Guía de Despliegue - API RESTful con Docker

Estudiante: Luis Diego **Curso:** Node.js Avanzado **Repositorio GitHub:** https://github.com/luisdie13/apirestful.git

📋 Tabla de Contenidos

1. Información del Proyecto

Descripción

API RESTful desarrollada con Node.js y Express.js que proporciona operaciones CRUD para gestionar productos. La aplicación está containerizada con Docker y utiliza PostgreSQL como base de datos.

Tecnologías Utilizadas

- **Node.js** v18 (Alpine Linux)
- **Express.js** - Framework web
- **PostgreSQL** v14 - Base de datos
- **Docker & Docker Compose** - Containerización
- **Jest** - Testing

Repositorio

```
https://github.com/luisdie13/apirestful.git
```

2. Construcción de la Imagen Docker

📄 Dockerfile

```
# Use official Node.js LTS (Long Term Support) image
FROM node:18-alpine

# Set working directory inside the container
WORKDIR /usr/src/app
```

```
# Copy package.json and package-lock.json
COPY package*.json ./

# Install production dependencies only
RUN npm ci --only=production

# Copy the rest of the application code
COPY . .

# Expose the port the app runs on
EXPOSE 3000

# Define environment variable
ENV NODE_ENV=production

# Start the application
CMD ["npm", "start"]
```

Comando de Construcción

```
docker build -t apirestful:1.0 .
```

Salida Esperada

```
[+] Building 24.6s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                0.0s
 => => transferring dockerfile: 534B                                0.0s
 => [internal] load metadata for docker.io/library/node:18-alpine   0.4s
 => [internal] load .dockerignore                                   0.0s
 => [1/5] FROM docker.io/library/node:18-alpine                    21.5s
 => [internal] load build context                                   0.1s
 => [2/5] WORKDIR /usr/src/app                                      0.2s
 => [3/5] COPY package*.json ./                                     0.0s
 => [4/5] RUN npm ci --only=production                              2.2s
 => [5/5] COPY . .                                                  0.0s
 => exporting to image                                              0.2s
 => => naming to docker.io/library/apirestful:1.0                   0.0s
```

Verificar Imagen Creada

```
docker images apirestful
```

Resultado:

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
apirestful   1.0       027aec339d26   7 seconds ago   135MB
```

☑ **Imagen optimizada de solo 135MB**

---

3. Ejecución Local con Docker Compose

📄 docker-compose.yml

```yaml
version: '3.8'
services:
  db:
    image: postgres:14-alpine
    restart: always
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_DATABASE}
    ports:
      - "5433:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - app-network

  app:
    build: .
    restart: always
    ports:
      - "3000:3000"
    environment:
      NODE_ENV: production
      PORT: 3000
      DB_USER: ${DB_USER}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_DATABASE: ${DB_DATABASE}
      DB_HOST: db
      DB_PORT: 5432
    depends_on:
      - db
    networks:
      - app-network

volumes:
  postgres_data:

networks:
  app-network:
    driver: bridge
```

Configuración de Variables de Entorno

Archivo .env:

```
# Database Configuration
DB_USER=postgres
DB_PASSWORD=mysecretpassword
DB_DATABASE=apirestful
DB_HOST=localhost
DB_PORT=5433

# Application Configuration
PORT=3000
NODE_ENV=development
```

Comando para Levantar los Servicios

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

Salida Esperada

```
[+] Building 0.9s (11/11) FINISHED
[+] Running 3/3
 ✔ Network apirestfulv1_app-network  Created
 ✔ Container apirestfulv1-db-1       Started
 ✔ Container apirestfulv1-app-1      Started
```

Verificar Estado de los Contenedores

```
docker-compose ps
```

Resultado:

```
NAME                IMAGE               COMMAND                  SERVICE
STATUS         PORTS
apirestfulv1-app-1   apirestfulv1-app     "docker-entrypoint.s…"   app      Up 6
seconds   0.0.0.0:3000->3000/tcp
apirestfulv1-db-1    postgres:14-alpine   "docker-entrypoint.s…"   db       Up 7
seconds   0.0.0.0:5433->5432/tcp
```

Ver Logs de la Aplicación

```
docker-compose logs app
```

Resultado:

```
app-1  | > apirestful@1.0.0 start
app-1  | > node index.js
app-1  |
app-1  | Servidor escuchando en el puerto 3000
app-1  | ☑ Conexión a la base de datos establecida.
```

☑ **Aplicación corriendo exitosamente**

---

4. Despliegue en Play With Docker

Pasos para Desplegar en Play With Docker

1. Acceder a Play With Docker

- Visita: https://labs.play-with-docker.com
- Inicia sesión con tu cuenta de Docker Hub
- Click en "Start"

2. Crear una Nueva Instancia

- Click en "+ ADD NEW INSTANCE"
- Se creará un terminal de Linux con Docker preinstalado

3. Clonar el Repositorio

```
git clone https://github.com/luisdie13/apirestful.git
cd apirestful
```

4. Crear Archivo .env

```
cat > .env << EOF
DB_USER=postgres
DB_PASSWORD=mysecretpassword
DB_DATABASE=apirestful
DB_HOST=db
DB_PORT=5432
PORT=3000
NODE_ENV=production
EOF
```

5. Construir y Ejecutar con Docker Compose

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

6. Verificar que los Contenedores Estén Corriendo

```
docker-compose ps
```

7. Ver Logs

```
docker-compose logs -f app
```

8. Probar la API

Play With Docker automáticamente genera URLs públicas para los puertos expuestos.

Encuentra el botón "3000" en la parte superior de la interfaz, que te llevará a la API.

O usa curl dentro del contenedor:

```
# Listar productos
curl http://localhost:3000/productos

# Crear un producto
curl -X POST http://localhost:3000/productos \
  -H "Content-Type: application/json" \
  -d '{"nombre":"Monitor 4K","precio":450.00,"descripcion":"Monitor de alta
resolución"}'

# Obtener producto por ID
curl http://localhost:3000/productos/1

# Actualizar producto
curl -X PUT http://localhost:3000/productos/1 \
  -H "Content-Type: application/json" \
  -d '{"nombre":"Monitor 4K UHD","precio":500.00,"descripcion":"Monitor
actualizado"}'

# Eliminar producto
curl -X DELETE http://localhost:3000/productos/1
```

5. Pruebas de la API

Endpoints Disponibles

Método	Endpoint	Descripción
GET	`/productos`	Obtener todos los productos
GET	`/productos/:id`	Obtener un producto por ID
POST	`/productos`	Crear un nuevo producto
PUT	`/productos/:id`	Actualizar un producto
DELETE	`/productos/:id`	Eliminar un producto

Ejemplo de Respuesta GET /productos

```
[
  {
    "id": 9,
    "nombre": "Monitor 4K",
    "descripcion": null,
    "precio": "450.00"
  }
]
```

Ejemplo de Request POST /productos

```
curl -X POST http://localhost:3000/productos \
  -H "Content-Type: application/json" \
  -d '{
    "nombre": "Laptop Gaming",
    "precio": 1200.00,
    "descripcion": "Laptop de alto rendimiento"
  }'
```

Respuesta:

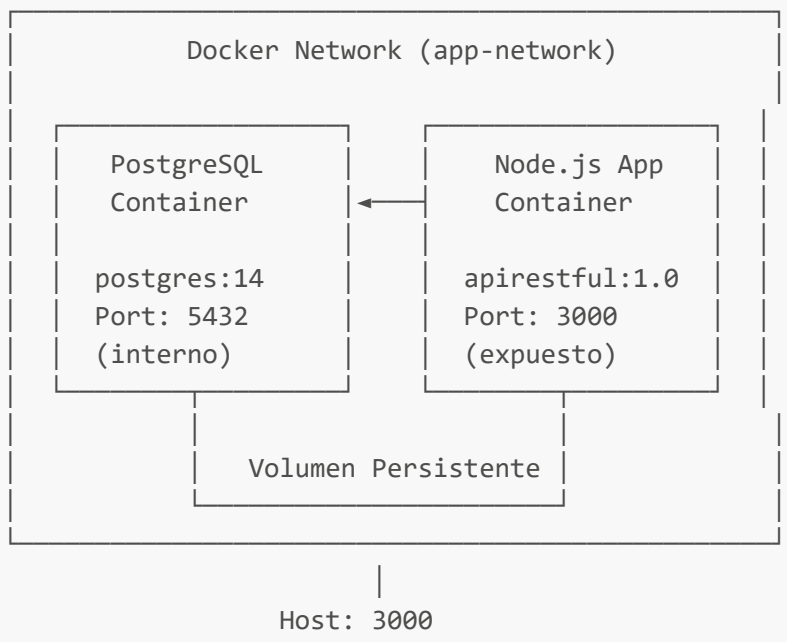
```
{
  "id": 10,
  "nombre": "Laptop Gaming",
  "descripcion": "Laptop de alto rendimiento",
  "precio": "1200.00"
}
```

---

6. Estructura del Proyecto

```
APIrestfulv1/
├── src/
│   ├── controllers/
│   │   └── entidad.controller.js    # Controladores de las rutas
│   ├── database/
│   │   └── db.js                     # Configuración de PostgreSQL
│   ├── middlewares/
│   │   └── validator.js              # Validación de datos
│   ├── routes/
│   │   └── entidad.route.js          # Definición de rutas
│   └── services/
│       └── entidad.service.js        # Lógica de negocio
├── tests/
│   └── entidad.test.js               # Tests unitarios
├── .dockerignore                      # Archivos excluidos del build
├── .env.example                       # Ejemplo de variables de entorno
├── .gitignore                         # Archivos ignorados por Git
├── app.js                             # Configuración de Express
├── babel.config.cjs                   # Configuración de Babel
├── docker-compose.yml                 # Orquestación de contenedores
├── Dockerfile                         # Definición de la imagen Docker
├── index.js                           # Punto de entrada
├── jest.config.cjs                    # Configuración de Jest
├── package.json                       # Dependencias del proyecto
├── package-lock.json                  # Lockfile de dependencias
└── README.md                          # Documentación del proyecto
```

📊 Arquitectura del Despliegue



```
                    |
                Internet
```

Características de Seguridad y Optimización

Dockerfile Optimizado

- **Imagen base ligera**: `node:18-alpine` (135MB total)
- **Multi-layer caching**: Optimiza rebuilds
- **Solo dependencias de producción**: `npm ci --only=production`
- **Variables de entorno**: Configuración flexible

.dockerignore

Excluye archivos innecesarios:

```
node_modules
npm-debug.log
.env
.git
.gitignore
tests/
*.test.js
README.md
```

Docker Compose

- **Redes aisladas**: Comunicación segura entre contenedores
- **Volúmenes persistentes**: Los datos de PostgreSQL se mantienen
- **Gestión de dependencias**: `depends_on` asegura orden de inicio
- **Variables de entorno**: Configuración centralizada

Seguridad

- Credenciales en archivos `.env` (no versionados)
- Red interna para comunicación entre servicios
- Puerto de base de datos no expuesto externamente en producción
- Reinicio automático de contenedores

Comandos Útiles

Docker Compose

```
# Iniciar servicios
docker-compose up -d
```

```
# Detener servicios
docker-compose down

# Ver logs
docker-compose logs -f app

# Reconstruir sin cache
docker-compose up -d --build --force-recreate

# Ver estado de contenedores
docker-compose ps

# Ejecutar comando en contenedor
docker-compose exec app sh
```

Docker

```
# Listar imágenes
docker images

# Listar contenedores
docker ps -a

# Ver logs de un contenedor
docker logs <container_id>

# Eliminar imagen
docker rmi apirestful:1.0

# Limpiar sistema
docker system prune -a
```

Notas Importantes

1. **Puerto de Base de Datos**:
   - Interno (contenedor): 5432
   - Externo (host): 5433 (para evitar conflictos)
2. **Persistencia de Datos**:
   - Los datos de PostgreSQL se guardan en el volumen `postgres_data`
   - Para eliminar datos: `docker-compose down -v`
3. **Variables de Entorno**:
   - Crear archivo `.env` basado en `.env.example`
   - No commitear el archivo `.env` a Git

4. **Desarrollo vs Producción**:
    - En desarrollo local: `DB_HOST=localhost`
    - Con Docker Compose: `DB_HOST=db`

Referencias

- **Repositorio GitHub**: https://github.com/luisdie13/apirestful.git
- **Docker Hub**: https://hub.docker.com/
- **Play With Docker**: https://labs.play-with-docker.com
- **Node.js**: https://nodejs.org/
- **PostgreSQL**: https://www.postgresql.org/
- **Express.js**: https://expressjs.com/
