SOaC Framework - Deployment Guide

Production Deployment Checklist

Pre-Deployment Security

- [ ] Change default admin password
- [ ] Generate strong SECRET_KEY for JWT
- [ ] Update database credentials
- [ ] Enable HTTPS/SSL
- [ ] Configure CORS for production domain
- [ ] Set up secrets management (AWS Secrets Manager, Azure Key Vault, etc.)
- [ ] Review and update firewall rules
- [ ] Enable database encryption at rest
- [ ] Set up database backups
- [ ] Configure rate limiting
- [ ] Set up monitoring and alerting

Infrastructure Setup

Option 1: Cloud Deployment (Recommended)

AWS Deployment Example:

1. **Database**: Use AWS RDS PostgreSQL
```bash
   # Create RDS instance
   aws rds create-db-instance \
     --db-instance-identifier soac-postgres \
     --db-instance-class db.t3.medium \
     --engine postgres \
     --master-username soac_admin \
     --master-user-password <STRONG_PASSWORD> \
     --allocated-storage 20 \
     --vpc-security-group-ids sg-xxx \
     --db-subnet-group-name soac-subnet-group
```

2. **Backend**: Deploy to AWS ECS or EC2
```bash
   # Build and push Docker image
   docker build -t soac-backend:latest ./backend
   docker tag soac-backend:latest <ECR_REPO>:latest
   docker push <ECR_REPO>:latest
```

3. **Frontend**: Deploy to S3 + CloudFront
```bash
   cd frontend
   npm run build
   aws s3 sync dist/ s3://soac-frontend-bucket/
   aws cloudfront create-invalidation --distribution-id <ID> --paths "/*"
```

Azure Deployment Example:

1. **Database**: Azure Database for PostgreSQL
2. **Backend**: Azure App Service or Container Instances
3. **Frontend**: Azure Static Web Apps or Blob Storage + CDN

GCP Deployment Example:

1. **Database**: Cloud SQL for PostgreSQL
2. **Backend**: Cloud Run or GKE
3. **Frontend**: Firebase Hosting or Cloud Storage + CDN

Option 2: On-Premises/VM Deployment

1. Server Setup (Ubuntu 22.04 LTS)

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

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

# Install Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Install nginx
sudo apt-get install nginx -y

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

2. Clone Repository

```
cd /opt
sudo git clone <your-repo-url> soac-framework
cd soac-framework
```

3. Configure Environment

```
# Backend configuration
cd backend
cp .env.example .env
nano .env
```

Update `.env`:

```
DATABASE_URL=postgresql://soac_user:STRONG_PASSWORD@postgres:5432/soac_db
SECRET_KEY=<GENERATE_STRONG_SECRET_KEY>
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30
FRONTEND_URL=https://soac.yourdomain.com
ENVIRONMENT=production
```

```
# Frontend configuration
cd ../frontend
cp .env.example .env
nano .env
```

Update `.env`:

```
VITE_API_BASE_URL=https://api.soac.yourdomain.com
```

4. Set Up SSL Certificate

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

5. Configure nginx

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

```
# Frontend
server {
    listen 80;
    server_name soac.yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name soac.yourdomain.com;

    ssl_certificate /etc/letsencrypt/live/soac.yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/soac.yourdomain.com/privkey.pem;

    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;
        proxy_cache_bypass $http_upgrade;
    }
}

# Backend API
server {
    listen 80;
    server_name api.soac.yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name api.soac.yourdomain.com;

    ssl_certificate /etc/letsencrypt/live/api.soac.yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.soac.yourdomain.com/privkey.pem;

    location / {
        proxy_pass http://localhost:8000;
        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;
    }
}
```

Enable the site:

```
sudo ln -s /etc/nginx/sites-available/soac /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

6. Production Docker Compose

Create `docker-compose.prod.yml`:

```yaml
version: '3.8'

services:
  postgres:
    image: postgres:15-alpine
    container_name: soac-postgres
    restart: always
    environment:
      POSTGRES_USER: soac_user
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: soac_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - soac-network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U soac_user -d soac_db"]
      interval: 10s
      timeout: 5s
      retries: 5

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile.prod
    container_name: soac-backend
    restart: always
    ports:
      - "8000:8000"
    environment:
      DATABASE_URL: postgresql://soac_user:${DB_PASSWORD}@postgres:5432/soac_db
      SECRET_KEY: ${SECRET_KEY}
      ALGORITHM: HS256
      ACCESS_TOKEN_EXPIRE_MINUTES: 30
      FRONTEND_URL: https://soac.yourdomain.com
      ENVIRONMENT: production
    depends_on:
      postgres:
        condition: service_healthy
    networks:
      - soac-network
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile.prod
      args:
        VITE_API_BASE_URL: https://api.soac.yourdomain.com
    container_name: soac-frontend
    restart: always
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
      - soac-network
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"

volumes:
  postgres_data:

networks:
  soac-network:
    driver: bridge
```

7. Create Production Dockerfiles

`backend/Dockerfile.prod`:

```
FROM python:3.11-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    gcc \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["gunicorn", "app.main:app", "-w", "4", "-k", "uvicorn.workers.UvicornWorker", "-b", "0.0.0.0:8000", "--access-logfile", "-", "--error-logfile", "-"]
```

`frontend/Dockerfile.prod`:

```
FROM node:18-alpine as build

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .
ARG VITE_API_BASE_URL
ENV VITE_API_BASE_URL=$VITE_API_BASE_URL

RUN npm run build

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

EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

8. Start Production Services

```
# Set environment variables
export DB_PASSWORD="<STRONG_DB_PASSWORD>"
export SECRET_KEY="<STRONG_SECRET_KEY>"

# Start services
docker compose -f docker-compose.prod.yml up -d

# Check logs
docker compose -f docker-compose.prod.yml logs -f
```

Database Backup

Automated Backup Script (`scripts/backup-db.sh`):

```
#!/bin/bash
BACKUP_DIR="/opt/backups/soac"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/soac_backup_$DATE.sql"

mkdir -p $BACKUP_DIR

docker compose exec -T postgres pg_dump -U soac_user soac_db > $BACKUP_FILE

gzip $BACKUP_FILE

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

echo "Backup completed: $BACKUP_FILE.gz"
```

Add to crontab:

```
# Daily backup at 2 AM
0 2 * * * /opt/soac-framework/scripts/backup-db.sh
```

Monitoring Setup

1. Application Monitoring

Create `docker-compose.monitoring.yml` :

```yaml
version: '3.8'

services:
  prometheus:
    image: prom/prometheus
    container_name: soac-prometheus
    volumes:
      - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus_data:/prometheus
    ports:
      - "9090:9090"
    networks:
      - soac-network

  grafana:
    image: grafana/grafana
    container_name: soac-grafana
    ports:
      - "3001:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
    networks:
      - soac-network

volumes:
  prometheus_data:
  grafana_data:

networks:
  soac-network:
    external: true
```

2. Log Aggregation

Consider using:
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Loki + Grafana
- Cloud provider’s logging service (CloudWatch, Azure Monitor, Stackdriver)

Health Checks

Create `/etc/cron.d/soac-health-check`:

```
*/5 * * * * curl -f http://localhost:8000/health || systemctl restart docker
```

Security Hardening

1. Firewall Configuration

```
# Allow only necessary ports
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp  # SSH
sudo ufw allow 80/tcp  # HTTP
sudo ufw allow 443/tcp # HTTPS
sudo ufw enable
```

2. Fail2Ban

```
sudo apt-get install fail2ban -y
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

3. Database Security

- Use strong passwords
- Enable SSL for database connections
- Restrict database access to application server only
- Regular security updates
- Enable audit logging

Performance Optimization

1. Backend

- Use Gunicorn with multiple workers
- Enable connection pooling
- Implement caching (Redis)
- Use CDN for static assets

2. Frontend

- Build for production (minification, tree-shaking)
- Enable gzip compression in nginx
- Set proper cache headers
- Use CDN

3. Database

- Regular vacuuming and analysis
- Create indexes on frequently queried columns
- Monitor query performance
- Set up read replicas for scaling

Rollback Procedure

```
# Stop current deployment
docker compose down

# Restore from backup
gunzip /opt/backups/soac/soac_backup_YYYYMMDD_HHMMSS.sql.gz
docker compose exec -T postgres psql -U soac_user soac_db < soac_backup_YYYYMMDD_HHMMS
S.sql

# Deploy previous version
git checkout <PREVIOUS_TAG>
docker compose up --build -d
```

Post-Deployment Verification

1. **Health Check**: Verify all services are running
   ```
   bash
      docker compose ps
      curl https://api.soac.yourdomain.com/health
   ```

2. **Login Test**: Verify authentication works

3. **Device Test**: Test device connections
4. **API Test**: Test all critical endpoints
5. **Frontend Test**: Verify all pages load correctly
6. **Database Test**: Check data integrity
7. **Monitoring**: Verify monitoring is working
8. **Backup Test**: Verify backups are working

Maintenance

Regular Tasks

- **Daily**: Monitor logs and metrics
- **Weekly**: Review security alerts
- **Monthly**: Apply security patches
- **Quarterly**: Review and update dependencies
- **Annually**: Security audit

Scaling

Horizontal Scaling:
- Deploy multiple backend instances behind a load balancer
- Use database read replicas
- Implement Redis for caching and session storage

Vertical Scaling:
- Increase server resources (CPU, RAM)
- Optimize database queries
- Enable query caching

Support

For questions or issues, contact the SOaC Framework Team.