CasperVPN Deployment Guide

📋 Table of Contents

Prerequisites

System Requirements

Minimum Requirements:
- CPU: 4 cores
- RAM: 8 GB
- Disk: 50 GB SSD
- OS: Ubuntu 20.04+ / Debian 11+ / CentOS 8+

Recommended for Production:
- CPU: 8+ cores
- RAM: 16+ GB
- Disk: 100+ GB SSD
- OS: Ubuntu 22.04 LTS

Required Software

1. **Docker**
   `bash`
   ```
      curl -fsSL https://get.docker.com -o get-docker.sh
      sudo sh get-docker.sh
      sudo usermod -aG docker $USER
   ```

2. **Docker Compose**
   `bash`
   ```
      sudo curl -L "https://github.com/docker/compose/releases/download/v2.23.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
      sudo chmod +x /usr/local/bin/docker-compose
   ```

3. **Git**
   `bash`

`sudo apt-get update`
`sudo apt-get install -y git`

4. **Optional Tools**
   ```bash
   # AWS CLI (for S3 backups)
   sudo apt-get install -y awscli

# certbot (for SSL certificates)
sudo apt-get install -y certbot
```

Network Requirements

- **Ports to Open:**
- 80 (HTTP)
- 443 (HTTPS)
- 8080 (API - optional for direct access)

- **Firewall Configuration:**
  `bash`
  `sudo ufw allow 80/tcp`
  `sudo ufw allow 443/tcp`
  `sudo ufw allow 22/tcp  # SSH`
  `sudo ufw enable`

Initial Setup

1. Clone Repository

```
# Clone the repository
git clone https://github.com/yourusername/caspervpn.git
cd caspervpn

# Or if deploying from a specific branch
git clone -b main https://github.com/yourusername/caspervpn.git
cd caspervpn
```

2. Run Setup Script

```
# Make scripts executable
chmod +x scripts/*.sh

# Run initial setup
./scripts/setup.sh
```

The setup script will:
- Check for Docker and Docker Compose
- Create `.env` file from template
- Generate self-signed SSL certificates
- Build Docker images

- Initialize database services
- Add local host entries

3. Configure Environment

Edit the `.env` file with your specific configuration:

```
nano .env
```

Critical settings to update:
- Database passwords
- Redis password
- JWT secret
- API keys
- SMTP settings
- Backup configuration

Environment Configuration

Development Environment

```
# Use development configuration
cp config/development.env .env

# Start development environment
./scripts/deploy.sh dev
```

Development Features:
- Hot reload enabled
- Debug logging
- Development tools (pgAdmin, Redis Commander)
- Non-optimized builds

Staging Environment

```
# Use staging configuration
cp config/staging.env .env

# Update secrets in .env
nano .env

# Deploy to staging
./scripts/deploy.sh staging
```

Staging Features:
- Production-like environment
- Real SSL certificates
- Performance monitoring
- Integration testing

Production Environment

```
# Use production configuration
cp config/production.env.example .env

# Update ALL secrets in .env
nano .env

# Deploy to production
./scripts/deploy.sh production
```

Production Features:
- Optimized builds
- SSL/TLS enabled
- Full monitoring
- Automated backups
- Rate limiting

Development Deployment

Quick Start

```
# 1. Setup (first time only)
./scripts/setup.sh

# 2. Deploy
./scripts/deploy.sh dev

# 3. Check status
./scripts/health-check.sh

# 4. View logs
./scripts/logs.sh all -f
```

Development Services

Access services at:
- **React Admin:** http://localhost:3000
- **API:** http://localhost:8080
- **PHP Admin:** http://localhost:9000
- **Server Agent:** http://localhost:8081
- **pgAdmin:** http://localhost:5050
- **Redis Commander:** http://localhost:8082
- **Grafana:** http://localhost:3001

Development Workflow

```
# 1. Make code changes
# 2. Changes auto-reload (hot reload enabled)

# 3. Check logs
./scripts/logs.sh api -f

# 4. Restart specific service if needed
docker-compose -f docker-compose.dev.yml restart api

# 5. Rebuild if dependencies changed
docker-compose -f docker-compose.dev.yml up -d --build api
```

Staging Deployment

Prerequisites

1. **Server Access:** SSH access to staging server
2. **DNS:** Staging domain configured
3. **SSL:** SSL certificates (Let's Encrypt recommended)

Deployment Steps

```
# 1. SSH into staging server
ssh user@staging.caspervpn.com

# 2. Navigate to application directory
cd /opt/caspervpn

# 3. Pull latest code
git pull origin develop

# 4. Update environment
nano .env

# 5. Create backup before deployment
./scripts/backup.sh

# 6. Deploy
./scripts/deploy.sh staging

# 7. Verify deployment
./scripts/health-check.sh

# 8. Run smoke tests
curl -f https://staging.caspervpn.com/health
curl -f https://staging-api.caspervpn.com/health
```

Staging Configuration

DNS Records:

```
staging.caspervpn.com        A      YOUR_SERVER_IP
staging-api.caspervpn.com    A      YOUR_SERVER_IP
staging-admin.caspervpn.com  A      YOUR_SERVER_IP
```

Nginx Configuration:
- Copy production nginx configs
- Update server names to staging domains
- Configure SSL certificates

Production Deployment

Pre-Deployment Checklist

- [ ] All tests passing in staging
- [ ] Database backup completed
- [ ] SSL certificates valid
- [ ] Environment variables verified
- [ ] Monitoring alerts configured
- [ ] Rollback plan documented
- [ ] Team notified of deployment
- [ ] Maintenance window scheduled (if needed)

Zero-Downtime Deployment

```
# 1. SSH into production server
ssh user@caspervpn.com

# 2. Navigate to application directory
cd /opt/caspervpn

# 3. Create backup
./scripts/backup.sh

# 4. Pull latest code
git pull origin main

# 5. Build new images
docker-compose build --parallel

# 6. Deploy with rolling update
docker-compose up -d --no-deps --build api
docker-compose up -d --no-deps --build admin-react
docker-compose up -d --no-deps --build admin-php
docker-compose up -d --no-deps --build server-agent

# 7. Verify each service
./scripts/health-check.sh

# 8. Update nginx if needed
docker-compose up -d nginx
```

Blue-Green Deployment

```
# 1. Deploy to "green" environment
docker-compose -p caspervpn-green up -d

# 2. Run smoke tests on green
curl -f http://localhost:8081/health

# 3. Switch traffic to green (update nginx config)
# 4. Monitor for issues
# 5. If successful, remove blue environment
docker-compose -p caspervpn-blue down
```

Production Configuration

DNS Records:

```
caspervpn.com           A      YOUR_SERVER_IP
api.caspervpn.com       A      YOUR_SERVER_IP
admin.caspervpn.com     A      YOUR_SERVER_IP
agent.caspervpn.com     A      YOUR_SERVER_IP
grafana.caspervpn.com   A      YOUR_SERVER_IP
```

SSL/TLS Configuration

Let’s Encrypt (Recommended)

```
# 1. Install certbot
sudo apt-get install certbot python3-certbot-nginx

# 2. Obtain certificates
sudo certbot --nginx -d caspervpn.com -d www.caspervpn.com \
  -d api.caspervpn.com -d admin.caspervpn.com

# 3. Copy certificates to nginx directory
sudo cp /etc/letsencrypt/live/caspervpn.com/fullchain.pem nginx/ssl/caspervpn.crt
sudo cp /etc/letsencrypt/live/caspervpn.com/privkey.pem nginx/ssl/caspervpn.key

# 4. Update nginx configuration
# Uncomment HTTPS server blocks in nginx/sites/*.conf

# 5. Restart nginx
docker-compose restart nginx

# 6. Setup auto-renewal
sudo certbot renew --dry-run
```

Custom SSL Certificates

```
# 1. Copy your certificates
cp your-certificate.crt nginx/ssl/caspervpn.crt
cp your-private-key.key nginx/ssl/caspervpn.key

# 2. Set proper permissions
chmod 600 nginx/ssl/*.key

# 3. Update nginx configuration
# 4. Restart nginx
docker-compose restart nginx
```

Database Migration

Initial Migration

```
# 1. Ensure database is running
docker-compose up -d postgres

# 2. Run migrations (example for .NET)
docker-compose exec api dotnet ef database update

# 3. Seed initial data (if needed)
docker-compose exec api dotnet run --seed-data
```

Migration During Deployment

```
# 1. Create backup BEFORE migration
./scripts/backup.sh

# 2. Stop services (except database)
docker-compose stop api admin-php

# 3. Run migrations
docker-compose exec postgres psql -U casperuser -d caspervpn -f /path/to/migration.sql

# 4. Verify migration
docker-compose exec postgres psql -U casperuser -d caspervpn -c "SELECT version FROM schema_migrations;"

# 5. Start services
docker-compose start api admin-php
```

Rollback Procedures

Quick Rollback

```
# 1. Stop current services
docker-compose down

# 2. Checkout previous version
git log --oneline  # Find previous commit
git checkout <previous-commit>

# 3. Restore from backup
./scripts/restore.sh backups/YYYYMMDD_HHMMSS.tar.gz

# 4. Deploy previous version
./scripts/deploy.sh production

# 5. Verify
./scripts/health-check.sh
```

Database Rollback

```
# 1. Stop services
docker-compose stop api admin-php

# 2. Restore database from backup
./scripts/restore.sh backups/YYYYMMDD_HHMMSS.tar.gz

# 3. Restart services
./scripts/deploy.sh production
```

Docker Image Rollback

```
# 1. List previous images
docker images | grep caspervpn

# 2. Tag previous image as latest
docker tag caspervpn-api:previous caspervpn-api:latest

# 3. Restart service with previous image
docker-compose up -d --no-deps api
```

Post-Deployment Verification

Health Checks

```
# 1. Run automated health check
./scripts/health-check.sh

# 2. Check individual services
curl -f https://api.caspervpn.com/health
curl -f https://admin.caspervpn.com/health
curl -f https://agent.caspervpn.com/health

# 3. Check logs for errors
./scripts/logs.sh all --tail=100
```

Smoke Tests

```
# API Tests
curl -X GET https://api.caspervpn.com/api/vpn/servers
curl -X GET https://api.caspervpn.com/api/vpn/status

# Admin Panel Tests
curl -f https://admin.caspervpn.com/
curl -f https://legacy.caspervpn.com/

# Monitoring Tests
curl -f https://grafana.caspervpn.com:3001/api/health
curl -f http://localhost:9090/-/healthy  # Prometheus
```

Performance Checks

```
# Response time
time curl https://api.caspervpn.com/health

# Load test (using Apache Bench)
ab -n 1000 -c 10 https://api.caspervpn.com/api/vpn/servers

# Monitor resource usage
docker stats --no-stream
```

Database Verification

```
# Check connections
docker-compose exec postgres psql -U casperuser -d caspervpn -c "SELECT count(*) FROM
pg_stat_activity;"

# Check table counts
docker-compose exec postgres psql -U casperuser -d caspervpn -c "SELECT schemaname,
tablename, n_live_tup FROM pg_stat_user_tables;"

# Verify migrations
docker-compose exec postgres psql -U casperuser -d caspervpn -c "SELECT version FROM
schema_migrations ORDER BY version DESC LIMIT 5;"
```

Monitoring Post-Deployment

Grafana Dashboards

1. Access Grafana: https://grafana.caspervpn.com:3001
2. Login with admin credentials
3. Check “CasperVPN Overview” dashboard
4. Monitor for 15-30 minutes post-deployment

Key Metrics to Watch

- **Response Time:** Should be < 500ms
- **Error Rate:** Should be < 1%
- **CPU Usage:** Should be < 70%
- **Memory Usage:** Should be < 80%
- **Database Connections:** Should be stable

Alert Configuration

Ensure alerts are enabled for:
- Service downtime
- High error rates
- High resource usage
- Database connection issues
- SSL certificate expiration

Troubleshooting Deployment Issues

Services Won’t Start

```
# Check logs
./scripts/logs.sh <service-name>

# Check Docker status
docker-compose ps

# Rebuild service
docker-compose up -d --build <service-name>
```

Database Connection Errors

```
# Check database is running
docker-compose ps postgres

# Test connection
docker-compose exec postgres pg_isready

# Check credentials in .env
cat .env | grep DB_
```

SSL Certificate Issues

```
# Verify certificates exist
ls -la nginx/ssl/

# Check certificate validity
openssl x509 -in nginx/ssl/caspervpn.crt -text -noout

# Test SSL configuration
openssl s_client -connect caspervpn.com:443
```

Automated Deployment (CI/CD)

GitHub Actions

Deployments are automated via GitHub Actions when:
- **Staging:** Push to `develop` branch
- **Production:** Push to `main` branch (with manual approval)

Manual Trigger

```
# Trigger deployment via GitHub UI
# Go to Actions → CI/CD Pipeline → Run workflow
# Select environment and confirm
```

Best Practices

1. **Always create backups before deployment**
2. **Deploy during low-traffic periods**
3. **Monitor for at least 30 minutes post-deployment**
4. **Keep rollback plan ready**
5. **Test in staging before production**
6. **Document any manual steps**
7. **Notify team before and after deployment**
8. **Update documentation if process changes**

Support

For deployment issues:
- Check Troubleshooting Guide (./TROUBLESHOOTING.md)
- View logs: `./scripts/logs.sh <service>`
- Contact DevOps team
- Create GitHub issue with logs

Last Updated: December 5, 2025
Version: 1.0.0