

---

# Guia de Repaso

## Docker, Bases de Datos y REST

---

Semana 01 - Viernes 13 de Febrero de 2026

Academia Java - Ciudad de Mexico

### **Temas del dia:**

1. Docker: conceptos e instalacion
2. MySQL y MongoDB en contenedores
3. SQL vs NoSQL
4. DBeaver y MongoDB Compass
5. API REST con Spring Boot: CRUD de alumnos
6. Conceptos REST y HTTP

*Instructor: Miguel Rugerio*

# 1. Docker: conceptos e instalacion

Docker permite empaquetar una aplicacion junto con todo lo que necesita (Java, base de datos, dependencias) dentro de un **contenedor**. Un contenedor es como una caja portatil: lo que funciona en tu maquina, funciona igual en cualquier otro entorno.

## ¿Por que usar Docker?

El problema clasico: *"en mi maquina si funciona"*. Cada desarrollador instala versiones diferentes de Java, MySQL, etc. Docker elimina ese problema.

Sin Docker	Con Docker
Instalar Java, MySQL, configurar variables, crear BD manualmente	<code>docker compose up</code>
Cada dev tiene versiones diferentes	Todos usan el mismo contenedor
Conflictos entre proyectos	Cada proyecto aislado

## Instalacion en Windows

**Paso 1:** Habilitar WSL 2 (PowerShell como Administrador):

```
wsl --install
```

**Paso 2:** Descargar Docker Desktop desde [docker.com](https://docker.com) y ejecutar el instalador.

**Paso 3:** Verificar la instalacion:

```
docker --version  
docker run hello-world
```

---

## Comandos esenciales

---

Comando	Descripcion
<code>docker run &lt;imagen&gt;</code>	Crear y ejecutar un contenedor
<code>docker ps</code>	Ver contenedores en ejecucion
<code>docker stop &lt;id&gt;</code>	Detener un contenedor
<code>docker start &lt;nombre&gt;</code>	Iniciar un contenedor detenido
<code>docker rm &lt;id&gt;</code>	Eliminar un contenedor
<code>docker images</code>	Ver imagenes descargadas
<code>docker logs &lt;nombre&gt;</code>	Ver logs de un contenedor

## 2. MySQL y MongoDB en contenedores

### MySQL 8 en Docker

```
docker pull mysql:8

docker run --name mysql-academia \
  -e MYSQL_ROOT_PASSWORD=root123 \
  -p 3306:3306 \
  -d mysql:8
```

Parametro	Que hace
<code>--name mysql-academia</code>	Nombre del contenedor
<code>-e MYSQL_ROOT_PASSWORD=root123</code>	Contraseña del usuario root
<code>-p 3306:3306</code>	Expone el puerto 3306 en tu maquina
<code>-d</code>	Ejecuta en segundo plano

Conectarse desde terminal:

```
docker exec -it mysql-academia mysql -u root -p
```

### MongoDB 7 en Docker

```
docker pull mongo:7
```

```
docker run --name mongo-academia \  
  -e MONGO_INITDB_ROOT_USERNAME=root \  
  -e MONGO_INITDB_ROOT_PASSWORD=root123 \  
  -p 27017:27017 \  
  -d mongo:7
```

Conectarse desde terminal:

```
docker exec -it mongo-academia mongosh -u root -p root123
```

## Comandos del día a día

```
docker stop mysql-academia mongo-academia    # Detener  
docker start mysql-academia mongo-academia    # Iniciar de nuevo  
docker logs mysql-academia                    # Ver logs
```

## 3. SQL vs NoSQL

En SQL (MySQL), la tabla tiene columnas fijas. **Todos los registros deben tener la misma estructura.**

```
CREATE TABLE alumnos (  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  nombre VARCHAR(100) NOT NULL,  
  apellido VARCHAR(100) NOT NULL,  
  email VARCHAR(150),  
  fecha_registro DATE DEFAULT (CURRENT_DATE)  
);
```

Si quieres que UN alumno tenga telefono, debes agregar la columna a TODOS:

```
ALTER TABLE alumnos ADD COLUMN telefono VARCHAR(20);  
-- Ahora TODOS tienen telefono (con NULL en la mayoria)
```

## En MongoDB cada documento puede ser diferente

Un alumno con telefono y otro sin telefono, sin problema:

```
db.alumnos.insertMany([  
  {  
    nombre: "Maria",  
    apellido: "Garcia",  
    email: "maria@mail.com",  
    telefono: "55-1234-5678"  
  },  
  {  
    nombre: "Carlos",  
    apellido: "Hernandez",  
    email: "carlos@mail.com"
```

```
}  
])
```

Un alumno con datos completamente diferentes a los demas:

```
db.alumnos.insertOne({  
  nombre: "Laura",  
  apellido: "Sanchez",  
  empresa: "Tech Corp",  
  puesto: "Developer",  
  tecnologias: ["Java", "Python", "Docker", "AWS"]  
})
```

! En MongoDB no se necesita ALTER TABLE, ni JOINS para datos anidados, ni tablas intermedias para listas. Cada documento es independiente.

## Tabla comparativa

Aspecto	SQL (MySQL)	NoSQL (MongoDB)
Estructura	Fija, definida por la tabla	Flexible, cada documento puede ser diferente
Agregar un campo	ALTER TABLE (afecta toda la tabla)	Solo agregalo al documento que lo necesite
Datos anidados	Requiere tablas extra + JOINS	Se guardan directamente en el documento
Listas/Arrays	Requiere tabla intermedia	Campo tipo array nativo

## 4. DBeaver y MongoDB Compass

### DBeaver Community (para MySQL)

Cliente gratuito de bases de datos. Descarga en [dbeaver.io/download](https://dbeaver.io/download).

Datos de conexión a MySQL en Docker:

```
Host:      localhost
Puerto:   3306
Usuario:   root
Password:  root123
```

### Problema comun: "Public Key Retrieval is not allowed"

MySQL 8 usa `caching_sha2_password` que necesita un permiso adicional.

**Solucion:** En la conexión, pestaña **Driver properties**, cambiar `allowPublicKeyRetrieval` a **TRUE**.

### MongoDB Compass (para MongoDB)

DBeaver Community **no soporta MongoDB** (requiere version PRO). Usamos MongoDB Compass.

Descarga en [mongodb.com/products/tools/compass](https://mongodb.com/products/tools/compass).

Connection string:

```
mongodb://root:root123@localhost:27017
```

! Antes de ejecutar cualquier comando en la shell de Compass, selecciona la base de datos con `use academia`.

## 5. API REST con Spring Boot

Se construyo una aplicacion REST con CRUD completo para la tabla `alumnos` de MySQL, usando la arquitectura de 3 capas.

### Flujo de una peticion

```
Cliente (Postman/curl)
  ↓
Controller → Recibe la peticion HTTP
  ↓
Service     → Ejecuta la logica de negocio
  ↓
Repository  → Ejecuta el query en la base de datos
  ↓
MySQL       → Devuelve los datos
```

### Entity - Representa la tabla

```
@Entity
@Table(name = "alumnos")
public class Alumno {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, length = 100)
    private String nombre;

    @Column(nullable = false, length = 100)
    private String apellido;
```

```

    @Column(length = 150)
    private String email;

    @Column(name = "fecha_registro")
    private LocalDate fechaRegistro;

    // Constructor vacio + Getters y Setters
}

```

Anotacion	Que hace
@Entity	Esta clase representa una tabla
@Table(name = "alumnos")	Nombre de la tabla en MySQL
@Id	Llave primaria
@GeneratedValue(IDENTITY)	AUTO_INCREMENT de MySQL
@Column(name = "fecha_registro")	Mapea atributo Java a columna MySQL

## Repository - Acceso a datos

```

public interface AlumnoRepository extends JpaRepository<Alumno, Integer> {
}

```

Con solo heredar de `JpaRepository`, obtienes estos metodos gratis:

Metodo	SQL equivalente
<code>findAll()</code>	SELECT * FROM alumnos
<code>findById(id)</code>	SELECT * FROM alumnos WHERE id = ?
<code>save(alumno)</code>	INSERT / UPDATE
<code>deleteById(id)</code>	DELETE FROM alumnos WHERE id = ?

---

## Service - Logica de negocio

```
@Service
public class AlumnoService {

    private final AlumnoRepository repository;

    public AlumnoService(AlumnoRepository repository) {
        this.repository = repository;
    }

    public List<Alumno> listarTodos() {
        return repository.findAll();
    }

    public Alumno crear(Alumno alumno) {
        if (alumno.getFechaRegistro() == null) {
            alumno.setFechaRegistro(LocalDate.now());
        }
        return repository.save(alumno);
    }

    // buscarPorId, actualizar, eliminar...
}
```

## Controller - Endpoints HTTP

```
@RestController
@RequestMapping("/api/alumnos")
public class AlumnoController {

    private final AlumnoService service;

    public AlumnoController(AlumnoService service) {
        this.service = service;
    }

    @GetMapping
    public List<Alumno> listarTodos() {
```

```
        return service.listarTodos();
    }

    @PostMapping
    public ResponseEntity<Alumno> crear(@RequestBody Alumno alumno) {
        Alumno creado = service.crear(alumno);
        return ResponseEntity.status(HttpStatus.CREATED).body(creado);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Alumno> actualizar(@PathVariable Integer id,
                                             @RequestBody Alumno alumno) {
        return service.actualizar(id, alumno)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> eliminar(@PathVariable Integer id) {
        if (service.eliminar(id)) {
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

## Endpoints del CRUD

Metodo	Endpoint	Accion	Codigo HTTP
GET	/api/alumnos	Listar todos	200 OK
GET	/api/alumnos/{id}	Buscar por id	200 / 404
POST	/api/alumnos	Crear alumno	201 Created
PUT	/api/alumnos/{id}	Actualizar alumno	200 / 404
DELETE	/api/alumnos/{id}	Eliminar alumno	204 / 404

## 6. Conceptos REST y HTTP

### Metodos HTTP

Cada metodo representa una accion diferente sobre un recurso:

Metodo	Accion	¿Envia body?
GET	Obtener datos (solo lectura)	No
POST	Crear un recurso nuevo	Si (JSON)
PUT	Actualizar un recurso existente	Si (JSON)
DELETE	Eliminar un recurso	No

### Codigos de estado HTTP

Rango	Significado	Ejemplos
<b>2xx</b>	Todo salio bien	200 OK, 201 Created, 204 No Content
<b>4xx</b>	Error del cliente	400 Bad Request, 404 Not Found
<b>5xx</b>	Error del servidor	500 Internal Server Error

**Regla para recordar:** 2xx = todo bien, 4xx = tu hiciste algo mal, 5xx = el servidor tiene un problema.

## JSON

Formato en que viajan los datos entre cliente y servidor:

```
{
  "id": 1,
  "nombre": "Juan",
  "apellido": "Lopez",
  "email": "juan@mail.com",
  "fechaRegistro": "2026-02-13"
}
```

Spring Boot convierte automaticamente entre JSON y objetos Java. Las llaves van entre comillas dobles, los textos entre comillas dobles, los numeros sin comillas.

## URL REST

```
http://localhost:8080/api/alumnos/1
```

- ID del recurso
- Recurso (plural)
- Prefijo de la API
- Puerto
- Host
- Protocolo

**Convencion:** Las URLs van en plural ( `/api/alumnos` , no `/api/alumno` ), en minusculas y sin verbos. La accion la define el metodo HTTP, no la URL.