Guía paso a paso para recrear grpc-H2db desde cero
==

1. Crear carpeta y pom.xml
```bash
mkdir grpc-H2db && cd grpc-H2db
cat > pom.xml <<'EOF'
# pega aquí el contenido completo del pom (paso 2)
EOF
```

2. Sustituir pom.xml (idéntico al proyecto actual)
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.grpc</groupId>
  <artifactId>ej5</artifactId>
  <version>1.0</version>

  <dependencies>
    <dependency>
      <groupId>com.google.protobuf</groupId>
      <artifactId>protobuf-java</artifactId>
      <version>3.22.2</version>
    </dependency>
    <dependency>
      <groupId>io.grpc</groupId>
      <artifactId>grpc-netty-shaded</artifactId>
      <version>1.54.0</version>
    </dependency>
    <dependency>
      <groupId>io.grpc</groupId>
      <artifactId>grpc-protobuf</artifactId>
      <version>1.54.0</version>
    </dependency>
    <dependency>
      <groupId>io.grpc</groupId>
    <artifactId>grpc-stub</artifactId>
    <version>1.54.0</version>
  </dependency>
  <dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.2</version>
  </dependency>
  <dependency>
    <groupId>io.reactivex.rxjava3</groupId>
    <artifactId>rxjava</artifactId>
    <version>3.1.8</version>
  </dependency>
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20230227</version>
  </dependency>

  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.2.224</version>
    <scope>runtime</scope>
  </dependency>

</dependencies>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
  <defaultGoal>clean generate-sources compile install</defaultGoal>
  <plugins>
    <plugin>
      <groupId>com.github.os72</groupId>
      <artifactId>protoc-jar-maven-plugin</artifactId>
      <version>3.6.0.1</version>
      <executions>
        <execution>
```

```
            <phase>generate-sources</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <includeMavenTypes>direct</includeMavenTypes>
              <inputDirectories>
                <include>src/main/resources</include>
              </inputDirectories>
              <!-- Descomentar SOLO SI TIENES macOS con arquitectura Intel (x86_64) --
>
              <protocArtifact>com.google.protobuf:protoc:3.22.2:exe:osx-
x86_64</protocArtifact>
              <outputTargets>
                <outputTarget>
                  <type>java</type>
                  <outputDirectory>src/main/java</outputDirectory>
                </outputTarget>
                <outputTarget>
                  <type>grpc-java</type>
                  <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.54.0</pluginArtifact>
                  <outputDirectory>src/main/java</outputDirectory>
                </outputTarget>
              </outputTargets>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>17</source>
          <target>17</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

3. Crear estructura de carpetas
```bash
mkdir -p src/main/resources
mkdir -p src/main/java/client
mkdir -p src/main/java/server
mkdir -p src/main/java/service
mkdir -p src/main/java/db
```

4. Definir contrato gRPC en src/main/resources/product.proto
```proto
syntax = "proto3";

option java_multiple_files = true;
option java_package = "com.grpc.ej5";
option java_outer_classname = "ProductProto";

service ProductService {

  // 1. UNARY: Obtener producto por ID
  rpc GetProductById (ProductIdRequest) returns (ProductResponse);

  // 2. SERVER STREAMING: Productos más baratos que un precio
  rpc StreamCheaperProducts (PriceRequest) returns (stream ProductResponse);

  // 3. CLIENT STREAMING: Insertar varios productos
  rpc InsertProducts (stream NewProductRequest) returns (InsertSummary);

  // 4. BIDIRECTIONAL STREAMING: Consultas dinámicas de precio
  rpc LivePriceQuery (stream PriceRequest) returns (stream ProductResponse);
}

message ProductIdRequest {
  int32 id = 1;
}

message PriceRequest {
  double maxPrice = 1;
}
```

```
message ProductResponse {
  int32 id = 1;
  string name = 2;
  double price = 3;
}

message NewProductRequest {
  string name = 1;
  double price = 2;
}

message InsertSummary {
  int32 insertedCount = 1;
}
```

5. Generar código Java desde el .proto

```bash
mvn clean compile
```

(Genera clases en src/main/java/com/grpc/ej5).

6. Implementar acceso H2 en src/main/java/db/H2Database.java

```java
package db;

import java.sql.Connection;
import java.sql.DriverManager;

public class H2Database {

    private static final String URL = "jdbc:h2:mem:productos;DB_CLOSE_DELAY=-1";
    private static final String USER = "sa";
    private static final String PASS = "";

    public static void init() {
        try (Connection conn = DriverManager.getConnection(URL, USER, PASS)) {

            conn.createStatement().execute("""
                CREATE TABLE productos (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    name VARCHAR(100),
                    price DOUBLE
                );
            """);

            conn.createStatement().execute("""
                INSERT INTO productos (name, price) VALUES
                ('Teclado mecánico', 59.99),
                ('Ratón inalámbrico', 19.90),
                ('Auriculares gaming', 79.90),
                ('Alfombrilla XL', 10.00),
                ('Webcam HD', 29.95);
            """);

            System.out.println("[H2] Base de datos inicializada");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws Exception {
        return DriverManager.getConnection(URL, USER, PASS);
    }
}
```

7. Implementar servicio gRPC en src/main/java/service/ProductService.java

```java
package service;

import com.grpc.ej5.*;
import db.H2Database;
import io.grpc.stub.StreamObserver;

import java.sql.*;

public class ProductService extends
ProductServiceGrpc.ProductServiceImplBase {

    // 1. UNARY
```

```
 @Override
 public void getProductById(ProductIdRequest req,
                 StreamObserver<ProductResponse> obs) {

   System.out.println("[SERVER](UNARY) Recibido ID del cliente = " + req.getId());

   try (Connection conn = H2Database.getConnection()) {

     PreparedStatement ps = conn.prepareStatement(
         "SELECT id, name, price FROM productos WHERE id = ?");
     ps.setInt(1, req.getId());

     ResultSet rs = ps.executeQuery();

     if (!rs.next()) {
       obs.onError(new RuntimeException("Producto no encontrado"));
       return;
     }

     ProductResponse resp = ProductResponse.newBuilder()
         .setId(rs.getInt("id"))
         .setName(rs.getString("name"))
         .setPrice(rs.getDouble("price"))
         .build();

     System.out.println("[SERVER](UNARY) Enviando producto → " +
resp.getName());

     obs.onNext(resp);
     obs.onCompleted();

   } catch (Exception e) {
     obs.onError(e);
   }
 }


 // 2. SERVER STREAMING
 @Override
 public void streamCheaperProducts(PriceRequest req,
                    StreamObserver<ProductResponse> obs) {
```

```
        System.out.println("[SERVER](SERVERSTREAMING) Recibido maxPrice = " +
req.getMaxPrice());

        try (Connection conn = H2Database.getConnection()) {

            PreparedStatement ps = conn.prepareStatement(
                    "SELECT id, name, price FROM productos WHERE price <= ?");
            ps.setDouble(1, req.getMaxPrice());

            ResultSet rs = ps.executeQuery();

            while (rs.next()) {

                ProductResponse p = ProductResponse.newBuilder()
                        .setId(rs.getInt("id"))
                        .setName(rs.getString("name"))
                        .setPrice(rs.getDouble("price"))
                        .build();

                System.out.println("[SERVER](SERVERSTREAMING) Enviando → " +
p.getName());

                obs.onNext(p);
            }

            obs.onCompleted();

        } catch (Exception e) {
            obs.onError(e);
        }
    }


    // 3. CLIENT STREAMING
    @Override
    public StreamObserver<NewProductRequest> insertProducts(
            StreamObserver<InsertSummary> obs) {

        return new StreamObserver<>() {
```

```
        int count = 0;

        @Override
        public void onNext(NewProductRequest req) {
          System.out.println("[SERVER](CLIENTSTREAMING) Recibido producto → " +
req.getName());

          try (Connection conn = H2Database.getConnection()) {

            PreparedStatement ps = conn.prepareStatement(
                "INSERT INTO productos (name, price) VALUES (?, ?)");
            ps.setString(1, req.getName());
            ps.setDouble(2, req.getPrice());
            ps.executeUpdate();

            System.out.println("[SERVER](CLIENTSTREAMING) Insertado en BD");
            count++;

          } catch (Exception e) {
            obs.onError(e);
          }
        }

        @Override
        public void onCompleted() {
          System.out.println("[SERVER](CLIENTSTREAMING) Finalizando stream →
total insertados = " + count);

          InsertSummary summary = InsertSummary.newBuilder()
              .setInsertedCount(count)
              .build();

          obs.onNext(summary);
          obs.onCompleted();
        }

        @Override
        public void onError(Throwable t) {
          System.out.println("[SERVER](CLIENTSTREAMING) Error → " +
t.getMessage());
        }
```

```
      };
    }


    // 4. BIDI STREAMING
    @Override
    public StreamObserver<PriceRequest> livePriceQuery(
            StreamObserver<ProductResponse> obs) {

        return new StreamObserver<>() {

            @Override
            public void onNext(PriceRequest req) {
                System.out.println("[SERVER](BIDIRECTIONAL) Recibido maxPrice → " +
req.getMaxPrice());

                try (Connection conn = H2Database.getConnection()) {

                    PreparedStatement ps = conn.prepareStatement(
                            "SELECT id, name, price FROM productos WHERE price <= ?");
                    ps.setDouble(1, req.getMaxPrice());

                    ResultSet rs = ps.executeQuery();

                    while (rs.next()) {

                        ProductResponse p = ProductResponse.newBuilder()
                                .setId(rs.getInt("id"))
                                .setName(rs.getString("name"))
                                .setPrice(rs.getDouble("price"))
                                .build();

                        System.out.println("[SERVER](BIDIRECTIONAL) Enviando → " +
p.getName());

                        obs.onNext(p);
                    }

                } catch (Exception e) {
                    obs.onError(e);
                }
      }

      @Override
      public void onCompleted() {
        System.out.println("[SERVER](BIDIRECTIONAL) Cliente finalizó stream.");
        obs.onCompleted();
      }

      @Override
      public void onError(Throwable t) {
        System.out.println("[SERVER](BIDIRECTIONAL) Error → " + t.getMessage());
      }
    };
  }
}
```

8. Implementar servidor en src/main/java/server/ProductServer.java

```java
package server;

import db.H2Database;
import io.grpc.Server;
import io.grpc.ServerBuilder;
import service.ProductService;

public class ProductServer {

  public static void main(String[] args) throws Exception {

    H2Database.init();

    Server server = ServerBuilder
        .forPort(50051)
        .addService(new ProductService())
        .build();

    server.start();
    System.out.println("Servidor gRPC con H2 ejecutándose en puerto 50051...");

    server.awaitTermination();
    }
}
```

9. Implementar cliente en src/main/java/client/ProductClient.java

```java
package client;

import com.grpc.ej5.*;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;

import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

public class ProductClient {

    private final ManagedChannel channel;
    private final ProductServiceGrpc.ProductServiceBlockingStub blockingStub;
    private final ProductServiceGrpc.ProductServiceStub asyncStub;

    public ProductClient() {
        channel = ManagedChannelBuilder
                .forAddress("localhost", 50051)
                .usePlaintext()
                .build();

        blockingStub = ProductServiceGrpc.newBlockingStub(channel);
        asyncStub = ProductServiceGrpc.newStub(channel);
    }

    // 1. UNARY
    public void unary() {

        ProductIdRequest req = ProductIdRequest.newBuilder().setId(1).build();
        System.out.println("[CLIENT](UNARY) Enviando ID = " + req.getId());

        ProductResponse resp = blockingStub.getProductById(req);

        System.out.println("[SERVER](UNARY) Recibido producto:");
        System.out.println("   Nombre = " + resp.getName());
        System.out.println("   Precio = " + resp.getPrice());
    }


    // 2. SERVER STREAMING
    public void serverStreaming() {

        PriceRequest req = PriceRequest.newBuilder().setMaxPrice(50).build();
        System.out.println("[CLIENT](SERVERSTREAMING) Max price = " +
req.getMaxPrice());

        System.out.println("[SERVER](SERVERSTREAMING) Productos recibidos:");

        blockingStub.streamCheaperProducts(req).forEachRemaining(
            p -> System.out.println("   → " + p.getName() + " | " + p.getPrice())
        );
    }


    // 3. CLIENT STREAMING
    public void clientStreaming() throws Exception {

        CountDownLatch latch = new CountDownLatch(1);

        StreamObserver<InsertSummary> respObs = new StreamObserver<>() {
            public void onNext(InsertSummary s) {
                System.out.println("[SERVER](CLIENTSTREAMING) Insertados total = " +
s.getInsertedCount());
            }
            public void onError(Throwable t) { latch.countDown(); }
            public void onCompleted() { latch.countDown(); }
        };

        StreamObserver<NewProductRequest> reqStream =
asyncStub.insertProducts(respObs);

        // ENVÍO 1
        NewProductRequest p1 = NewProductRequest.newBuilder()
            .setName("Mesa gaming").setPrice(120).build();
        System.out.println("[CLIENT](CLIENTSTREAMING) Enviando → " +
p1.getName() + "con precio " + p1.getPrice());
        reqStream.onNext(p1);

        // ENVÍO 2
        NewProductRequest p2 = NewProductRequest.newBuilder()
            .setName("Monitor 27''").setPrice(199).build();
        System.out.println("[CLIENT](CLIENTSTREAMING) Enviando → " +
p2.getName() + "con precio " + p2.getPrice());
        reqStream.onNext(p2);

        reqStream.onCompleted();
        latch.await();
    }


    // 4. BIDIRECTIONAL STREAMING
    public void bidiStreaming() throws Exception {

        CountDownLatch latch = new CountDownLatch(1);

        StreamObserver<ProductResponse> respObs = new StreamObserver<>() {

            public void onNext(ProductResponse p) {
            System.out.println("[SERVER](BIDIRECTIONAL) Producto → " + p.getName()
+ " | " + p.getPrice());
            }

            public void onError(Throwable t) { latch.countDown(); }

            public void onCompleted() {
            System.out.println("[SERVER](BIDIRECTIONAL) Fin del stream");
            latch.countDown();
            }
        };

        StreamObserver<PriceRequest> reqStream =
asyncStub.livePriceQuery(respObs);

        PriceRequest r1 = PriceRequest.newBuilder().setMaxPrice(20).build();
        System.out.println("[CLIENT](BIDIRECTIONAL) MaxPrice enviado = " +
r1.getMaxPrice());
        reqStream.onNext(r1);
        Thread.sleep(500);

        PriceRequest r2 = PriceRequest.newBuilder().setMaxPrice(60).build();
        System.out.println("[CLIENT] (BIDIRECTIONAL) MaxPrice enviado = " +
r2.getMaxPrice());
        reqStream.onNext(r2);
        Thread.sleep(500);

        PriceRequest r3 = PriceRequest.newBuilder().setMaxPrice(200).build();
        System.out.println("[CLIENT](BIDIRECTIONAL) MaxPrice enviado = " +
r3.getMaxPrice());
        reqStream.onNext(r3);

        reqStream.onCompleted();
        latch.await();
    }

    public void shutdown() throws InterruptedException {
        channel.shutdown().awaitTermination(2, TimeUnit.SECONDS);
    }

    public static void main(String[] args) throws Exception {
        ProductClient c = new ProductClient();
        c.unary();
        c.serverStreaming();
        c.clientStreaming();
        c.bidiStreaming();
        c.shutdown();
    }
}
```

10. Empaquetar

```bash
mvn clean package
```

11. Ejecutar

```bash
# Terminal 1: servidor
mvn exec:java -Dexec.mainClass=server.ProductServer

# Terminal 2: cliente
mvn exec:java -Dexec.mainClass=client.ProductClient
```

12. Flujo esperado
- Servidor arranca H2 en memoria con datos de ejemplo.
- Cliente realiza cuatro llamadas: unary, server-streaming, client-streaming (inserta y recibe resumen) y bidirectional-streaming (consultas dinámicas).
- Ajusta IDs o precios en el cliente para probar otros casos.