

# Documentación completa – IAW + BBDD (MediCloud)

## 1) Visión general del entorno local

### Objetivo

Montar un entorno local completo para validar la implantación de la aplicación web de MediCloud antes del despliegue en Azure, con:

- Base de datos MySQL **en contenedor Docker**
- API backend **Node + Express** conectada a MySQL
- Frontend **Angular** consumiendo la API

### Arquitectura (local)

- **MySQL (Docker):** localhost:3306
- **API Node:** <http://localhost:3000>
- **Angular:** <http://localhost:4200>

Flujo:

1. Usuario accede a Angular
2. Angular hace login contra Node (/auth/login)
3. Node consulta MySQL (cuenta) y devuelve JWT
4. Angular guarda token y consume endpoints protegidos (/files)
5. Node registra eventos en evento\_seguridad

## 2) Carpeta medicloud-local (Docker + MySQL)

### 2.1 Archivo modificado/creado: docker-compose.yml

**Ruta:** medicloud-local/docker-compose.yml

**Finalidad:** levantar MySQL 8.0 con base de datos inicial y usuario de aplicación.

**Contenido actual:**

- Servicio: mysql
- Imagen: mysql:8.0
- Nombre contenedor: **medicloud-mysql**
- Variables de entorno:
  - MYSQL\_ROOT\_PASSWORD=rootpass
  - MYSQL\_DATABASE=medicloud
  - MYSQL\_USER=medicloud\_app
  - MYSQL\_PASSWORD=apppass
- Puerto expuesto: 3306:3306
- Volumen persistente: mysql\_data:/var/lib/mysql

#### **Implicaciones:**

- La base de datos persiste aunque apagues el contenedor (por el volumen).
- La API Node se conecta desde el host a localhost:3306 usando el usuario medicloud\_app.

### **2.2 Arranque del contenedor**

Desde medicloud-local:

```
docker compose up -d
docker ps
```

### **2.3 Importación del esquema SQL (v2)**

Se importó el script SQL del esquema v2 dentro del contenedor (ejemplo):

```
docker exec -i medicloud-mysql mysql -u root -prootpass <
medicloud_v2_mysql.sql
```

### **2.4 Verificación de estado (comprobaciones)**

Comprobación de bases de datos:

```
docker exec -it medicloud-mysql mysql -u root -prootpass -e "SHOW
DATABASES;"
```

Comprobación de tablas:

```
docker exec -it medicloud-mysql mysql -u root -prootpass -D medicloud -e "SHOW TABLES;"
```

Comprobación de estructura:

```
docker exec -it medicloud-mysql mysql -u root -prootpass -D medicloud -e "DESCRIBE cuenta;"  
docker exec -it medicloud-mysql mysql -u root -prootpass -D medicloud -e "DESCRIBE archivo;"
```

## 2.5 Backup local (antes de Azure)

Se genera un backup en formato SQL:

```
docker exec -it medicloud-mysql mysqldump -u root -prootpass medicloud > medicloud_backup.sql
```

# 3) Carpeta medicloud-api (Node + Express + MySQL)

## 3.1 Dependencias instaladas ( proyecto Node )

Se creó un backend Node/Express con:

- express (servidor)
- cors (permitir peticiones desde Angular)
- dotenv (variables de entorno)
- mysql2 (cliente MySQL con promesas)
- jsonwebtoken (JWT)

Modo desarrollo:

- nodemon para recargar en caliente

Instalación típica:

```
npm init -y  
npm i express mysql2 cors dotenv jsonwebtoken  
npm i -D nodemon
```

## 3.2 Archivo: index.js (contenido real documentado)

Ruta: medicloud-api/index.js

### 3.2.1 Carga de configuración y módulos

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const jwt = require('jsonwebtoken');
const mysql = require('mysql2/promise');
```

### 3.2.2 Middlewares aplicados (CORS + JSON)

index.js actual

```
const app = express();
app.use(cors({ origin: true }));
app.use(express.json());

app.use(cors({ origin: 'http://localhost:4200' }));
```

### 3.2.3 Variables de entorno utilizadas

```
const { PORT = 3000, DB_HOST, DB_PORT, DB_USER, DB_PASSWORD,
DB_NAME, JWT_SECRET } = process.env;
```

### 3.2.4 Pool MySQL

```
const pool = mysql.createPool({
  host: DB_HOST,
  port: Number(DB_PORT),
  user: DB_USER,
  password: DB_PASSWORD,
  database: DB_NAME,
  waitForConnections: true,
  connectionLimit: 10,
});
```

### **Conexión esperada (local):**

- DB\_HOST=127.0.0.1 o localhost
- DB\_PORT=3306
- DB\_USER=medicloud\_app
- DB\_PASSWORD=apppass
- DB\_NAME=medicloud

### **3.2.5 Autenticación JWT**

- signToken(user) genera JWT con:
  - sub = id\_cuenta
  - email = correo
  - role = tipo\_cuenta
- auth() valida Authorization: Bearer <token> y bloquea si falta o es inválido.

### **3.2.6 Endpoints implementados**

**POST /auth/login**

- Input: { "email": "..." }
- Lógica:
  - consulta cuenta por correo
  - valida que la cuenta esté ACTIVA
  - crea token
  - inserta evento en evento\_seguridad (LOGIN\_OK)
- Output:

```
{  
  "token": "...",  
  "user": { "id": 1, "email": "...", "role": "CLIENTE" }  
}
```

**GET /me (protegido)**

- Requiere token
- Devuelve datos de cuenta:
  - id\_cuenta, correo, nombre, tipo\_cuenta, estado

**GET /files (protegido)**

- Requiere token
- Devuelve archivos donde:
  - `archivo.id_cuenta_propietaria = id_cuenta` (regla demo)
- Registra evento LISTA\_ARCHIVOS

**POST /events (protegido)**

- Requiere token
- Inserta evento personalizado en `evento_seguridad`
- Campos: `tipo_evento`, `resultado`, `detalle`, `id_archivo` opcional, `id_factura` opcional

**GET /health**

- Consulta `SELECT 1 AS ok`
- Respuesta esperada:

```
{ "status": "ok", "db": 1 }
```

**GET /**

- Respuesta de texto:  
MediCloud API OK. Usa `/health`, `/auth/login`, `/me`, `/files`

### **3.2.7 Arranque del servidor**

```
app.listen(PORT, () => console.log(`MediCloud API running on  
http://localhost:\${PORT}`));
```

## **3.3 Pruebas realizadas**

- GET <http://localhost:3000/health> → {status:"ok", db:1}
- POST `/auth/login` con email existente → devuelve token
- GET `/files` con token → lista archivos y genera evento

## 4) Carpeta medicloud-web (Angular)

### 4.1 Rutas (archivo real)

Ruta: src/app/app.routes.ts

Contenido actual:

- / → Home
- /login → Login
- /cliente/archivos → ClienteArchivos
- /trabajador/archivos → TrabajadorArchivos
- /forbidden → Forbidden
- \*\* → NotFound

Con lazy loading de standalone components:

```
{ path: '', loadComponent: () =>
import('./features/public/home/home').then(m => m.Home) },
{ path: 'login', loadComponent: () =>
import('./features/public/login/login').then(m => m.Login) },
{ path: 'cliente/archivos', loadComponent: () =>
import('./features/cliente/cliente-archivos/cliente-
archivos').then(m => m.ClienteArchivos) },
{ path: 'trabajador/archivos', loadComponent: () =>
import('./features/trabajador/trabajador-archivos/trabajador-
archivos').then(m => m.TrabajadorArchivos) },
{ path: 'forbidden', loadComponent: () =>
import('./shared/forbidden/forbidden').then(m => m.Forbidden) },
{ path: '**', loadComponent: () => import('./shared/not-found/not-
found').then(m => m.NotFound) },
```

### 4.2 Estructura del proyecto (standalone, sin .component.ts)

Tu Angular usa componentes standalone con archivos:

- .ts (clase con @Component)
- .html (templateUrl)
- .css (styleUrl)

Ejemplo de patrón real:

```

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [...],
  templateUrl: './login.html',
  styleUrls: ['./login.css'],
})
export class Login {}

```

## 4.3 HttpClient y consumo de API

Para consumir la API Node desde Angular se hizo:

1. Activar provideHttpClient() en app.config.ts.
2. Crear un servicio de API (p.e. src/app/shared/api/api.service.ts) con:
  - a. login(email)
  - b. files(token)
  - c. me(token)

Importante: en Angular “nuevo” puede no existir src/environments. Se creó manualmente.

## 4.4 Flujo funcional implementado

1. Usuario abre /login
2. Introduce email (demo)
3. Angular llama POST <http://localhost:3000/auth/login>
4. Guarda token en localStorage
5. Redirige a:
  - a. cliente → /cliente/archivos
  - b. trabajador → /trabajador/archivos
6. En /cliente/archivos Angular llama GET /files con Authorization: Bearer <token>
7. Se listan archivos reales de MySQL

## 4.5 Comandos usados (Angular)

Arranque:

`ng serve`

Acceso:

- <http://localhost:4200>

## 5) Orden de arranque recomendado (para repetirlo en el futuro)

### 1. Base de datos

```
cd medicloud-local  
docker compose up -d
```

### 2. API

```
cd medicloud-api  
npm run dev
```

### 3. Angular

```
cd medicloud-web  
ng serve
```

## 6) Credenciales de prueba (demo actual)

El login actual no valida password. Solo email debe existir en cuenta y estar ACTIVA.

Ejemplos si el seed sigue:

- [cliente1@demo.com](mailto:cliente1@demo.com)
- [empleado@demo.com](mailto:empleado@demo.com)
- [admin@demo.com](mailto:admin@demo.com)