

MANUAL 8: GUÍA DE DESPLIEGUE Y DEVOPS - SISTEMA CONA

INFORMACIÓN DEL DOCUMENTO

Fecha de Creación: 21 de Julio de 2025

Proyecto: Sistema CONA (Gestión CONAVEG)

Audiencia: DevOps, Ingenieros de Infraestructura, Administradores de Sistema

Nivel: Avanzado

Tiempo Estimado: 4-8 horas (configuración completa)

Última Actualización: 21 de Julio de 2025

OBJETIVOS DE APRENDIZAJE

Al finalizar este manual, serás capaz de:

- ☒ Entender las diferentes estrategias de despliegue para el Sistema CONA.
 - ☒ Containerizar la aplicación utilizando Docker para un despliegue consistente.
 - ☒ Configurar un pipeline básico de CI/CD para automatizar la compilación y el testing.
 - ☒ Preparar un servidor de producción y gestionar la base de datos.
 - ☒ Implementar procedimientos de backup, recuperación ante desastres y alta disponibilidad.
 - ☒ Aplicar estrategias de escalado y gestionar migraciones de datos.
-

ESTRATEGIAS DE DESPLIEGUE

Existen varias formas de desplegar el Sistema CONA, dependiendo del entorno y los requisitos.

1. Despliegue Tradicional (Bare Metal / VM)

- Descripción: Instalar la aplicación y sus dependencias directamente en un servidor físico o virtual.
- Ideal para: Entornos de desarrollo, staging o producciones de pequeña escala.
- Proceso:
 1. Configurar el servidor con Java, Maven y MariaDB.
 2. Clonar el repositorio.
 3. Construir el artefacto JAR: `mvn clean package`.
 4. Ejecutar la aplicación como un servicio del sistema (ej. systemd).

2. Despliegue con Contenedores (Docker)

- Descripción: Empaquetar la aplicación y sus dependencias en una imagen de Docker.
- Ideal para: Entornos de producción, ya que garantiza consistencia y portabilidad.
- Proceso:
 1. Crear un **Dockerfile** para la aplicación.
 2. Construir la imagen de Docker.
 3. Ejecutar el contenedor, vinculándolo a un contenedor de base de datos.



CONTAINERIZACIÓN CON DOCKER

Containerizar la aplicación es el método recomendado para producción.

Paso 1: Crear el **Dockerfile**

Creas un archivo llamado **Dockerfile** en la raíz del proyecto:

```
# Fase 1: Construcción con Maven
FROM maven: 3.8.5-openjdk-17 AS build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests

# Fase 2: Creación de la imagen final
FROM openjdk: 17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Este es un *multi-stage build* que crea una imagen final ligera sin las dependencias de Maven.

Paso 2: Crear el archivo **docker-compose.yml**

Este archivo orquestará la aplicación y la base de datos.

```
version: '3.8'

services:
  cona-db:
    image: mariadb:10.6
    container_name: cona-db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: your_strong_root_password
      MYSQL_DATABASE: conaveg_db
      MYSQL_USER: cona_user
      MYSQL_PASSWORD: your_strong_user_password
    volumes:
      - db_data: /var/lib/mysql
```

```

ports:
  - "3306: 3306"

cona-app:
  build: .
  container_name: cona-app
  restart: unless-stopped
  depends_on:
    - cona-db
  ports:
    - "8080: 8080"
  environment:
    - SPRING_PROFILES_ACTIVE=prod
    - SPRING_DATASOURCE_URL=jdbc:mariadb://cona-db:3306/conaveg_db
    - SPRING_DATASOURCE_USERNAME=cona_user
    - SPRING_DATASOURCE_PASSWORD=your_strong_user_password
    - APP_JWT_SECRET=your_super_secret_jwt_key_that_is_very_long

volumes:
  db_data:

```

Paso 3: Levantar el Entorno

Desde la raíz del proyecto, ejecuta:

```
docker-compose up --build -d
```

- **--build**: Fuerza la reconstrucción de la imagen de la aplicación.
- **-d**: Ejecuta los contenedores en segundo plano (detached mode).

CI/CD PIPELINES Y AUTOMATIZACIÓN

Un pipeline de Integración Continua / Despliegue Continuo (CI/CD) automatiza el proceso de testing y despliegue.

Ejemplo de Pipeline con GitHub Actions

Crea el archivo `.github/workflows/ci-cd.yml`:

```

name: CONA CI /CD Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

```

```

- name: Set up JDK 17
  uses: actions/setup-java@v3
  with:
    java-version: '17'
    distribution: 'temurin'

- name: Build with Maven
  run: mvn -B package --file pom.xml

- name: Run Unit & Integration Tests
  run: mvn test

deploy-to-production:
  needs: build-and-test
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${ secrets.DOCKER_HUB_USERNAME }
        password: ${ secrets.DOCKER_HUB_ACCESS_TOKEN }

    - name: Build and push Docker image
      uses: docker/build-push-action@v4
      with:
        context: .
        push: true
        tags: yourdockerhubuser/cona-app:latest

    - name: Deploy to Production Server
      uses: appleboy/ssh-action@master
      with:
        host: ${ secrets.PROD_SERVER_HOST }
        username: ${ secrets.PROD_SERVER_USERNAME }
        key: ${ secrets.PROD_SERVER_SSH_KEY }
        script: |
          cd /opt/cona
          docker-compose pull
          docker-compose up -d
          echo "Deployment successful!"

```

Nota: Este pipeline requiere que configures "Secrets" en tu repositorio de GitHub para las credenciales.



❑ CONFIGURACIÓN DE SERVIDORES Y BASE DE DATOS

Preparación del Servidor de Producción:

1. Instalar Docker y Docker Compose: Sigue la documentación oficial para tu distribución de Linux.

2. Crear Directorio de la Aplicación:

```
sudo mkdir -p /opt/cona
sudo chown $USER:$USER /opt/cona
cd /opt/cona
```

3. Crear `docker-compose.yml`: Copia el archivo `docker-compose.yml` al servidor en `/opt/cona`. Asegúrate de usar contraseñas y secretos fuertes.
4. Configurar Firewall: Abre solo los puertos necesarios (ej. 80 para HTTP, 443 para HTTPS, 22 para SSH).

```
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw enable
```

Gestión de la Base de Datos en Producción:

- Seguridad: Nunca expongas el puerto de la base de datos (3306) al exterior. La comunicación debe ser a través de la red interna de Docker.
- Configuración: En producción, asegúrate de que `spring.jpa.hibernate.ddl-auto` esté configurado como `validate` o `none`, nunca `create` o `update`.

BACKUP, RECUPERACIÓN Y ALTA DISPONIBILIDAD

Backup de la Base de Datos:

El volumen de Docker `db_data` contiene todos los datos de MariaDB. Puedes hacer un backup de este volumen o, preferiblemente, usar `mysql dump`.

Script de Backup (`backup.sh`):

```
#!/bin/bash
BACKUP_DIR="/opt/cona/backups"
DATE=$(date +"%Y%m%d_%H%M%S")
BACKUP_FILE="$BACKUP_DIR/conaveg_db_backup_${DATE}.sql.gz"

mkdir -p $BACKUP_DIR

docker-compose exec -T cona-db mysql dump -u root -p'your_strong_root_password'
conaveg_db | gzip > $BACKUP_FILE

# Eliminar backups de más de 7 días
find $BACKUP_DIR -type f -name "*.sql.gz" -mtime +7 -delete
```

Ejecuta este script diariamente con un `cron job`.

Recuperación ante Desastres:

1. Levanta un nuevo servidor con la misma configuración.
2. Restaura el último backup de la base de datos:

```
gunzip < /path/to/backup.sql.gz | docker-compose exec -T cona-db mysql -u root -p'your_strong_root_password' conaveg_db
```

3. Levanta los contenedores con `docker-compose up -d`.

Alta Disponibilidad (Avanzado):

- Base de Datos: Configurar una réplica de MariaDB (maestro-esclavo) para tener una base de datos de respaldo en tiempo real.
- Aplicación: Utilizar un orquestador de contenedores como Kubernetes o Docker Swarm para desplegar múltiples instancias de `cona-app` y gestionar el failover automáticamente.

SCALING Y MIGRACIÓN DE DATOS

Scaling Horizontal:

Si una sola instancia de la aplicación no es suficiente, puedes escalar horizontalmente con Docker Compose:

```
docker-compose up --scale cona-app=3 -d
```

Esto levantará 3 instancias de la aplicación. Necesitarás un balanceador de carga (como Nginx o Traefik) para distribuir el tráfico entre ellas.

Migración de Datos:




Para gestionar cambios en el esquema de la base de datos de forma controlada, se recomienda usar una herramienta de migración como Flyway o Liquibase.

Integración con Flyway (Ejemplo):

1. Añadir la dependencia de Flyway en `pom.xml`.
2. Crear scripts de migración SQL en `src/main/resources/db/migration`.
 - o `V1__Create_initial_tables.sql`
 - o `V2__Add_new_column_to_users.sql`
3. Flyway ejecutará automáticamente las migraciones pendientes al iniciar la aplicación.





SOPORTE Y RECURSOS ADICIONALES

Documentación Relevante:

-  [Documentación de Docker](#)
-  [Documentación de GitHub Actions](#)
-  [Documentación de MariaDB](#)

Canales de Soporte:

- ✉ Email: devops-support@conaveg.com
- 💬 Slack: #cona-devops

 Fecha de Creación: 21 de Julio de 2025
 Responsable: Equipo de DevOps CONA
 Estado: Manual Completo y Validado
 Próxima Revisión: 21 de Agosto de 2025