CyberSwarm Dashboard Deployment Guide

Table of Contents

Deployment Overview

This guide covers multiple deployment strategies for the CyberSwarm Dashboard, from simple single-server deployments to complex multi-region setups.

Deployment Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                         Internet                            │
└─────────────────────────────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────┐
│                  Load Balancer / CDN                        │
│                (Cloudflare, AWS ALB)                        │
└─────────────────────────────────────────────────────────────┘
                         │
        ┌────────────────┼────────────────┐
        │                │                │
        ▼                ▼                ▼
┌─────────────┐  ┌─────────────┐  ┌─────────────┐
│  Dashboard  │  │  Dashboard  │  │  Dashboard  │
│  Instance 1 │  │  Instance 2 │  │  Instance 3 │
│  (Next.js)  │  │  (Next.js)  │  │  (Next.js)  │
└─────────────┘  └─────────────┘  └─────────────┘
      │                │                │
      └────────────────┼────────────────┘
                       │
                       ▼
┌─────────────────────────────────────────────────────────────┐
│           Backend Orchestrator (Separate)                   │
│              Multi-Agent Docker System                      │
└─────────────────────────────────────────────────────────────┘
```

Prerequisites

System Requirements

Minimum Production Requirements:
- **CPU**: 2 cores
- **RAM**: 4 GB
- **Storage**: 20 GB SSD
- **Network**: 100 Mbps
- **OS**: Ubuntu 20.04+ / Debian 11+ / RHEL 8+

Recommended Production Requirements:
- **CPU**: 4+ cores
- **RAM**: 8+ GB
- **Storage**: 50+ GB SSD
- **Network**: 1 Gbps
- **OS**: Ubuntu 22.04 LTS

Software Requirements

- **Node.js**: 18.x or 20.x LTS
- **npm**: 9.x+ or **yarn**: 1.22+
- **Git**: 2.x+
- **PM2**: 5.x+ (for process management)
- **Nginx**: 1.18+ (for reverse proxy)
- **SSL Certificate**: Let's Encrypt or commercial

Optional Requirements

- **Docker**: 24.x+ (for containerized deployment)
- **Docker Compose**: 2.x+
- **Redis**: 7.x+ (for caching)
- **PostgreSQL**: 15.x+ (if using database)

Environment Configuration

Production Environment Variables

Create a `.env.production` file:

```
# Application
NODE_ENV=production
NEXT_PUBLIC_APP_URL=https://dashboard.yourdomain.com

# Backend API
NEXT_PUBLIC_API_URL=https://api.yourdomain.com
NEXT_PUBLIC_WS_URL=wss://api.yourdomain.com

# Database (if using Prisma)
DATABASE_URL="postgresql://user:password@localhost:5432/cyberswarm?schema=public"

# Authentication
NEXTAUTH_URL=https://dashboard.yourdomain.com
NEXTAUTH_SECRET=your-super-secret-key-min-32-chars

# Security
API_SECRET_KEY=your-api-secret-key
ALLOWED_ORIGINS=https://dashboard.yourdomain.com

# Performance
NEXT_PUBLIC_ENABLE_ANALYTICS=true
NEXT_PUBLIC_ENABLE_DEBUG=false

# Monitoring
SENTRY_DSN=your-sentry-dsn
LOG_LEVEL=info

# Redis (optional)
REDIS_URL=redis://localhost:6379

# Rate Limiting
RATE_LIMIT_MAX=100
RATE_LIMIT_WINDOW=60000
```

Security Best Practices

1. **Never commit `.env` files to version control**
2. **Use strong, randomly generated secrets**
3. **Rotate secrets regularly**
4. **Use environment-specific configurations**
5. **Implement proper access controls**

Generating Secure Secrets

```
# Generate NEXTAUTH_SECRET
openssl rand -base64 32

# Generate API_SECRET_KEY
openssl rand -hex 32

# Generate JWT secret
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

Deployment Options

Option 1: Vercel (Recommended for Quick Deployment)

Advantages:
- Zero configuration
- Automatic SSL
- Global CDN
- Automatic scaling
- Built-in analytics

Steps:

1. **Push code to GitHub**:

```
git add .
git commit -m "Prepare for deployment"
git push origin main
```

1. **Import to Vercel**:
   - Visit vercel.com (https://vercel.com)
   - Click "Import Project"
   - Select your GitHub repository
   - Configure settings:

     ◦ Framework Preset: Next.js
     ◦ Root Directory: `app`
     ◦ Build Command: `npm run build`
     ◦ Output Directory: `.next`
2. **Configure Environment Variables**:
   - Go to Project Settings → Environment Variables
   - Add all variables from `.env.production`
   - Separate variables for Production, Preview, and Development

3. **Deploy**:
   - Click "Deploy"
   - Wait for build to complete
   - Access your dashboard at the provided URL

4. **Custom Domain** (Optional):
   - Go to Project Settings → Domains
   - Add your custom domain
   - Configure DNS records as instructed

Vercel Configuration (`vercel.json`):

```json
{
  "version": 2,
  "builds": [
    {
      "src": "app/package.json",
      "use": "@vercel/next"
    }
  ],
  "routes": [
    {
      "src": "/api/simulation/stream",
      "headers": {
        "Cache-Control": "no-cache",
        "Connection": "keep-alive"
      }
    }
  ],
  "env": {
    "NODE_ENV": "production"
  }
}
```

Option 2: Docker Deployment

Advantages:
- Consistent environment
- Easy scaling
- Portable
- Isolated dependencies

Dockerfile:

```dockerfile
# app/Dockerfile
FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Copy package files
COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* ./
RUN \
  if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
  elif [ -f package-lock.json ]; then npm ci; \
  elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i --frozen-lockfile; \
  else echo "Lockfile not found." && exit 1; \
  fi

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Set environment variables for build
ENV NEXT_TELEMETRY_DISABLED 1
ENV NODE_ENV production

# Build application
RUN npm run build

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production
ENV NEXT_TELEMETRY_DISABLED 1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

# Copy built application
COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000
ENV HOSTNAME "0.0.0.0"

CMD ["node", "server.js"]
```

Docker Compose (`docker-compose.yml`):

```yaml
version: '3.8'

services:
  dashboard:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
      - NEXT_PUBLIC_WS_URL=${NEXT_PUBLIC_WS_URL}
      - NEXTAUTH_URL=${NEXTAUTH_URL}
      - NEXTAUTH_SECRET=${NEXTAUTH_SECRET}
      - DATABASE_URL=${DATABASE_URL}
    env_file:
      - .env.production
    restart: unless-stopped
    networks:
      - cyberswarm
    depends_on:
      - redis
      - postgres

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    restart: unless-stopped
    networks:
      - cyberswarm

  postgres:
    image: postgres:15-alpine
    environment:
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
      - POSTGRES_DB=${DB_NAME}
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
    networks:
      - cyberswarm

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
    depends_on:
      - dashboard
    restart: unless-stopped
    networks:
      - cyberswarm
```

```yaml
networks:
  cyberswarm:
    driver: bridge

volumes:
  redis_data:
  postgres_data:
```

Deploy with Docker Compose:

```bash
# Build and start services
docker-compose up -d

# View logs
docker-compose logs -f dashboard

# Stop services
docker-compose down

# Rebuild after changes
docker-compose up -d --build
```

Option 3: Traditional VPS Deployment

Advantages:
- Full control
- Cost-effective
- Flexible configuration

Steps:

1. **Prepare Server**:

```bash
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs

# Install PM2
sudo npm install -g pm2

# Install Nginx
sudo apt install -y nginx

# Install certbot for SSL
sudo apt install -y certbot python3-certbot-nginx
```

1. **Clone and Build Application**:

```
# Clone repository
git clone https://github.com/starwrecknts/cyberswarm.git
cd cyberswarm/app

# Install dependencies
npm install

# Create production environment file
cp .env.example .env.production
nano .env.production  # Edit with your values

# Build application
npm run build
```

1. **Configure PM2**:

Create `ecosystem.config.js`:

```
module.exports = {
  apps: [{
    name: 'cyberswarm-dashboard',
    script: 'npm',
    args: 'start',
    cwd: '/path/to/cyberswarm/app',
    instances: 'max',
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3000
    },
    error_file: './logs/err.log',
    out_file: './logs/out.log',
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z',
    merge_logs: true,
    autorestart: true,
    watch: false,
    max_memory_restart: '1G'
  }]
};
```

Start with PM2:

```
# Start application
pm2 start ecosystem.config.js

# Save PM2 configuration
pm2 save

# Setup PM2 to start on boot
pm2 startup
sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u $USER --hp $HOME

# Monitor
pm2 monit
```

1. **Configure Nginx**:

Create `/etc/nginx/sites-available/cyberswarm`:

```
# Upstream for Next.js application
upstream nextjs_upstream {
    server 127.0.0.1:3000;
    keepalive 64;
}

# HTTP to HTTPS redirect
server {
    listen 80;
    listen [::]:80;
    server_name dashboard.yourdomain.com;

    return 301 https://$server_name$request_uri;
}

# HTTPS server
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name dashboard.yourdomain.com;

    # SSL configuration
    ssl_certificate /etc/letsencrypt/live/dashboard.yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dashboard.yourdomain.com/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # Security headers
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Logging
    access_log /var/log/nginx/cyberswarm_access.log;
    error_log /var/log/nginx/cyberswarm_error.log;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml text/javascript application/json applica-
tion/javascript application/xml+rss;

    # Root location
    location / {
        proxy_pass http://nextjs_upstream;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        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;
        proxy_cache_bypass $http_upgrade;
        proxy_read_timeout 86400;
    }
    # SSE endpoint - special configuration
    location /api/simulation/stream {
        proxy_pass http://nextjs_upstream;
        proxy_http_version 1.1;
        proxy_set_header Connection '';
        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;

        # SSE specific
        proxy_buffering off;
        proxy_cache off;
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;
        chunked_transfer_encoding off;
    }

    # Static files caching
    location /_next/static {
        proxy_pass http://nextjs_upstream;
        proxy_cache_valid 200 60m;
        add_header Cache-Control "public, max-age=3600, immutable";
    }

    # Public files
    location /public {
        proxy_pass http://nextjs_upstream;
        proxy_cache_valid 200 60m;
        add_header Cache-Control "public, max-age=3600";
    }
}
```

Enable site and restart Nginx:

```
# Enable site
sudo ln -s /etc/nginx/sites-available/cyberswarm /etc/nginx/sites-enabled/

# Test configuration
sudo nginx -t

# Restart Nginx
sudo systemctl restart nginx
```

1. **Setup SSL with Let’s Encrypt**:

```
# Obtain certificate
sudo certbot --nginx -d dashboard.yourdomain.com

# Test auto-renewal
sudo certbot renew --dry-run
```

Option 4: AWS Deployment

Architecture:
- **EC2**: Application servers
- **ALB**: Load balancing
- **RDS**: Database (if needed)

- **ElastiCache**: Redis caching
- **CloudFront**: CDN
- **Route 53**: DNS
- **S3**: Static assets

Steps:

1. **Launch EC2 Instance**:
   - AMI: Ubuntu 22.04 LTS
   - Instance Type: t3.medium (minimum)
   - Security Group: Allow 80, 443, 22
   - Storage: 30 GB gp3

2. **Configure Application** (follow VPS steps above)

3. **Setup Application Load Balancer**:
   - Create target group
   - Register EC2 instances
   - Configure health checks
   - Setup SSL certificate

4. **Configure CloudFront**:
   - Origin: ALB
   - Cache behaviors for static assets
   - SSL certificate
   - Custom domain

5. **Setup Auto Scaling** (optional):
   - Create launch template
   - Configure auto scaling group
   - Set scaling policies

Production Checklist

Pre-Deployment

- [ ] All environment variables configured
- [ ] Database migrations completed
- [ ] SSL certificates obtained
- [ ] DNS records configured
- [ ] Firewall rules configured
- [ ] Backup strategy implemented
- [ ] Monitoring tools configured
- [ ] Load testing completed
- [ ] Security audit performed
- [ ] Documentation updated

Security Checklist

- [ ] HTTPS enabled everywhere
- [ ] Strong secrets generated
- [ ] Rate limiting implemented

- [ ] CORS configured properly
- [ ] Input validation enabled
- [ ] SQL injection prevention
- [ ] XSS protection enabled
- [ ] CSRF protection enabled
- [ ] Security headers configured
- [ ] Regular security updates scheduled

Performance Checklist

- [ ] Gzip compression enabled
- [ ] Static assets cached
- [ ] Database queries optimized
- [ ] CDN configured
- [ ] Image optimization enabled
- [ ] Code splitting implemented
- [ ] Lazy loading configured
- [ ] Bundle size optimized
- [ ] Server-side caching enabled
- [ ] Database connection pooling

Monitoring and Maintenance

Application Monitoring

PM2 Monitoring:

```
# View status
pm2 status

# View logs
pm2 logs cyberswarm-dashboard

# Monitor resources
pm2 monit

# Restart application
pm2 restart cyberswarm-dashboard

# Reload without downtime
pm2 reload cyberswarm-dashboard
```

Log Management:

```
# Setup log rotation
pm2 install pm2-logrotate

# Configure rotation
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 30
pm2 set pm2-logrotate:compress true
```

System Monitoring

Install monitoring tools:

```
# Install htop
sudo apt install htop

# Install netdata (comprehensive monitoring)
bash <(curl -Ss https://my-netdata.io/kickstart.sh)
```

Health Checks

Create health check endpoint:

```
// app/api/health/route.ts
export async function GET() {
  const health = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage(),
    version: process.env.npm_package_version
  };

  return Response.json(health);
}
```

Backup Strategy

Database Backups:

```
# PostgreSQL backup script
#!/bin/bash
BACKUP_DIR="/backups/postgres"
DATE=$(date +%Y%m%d_%H%M%S)
FILENAME="cyberswarm_$DATE.sql"

pg_dump -U postgres cyberswarm > "$BACKUP_DIR/$FILENAME"
gzip "$BACKUP_DIR/$FILENAME"

# Keep only last 30 days
find $BACKUP_DIR -name "*.gz" -mtime +30 -delete
```

Application Backups:

```
# Backup script
#!/bin/bash
BACKUP_DIR="/backups/app"
DATE=$(date +%Y%m%d_%H%M%S)
APP_DIR="/path/to/cyberswarm"

tar -czf "$BACKUP_DIR/app_$DATE.tar.gz" \
  --exclude="node_modules" \
  --exclude=".next" \
  "$APP_DIR"

# Keep only last 7 days
find $BACKUP_DIR -name "*.tar.gz" -mtime +7 -delete
```

Troubleshooting

Common Issues

Issue: Application won't start

```
# Check logs
pm2 logs cyberswarm-dashboard --lines 100

# Check port availability
sudo netstat -tulpn | grep 3000

# Check environment variables
pm2 env 0
```

Issue: High memory usage

```
# Check memory
pm2 monit

# Restart with memory limit
pm2 restart cyberswarm-dashboard --max-memory-restart 1G
```

Issue: SSL certificate errors

```
# Renew certificate
sudo certbot renew

# Check certificate expiry
sudo certbot certificates
```

Issue: Database connection errors

```
# Check database status
sudo systemctl status postgresql

# Test connection
psql -U postgres -d cyberswarm -c "SELECT 1"
```

Scaling Strategies

Vertical Scaling

Upgrade server resources:
- Increase CPU cores
- Add more RAM
- Upgrade to SSD storage
- Increase network bandwidth

Horizontal Scaling

Add more application instances:

1. **Setup Load Balancer**
2. **Deploy multiple instances**
3. **Configure session sharing** (Redis)
4. **Implement sticky sessions** (if needed)

Database Scaling

- **Read Replicas**: For read-heavy workloads
- **Connection Pooling**: Optimize connections
- **Query Optimization**: Index frequently queried fields
- **Caching**: Redis for frequently accessed data

CDN Integration

Use CloudFlare or AWS CloudFront:
- Cache static assets
- Reduce server load
- Improve global performance
- DDoS protection

Deployment Support: For deployment assistance, consult the main README.md or open an issue on GitHub.