/YOUR\_USERNAME/YOUR\_REPO\_NAME.git](https://github.com/YOUR\_USERNAME/YOUR\_REPO\_NAME.git) .

### Step 2: Install Dependencies

# Install frontend dependencies  
sudo -u deploy npm install  
# Install backend dependencies  
cd backend  
sudo -u deploy npm install  
cd ..

### Step 3: Build Application

# Build frontend  
sudo -u deploy npm run build

### Step 4: Configure Environment Variables

# Copy environment file  
sudo -u deploy cp env.example .env  
# Edit environment variables  
sudo -u deploy nano .env

## PM2 Process Management

### Step 1: Create PM2 Ecosystem File

# Create PM2 ecosystem file  
sudo -u deploy tee /var/www/your-app-name/ecosystem.config.cjs > /dev/null <<'EOF'  
module.exports = {  
 apps: [  
 {  
 name: 'app-frontend',  
 cwd: '/var/www/your-app-name',  
 script: 'npm',  
 args: 'run preview',  
 env: {  
 NODE\_ENV: 'production',  
 PORT: 4173  
 },  
 instances: 1,  
 autorestart: true,  
 watch: false,  
 max\_memory\_restart: '1G'  
 },  
 {  
 name: 'app-backend',  
 cwd: '/var/www/your-app-name/backend',  
 script: 'npm',  
 args: 'start',  
 env: {  
 NODE\_ENV: 'production',  
 PORT: 3000  
 },  
 instances: 1,  
 autorestart: true,  
 watch: false,  
 max\_memory\_restart: '1G'  
 }  
 ]  
};  
EOF

### Step 2: Start Services with PM2

# Start applications with PM2  
sudo -u deploy pm2 start /var/www/your-app-name/ecosystem.config.cjs  
# Save PM2 configuration  
sudo -u deploy pm2 save  
# Set up PM2 to start on boot  
sudo -u deploy pm2 startup  
# Copy and run the command that PM2 outputs with sudo  
# Check PM2 status  
sudo -u deploy pm2 status

### Step 3: PM2 Management Commands

# Check status  
sudo -u deploy pm2 status  
# View logs  
sudo -u deploy pm2 logs  
sudo -u deploy pm2 logs app-frontend  
sudo -u deploy pm2 logs app-backend  
# Restart services  
sudo -u deploy pm2 restart all  
sudo -u deploy pm2 restart app-frontend  
sudo -u deploy pm2 restart app-backend  
# Stop services  
sudo -u deploy pm2 stop all  
# Delete services  
sudo -u deploy pm2 delete all  
# Monitor resources  
sudo -u deploy pm2 monit

## Nginx Configuration

### Step 1: Create Nginx Configuration

# Create nginx site configuration  
sudo tee /etc/nginx/sites-available/your-app > /dev/null <<'EOF'  
# Frontend (HTTPS)  
server {  
 listen 443 ssl http2;  
 server\_name your-domain.com [www.your-domain.com](https://www.your-domain.com);  
  
 # SSL Configuration  
 ssl\_certificate /etc/letsencrypt/live/your-domain.com/fullchain.pem;  
 ssl\_certificate\_key /etc/letsencrypt/live/your-domain.com/privkey.pem;  
 ssl\_protocols TLSv1.2 TLSv1.3;  
 ssl\_prefer\_server\_ciphers off;  
  
 # Security headers  
 add\_header X-Frame-Options "SAMEORIGIN" always;  
 add\_header X-XSS-Protection "1; mode=block" always;  
 add\_header X-Content-Type-Options "nosniff" always;  
 add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;  
  
 # Frontend - serve static files  
 root /var/www/your-app-name/dist;  
 index index.html;  
  
 location / {  
 try\_files $uri $uri/ /index.html;  
 }  
  
 # API proxy  
 location /api/ {  
 proxy\_pass http://localhost:3000/;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}  
# Backend API (HTTPS)  
server {  
 listen 443 ssl http2;  
 server\_name api.your-domain.com;  
  
 # SSL Configuration  
 ssl\_certificate /etc/letsencrypt/live/your-domain.com/fullchain.pem;  
 ssl\_certificate\_key /etc/letsencrypt/live/your-domain.com/privkey.pem;  
 ssl\_protocols TLSv1.2 TLSv1.3;  
 ssl\_prefer\_server\_ciphers off;  
  
 # Security headers  
 add\_header X-Frame-Options "SAMEORIGIN" always;  
 add\_header X-XSS-Protection "1; mode=block" always;  
 add\_header X-Content-Type-Options "nosniff" always;  
 add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;  
  
 # CORS headers  
 add\_header Access-Control-Allow-Origin "https://your-domain.com" always;  
 add\_header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS" always;  
 add\_header Access-Control-Allow-Headers "DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range,Authorization" always;  
 add\_header Access-Control-Allow-Credentials "true" always;  
  
 # Backend API  
 location / {  
 proxy\_pass http://localhost:3000;  
 proxy\_set\_header Host $host;  
 proxy\_set\_header X-Real-IP $remote\_addr;  
 proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  
 proxy\_set\_header X-Forwarded-Proto $scheme;  
 }  
}  
# HTTP to HTTPS redirect  
server {  
 listen 80;  
 server\_name your-domain.com [www.your-domain.com](https://www.your-domain.com) api.your-domain.com;  
 return 301 https://$server\_name$request\_uri;  
}  
EOF  
# Enable the site  
sudo ln -sf /etc/nginx/sites-available/your-app /etc/nginx/sites-enabled/  
# Remove default site  
sudo rm -f /etc/nginx/sites-enabled/default  
# Test nginx configuration  
sudo nginx -t  
# Reload nginx  
sudo systemctl reload nginx

## Deployment Process

### Automatic Deployment

Once everything is set up, deployment happens automatically when you push to the main branch:

1. Push code to GitHub

2. GitHub Actions triggers

3. Code is built and tested

4. SSH connection to VPS is established

5. Code is pulled and deployed

6. PM2 services are restarted

### Manual Deployment

If you need to deploy manually:

# On your VPS  
cd /var/www/your-app-name  
sudo -u deploy git pull origin main  
sudo -u deploy npm install  
sudo -u deploy npm run build  
cd backend && sudo -u deploy npm install && cd ..  
sudo -u deploy pm2 restart all  
sudo systemctl reload nginx

## Monitoring and Maintenance

### Check Service Status

# Check PM2 services  
sudo -u deploy pm2 status  
# Check nginx  
sudo systemctl status nginx  
# Check all services  
sudo -u deploy pm2 status  
sudo systemctl status nginx

### Monitor Resources

# Check system resources  
htop  
df -h  
free -h  
# Check nginx access logs  
sudo tail -f /var/log/nginx/access.log  
sudo tail -f /var/log/nginx/error.log  
# Check PM2 logs  
sudo -u deploy pm2 logs --lines 50

### Backup Strategy

# Create backup script  
sudo -u deploy tee /var/www/your-app-name/backup.sh > /dev/null <<'EOF'  
#!/bin/bash  
DATE=$(date +%Y%m%d\_%H%M%S)  
BACKUP\_DIR="/var/backups/your-app"  
mkdir -p $BACKUP\_DIR  
# Backup application  
tar -czf $BACKUP\_DIR/app\_$DATE.tar.gz /var/www/your-app-name  
# Backup nginx config  
cp /etc/nginx/sites-available/your-app $BACKUP\_DIR/nginx\_$DATE.conf  
# Backup SSL certificates  
cp -r /etc/letsencrypt $BACKUP\_DIR/ssl\_$DATE  
# Keep only last 7 days of backups  
find $BACKUP\_DIR -name "\*.tar.gz" -mtime +7 -delete  
EOF  
# Make executable and add to crontab  
sudo chmod +x /var/www/your-app-name/backup.sh  
sudo -u deploy crontab -e  
# Add: 0 2 \* \* \* /var/www/your-app-name/backup.sh

## Troubleshooting

### Common Issues

#### 1. PM2 Services Not Starting

# Check PM2 status  
sudo -u deploy pm2 status  
# Check PM2 logs  
sudo -u deploy pm2 logs  
# Restart PM2 services  
sudo -u deploy pm2 restart all  
# Check if ports are in use  
sudo netstat -tlnp | grep :3000  
sudo netstat -tlnp | grep :4173

#### 2. SSL Certificate Issues

# Check certificate status  
sudo certbot certificates  
# Renew certificates manually  
sudo certbot renew  
# Check nginx SSL configuration  
sudo nginx -t

#### 3. Nginx Issues

# Test nginx configuration  
sudo nginx -t  
# Check nginx error logs  
sudo tail -f /var/log/nginx/error.log  
# Reload nginx  
sudo systemctl reload nginx

#### 4. Permission Issues

# Fix ownership  
sudo chown -R deploy:deploy /var/www/your-app-name  
# Fix permissions  
sudo chmod -R 755 /var/www/your-app-name

### Log Locations

* PM2 logs: `sudo -u deploy pm2 logs`
* Nginx logs: `/var/log/nginx/access.log`, `/var/log/nginx/error.log`
* System logs: `/var/log/syslog`

### Performance Optimization

#### 1. Enable Gzip Compression

Already configured in nginx.conf

#### 2. Set Up Caching

# Add to nginx.conf  
location ~\* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {  
 expires 1y;  
 add\_header Cache-Control "public, immutable";  
}

## Security Best Practices

### 1. Regular Updates

# Update system packages  
sudo apt update && sudo apt upgrade -y  
# Update Node.js packages  
sudo -u deploy npm audit fix

### 2. Firewall Configuration

# Check UFW status  
sudo ufw status  
# Block unnecessary ports  
sudo ufw deny 3000 # If not needed externally

### 3. SSL Security

* Use strong SSL configurations
* Regular certificate renewal
* HSTS headers (already configured)

### 4. Application Security

* Use environment variables for secrets
* Implement proper CORS policies
* Regular security audits

## Scaling Considerations

### 1. Load Balancing

For high traffic, consider:

* Multiple VPS instances
* Load balancer (HAProxy, Nginx)
* CDN for static assets

### 2. Database Scaling

* Separate database server
* Read replicas
* Caching layer (Redis)

### 3. Monitoring

* Application monitoring (New Relic, DataDog)
* Server monitoring (Nagios, Zabbix)
* Log aggregation (ELK Stack)

## Conclusion

This guide provides a complete CI/CD setup for your full-stack application with PM2. The setup includes:

* ✅ Automatic deployment on git push
* ✅ SSL certificates with auto-renewal
* ✅ Reverse proxy with Nginx
* ✅ Process management with PM2
* ✅ Security hardening
* ✅ Monitoring and logging

Your application will be available at:

* Frontend: `https://your-domain.com`
* Backend: `https://api.your-domain.com`

Remember to:

1. Replace placeholder values with your actual information

2. Test the setup thoroughly

3. Monitor logs and performance

4. Keep everything updated

5. Implement proper backup strategies

For any issues, check the troubleshooting section or refer to the official documentation of the tools used.

## Quick Reference Commands