Universidad Católica San Pablo

Login Usando Docker

Computer Science — Cloud Computing

Harold Alejandro Villanueva Borda

1. Introducción

Este tutorial muestra cómo implementar un sistema de login utilizando Docker con una arquitectura de tres componentes: frontend, backend y base de datos. La aplicación utilizará:

- React para el frontend
- Node.js con Express para el backend
- PostgreSQL para la base de datos

El sistema completo estará containerizado con Docker, permitiendo que cada componente se ejecute en su propio entorno aislado mientras se comunican a través de una red definida por Docker Compose.

2. Requisitos Previos

Para este tutorial, necesitarás tener instalado:

- Docker Desktop para Windows
- Node.js y npm
- Un editor de código (como Visual Studio Code)

3. Estructura del Proyecto

La estructura de carpetas del proyecto será la siguiente:

```
login-system/
│
├── frontend/
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── Login.js
│   │   │   ├── Register.js
│   │   │   └── Dashboard.js
│   │   ├── App.js
│   │   ├── App.css
│   │   └── index.js
│   ├── package.json
│   └── Dockerfile
│
├── backend/
│   ├── src/
│   │   ├── controllers/
│   │   │   └── authController.js
│   │   ├── models/
│   │   │   └── User.js
│   │   ├── routes/
│   │   │   └── authRoutes.js
│   │   ├── config/
│   │   │   └── db.js
│   │   ├── middlewares/
│   │   │   └── authMiddleware.js
│   │   └── server.js
│   ├── package.json
│   └── Dockerfile
│
├── database/
│   └── init.sql
│
└── docker-compose.yml
```

4. Creación del Proyecto

4.1. Preparación de la Estructura

Abre una terminal en Windows (CMD o PowerShell) y ejecuta los siguientes comandos para crear la estructura del proyecto:

```
mkdir login-system
cd login-system

mkdir frontend
mkdir backend
mkdir database

cd frontend
mkdir public
mkdir src
cd src
mkdir components
cd ..\..\

cd backend
mkdir src
cd src
mkdir controllers
mkdir models
mkdir routes
mkdir config
mkdir middlewares
cd ..\..\

cd database
```

4.2. Configuración del Frontend

Crea tu aplicación React para el frontend:

```
cd frontend
npm init -y
npm install react react-dom react-router-dom axios
npm install --save-dev @babel/core @babel/preset-env @babel/preset-react babel-loader css-loader style-loader webpack webpack-cli webpack-dev-server html-webpack-plugin
```

4.3. Configuración del Backend

Configura el backend con Node.js y Express:

```
cd ..\backend
npm init -y
npm install express pg bcrypt jsonwebtoken cors dotenv
npm install --save-dev nodemon
```

5. Configuración de Docker

Ahora, crearemos los archivos Dockerfile para cada componente y el archivo docker-compose.yml para orquestar los contenedores.

5.1. Dockerfile para el Frontend

Este Dockerfile configura un entorno optimizado para una aplicación frontend con Node.js, realizando:

- **Base**: Imagen oficial de Node.js 18 en Alpine.
- **Dependencias**: Instalación de paquetes definidos en `package.json`.
- **Build**: Compilación de la aplicación para producción.
- **Servidor**: Uso de `serve` para alojar los archivos estáticos resultantes (`build/`).
- **Despliegue**: Exposición del puerto `3000` y ejecución automática al iniciar el contenedor.

5.2. Dockerfile para el Backend

Este Dockerfile configura un entorno para ejecutar una aplicación backend en Node.js 18 (Alpine), optimizado para contenedores ligeros.

- **FROM**: Imagen base oficial de Node.js 18 en Alpine.
- **WORKDIR**: Define `/app` como directorio principal.
- **COPY + RUN**: Instala dependencias antes de copiar el código para aprovechar el caching de Docker.
- **COPY . .**: Clona el código fuente restante al contenedor.
- **EXPOSE**: Habilita el puerto 5000 para conexiones externas.
- **CMD**: Inicia la aplicación con `npm start`.

5.3. Docker Compose

Archivo `docker-compose.yml` que configura tres servicios interconectados para desplegar una aplicación full-stack en contenedores Docker.

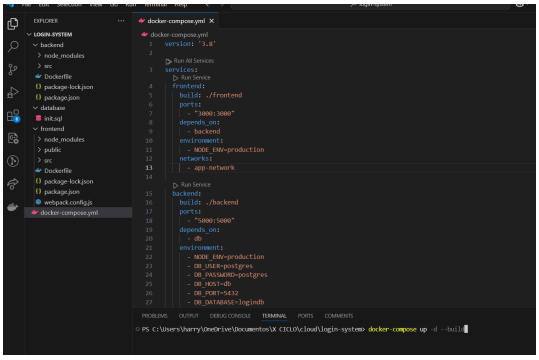
1. Frontend
   - **Construcción**: Desde el directorio `./frontend`
   - **Puerto**: 3000 (para frameworks como React/Vue)
   - **Dependencias**: Requiere el servicio `backend`
   - **Red**: Conectado a `app-network`
   - **Entorno**: Configurado para producción (`NODE_ENV=production`)
2. Backend
   - **Construcción**: Desde `./backend`
   - **Puerto**: 5000 (para Node.js/Python/etc)
   - **Configuración**:
     - Variables de entorno para conexión a BD
     - Secreto JWT para autenticación
   - **Dependencias**: Requiere el servicio `db`
3. Base de datos (PostgreSQL)
   - **Imagen**: `postgres:13-alpine`
   - **Configuración**:
     - Usuario/contraseña: `postgres`
     - Nombre BD: `logindb`
   - **Inicialización**: Ejecuta script `init.sql`
   - **Persistencia**: Volumen `pgdata` para datos persistentes

6. Ejecución del Proyecto

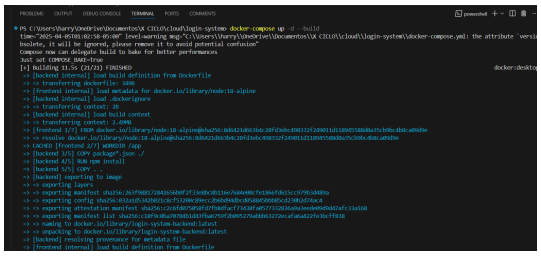
Una vez que tengas todos los archivos configurados, puedes ejecutar el proyecto usando Docker:

```
cd login-system
docker-compose up -d --build
```

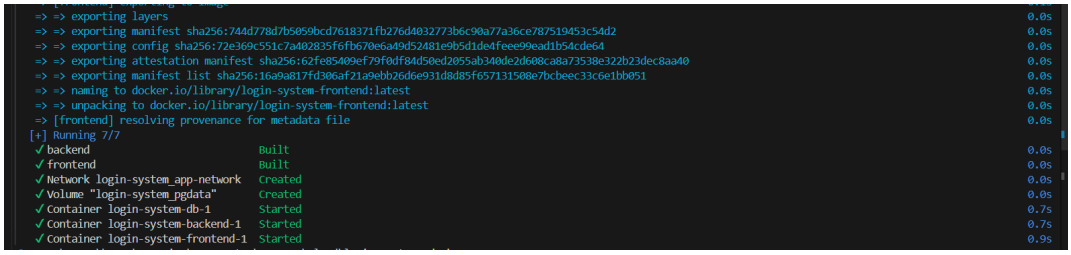
Este comando construirá las imágenes necesarias y ejecutará los tres contenedores (frontend, backend y base de datos) en segundo plano.



1: Resultado de la ejecución del primer comando



2: Resultado de la ejecución del primer comando



3: Resultado de la ejecución del primer comando

Figura 1: Ejecución general

6.1. Verificación de los Contenedores

Para verificar que los contenedores están ejecutándose correctamente:

```
docker-compose ps
```

Deberías ver tres contenedores en estado Üp”:

```
Name                          Command                       State                    Ports
-------------------------------------------------------------------------------------------------

login-system_backend_1     docker-entrypoint.sh npm start     Up
   0.0.0.0:5000->5000/tcp
login-system_db_1          docker-entrypoint.sh postgres      Up
   0.0.0.0:5432->5432/tcp
login-system_frontend_1    docker-entrypoint.sh serve ...     Up
   0.0.0.0:3000->3000/tcp
```

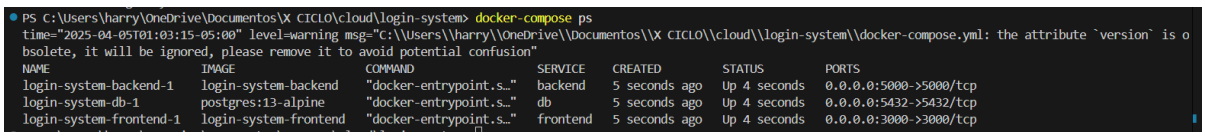


Figura 2: Resultado de la ejecución del segundo comando

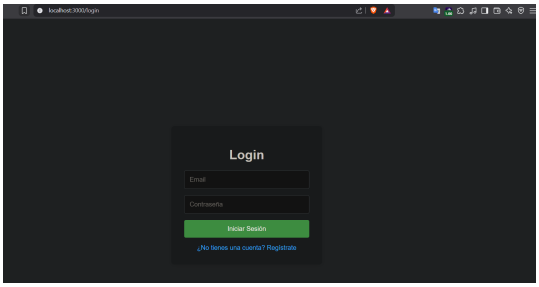
Para ver los logs de los contenedores:

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

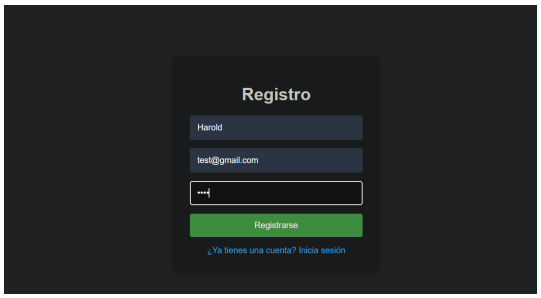
7. Probando el Sistema de Login

Para probar el sistema:

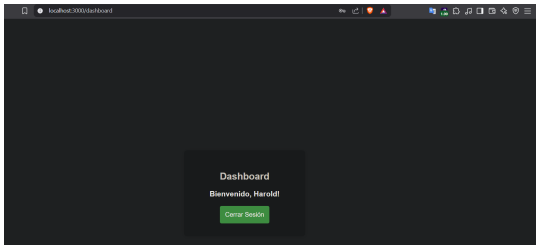
1. Ingresa a `http://localhost:3000`
2. Se abre la página de login
3. Clic en el enlace para registrarte
4. Completa el formulario de registro con tu nombre, email y contraseña
5. Una vez registrado, serás redirigido al dashboard
6. Cerrar sesión y volver a iniciar sesión para probar la funcionalidad



1: Resultado del loguin



2: Resultado del registro



3: Resultado del dashboard

Figura 3: Ejecución general

8. Interacción entre Componentes

- **Frontend (React):** Se ejecuta en el puerto 3000 y proporciona la interfaz de usuario. Se comunica con el backend mediante solicitudes HTTP utilizando Axios.
- **Backend (Node.js/Express):** Se ejecuta en el puerto 5000 y maneja las solicitudes del frontend. Se comunica con la base de datos para autenticar usuarios y almacenar/recuperar datos.
- **Base de Datos (PostgreSQL):** Almacena los datos de los usuarios de forma persistente.

8.1. Flujo de la Aplicación

1. El usuario accede a la aplicación frontend
2. Al registrarse o iniciar sesión, el frontend envía una solicitud al backend
3. El backend valida los datos y consulta la base de datos
4. Si la autenticación es exitosa, el backend genera un JWT (token) y lo devuelve al frontend
5. El frontend almacena el token en localStorage y lo usa para las solicitudes subsiguientes

9. Acceso a la Base de Datos

Para acceder y visualizar la base de datos PostgreSQL que está ejecutándose en el contenedor Docker:

9.1. Usando la línea de comandos directamente desde el contenedor

```
docker exec -it login-system_db_1 psql -U postgres -d logindb
```

Una vez conectado, puedes usar comandos SQL para interactuar con la base de datos:

```
-- Ver las tablas existentes
\dt
-- Ver la estructura de la tabla users
\d users
-- Consultar todos los usuarios
SELECT * FROM users;
-- Salir de psql
\q
```

9.2. Usando scripts desde la línea de comandos

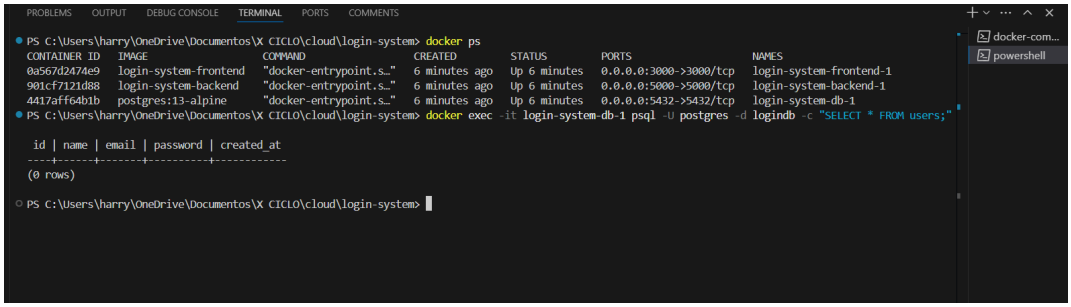
También puedes ejecutar scripts SQL sin entrar en el modo interactivo:

```
docker exec -it login-system_db_1 psql -U postgres -d logindb -c "SELECT * FROM
    users;"
```

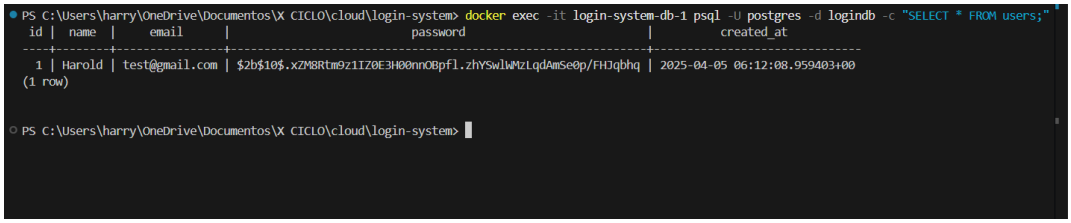
9.3. Creando un script para realizar un backup

Si quieres hacer un backup de la base de datos:

```
docker exec -it login-system_db_1 pg_dump -U postgres -d logindb > backup.sql
```



1: Resultado de la BD antes del registro



2: Resultado de la BD depues del registro

Figura 4: Ejecución general

9.4. Para verificar los logs de PostgreSQL

```
docker-compose logs db
```

10. Errores comunes y verificación

Si el sistema no funciona como se espera, puedes verificar:

```
# Ver los logs del backend
docker-compose logs backend

# Ver los ultimos errores en general
docker-compose logs --tail=100

# Verificar si el contenedor del backend se cre  pero se detuvo
docker ps -a

# Ver logs de un contenedor espec fico
docker logs login-system-backend-1
```

10.1. ¿El contenedor del backend se creó, pero se detuvo?

Ejecuta:

```
docker ps -a
```

Este comando te muestra todos los contenedores, incluso los que están detenidos. Si ves uno llamado `login-system-backend-1` con el estado `Exited`, entonces se inició pero falló. Por ejemplo:

```
login-system-backend-1   backend-image-name   ...   Exited (1)  5 seconds ago
```

En ese caso, revisa los logs para saber por qué falló:

```
docker logs login-system-backend-1
```

10.2. ¿Se olvidó construir el contenedor del backend?

Si hiciste cambios recientemente y no reconstruiste, asegúrate de forzar el rebuild con:

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

10.3. Errores con bcrypt

Un error común es:

```
Error: Error loading shared library /app/node_modules/bcrypt/lib/binding/napi-
    v3/bcrypt_lib.node: Exec format error
```

Esto indica que el archivo binario nativo de bcrypt no es compatible con la arquitectura del contenedor Docker. Esto sucede cuando se instala bcrypt en máquina local Windows y luego se copian los node_modules dentro del contenedor Linux.

Solución: Usar bcryptjs como alternativa:

```
npm uninstall bcrypt
npm install bcryptjs
```

Y en el código (backend/src/models/User.js) cambiar:

```
const bcrypt = require('bcryptjs');
```

10.4. Limpieza y reconstrucción completa

Después de hacer cambios, limpia y reconstruye tus contenedores:

```
docker-compose down --volumes --remove-orphans
docker-compose build --no-cache
docker-compose up
```

11. Seguridad

Este sistema incluye:

- Contraseñas encriptadas con bcrypt

- Autenticación basada en tokens JWT
- Protección de rutas en el backend
- Redirecciones condicionadas en el frontend

12. Conclusión

En este tutorial, has creado un sistema de login completo con Docker, que incluye:

- Frontend con React
- Backend con Node.js/Express
- Base de datos PostgreSQL

Este sistema es un buen punto de partida para cualquier aplicación que requiera autenticación. Puedes expandirlo agregando más funcionalidades como recuperación de contraseña, perfiles de usuario, roles y permisos, entre otras.

13. Repositorio

El código fuente está disponible en GitHub - Sistema de Login con Docker.