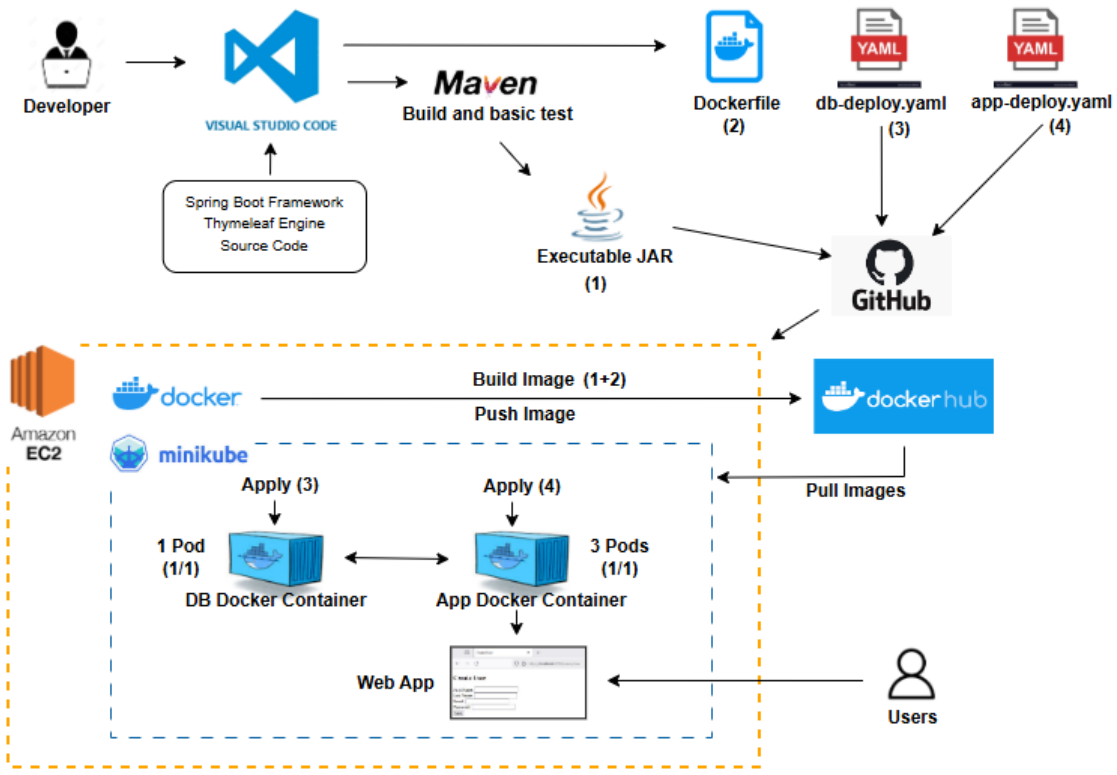
CREATE AND DEPLOY A SPRING BOOT CRUD APP WITH DOCKER AND KUBERNETES ON AWS

Prepared By: Enric Roca Abenza

This project provides a comprehensive overview of developing and deploying a **Spring Boot CRUD application**. I will guide you through the entire process, including:

1. **Application Setup**: Creating a sample Spring Boot CRUD application.
2. **Containerization & Orchestration Prep**: Developing the necessary configuration files, specifically the Dockerfile and Kubernetes deployment YAMLs.
3. **Cloud Infrastructure**: Launching an EC2 instance on AWS and installing all required deployment software (Docker, conntrack, Git, Minikube, kubectl, and Maven).
4. **Image Creation & Registry**: Building the Docker image for our application and pushing it to a container registry.
5. **Deployment & Troubleshooting**: Applying the Kubernetes deployment files to Minikube, systematically identifying and resolving various integration and runtime issues along the way.
6. **Validation**: Finally, verifying that the deployed solution is fully functional.

The architecture diagram can be summarized as follows:



In laptop

Part I: Create a Spring Boot Application
- I.1- Run and test the MySQL connection
- I.2- Create the DB and check it
- I.3- Create the Visual Studio project
- I.4- Change the DB connection settings in the App
- I.5- Create and modify project files
- I.6- Check that the application works
- I.7- Create the executable JAR file

Part II: Create the configuration files to deploy the App
- II.1- Create the Dockerfile
- II.2- Create the db deployment yaml file
- II.3- Create the app deployment yaml file
- II.4- Upload the files to GitHub

In AWS instance

Part III: Launch an instance on AWS and install the software
- III.1- Launch the Linux instance on AWS
- III.2- Install Docker, Minikube, kubectl, Git and Maven

Part IV: Download the files and deploy the solution
- IV.1- Download the files from GitHub to the instance
- IV.2- Re-built the executable jar file in the pipeline server
- IV.3- Create the Docker image and upload it to Docker Hub
- IV.4- Apply the db-deploy.yaml file
- IV.5- Apply the app-deploy.yaml file
- IV.6- Configure connectivity

Part V: Check that the solution works
- V.1- Test the App and values in DB
- V.2- Check Minikube Dashboard
- V.3- Summary and Conclusions

Part I: Create a Spring Boot Application

A **CRUD Spring Boot application** is a software application built using the **Spring Boot** framework that primarily implements the four basic functions of persistent storage: **Create**, **Read**, **Update**, and **Delete** (**CRUD**).

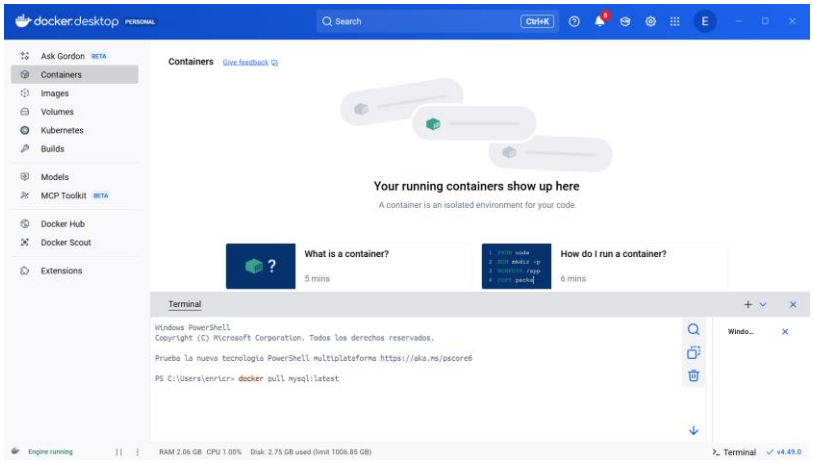
Additionally, our project will employ a template engine called **Thymeleaf** to generate dynamic web pages for the user interface. So, we will see that by writing few lines of code we get all the functionalities needed.

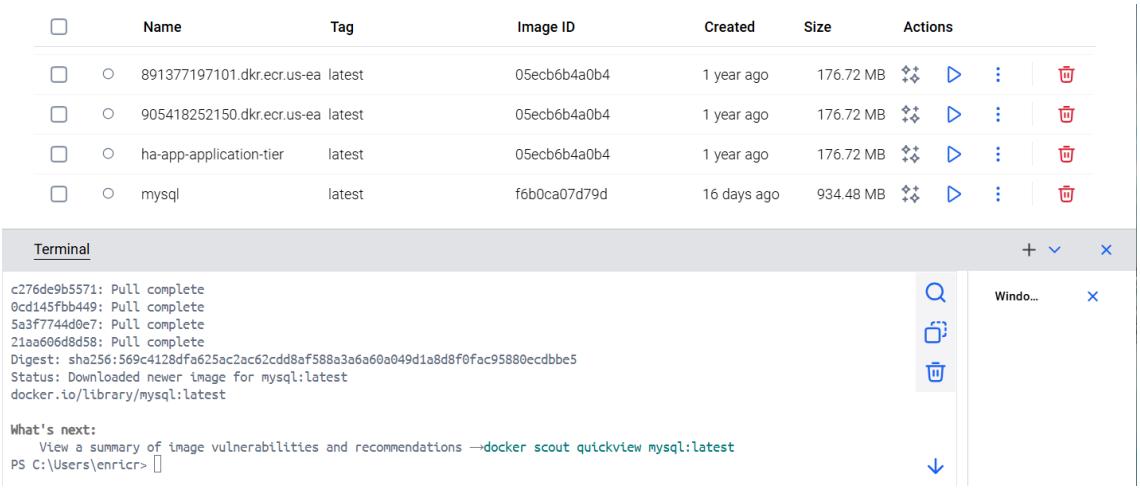
As development environment we will use **Visual Studio Code**, and we will need a **MySQL** server available. If you don't have one, I will explain how to get one up and running in a container using **Docker Desktop** in your machine.

I.1- Run and test the MySQL connection

You will need the data source parameters to connect to your MySQL server DB, this is: the connection URL (to a DB that will see later), the username and the password.

In my case, I don't currently have a MySQL engine installed on my laptop, so I use **Docker Desktop** running a basic server instance with the **mysql:latest** image on port 3306. If you want to do the same and don't have Docker Desktop in your machine, you can install it from https://docs.docker.com/desktop/install/windows-install/.
Once Docker Desktop is installed, using its terminal, download the MySQL image with "**docker pull mysql:latest**"





and then run the container with the command:

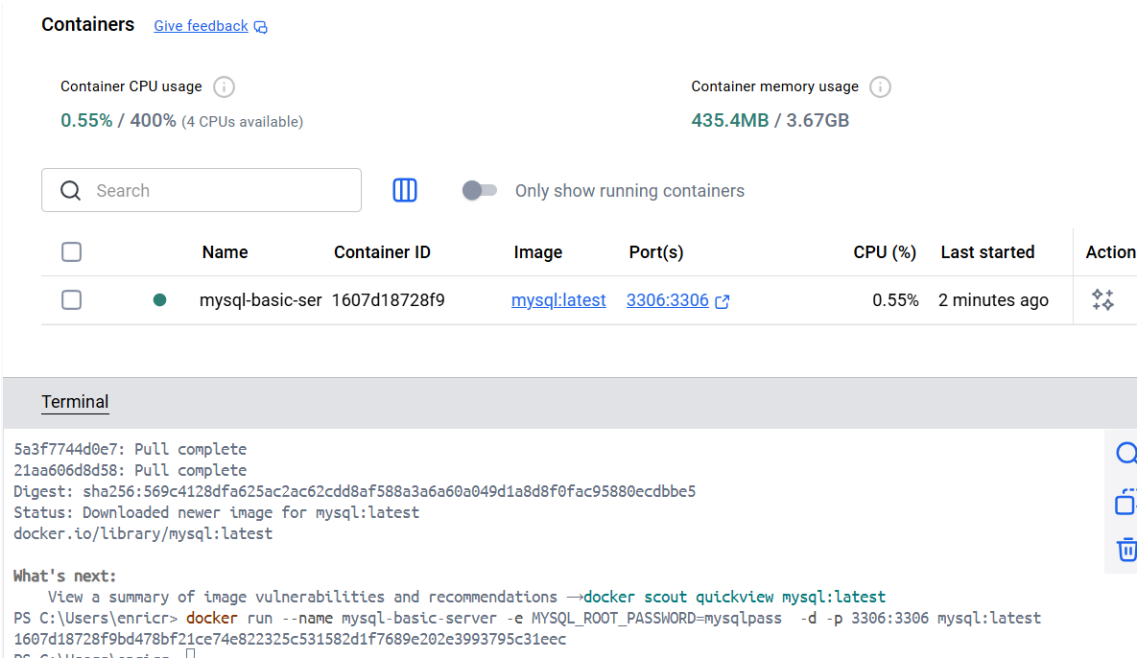
docker run --name mysql-basic-server -e MYSQL_ROOT_PASSWORD=mysqlpass -d -p 3306:3306 mysql:latest

Where:

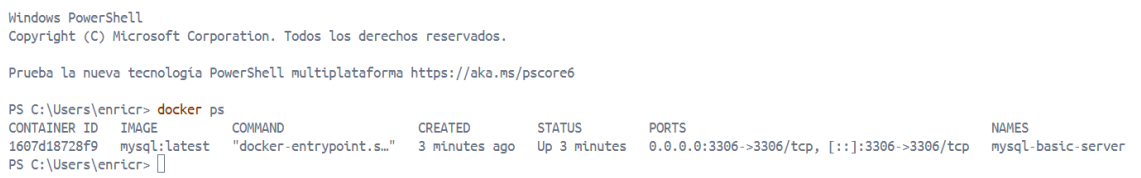
- **docker run** → This is the main command to create and start the container.
- **--name mysql-basic-server** → mysql-basic-server is the name that I give to the container.
- **-e MYSQL_ROOT_PASSWORD=mysqlpass** → Sets the environment variable MYSQL_ROOT_PASSWORD inside the container with the password mysqlpass, as the image mysql:latest uses it.
- **-d** → Runs the container in the background (detached mode)
- **-p 3306:3306** → Allow access from your host machine on port 3306 to the MySQL server inside the container om port 3306 (Format: host_port:container_port).
- **mysql:latest** → It is the Docker Image to use.

Since this is for temporary testing, I'm not using a persistent volume, so the data will be lost when the container terminates.

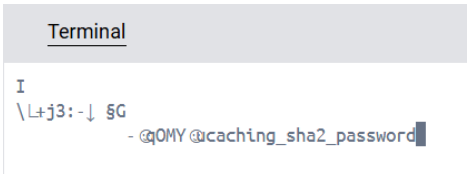
After executing this command, we can see my container running:



To check that my container is running, also in the Docker Desktop terminal I use "**docker ps**". We can see our container mysql-basic-server up and running:



We can do a simple test to check the port executing in the terminal or in the command prompt "**telnet localhost 3306** ". If there is an answer similar that this:



This means that there is a process listening on that port and that is asking the password, it must be our MySQL.

I.2- Create the DB and check it

Now that we have a MySQL server running, we must create our database that we will name **springCrud**.

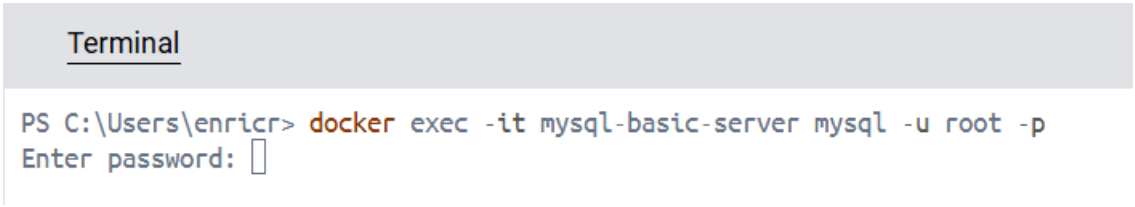
In my case, I go to execute the MySQL client inside the container. So, in the Docker Desktop terminal y do:

docker exec -it mysql-basic-server mysql -u root -p

In detail:

- **docker exec** → Runs a command in a running container.
- **-it** → For an interactive session with the MySQL client.
- **mysql-basic-server** → The name of my container
- **mysql -u root -p** → The command to execute inside the container to start the MySQL client, specifying the user (-u) and to prompt asking you for the password (-p)

It asks for the password:



I write **mysqlpass**, the one that I specified when creating the container, and push ENTER.

```
PS C:\Users\enricr> docker exec -it mysql-basic-server mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 9.5.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

To create the new database with the name **springCrud**, in the MySQL command-line interface:

```
CREATE DATABASE springCrud;
```

```
mysql> CREATE DATABASE springCrud;
Query OK, 1 row affected (0.548 sec)

mysql>
```

To verify its creation by listing all databases:

SHOW DATABASES;

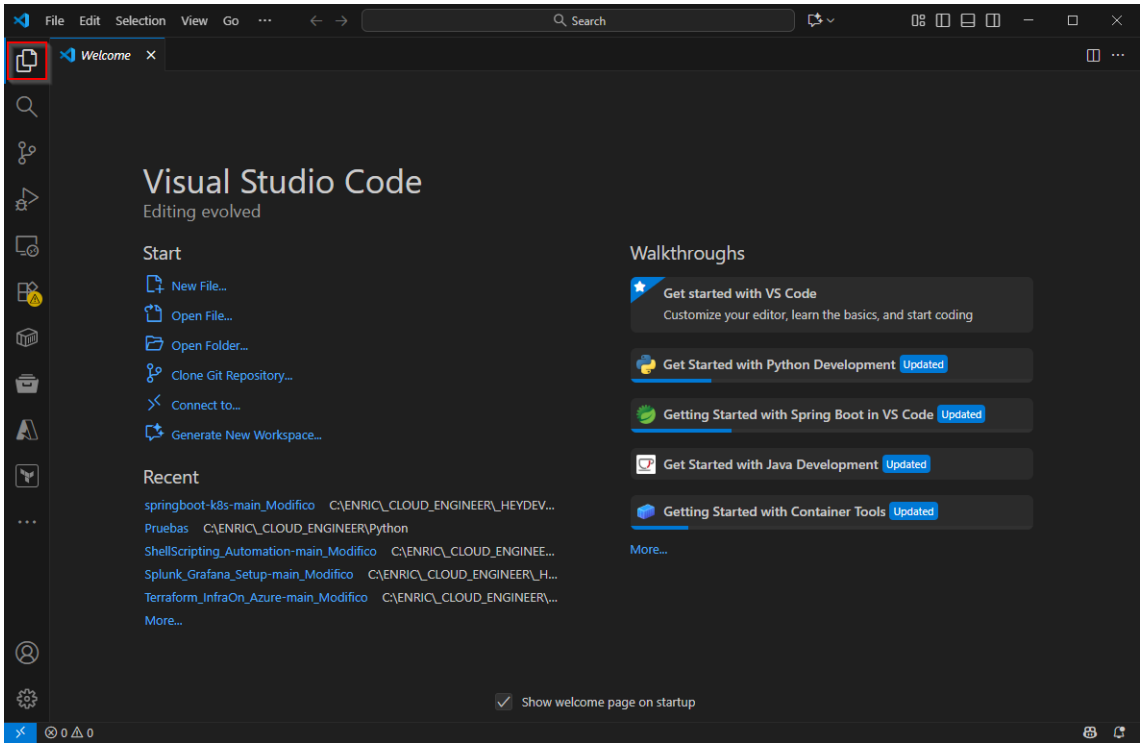
```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| springCrud         |
| sys                |
+--------------------+
5 rows in set (0.137 sec)

mysql>
```

So, the data source parameters to connect to the database with a jdbc data provider will be:

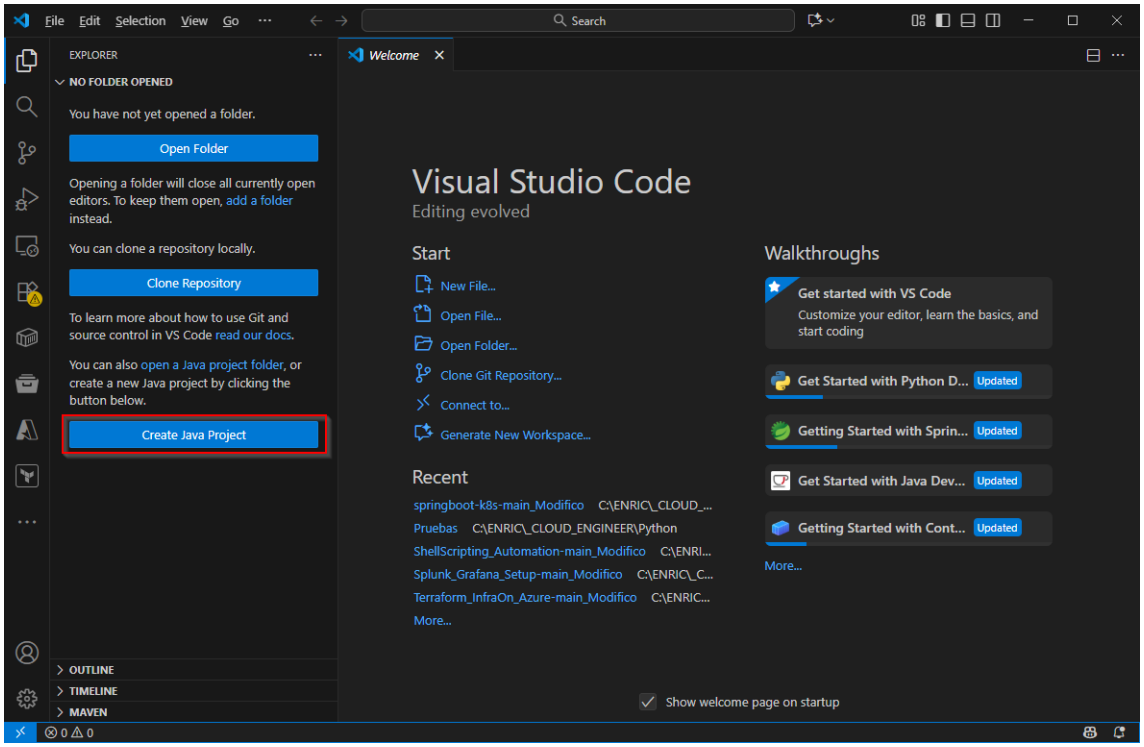
url=**jdbc:mysql://localhost:3306/springCrud**
username=**root**
password= **mysqlpass**

I.3- Create the Visual Studio project

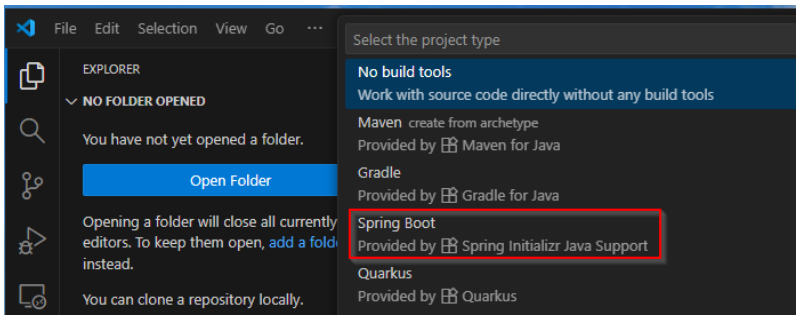
In Visual Studio Code, select the **Explorer** icon:



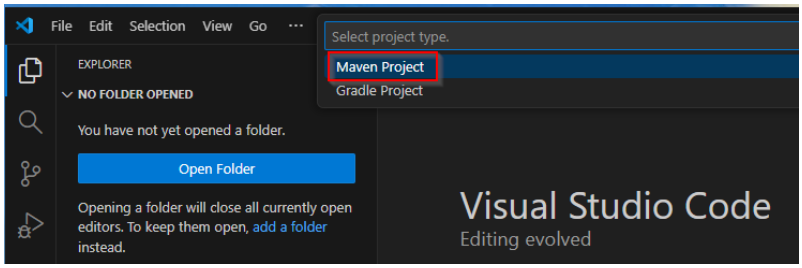
Click on **Create Java Project**:



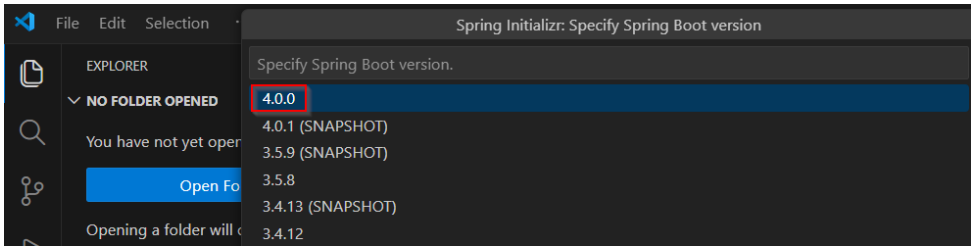
The Project Will be **Spring Boot**:



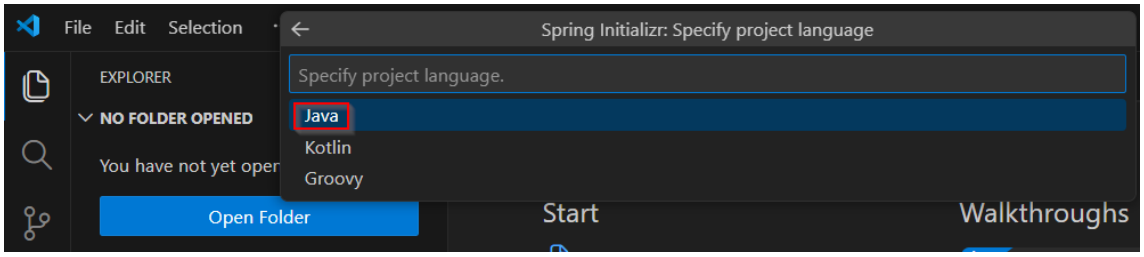
And **Maven Project**:



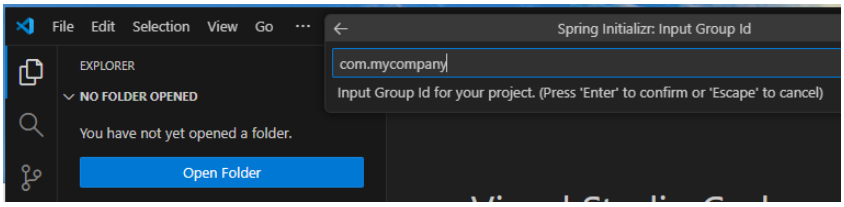
The version must be **4.0.0** :



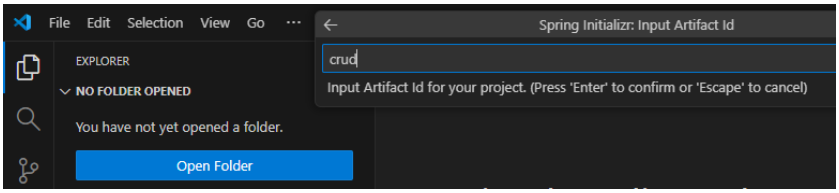
Java:

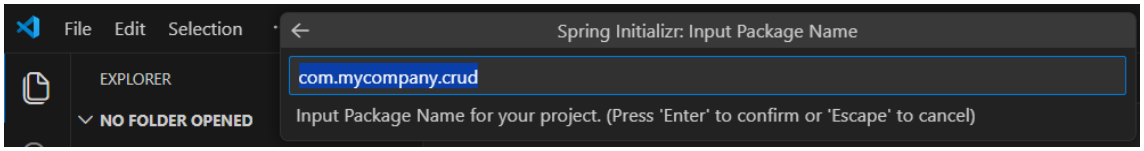


Enter the name of the **Group Id** for the project. I write **com.mycompany**:

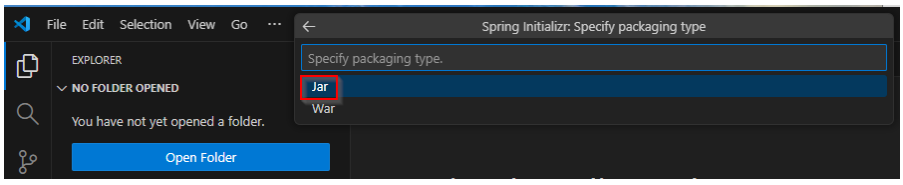


As **Artifact Id** of our Project, I put **crud** :

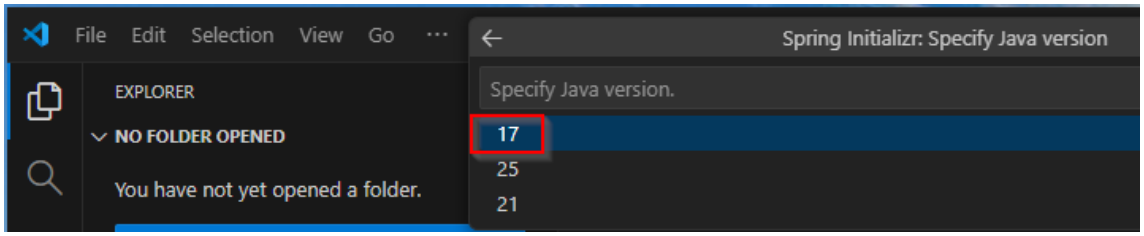




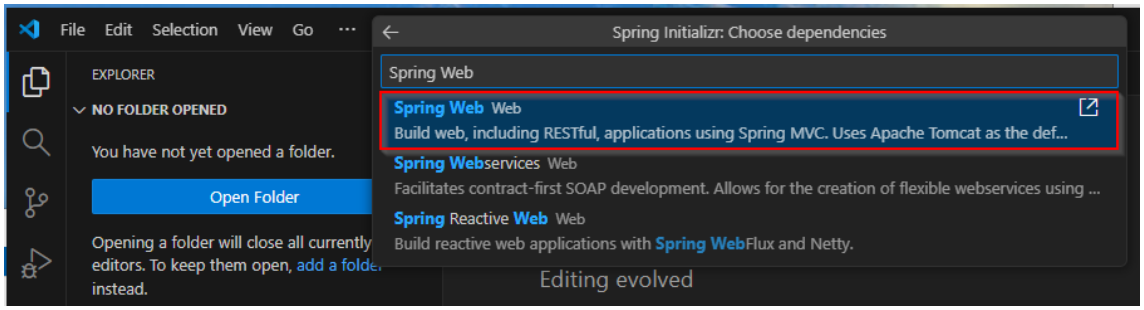
Then select **Jar**:

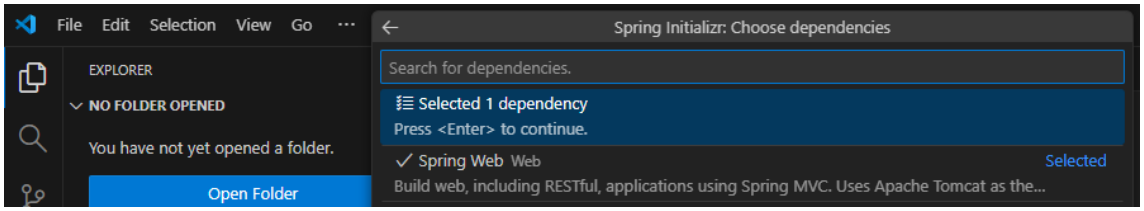


Version **17** :



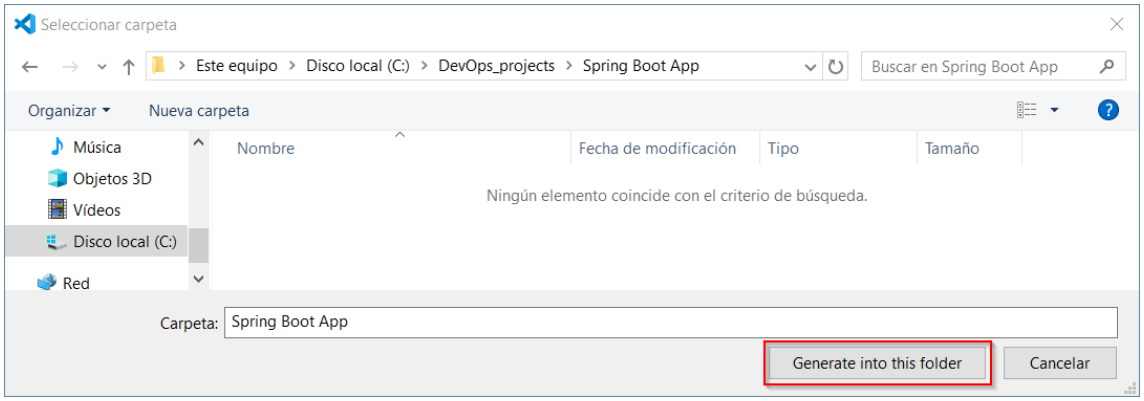
Look for **Spring Web**:



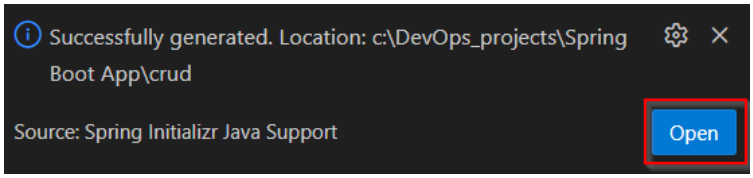


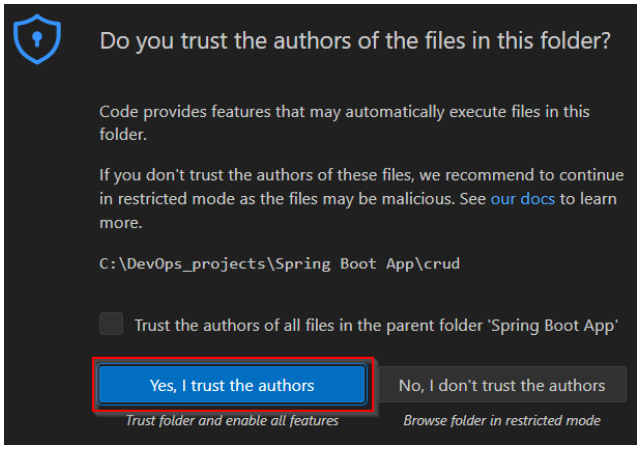
Press <Enter>

Now the Windows Explorer is opened and I select the folder where to create the project, in my case, C:\DevOps_projects\Spring Boot App. Press "**Generate into this folder**":

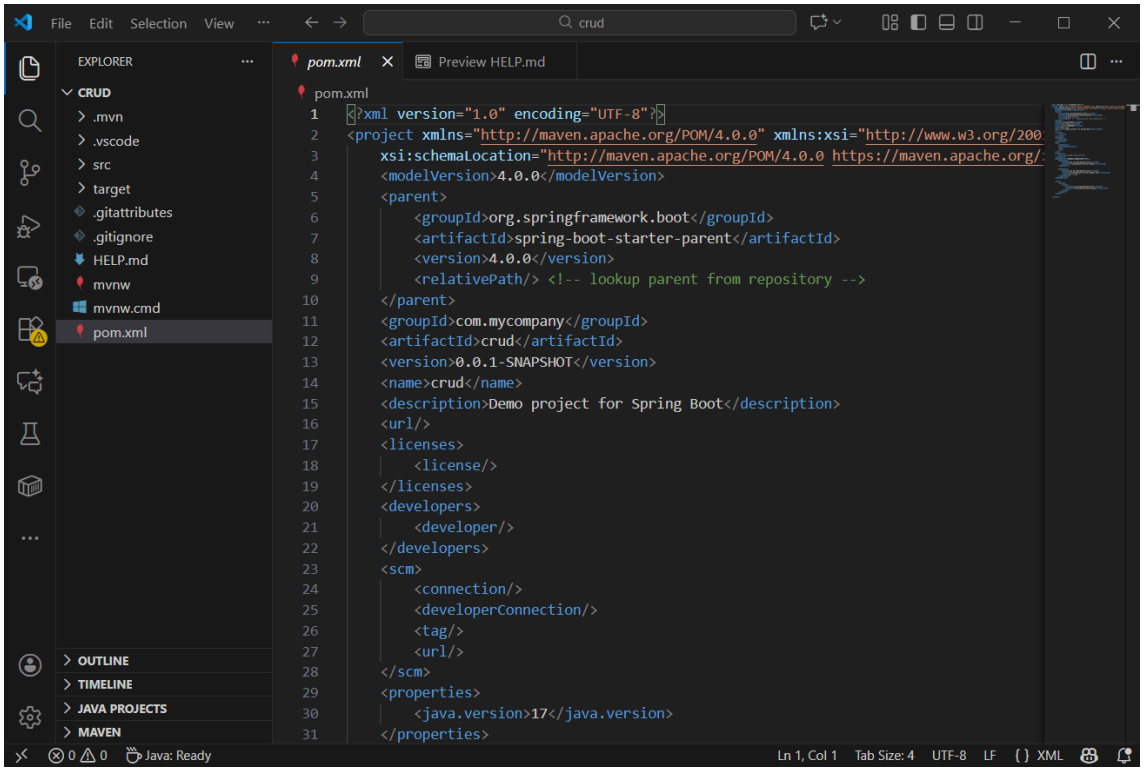


A message appears in the bottom right corner. Click in **Open** to open the Project:





In the project we can see different folders and files. Open the **pom.xml** file:



```
<?xml version="1.0" encoding="UTF-8"?>
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>4.0.0</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.mycompany</groupId>
	<artifactId>crud</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>crud</name>
	<description>Demo project for Spring Boot</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
		<url/>
	</scm>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-webmvc</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-webmvc-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

🗐 This is typical **pom.xml** file of a Spring Boot **CRUD** (Create, Read, Update. Delete) application, managed by **Maven**. The important parts are:

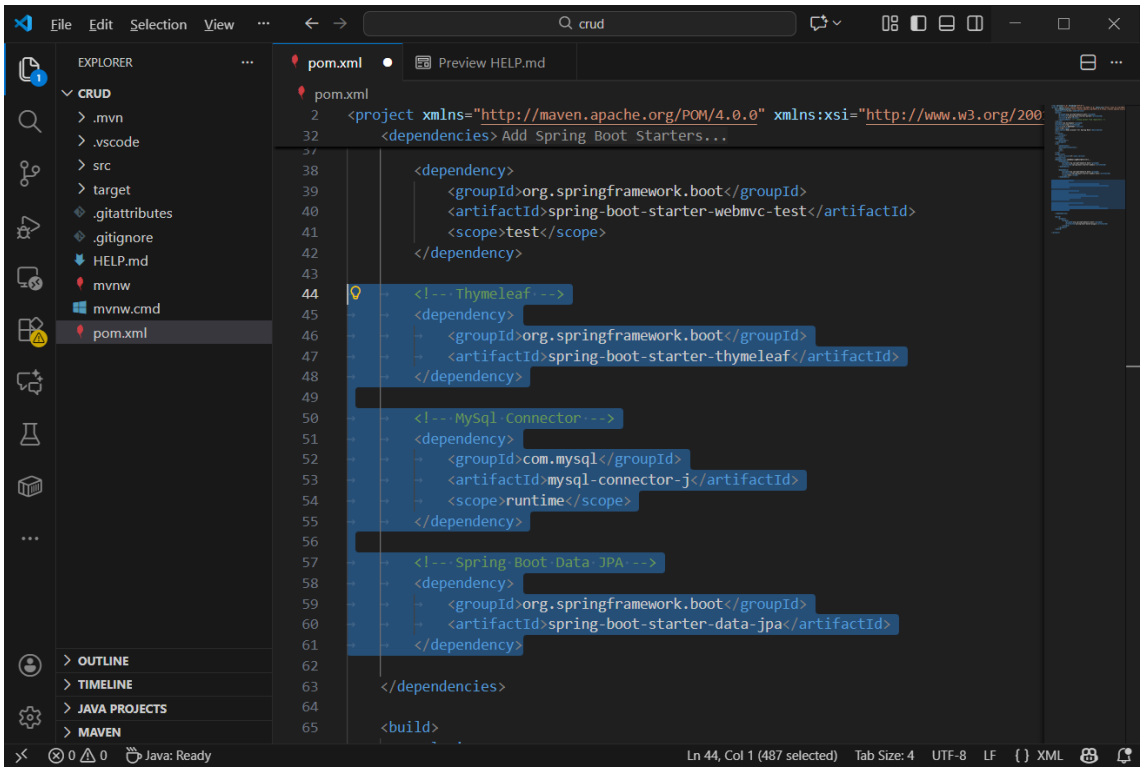
- **Project Identification and Inheritance** (<parent>, <groupId>, <artifactId>, <version>): **<parent>** specifies the parent project from which our application inherits its configuration, dependency versions, and default settings; **<groupId>** is the identifier of our organization (com.mycompany); **<artifactId>** is the name of our project (crud); **<version>** is the current version of our project (0.0.1-SNAPSHOT).
- **Properties** (<properties>). This section defines reusable variables throughout the pom.xml: **<java.version>** is the Java version for compiling and running the application (17).
- **Dependencies** (<dependencies>): JAR libraries needed for the application to function. For a Spring Boot CRUD application, these are the typical essentials:
  - **spring-boot-starter-webmvc**: It automatically bundles all necessary components for building a Web MVC (Model-View-Controller) application, which typically includes the Spring Web Framework and an embedded web server (like Tomcat). This indicates the project is a web application (like a REST API).
  - **spring-boot-starter-webmvc-test**: This starter provides libraries for testing Spring MVC applications. **<scope>test</scope>**: Indicates that this dependency is only required for compiling and running tests, not in the final production JAR.
- **Build Configuration** (<build>): This defines how the project should be constructed. **spring-boot-maven-plugin** packages the application as a single, executable JAR (or WAR) file, including all dependencies and the embedded server.

Now we are going to add the next code in the **<dependencies>** section. So, copy the next lines:

```
<!-- Thymeleaf -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<!-- MySql Connector -->
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>
</dependency>

<!-- Spring Boot Data JPA -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

and paste them before the </dependencies> label.

Once done:



The added Dependencies have the nest purposes:

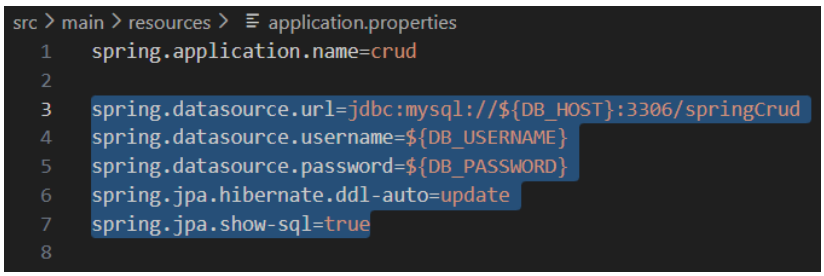
- **spring-boot-starter-thymeleaf**: Library that implements the generation of dynamic web pages for the user interface.
- **mysql-connector-j**: Database drive that allows the application to connect to a MySQL system.
- **spring-boot-starter-data-jpa**: Enables interaction with relational databases using the Java Persistence API (JPA) and Hibernate (the default implementation).

I.4- Change the DB connection settings in the App

To set the DB connection parameters in our App we must go to **CRUD > src > main > resources** and open the file **application.properties**. Change the values to:

```
spring.application.name=crud

spring.datasource.url=jdbc:mysql://${DB_HOST}:3306/springCrud
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

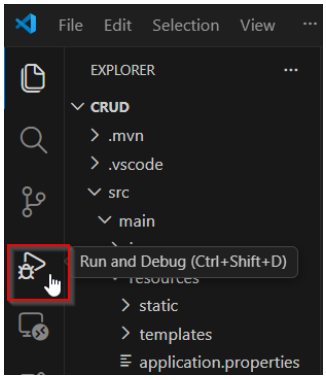
When done:



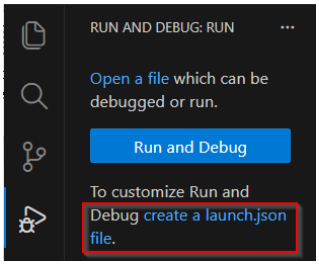
Where:

- **DB_HOST**, **DB_USERNAME** and **DB_PASSWORD** are environment variables for the host, username and password respectively to access to the MySQL instance. So, we can pass this information as parameters when executing the application.
- **spring.jpa.hibernate.ddl-auto=update** → update means Hibernate will automatically update the database schema (create, alter, or drop tables/columns).
- **spring.jpa.show-sql=true** → true makes Hibernate will log and print every SQL statement it executes to the console, very useful for debugging and seeing what the application is doing.
  (*Hibernate* is an Object-Relational Mapping (ORM) that allows interact with relational databases, like MySQL, using objects from our code instead of writing raw SQL queries).

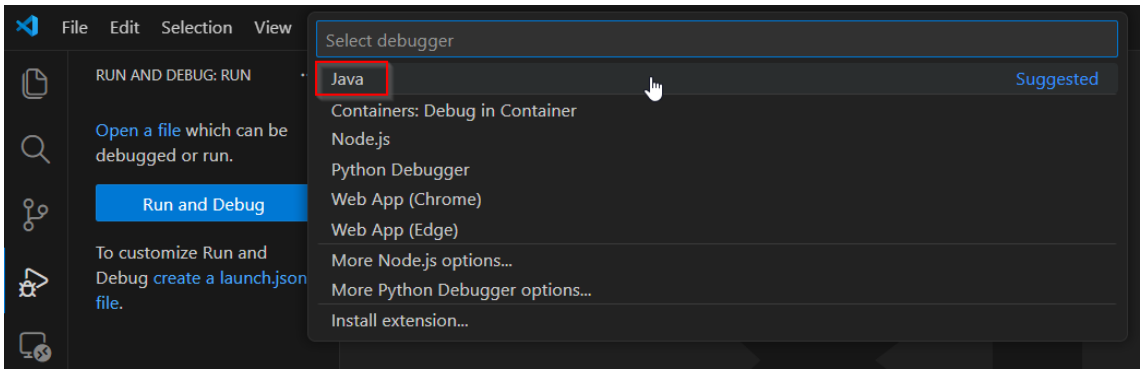
Now we must configure the VS Code project to add these environment variables, so when select run (Debug) automatically pass these parameters. For that, we can use "**Run and Debug**" icon on the left-side activity bar, or we can use the keyboard shortcut Ctrl + Shift + D:



Click on the link that says "**create a launch.json file**":



Select **Java**:

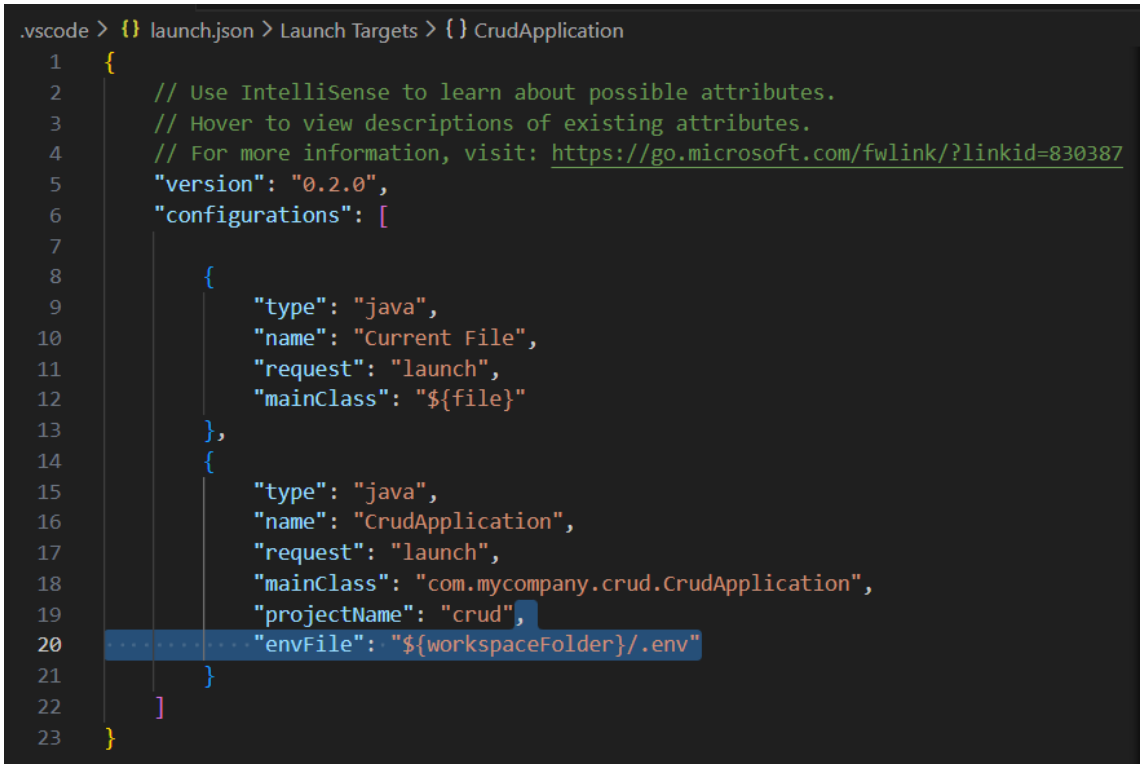


Then it appears the **launch.json** file:

Now we are going to add a “envFile” property to load environment variables from a file. So, add the line:

`"envFile": "${workspaceFolder}/.env"`

after ““projectName": "crud"”. Don’t forget to include a “,” between them. Once done:

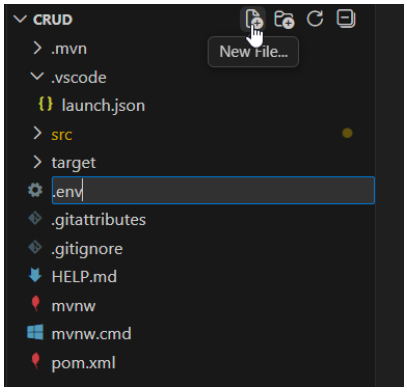



```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
```

```
  "configurations": [

    {
        "type": "java",
        "name": "Current File",
        "request": "launch",
        "mainClass": "${file}"
    },
    {
        "type": "java",
        "name": "CrudApplication",
        "request": "launch",
        "mainClass": "com.mycompany.crud.CrudApplication",
        "projectName": "crud",
        "envFile": "${workspaceFolder}/.env"
    }
  ]
}
```

Save the file.

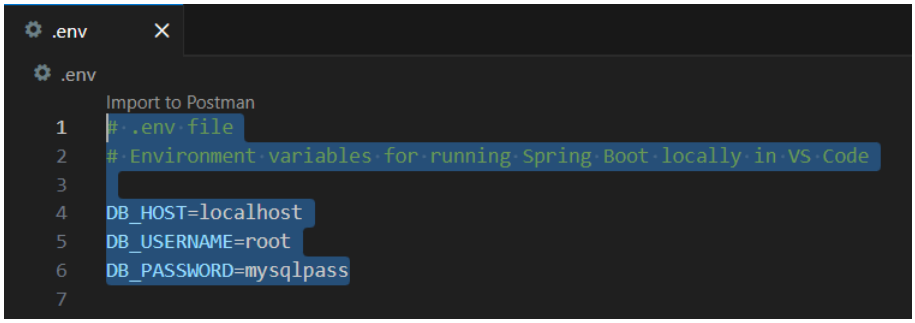
The next is to create a file that must be named **.env** and write on it the environment variables and their values. For that, in the root folder of the project we create a new file with the name .env:



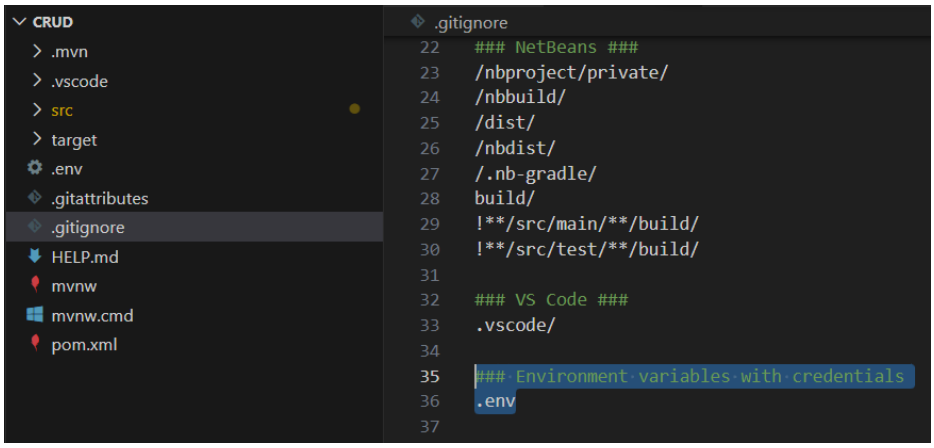
Copy the next code:

```
# .env file
# Environment variables for running Spring Boot locally in VS Code

DB_HOST=localhost
DB_USERNAME=root
DB_PASSWORD=mysqlpass
```

And paste it inside the file:

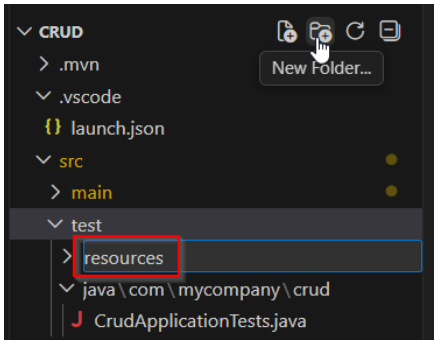


There is an additional recommendation. If we plan to use the Git version control, it's important to add the **.env** file to the **.gitignore** file of the project to avoid uploading sensitive credentials (local passwords, etc.) to the public repository. Once done:



Another thing we must do is to configure the **application testing context**, because later when we create the jar executable of our App, Maven will look for the test scope to do some basic unit test. If this scope doesn't exist it will crash into the building.

First, we must create a file named **application-test.properties** in the folder **CRD > src > test > resources**. As the **resources** directory doesn't exist, we create it:

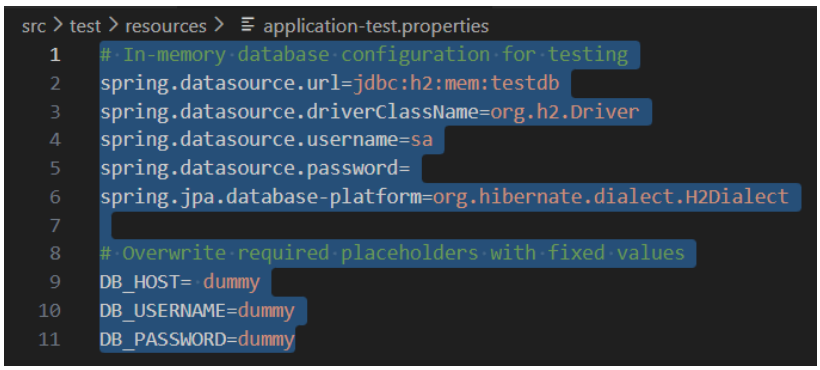


Copy the code:

```
# In-memory database configuration for testing
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
# Overwrite required placeholders with fixed values
DB_HOST= dummy
DB_USERNAME=dummy
DB_PASSWORD=dummy
```

And paste inside the file:

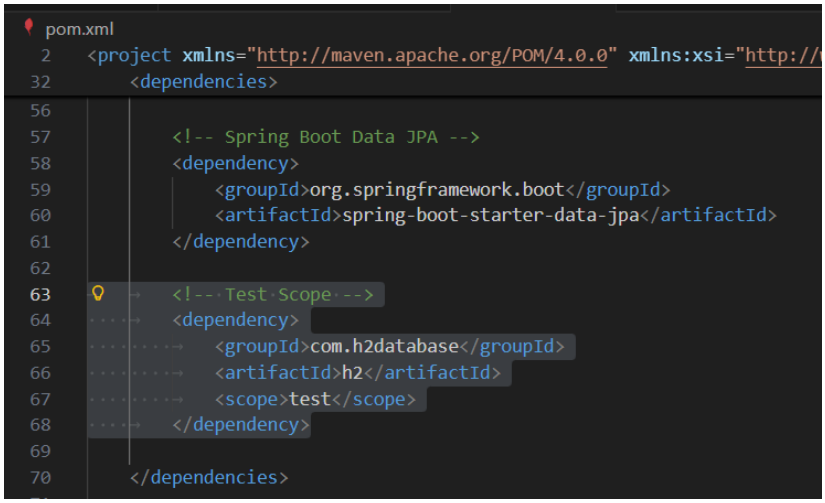


This file defines the data source properties that Spring Boot needs to load the test context. Otherwise, when creating the Java executable (JAR) with "mvn clean package", we'll get an error saying that it can't find the application context during the execution of the unit tests (it doesn't know the .env file). To solve this, we use an in-memory database (H2) with dummy values, since it only checks for the existence of a string and doesn't attempt to connect to the database.

Additionally, we need to add a dependency to **pom.xml** to include the test scope. Copy the code:

```
<!-- Test Scope -->
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
  </dependency>
```

and paste it before the label </dependencies> in the pom.xml file:



Finally, to get the test scope ready, we must add the line

`@ActiveProfiles("test")`

in the file **CRUD > src > java > com > mycompany > crud > CrudApplicationTest.java**

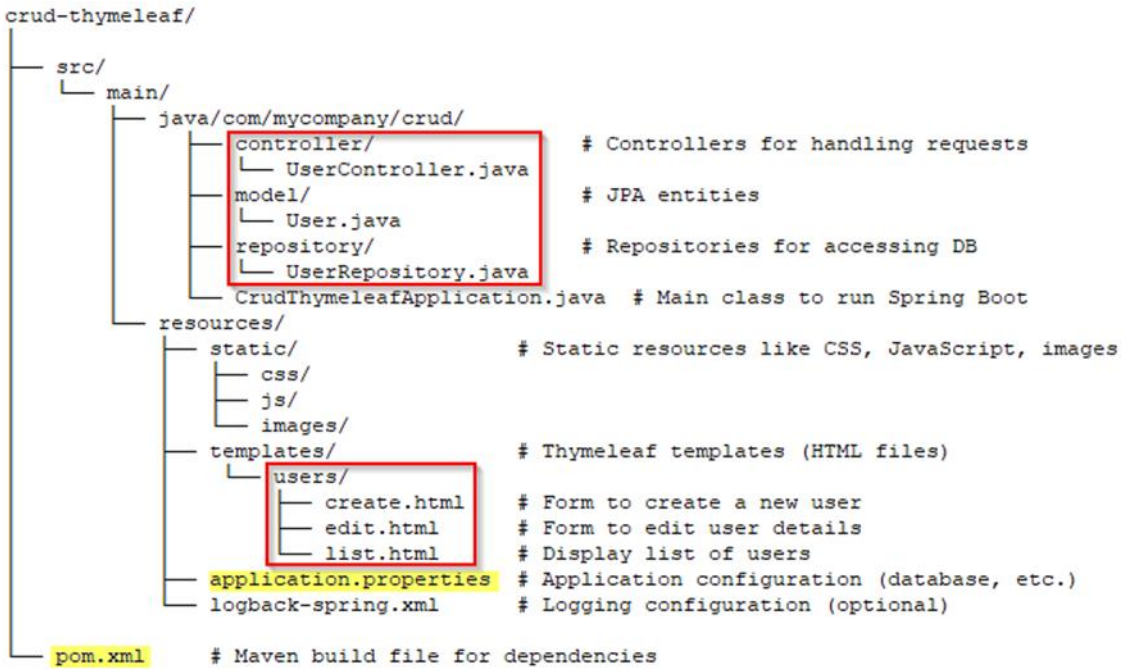
before the definition of the class:

src > test > java > com > mycompany > crud > CrudApplicationTests.java > CrudApplicationTests

```java
package com.mycompany.crud;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ActiveProfiles;

@SpringBootTest
@ActiveProfiles("test")
class CrudApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

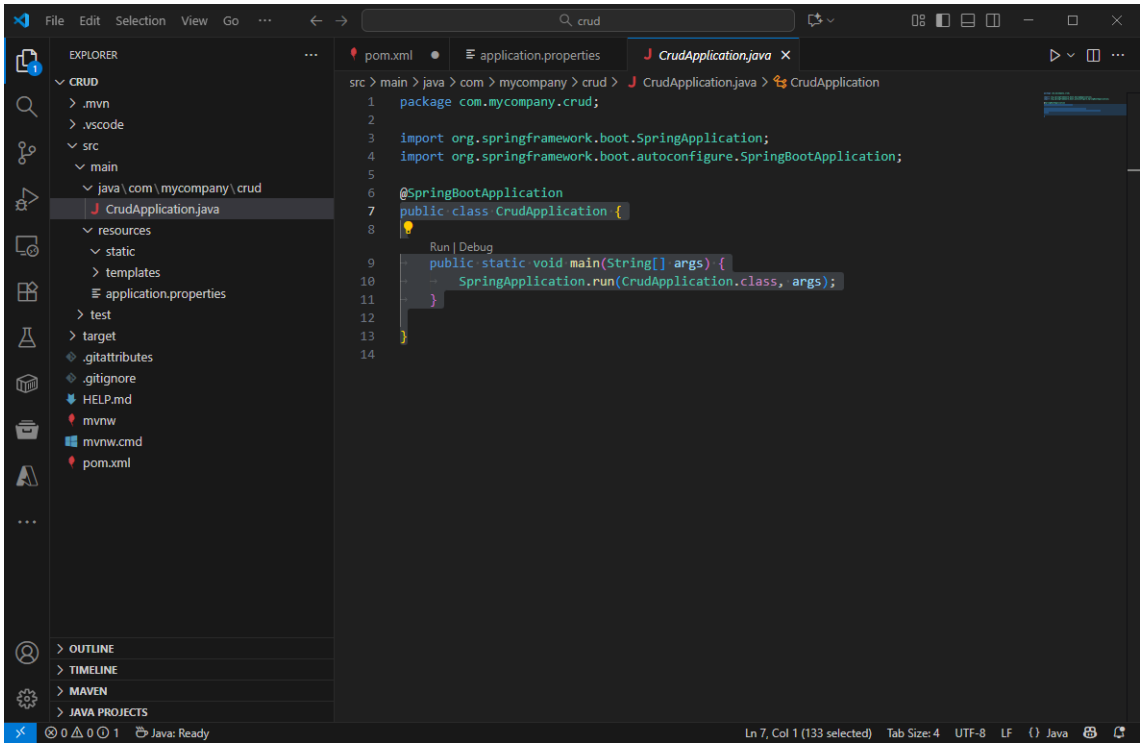
I.5- Create and modify project files

The typical structure of a CRUD-Thymeleaf application can be seen in the next image. The folders and files that we will create are shown inside red squares, and in yellow the files that we modify:

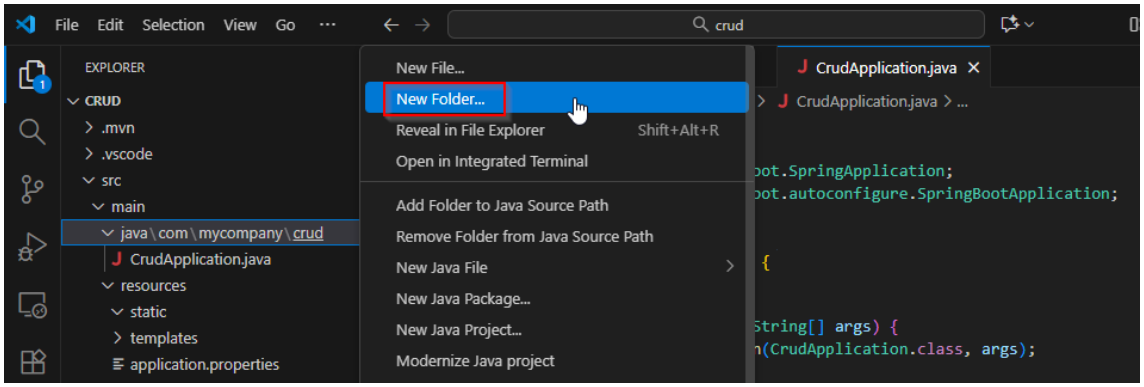


We will see step by step how to proceed with writing the needed code in VS Code.

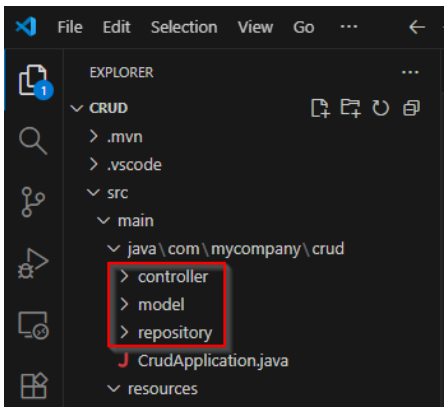
The class for our application already exists. We can see it in:



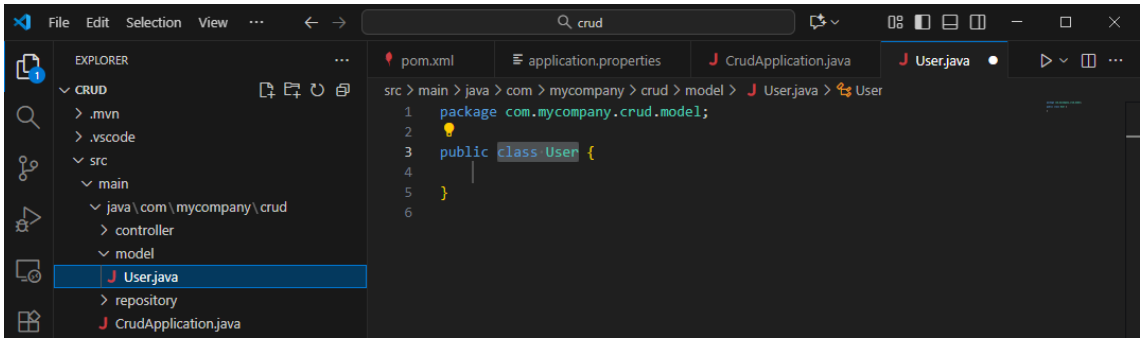
We need to create the **User.java** model class. For that, we go to **CRUD > src > main > java** and create the folder **model**:



and the folders **controller** and **repository** in the same way. So, now we have these three new directories:



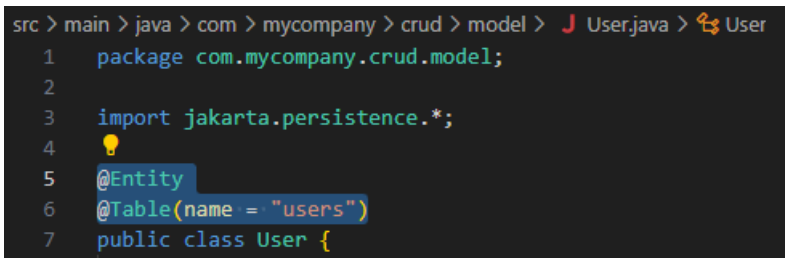
To create the **User.java** model, in VS Code go to **CRUD > src > main > java > model** folder and create a file with the name **User.java**. When created, it appears with the next code for default:



Copy the next code:

```
import jakarta.persistence.*;

@Entity
@Table(name = "users")
```

And paste it before the definition of the class. When done:



Now copy the code:

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

 @Column(name = "first_name")
 private String firstName;

 @Column(name = "last_name")
 private String lastName;

 @Column(name = "email", unique = true)
 private String email;

 @Column(name = "password")
 private String password;

 // Getters and setters
 public Long getId() {
   return id;
 }

 public void setId(Long id) {
```

```
    this.id = id;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public String getEmail() {
    return email;
  }

  public void setEmail(String email) {
    this.email = email;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }
```

and paste it inside the class. It must appear so:

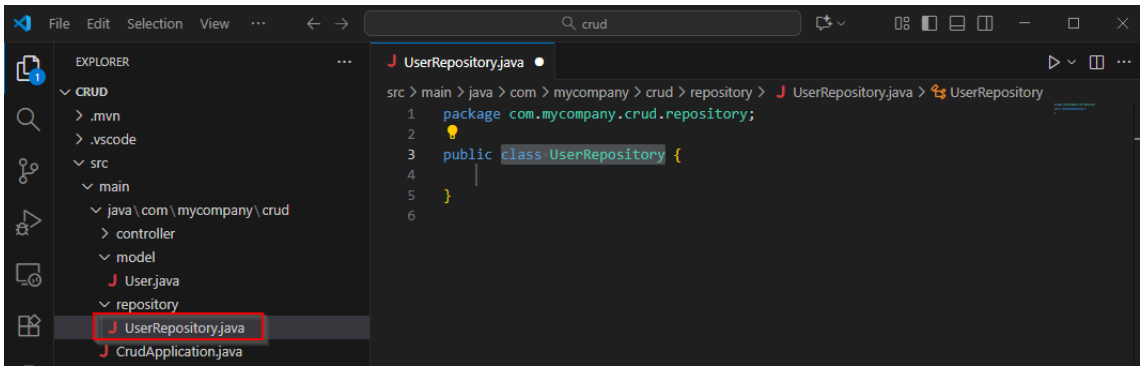
src > main > java > com > mycompany > crud > model > User.java > User > setId(Long)

```java
package com.mycompany.crud.model;

import jakarta.persistence.*;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email", unique = true)
    private String email;

    @Column(name = "password")
    private String password;

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

This Java code defines a **User model class** that represents a user entity, typically for an application that uses a CRUD database. It uses Jakarta Persistence Api (JPA) annotations, which are common in frameworks like Spring Boot, to map the class fields to columns in a database table. We can comment:

- **package com.mycompany.crud.model;**; Declares the package where this class resides.
- **import jakarta.persistence.***: Imports all necessary classes from the Jakarta Persistence API, which provides the standard for Object-Relational Mapping (ORM) in Java.
- **@Entity**: This is a mandatory JPA annotation that marks the User class as a JPA entity. An entity represents a table in a relational database.
- **@Table(name = "users")**: This annotation specifies the database table name that this entity maps to. In this case, the User class maps to a table named users.
- **@Id, @GeneratedValue(strategy = GenerationType.IDENTITY)**: @Id marks it as the primary key. @GeneratedValue specifies that its value is generated by the database (usually auto-increment).
- **@Column(name = "first_name")**: Stores the user's first name. @Column explicitly maps the field to the specified column name.
- **@Column(name = "email", unique = true)**: Stores the user's email address. unique = true adds a unique constraint at the database level, ensuring no two users can have the same email.
- **Getters and Setters**: The class includes standard public getter and setter methods for all its private fields (id, firstName, lastName, email, password). These methods are used to access and modify the private data fields from outside the class, adhering to the principle of encapsulation.
- For development, testing, or smaller applications, we can configure our JPA provider (Hibernate) to automatically manage the schema using the **spring.jpa.hibernate.ddl-auto** in **application.properties** to **update**, then our JPA provider will automatically create the users table and its columns if they do not already exist when your application starts up.

Now we need to create **UserRepository.java** class. To do this, we go to **CRUD > src > main > java > repository** folder and create the file **UserRepository.java** :



Delete the definition of the class, copy the code:

```
import com.mycompany.crud.model.User;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long>{
}
```

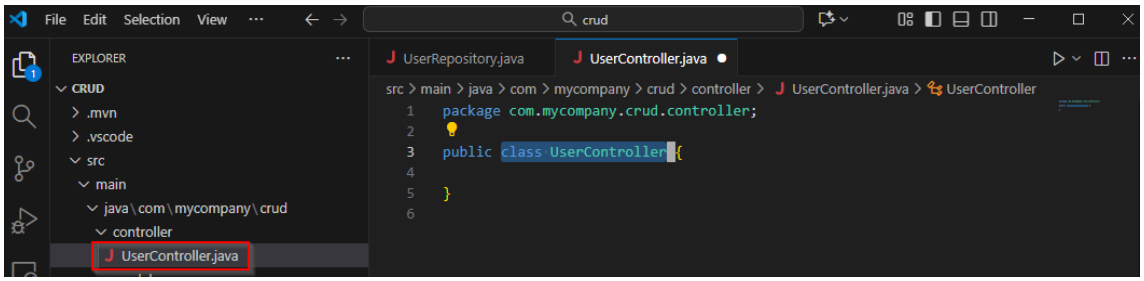
and paste them after the package line:

```
src > main > java > com > mycompany > crud > repository > J UserRepository.java > ...
1   package com.mycompany.crud.repository;
2
3   import com.mycompany.crud.model.User;
4
5   import org.springframework.data.jpa.repository.JpaRepository;
6   import org.springframework.stereotype.Repository;
7
8   @Repository
9   public interface UserRepository extends JpaRepository<User, Long> {
10  }
11
```

This is a very concise piece of Java code defining a Spring Data JPA Repository . Its primary purpose is to provide an easy way to perform standard database operations (like saving, finding, and deleting data) on the User entity without writing a lot of standard code. To remark:

- **@Repository**: This is a Spring stereotype annotation that marks this interface as a Data Access Object (DAO) or repository. When the Spring application starts, it detects this annotation and automatically manages this interface, handling exceptions and allowing it to be injected into other components (like services or controllers).
- **extends JpaRepository**: By extending JpaRepository, the UserRepository inherits a complete set of methods for interacting with the database, including:
  - save(User entity): To insert or update a user.
  - findById(Long id): To retrieve a user by their primary key.
  - findAll(): To retrieve all users.
  - delete(User entity): To delete a user.
  - ... and many more methods for sorting and pagination.
- **<User, Long>**: The JpaRepository is a generic interface that requires two parameters:
  - The **Entity Type** it manages: User (the Java class representing the database table).
  - The **Primary Key Type** of that entity: Long (the type of the id field in the User class).

The next is to create the **UserController.java** class. This class handles HTTP requests for CRUD operations on User. So, we go to **CRUD > src > main > java > controller** folder and create the file **UserRepository.java** :



Delete the definition of the class, copy the code:

```
import com.mycompany.crud.model.User;
import com.mycompany.crud.repository.UserRepository;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/users")
```

```
public class UserController {

  private final UserRepository userRepository;

  public UserController(UserRepository userRepository) {
     this.userRepository = userRepository;
  }

  @GetMapping
  public String listUsers(Model model) {
     model.addAttribute("users", userRepository.findAll());
     return "users/list";
  }

  @GetMapping("/new")
  public String showCreateForm(Model model) {
     model.addAttribute("user", new User());
     return "users/create";
  }

  @PostMapping
  public String createUser(@ModelAttribute("user") User user) {
     userRepository.save(user);
     return "redirect:/users";
  }

  @GetMapping("/edit/{id}")
  public String showEditForm(@PathVariable("id") Long id, Model model) {
     model.addAttribute("user", userRepository.findById(id).orElseThrow(() -> new
IllegalArgumentException("Invalid user Id:" + id)));
     return "users/edit";
  }

  @PostMapping("/update/{id}")
  public String updateUser(@PathVariable("id") Long id, @ModelAttribute("user") User user) {
     user.setId(id);
     userRepository.save(user);
     return "redirect:/users";
  }

  @GetMapping("/delete/{id}")
  public String deleteUser(@PathVariable("id") Long id) {
     userRepository.deleteById(id);
     return "redirect:/users";
  }
}
```

and paste it after the package line. Now, we must have:

src > main > java > com > mycompany > crud > controller > UserController.java > UserController

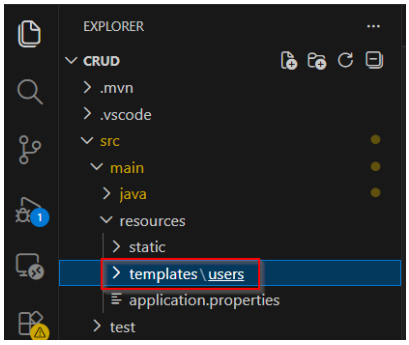
```java
package com.mycompany.crud.controller;

import com.mycompany.crud.model.User;
import com.mycompany.crud.repository.UserRepository;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;


@Controller
@RequestMapping("/users")

public class UserController {

    private final UserRepository userRepository;


    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @GetMapping
    public String listUsers(Model model) {
        model.addAttribute("users", userRepository.findAll());
        return "users/list";
    }

    @GetMapping("/new")
    public String showCreateForm(Model model) {
        model.addAttribute("user", new User());
        return "users/create";
    }

```

```java
    @PostMapping
    public String createUser(@ModelAttribute("user") User user) {
        userRepository.save(user);
        return "redirect:/users";
    }

    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable("id") Long id, Model model) {
        model.addAttribute("user", userRepository.findById(id).orElseThrow
        return "users/edit";
    }

    @PostMapping("/update/{id}")
    public String updateUser(@PathVariable("id") Long id, @ModelAttribute(
        user.setId(id);
        userRepository.save(user);
        return "redirect:/users";
    }

    @GetMapping("/delete/{id}")
    public String deleteUser(@PathVariable("id") Long id) {
        userRepository.deleteById(id);
        return "redirect:/users";
    }
}
```

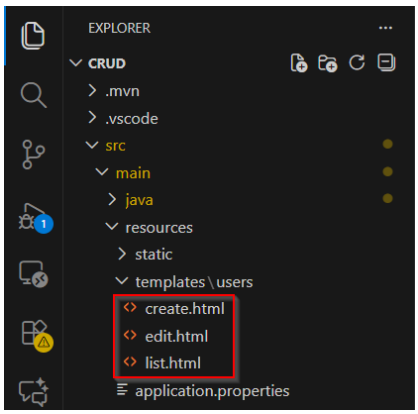
This Java code defines the UserController, which is a Spring MVC (Model View Controller) responsible for handling incoming web requests related to the User resource. It implements all the necessary methods to perform CRUD (Create, Read, Update, Delete) operations for users in a web application. To comment:

- **@Controller**: This is a Spring stereotype annotation marking the class as a web controller, capable of processing incoming requests and returning a view name or data.
- **@RequestMapping("/users")**: This class-level annotation maps all handler methods inside this controller to the base URL path /users. For example, a method with @GetMapping will handle requests to /users.

Now, we must create the HTML files that create, edit and list the users. These must be Thymeleaf templates. For this, we create the **users** folder in **CRUD > src > main > resources > templates** :



And inside of it we create the files **create.html**, **edit.html** and **list.html** :



Now, copy the next code to the corresponding files:

create.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Create User</title>
</head>
<body>
<h2>Create User</h2>
<form th:action="@{/users}" th:object="${user}" method="post">
  <label for="firstName">First Name:</label>
```

```
  <input type="text" th:field="*{firstName}" required><br>
  <label for="lastName">Last Name:</label>
  <input type="text" th:field="*{lastName}" required><br>
  <label for="email">Email:</label>
  <input type="email" th:field="*{email}" required><br>
  <label for="password">Password:</label>
  <input type="password" th:field="*{password}" required><br>
  <button type="submit">Save</button>
</form>
</body>
</html>
```

src > main > resources > templates > users > create.html > html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Create User</title>
</head>
<body>
<h2>Create User</h2>
<form th:action="@{/users}" th:object="${user}" method="post">
    <label for="firstName">First Name:</label>
    <input type="text" th:field="*{firstName}" required><br>
    <label for="lastName">Last Name:</label>
    <input type="text" th:field="*{lastName}" required><br>
    <label for="email">Email:</label>
    <input type="email" th:field="*{email}" required><br>
    <label for="password">Password:</label>
    <input type="password" th:field="*{password}" required><br>
    <button type="submit">Save</button>
</form>
</body>
</html>
```

edit.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Edit User</title>
</head>
<body>
<h2>Edit User</h2>
<form th:action="@{/users/update/{id}(id=${user.id})}" th:object="${user}" method="post">
  <label for="firstName">First Name:</label>
  <input type="text" th:field="*{firstName}" required><br>
  <label for="lastName">Last Name:</label>
  <input type="text" th:field="*{lastName}" required><br>
  <label for="email">Email:</label>
  <input type="email" th:field="*{email}" required><br>
  <label for="password">Password:</label>
  <input type="password" th:field="*{password}" required><br>
  <button type="submit">Save</button>
</form>
</body>
</html>
```

src > main > resources > templates > users > edit.html > html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit User</title>
</head>
<body>
<h2>Edit User</h2>
<form th:action="@{/users/update/{id}(id=${user.id})}" th:object="${user}" method="post">
    <label for="firstName">First Name:</label>
    <input type="text" th:field="*{firstName}" required><br>
    <label for="lastName">Last Name:</label>
    <input type="text" th:field="*{lastName}" required><br>
    <label for="email">Email:</label>
    <input type="email" th:field="*{email}" required><br>
    <label for="password">Password:</label>
    <input type="password" th:field="*{password}" required><br>
    <button type="submit">Save</button>
</form>
</body>
</html>
```

list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Users</title>
</head>
<body>
<h2>Users List</h2>
<a href="/users/new">Create New User</a>
<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Email</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="user : ${users}">
      <td th:text="${user.firstName}"></td>
      <td th:text="${user.lastName}"></td>
      <td th:text="${user.email}"></td>
      <td>
        <a th:href="@{/users/edit/{id}(id=${user.id})}">Edit</a> |
        <a th:href="@{/users/delete/{id}(id=${user.id})}">Delete</a>
      </td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

src > main > resources > templates > users > list.html > html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Users</title>
</head>
<body>
<h2>Users List</h2>
<a href="/users/new">Create New User</a>
<table>
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="user : ${users}">
            <td th:text="${user.firstName}"></td>
            <td th:text="${user.lastName}"></td>
            <td th:text="${user.email}"></td>
            <td>
                <a th:href="@{/users/edit/{id}(id=${user.id})}">Edit</a> |
                <a th:href="@{/users/delete/{id}(id=${user.id})}">Delete</a>
            </td>
        </tr>
    </tbody>
</table>
</body>
</html>
```

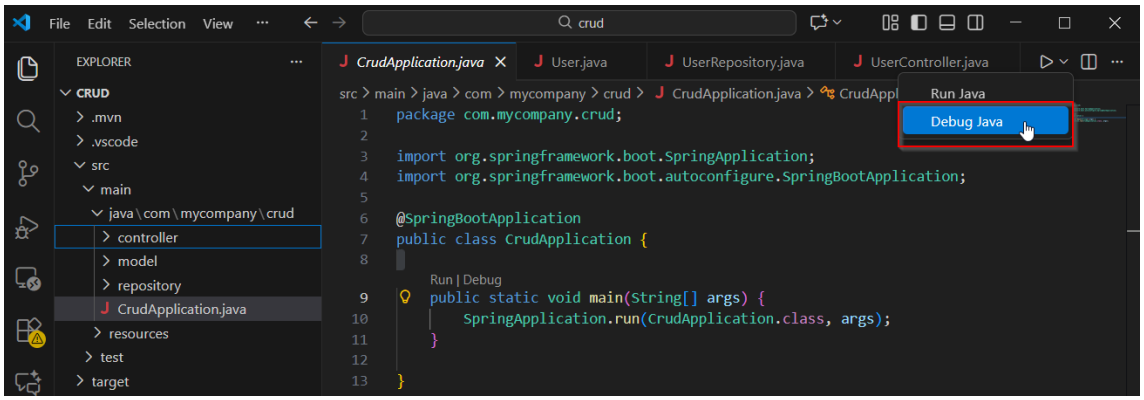
Before we run the project, we are going to check the data in the DB as we did before, connected to our Docker container:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| springCrud         |
| sys                |
+--------------------+
5 rows in set (0.132 sec)
```

```
mysql> USE springCrud;
Database changed
mysql> SHOW TABLES;
Empty set (0.004 sec)
```

We can see that no tables are yet in the DB.

To run the application, we go to the UserController.java file and select **Debug java** on the run icom:



We can see:

```
$  /usr/bin/env DB_HOST=localhost DB_USERNAME=root DB_PASSWORD=mysqlpass C:\\Users\\enricr\\.vscode
\\extensions\\redhat.java-1.49.0-win32-x64\\jre\\21.0.9-win32-x86_64\\bin\\java.exe -agentlib:jdwp=
transport=dt_socket,server=n,suspend=y,address=localhost:61323 @C:\\Users\\enricr\\AppData\\Local\\
Temp\\cp_ai8fy4q4x4jbdamfhkiskqanz.argfile com.mycompany.crud.CrudApplication

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v4.0.0)

2025-11-25T11:12:39.482+01:00  INFO 19928 --- [crud] [           main] com.mycompany.crud.CrudAppli
cation       : Starting CrudApplication using Java 21.0.9 with PID 19928 (C:\DevOps_projects\Spring
 Boot App\crud\target\classes started by enricr in C:\DevOps_projects\Spring Boot App\crud)
```

In the messages we can check that the table users have been created, and the embedded Tomcat web server started:

...

2025-11-25T11:12:51.438+01:00 INFO 19928 --- [crud] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)

Hibernate: create table users (id bigint not null auto_increment, email varchar(255), first_name varchar(255), last_name varchar(255), password varchar(255), primary key (id)) engine=InnoDB

Hibernate: alter table users drop index UK6dotkott2kjsp8vw4d0m25fb7

Hibernate: alter table users add constraint UK6dotkott2kjsp8vw4d0m25fb7 unique (email)

2025-11-25T11:12:52.114+01:00 INFO 19928 --- [crud] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'

2025-11-25T11:12:52.301+01:00 INFO 19928 --- [crud] [main] o.s.d.j.r.query.QueryEnhancerFactories : Hibernate is in classpath; If applicable, HQL parser will be used.

2025-11-25T11:12:52.590+01:00 WARN 19928 --- [crud] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning2025-11-25T11:12:53.752+01:00 INFO 19928 --- [crud] [main] o.s.boot.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'

2025-11-25T11:12:53.781+01:00 INFO 19928 --- [crud] [main] com.mycompany.crud.CrudApplication : Started CrudApplication in 15.858 seconds (process running for 17.568)

We can see that the table and columns have been created. Let's check in our container:

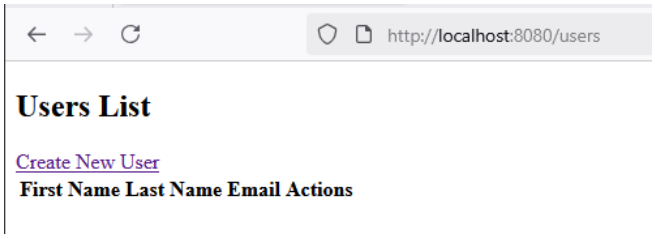
```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_springCrud |
+----------------------+
| users                |
+----------------------+
1 row in set (0.016 sec)
```

```
mysql> DESCRIBE users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint       | NO   | PRI | NULL    | auto_increment |
| email      | varchar(255) | YES  | UNI | NULL    |                |
| first_name | varchar(255) | YES  |     | NULL    |                |
| last_name  | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
5 rows in set (0.015 sec)

mysql> SELECT * FROM users;
Empty set (0.007 sec)
```

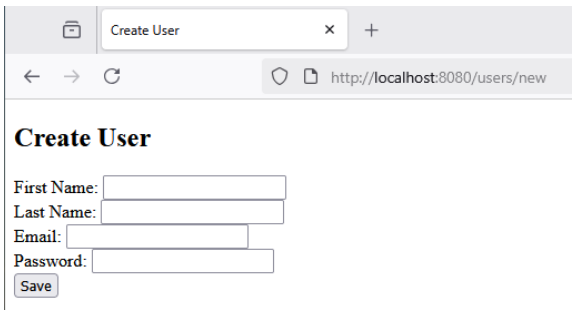
So, the table and his columns are created.

I.6- Check that the application works

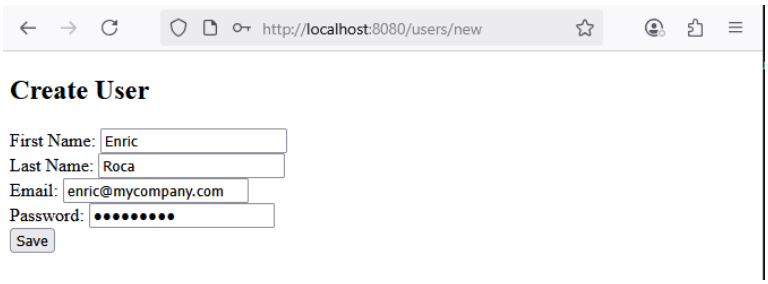
Now we are going to test creating a register. Go to the browser and type http://localhost:8080/users



Click on **Create New User**:

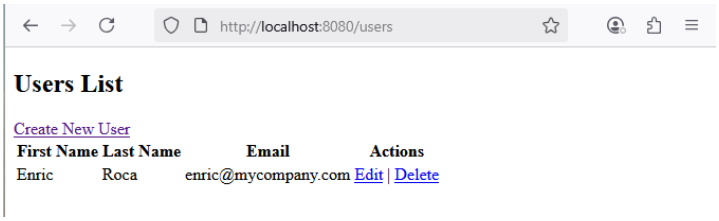


We type the details of our first user:



As password I have write password1.

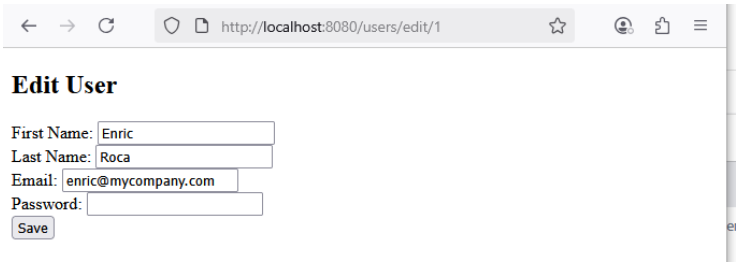
Click on **Save** and it lists the user:



So, obviously the data has been inserted in the DB:

```
mysql> SELECT * FROM users;
+----+---------------------+------------+-----------+-----------+
| id | email               | first_name | last_name | password  |
+----+---------------------+------------+-----------+-----------+
|  1 | enric@mycompany.com | Enric      | Roca      | password1 |
+----+---------------------+------------+-----------+-----------+
1 row in set (0.009 sec)
```

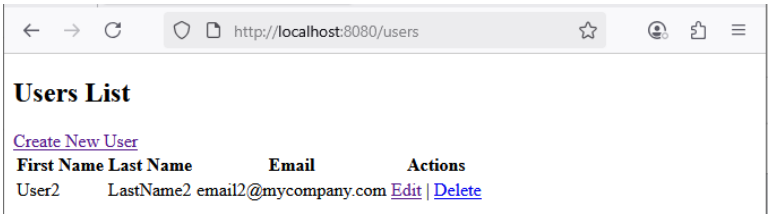
If we now click on **Edit**, it appears a form to modify the data:



As the URL we can see: http://localhost:8080/users/edit/1

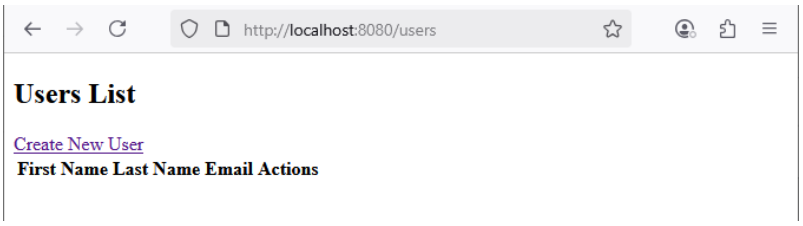
Now change the details and specify:

With password2 as the new password. Push on **Save** and we get:

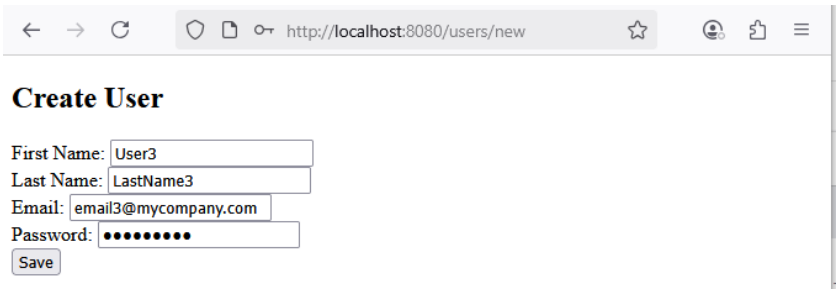



```
mysql> SELECT * FROM users;
+----+----------------------+------------+-----------+-----------+
| id | email                | first_name | last_name | password  |
+----+----------------------+------------+-----------+-----------+
|  1 | email2@mycompany.com | User2      | LastName2 | password2 |
+----+----------------------+------------+-----------+-----------+
1 row in set (0.012 sec)
```

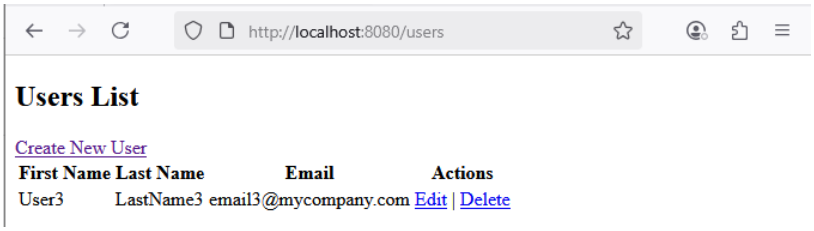
And the last to check is delete. For that we click on **Delete**:



The user has been deleted. If we try creating a user again click on **Create New User** and enter the data:



With password3 as the password. Click **Save**:

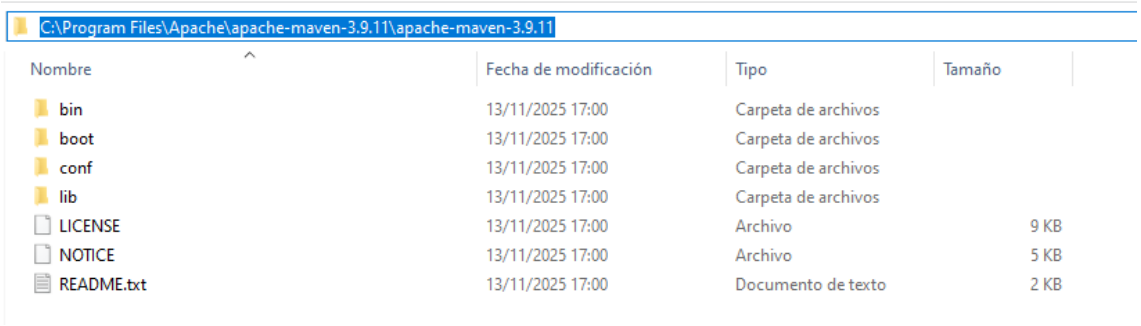


So, it works fine. We can create, edit, list and delete the data.

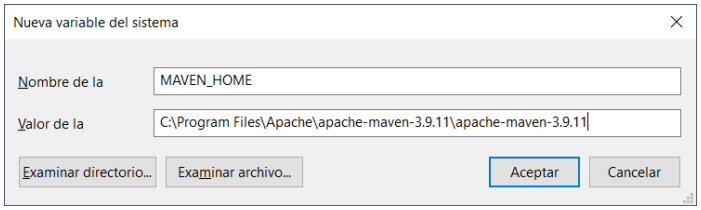
I.7- Create the executable JAR file

To generate the executable jar file of our application, first we install Maven in our machine if we don't have it. For that we access with the browser to https://maven.apache.org/download.cgi and download the last Binary zip archive file, that when I did was apache-maven-3.9.11-bin.zip

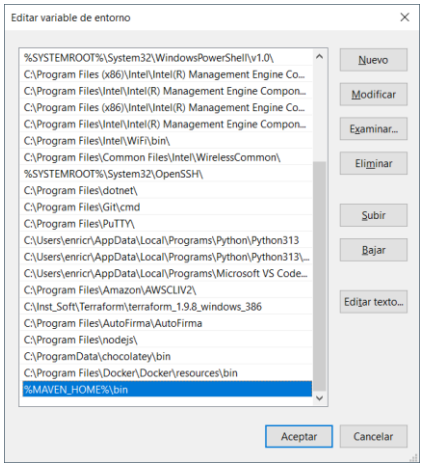
Now create the folders C:\Program Files**Apache\apache-maven-3.9.11** and unzip the file inside this folder. So, we have:



Now we must create the system environment variable **MAVEN_HOME** and given it the path of Maven, this is, C:\Program Files\Apache\apache-maven-3.9.11\apache-maven-3.9.11 :



and edit the system environment variable **Path** adding **%MAVEN_HOME%\bin** :



If we test the installation in a system command prompt with **mvn -v**, it works:

```
C:\Users\enricr>mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: C:\Program Files\Apache\apache-maven-3.9.11\apache-maven-3.9.11
Java version: 21.0.5, vendor: Eclipse Adoptium, runtime: C:\Program Files\Eclipse Adoptium\jdk-21.0.5.11-hotspot
Default locale: es_ES, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\enricr>
```

We must close VS Code and open again with the project, and test the same in a PowerShell terminal:

```
PS C:\DevOps_projects\Spring Boot App\crud> mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: C:\Program Files\Apache\apache-maven-3.9.11\apache-maven-3.9.11
Java version: 21.0.5, vendor: Eclipse Adoptium, runtime: C:\Program Files\Eclipse Adoptium\jdk-21.0
.5.11-hotspot
Default locale: es_ES, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
PS C:\DevOps_projects\Spring Boot App\crud>
```

So, Maven is properly installed.

Now we try to create the executable jar file. For that execute the next command in PowerShell terminal:

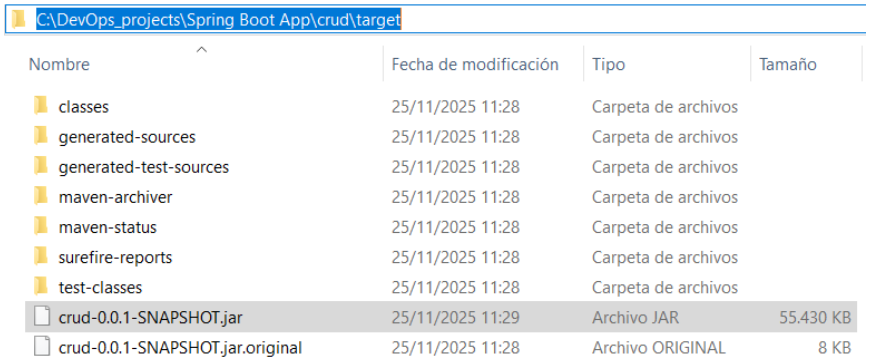
mvn clean package

```
PS C:\DevOps_projects\Spring Boot App\crud> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.mycompany:crud >------------------------
[INFO] Building crud 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean
-plugin/3.5.0/maven-clean-plugin-3.5.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-
plugin/3.5.0/maven-clean-plugin-3.5.0.pom (5.7 kB at 7.0 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugi
ns/44/maven-plugins-44.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugin
s/44/maven-plugins-44.pom (8.4 kB at 109 kB/s)
[INFO]
[INFO] --- clean:3.5.0:clean (default-clean) @ crud ---
[INFO] Deleting C:\DevOps_projects\Spring Boot App\crud\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ crud ---
[INFO] Copying 1 resource from src\main\resources to target\classes
[INFO] Copying 3 resources from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.14.1:compile (default-compile) @ crud ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 4 source files with javac [debug parameters release 17] to target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ crud ---
[INFO] Copying 1 resource from src\test\resources to target\test-classes
[INFO]
```

...

```
[INFO] Replacing main artifact C:\DevOps_projects\Spring Boot App\crud\target\crud-0.0.1-SNAPSHOT.jar with repac
kaged archive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to C:\DevOps_projects\Spring Boot App\crud\target\crud-0.0.1-SNAPS
HOT.jar.original
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  33.703 s
[INFO] Finished at: 2025-11-25T11:29:04+01:00
[INFO] ------------------------------------------------------------------------
PS C:\DevOps_projects\Spring Boot App\crud>
```

The building of the jar file has finished fine. We can check that it has been created in the target folder:



As we can see, the name of the executable jar has the format: <artifactId>-<version>.jar. To use a different name, update these values in pom.xml:

pom.xml

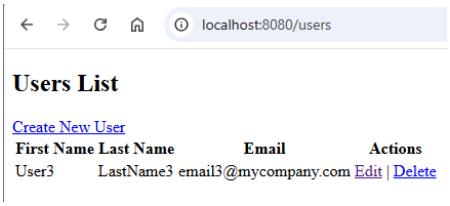
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.0.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.mycompany</groupId>
    <artifactId>crud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>crud</name>
    <description>Demo project for Spring Boot</description>
    <url/>
```

If we want to directly execute this jar from a PowerShell terminal, we go to the **target** folder and run the next command, passing the environment variables as parameters:

$env:DB_HOST='localhost'; $env:DB_USERNAME='root'; $env:DB_PASSWORD='mysqlpass'; java -jar crud-0.0.1-SNAPSHOT.jar

```
PS C:\DevOps_projects\Spring Boot App\crud> cd target
PS C:\DevOps_projects\Spring Boot App\crud\target> $env:DB_HOST='localhost'; $env:DB_USERNAME='root'; $env:DB_PASSWORD='mysqlpass'; java
 -jar crud-0.0.1-SNAPSHOT.jar
```

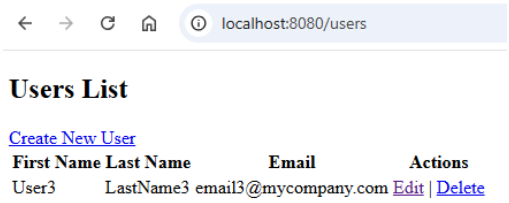
And trying in the browser, it works:



If we want to execute the jar file from a bash terminal, the command is:

DB_HOST=localhost DB_USERNAME=root DB_PASSWORD=mysqlpass java -jar crud-0.0.1-SNAPSHOT.jar

```
enricr@DESKTOP-N45K1AL MINGW64 /c/DevOps_projects/Spring Boot App/crud/target
$ DB_HOST=localhost DB_USERNAME=root DB_PASSWORD=mysqlpass java -jar crud-0.0.1-SNAPSHOT.jar
```



Part II: Create the configuration files to deploy the App

To deploy our application, we will use **Docker**. Docker as a platform to build, test and deploy applications quickly by packaging them into standardized, isolated units called containers. A **Docker Image** is a package that includes everything needed to run a piece of software, and a **Docker Container** is a runtime instance of a Docker Image.

In this part we will create a **Dockerfile**, which is used by Docker to build a Docker Image. Besides, we will upload this image to a remote repository called **Docker Hub** for use later.

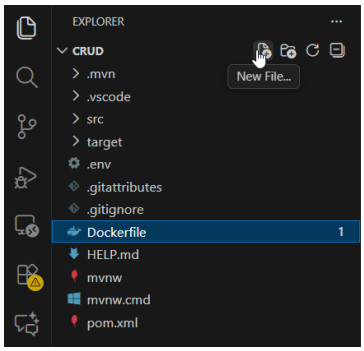
Another great platform is **Kubernetes**. Later in part III we will talk about it. But, for the moment be in mind that to deploy resources we can use configuration files, known as **manifests**, typically written in YAML or JSON format. We will create two yaml deployment files: one for the DB and another for the Application. In each of these files we will specify things like the image to use to create the containers, the number of replicas, the ports to be used, the volumes, the services, etc.

Another important concept is the **Pod**. Pods are the smallest, most basic deployable unit in Kubernetes. Kubernetes does not manage containers directly; it manages pods which contain the containers. Normally one pod contains only one container, but there are situations in which we can have more than one container inside a pod.

II.1- Create the Dockerfile

A Dockerfile is a plain text file that contains a set of instructions for the Docker Daemon to execute during the image-building process.

Create a file named Dockerfile without extension in the text editor that you prefer, we will use VS Code:



Copy the code:

```
FROM openjdk:8
EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","/crud-0.0.1-SNAPSHOT.jar"]
```

And paste it inside the file and save it:

```
Dockerfile > ...
1   FROM openjdk:8
2   EXPOSE 8080
3   ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar
4   ENTRYPOINT ["java","-jar","/crud-0.0.1-SNAPSHOT.jar"]
5
```

Line by line:

- **FROM openjdk:8** : It pulls (downloads) the official Docker image for OpenJDK version 8. This image already contains the necessary Java Runtime Environment (JRE) needed to execute our Java application.
- **EXPOSE 8080** : It documents that the containerized application inside will be listening on TCP port 8080 at runtime. It does not automatically publish the port or make it accessible from the host machine.
- **ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar** : It copies files from the local machine where we run docker build into the new Docker Image; target/crud-0.0.1-SNAPSHOT.jar: This is the source file path on your host machine. This assumes you have compiled your Spring Boot application and created an executable JAR file in a target/ directory; crud-0.0.1-SNAPSHOT.jar: This is the destination path and filename inside the image's filesystem (it is copied to the root directory /).
- **ENTRYPOINT ["java","-jar","/crud-0.0.1-SNAPSHOT.jar"]** : It defines the main command that will be executed when a container is started from this image; "java": The executable program (the Java runtime); "-jar": A standard Java flag to execute a JAR file; "/crud-0.0.1-SNAPSHOT.jar": The absolute path to the JAR file copied in the previous step. When you run the container, this command starts the application.

II.2- Create the db deployment yaml file

Now we are going to create a Kubernetes deployment yaml file to deploy the MySQL pod (with one container inside). We will use a **ConfigMap** object to specify the database name, thinking that it is a configuration detail and not sensible information, but we will use a **Secret** object to avoid embedding the highly-privileged root user database username and the database password in the manifest file, which would be a significant security risk.

Create a file named **db-deploy.yaml** .

Copy the next code:

```
# Define the ConfigMap for the non-sensitive username
apiVersion: v1
kind: ConfigMap
metadata:
 name: mysql-config
 labels:
   app: mysql
   tier: database
data:
 MYSQL_DATABASE: springCrud                    # The name of the database
---
# Define the Secret
apiVersion: v1
kind: Secret
metadata:
 name: mysql-secret
 labels:
   app: mysql
   tier: database
type: Opaque
```

```
data:
 MYSQL_USER: cm9vdA==                    # Base64 encoded 'root'
 MYSQL_ROOT_PASSWORD: bXlzcWxwb3Nz  # Base64 encoded 'mysqlpass'
---
# Define a 'Persistent Volume Claim'(PVC) for Mysql Storage, dynamically provisioned by cluster
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mysql-pv-claim      # name of PVC essential for identifying the storage data
 labels:
  app: mysql
  tier: database
spec:
 accessModes:
  - ReadWriteOnce          #This specifies the mode of the claim that we are trying to create.
 resources:
  requests:
   storage: 1Gi            #This will tell kubernetes about the amount of space we are trying to claim.
---
# Configure 'Deployment' of mysql server
apiVersion: apps/v1
kind: Deployment
metadata:
 name: mysql
 labels:
  app: mysql
  tier: database
spec:
 selector:                 # mysql Pod Should contain same labels
  matchLabels:
   app: mysql
   tier: database
 strategy:
  type: Recreate
 template:
  metadata:
   labels:                 # Must match 'Service' and 'Deployment' selectors
    app: mysql
    tier: database
  spec:
   containers:
    - image: mysql:5.7     # image from docker-hub
      args:
       - "--ignore-db-dir=lost+found" # Workaround for https://github.com/docker-library/mysql/issues/186
      name: mysql
      env:
       # Get database name from the ConfigMap
       - name: MYSQL_DATABASE
         valueFrom:
           configMapKeyRef:
             name: mysql-config
             key: MYSQL_DATABASE
       # Get username from the Secret (Sensitive)
       - name: MYSQL_USER
         valueFrom:
          secretKeyRef:
            name: mysql-secret
            key: MYSQL_USER
```

```
        # Get root password from the Secret (Sensitive)
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: MYSQL_ROOT_PASSWORD
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:           # Mounting voulume obtained from Persistent Volume Claim
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql      #This is the path in the container on which the mounting will take place.
    volumes:
      - name: mysql-persistent-storage              # Obtaining 'vloume' from PVC
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
# Define a 'Service' To Expose mysql to Other Services
apiVersion: v1
kind: Service
metadata:
  name: mysql                   # DNS name
  labels:
    app: mysql
    tier: database
spec:
  ports:
    - port: 3306
      targetPort: 3306
  selector:                     # mysql Pod Should contain same labels
    app: mysql
    tier: database
  clusterIP: None               # We Use DNS, Thus ClusterIP is not relevant
```

paste it in the file and save.

In detail:

- This deployment file contains the definition of five resources separated by ---: a ConfigMap, a Secret, a PersistentVolumeClaim (PVC), a Deployment, and a Service.
- **ConfigMap**: The ConfigMap is used to store non-sensitive configuration data as key-value pairs that the application (the MySQL container) will consume as environment variables.
  - **kind: ConfigMap**: The object type for storing configuration data.
  - **data.MYSQL_DATABASE: springCrud**: Specifies the name of the database that MySQL will initialize when it starts up.
- **Secret**: This object securely stores sensitive credentials.
  - **kind: Secret**: Standard object for sensitive data.
  - **type: Opaque**: A general-purpose Secret type.
  - **data:**
    - **MYSQL_USER: cm9vdA==** : This is the Base64-encoded value for the username, which decodes to “root”.
    - **MYSQL_ROOT_PASSWORD: bXlzcWxwb3Nz** : This is the Base64-encoded value for our password. We will see two issues with it later, when trying to deploy and use.

  Note: Later we will see how to encode a string.

  Note on Encoding: While the string data must be encoded to meet the Kubernetes API requirement, Base64 **encoding is NOT encryption**. It only hides the password from casual view. For true protection, the Secret should be further secured using tools like **Vault** or **Kubernetes Secret Encryption at Rest** (which uses an external Key Management Service).
- **PersistentVolumeClaim**: This section requests a specific amount of storage from the cluster for the database to use, ensuring the data is not lost if the Pod restarts (persistency).

- **kind: PersistentVolumeClaim**: Defines the object as a request for storage.
- **metadata.name: mysql-pv-claim**: The name used by the Deployment to bind the storage.
- **spec.accessModes: - ReadWriteOnce**: Specifies that the volume can be mounted as read-write by a single Node. This is typical for databases.
- **spec.resources.requests.storage: 1Gi**: Requests 1 Gigabyte of storage space. The Kubernetes StorageClass (if configured) will dynamically provision a PersistentVolume (PV) to satisfy this claim.

- **Deployment**: The Deployment manages the Pod(s) running the MySQL container, defining its desired state, image, environment variables, and how it attaches to the storage defined in the PVC.
  - **kind: Deployment**: Defines the object that manage Pods and ReplicaSets (objects that maintain a stable set of identical Pods available and running).
  - **metadata.name: mysql**: The name of the Deployment.
  - **spec.selector.matchLabels**: Used to identify the Pods that belong to this Deployment. The labels ("app: mysql", "tier: database") must match the labels on the Pod template.
  - **spec.strategy.type: Recreate**: A critical setting for stateful applications like databases. It ensures that when the Deployment is updated, the old Pod is completely terminated before a new one is created. This prevents issues with multiple Pods trying to access the same ReadWriteOnce volume simultaneously.
  - **spec.template**: This defines the actual Pod that will be created by the Deployment.
    - **spec.containers.image: mysql:5.7**: Specifies the container image to use from Docker Hub.
    - **spec.containers.name: mysql**: The name of the container.
    - **spec.containers.env**: Variables are injected from the external objects:
      - ✓ MYSQL_DATABASE is sourced from the ConfigMap (configMapKeyRef).
      - ✓ MYSQL_USER and MYSQL_ROOT_PASSWORD are sourced from the Secret (secretKeyRef).
    - **spec.containers.ports: containerPort: 3306**: Exposes the standard MySQL port inside the container.
    - **spec.containers.volumeMounts**: Mounts the volume inside the container at the path /var/lib/mysql, which is where MySQL stores its data.
    - **spec.volumes**: Defines the source of the volume, which is the PersistentVolumeClaim named mysql-pv-claim (the one defined before). This is what connects the Pod to the persistent storage.

  Note: The **spec.replicas** field is omitted, but the Deployment Controller will interpret this as a desire for one replica, or what is the same, one Pod.
- **Service**: The Service provides a stable network endpoint (a static IP or DNS name) to access the Pod(s) managed by the Deployment, even if the Pods are destroyed and recreated with new IP addresses.
  - **kind: Service**: Defines the object as a method for exposing a set of Pods.
  - **metadata.name: mysql**: This name will become the DNS name used by other services in the cluster (e.g., a web app) to connect to the database (e.g., mysql:3306).
  - **spec.ports**: Maps the Service's port (port: 3306) to the container's exposed port (targetPort: 3306).
  - **spec.selector**: This is how the Service finds the Pods to route traffic to. It selects any Pods with the labels "app: mysql" and "tier: database" (the same labels used in the Deployment's Pod template).
  - **spec.clusterIP: None**: Creates a "Headless Service." Instead of assigning a single cluster IP, it returns the IP addresses of the individual Pods. This is often used for stateful applications like databases to allow for more advanced connection/failover logic by client applications, though for a single-replica database like this, it mainly emphasizes that the DNS name maps to the Pod's actual network endpoint.

II.3- Create de app deployment yaml file

The second Kubernetes deployment yaml file will deploy the Application pods (3 pods = 3 replicas) with their containers (one container in each pod). This time, we will use a **ConfigMap** object to configure the database host and the database name, while to specify the database username and his password we will use a **Secret** as before.

Create a file named **app-deploy.yaml** .

Copy the next code:

```
# Define the ConfigMap for non-sensitive configuration
apiVersion: v1
kind: ConfigMap
metadata:
 name: springboot-config
 labels:
   app: springboot-crud
data:
 DB_HOST: mysql                          # Hostname of the MySQL Service
 DB_NAME: springCrud                      # Name of the database
---
# Define the Secret for sensitive credentials
apiVersion: v1
kind: Secret
metadata:
 name: springboot-secret
 labels:
   app: springboot-crud
type: Opaque
data:
 DB_USERNAME: cm9vdA==                # Base64 encoded 'root'
 DB_PASSWORD: bXlzcWxwb3Nz      # Base64 encoded 'mysqlpass'
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: springboot-crud-deploy
spec:
 selector:
   matchLabels:
     app: springboot-crud
 replicas: 3
 template:
   metadata:
     labels:
       app: springboot-crud
   spec:
     containers:
       - name: springboot-crud-container
         image: enricr/enricr/springbootcrud-k8s:1.0
         ports:
           - containerPort: 8080
         env:                # Setting Environmental Variables using ConfigMap and Secret
           # Get DB_HOST from ConfigMap
           - name: DB_HOST
             valueFrom:
               configMapKeyRef:
                 name: springboot-config
                 key: DB_HOST
           # Get DB_NAME from ConfigMap
           - name: DB_NAME
             valueFrom:
               configMapKeyRef:
                 name: springboot-config
                 key: DB_NAME
```

```
        # Get DB_USERNAME from Secret
        - name: DB_USERNAME
          valueFrom:
            secretKeyRef:
              name: springboot-secret
              key: DB_USERNAME
        # Get DB_PASSWORD from Secret
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: springboot-secret
              key: DB_PASSWORD
---
apiVersion: v1                  # Kubernetes API version
kind: Service                   # Kubernetes resource kind we are creating
metadata:                       # Metadata of the resource kind we are creating
 name: springboot-crud-svc
spec:
 selector:
   app: springboot-crud
 ports:
  - protocol: "TCP"
    port: 8080                  # Port that the service is running on in the cluster
    targetPort: 8080            # The port exposed by the service
 type: NodePort                 # type of the service.
```

paste it in the file and save.

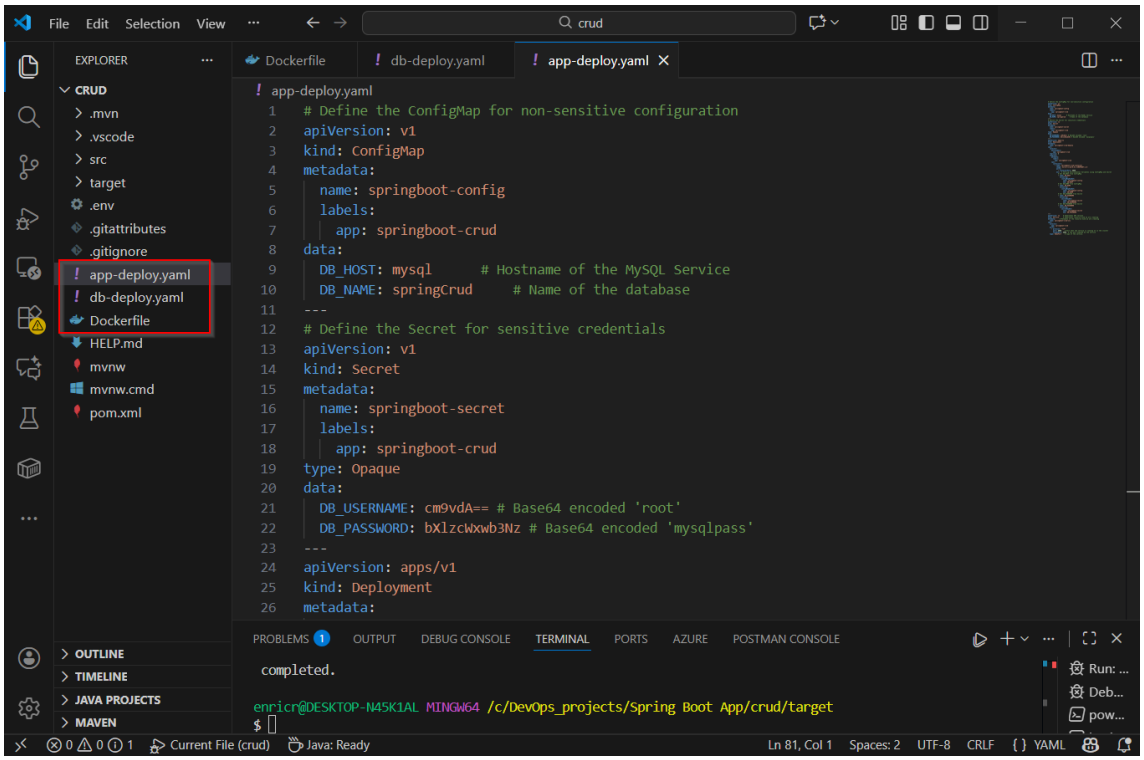
In detail:

- This deployment file contains the definition of four resources separated by ---: a ConfigMap, a Secret, a Deployment, and a Service.
- **ConfigMap**: The ConfigMap stores non-sensitive, general application settings that the Spring Boot application needs to connect to its database.
  - **kind: ConfigMap**: The object type for storing configuration data.
  - **data:**
    - **DB_HOST: mysql** : Specifies the hostname of the database server. It is the Kubernetes Service named mysql running the database that we created in db-deploy.yaml.
    - **DB_NAME: springCrud** : Specifies the name of the database the application should connect to.
- **Secret**: This object securely stores the sensitive credentials required to authenticate with the database.
  - **kind: Secret**: The object type for storing sensitive data.
  - **type: Opaque**: A general-purpose Secret type.
  - **data:**
    - **DB_USERNAME: cm9vdA==** : Base64-encoded username (root).
    - **DB_PASSWORD: bXlzcWxwb3Nz** : Base64-encoded password.
  - The application references these keys, and Kubernetes securely injects the decoded values as environment variables.
- **Deployment**: The Deployment manages the Pods running the Spring Boot application, defining their desired state and scale.
  - **kind: Deployment**: Manages the application's lifecycle, updates, and scaling.
  - **spec.replicas: 3**: This setting ensures that three identical instances (Pods) of the application are constantly running, providing high availability and basic load distribution.
  - **spec.template:** Defines the Pod structure:
    - **image: enricr/enricr/springbootcrud-k8s:1.0**: Specifies the Docker image to use for the application container.
    - **containerPort: 8080** : Defines the port the Spring Boot application listens on inside the container.
    - **env** (Environment Variables): This section links the application to the ConfigMap and Secret:

- ✓ DB_HOST and DB_NAME use configMapKeyRef to fetch values from the springboot-config ConfigMap.
- ✓ DB_USERNAME and DB_PASSWORD use secretKeyRef to fetch decoded values from the springboot-secret Secret.

- **Service**: The Service provides a stable, external point of access to the application Pods, regardless of which node they run on or if they restart.
  - ➢ **kind: Service**: Defines the object as a method for exposing a set of Pods.
  - ➢ **spec.selector:** app: springboot-crud: This targets the three Pods created by the Deployment (which all have this label).
  - ➢ **spec.ports:**
    - **port: 8080** : The port exposed by the Service within the cluster.
    - **targetPort: 8080** : The port on the actual application container.
  - ➢ **type: NodePort** : This service type exposes the application on a specific port on every Node's IP address in the cluster. This makes the application accessible from outside the cluster for testing or external clients.

Once the files are created, we will see them in the project's root directory:



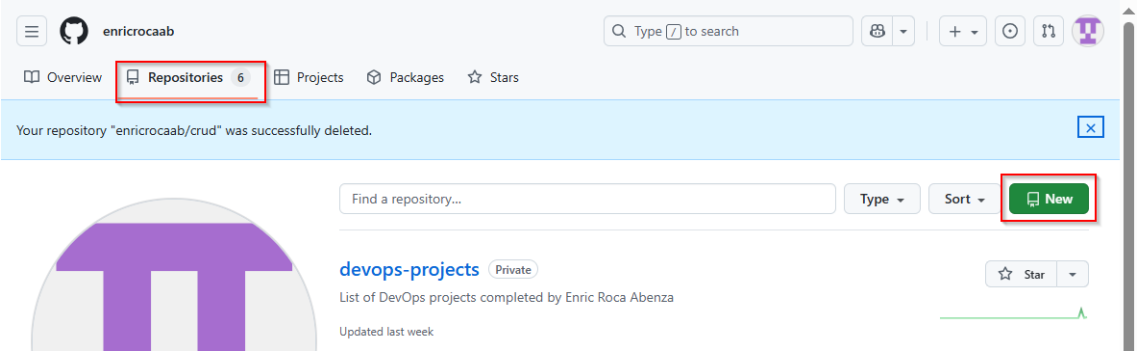
II.4- Upload the files to GitHub

Now we must upload (push) the project files to a remote repository, for instance **GitHub**, so they are available for download when needed.

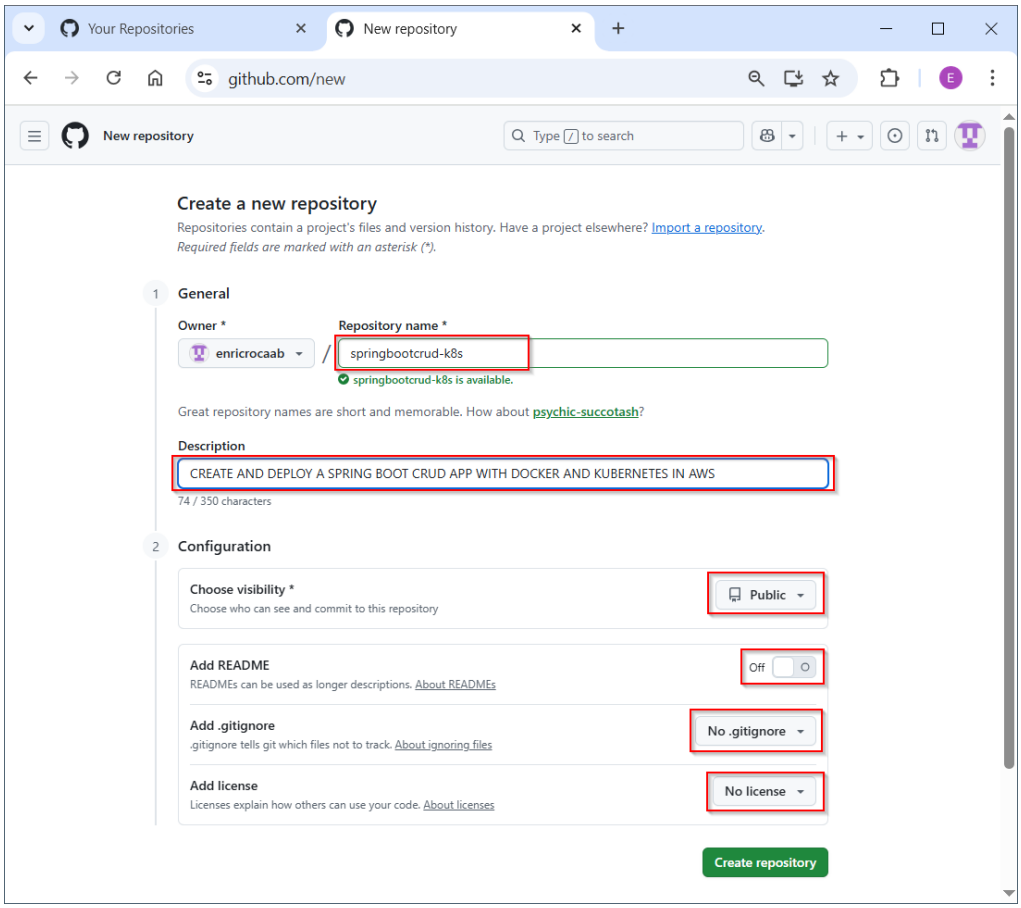
You should use your own GitHub account, so log in it (https://github.com/), or create an account if you don't have.

Our project has the name **crud**, but we would like to create a repository in GitHub with a little more detail in its name, so we will manually create the repository in the GitHub web interface and later we will link our local repository to it.

In your GitHub, select the **Repositories** tab and click on **New**:

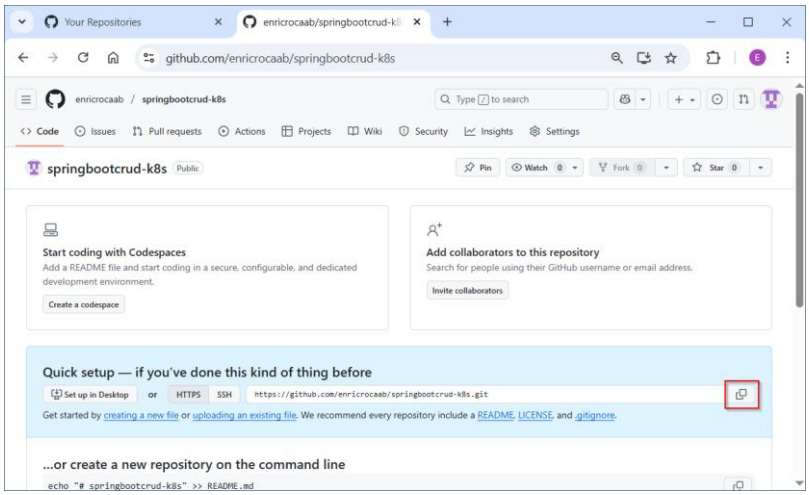


Write **springbootcrud-k8s** as the name of our new repository and specify a description. Choose the visibility as **public**, and **do not** initialize it with a README, .gitignore, or license:



Push on **Create repository**.

A webpage appears with suggestions of how to proceed. Click on the copy icon to copy the URL of the repository:

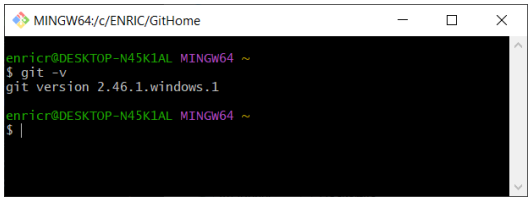


https://github.com/enricrocaab/springbootcrud-k8s.git

You should have **Git** installed in your laptop to push the project files and folders to the GitHub repository. If you don't have Git, install it. In case of a Windows machine, you can download the installer and follow the instructions on https://gitforwindows.org/.

Once you have Git installed in your working machine, open **Git Bash** and check that it is working by executing the command:

git -v



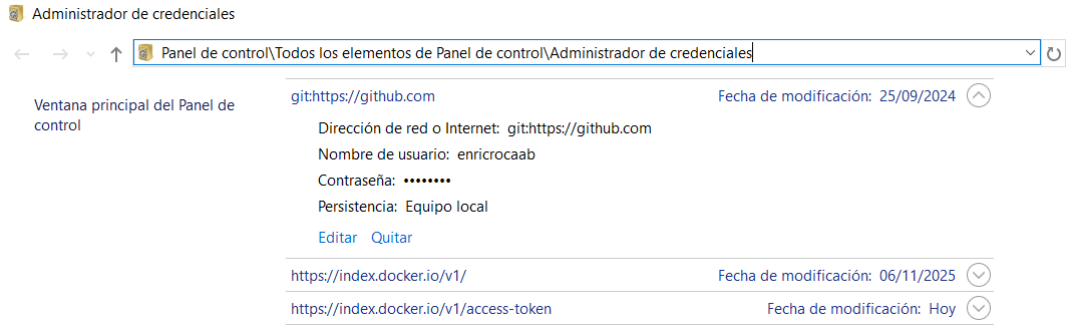
The process of pushing the project files to GitHub using git commands should be (don't do, later we will see a more integrated way with VS Code):

1. Navigate to our project's root folder (crud) in the VS Code Terminal.
2. Initialize a new local Git repository in our project folder with "**git init**" .
3. Stage all our project files for commit with "**git add .**" .
4. Create the first commit (snapshot) of our code in the local repository using the command "**git commit -m "Initial commit of springbootcrud-k8s"**"
5. Link our local repository to the URL of the repository copied before using the standard remote name **origin**:
   git remote add origin https://github.com/enricrocaab/springbootcrud-k8s.git
6. Verify the remote with "**git remote -v**".
7. Push our committed local code to the remote repository: "**git push -u origin main**".
   -u (or --set-upstream) Tells Git to remember that the local branch (main) is tracking the remote branch (main) on the origin remote. You only need to use this the first time.

Notes:

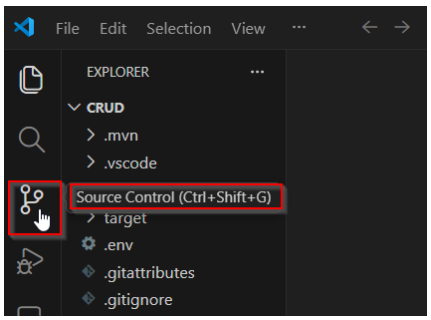
- Our Git username and email are used to label our commits (i.e., who made the changes). This is separate from the GitHub sign-in, but it is important to ensure our commits link to our GitHub profile. We can check or set these using the VS Code integrated terminal:
  - Check Username: git config user.name
  - Check Email: git config user.email
  - Set Username: git config --global user.name "Your Name"
  - Set Email: git config --global user.email "your.email@example.com"

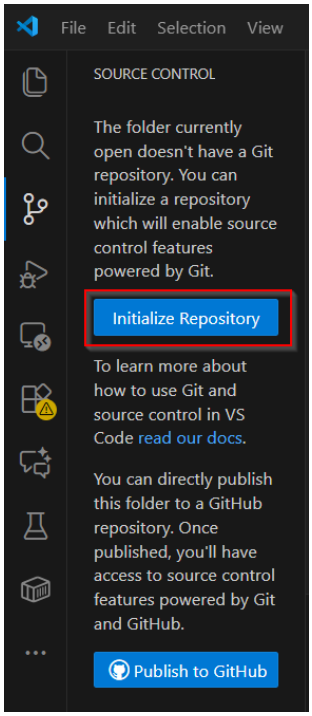
  Use the --global flag to apply the settings to all Git projects on your machine. You can omit it to set the details only for the current project.
- VS Code handles storing the GitHub credentials securely using our operating system's Credential Manager. So, if you need to switch accounts, search for and open "Credential Manager", and look for entries under Generic Credentials related to vscode or git:https://github.com. Delete the relevant entry to force VS Code to prompt you to sign in again.



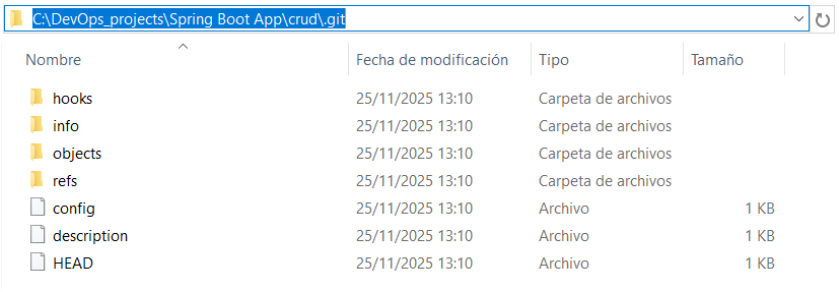
But the most integrated way to upload our existing Spring Boot Maven project from Visual Studio Code to GitHub is to use the **Source Control** view within VS Code. This process involves adding the remote **origin**, **initializing** a local Git repository, **committing** our project files, and then **publishing** it directly to a new repository on the GitHub account. Messages may appear in your browser at any time to authenticate your GitHub login.

With the project opened in VS Code, go the **Source Control** icon and select **Initialize Repository**:





VS Code will create a hidden **.git** folder in our project directory:



Remember that we already have in our project a file named **.gitignore**, where we added the .env file. Adding these lines prevents unnecessary, large, or security-sensitive files from being pushed to GitHub:

```
HELP.md
target/
.mvn/wrapper/maven-wrapper.jar
!**/src/main/**/target/
!**/src/test/**/target/

### STS ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/
build/
!**/src/main/**/build/
!**/src/test/**/build/

### VS Code ###
.vscode/

### Environment variables with credentials
.env
```

Someone might think that the **target** directory shouldn't be in the file since we're interested in having the JAR file to create the Docker image, but this wouldn't be the correct way to

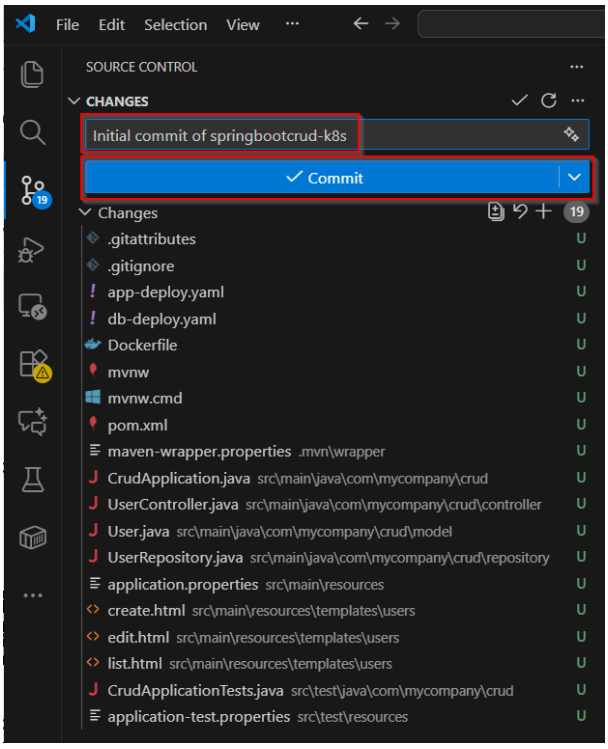
proceed. This is a fundamental concept in version control for almost all programming languages and build tools.

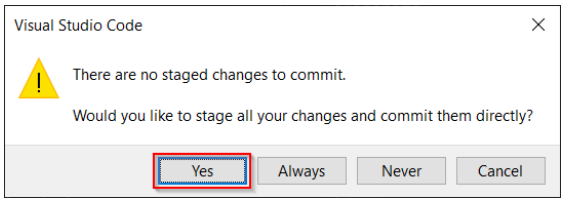
In a typical **Continuous Integration/Continuous Deployment (CI/CD)** pipeline, we have four basic steps:

Step	Location	Action
1. Development	Your Local Machine (VS Code)	You write code, commit, and push only the **source files** to GitHub.
2. CI/CD Trigger	GitHub	Pushing code to a specific branch (like main or release) triggers a pipeline (e.g., using GitHub Actions, Jenkins, GitLab CI, etc.).
3. Build	CI/CD Server	The pipeline server runs the command, e.g., **mvn clean install**. The JAR file is **re-built** from the source code checked out from GitHub.
4. Deploy	CI/CD Server → Target Server	The pipeline takes the newly generated JAR from the CI/CD server's temporary workspace (the equivalent of the /target folder) and deploys it to the production or staging server.

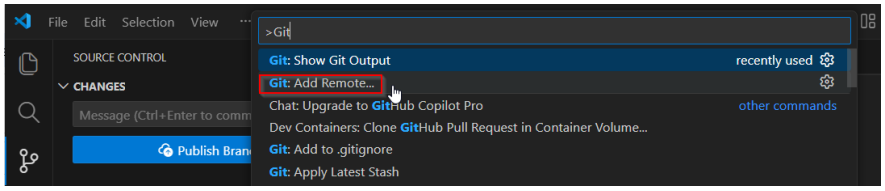
In short, our **local machine**, **Git**, and **GitHub** are for managing source code, while the CI/CD pipeline is responsible for **building** and **deploying** the final artifact.

Now, before pushing to GitHub, we need to save a **snapshot** of our project's current state with a **commit**. In the **Sorce Control** view, we can see all the files under **Changes**. Write a message like “Initial commit of springbootcrud-k8s” and click on **Commit**:

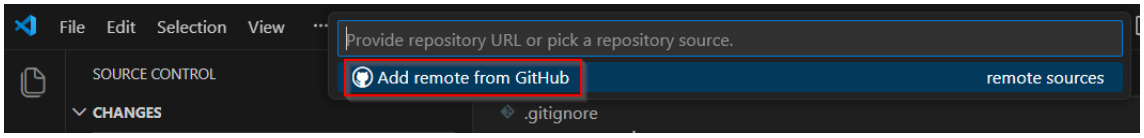




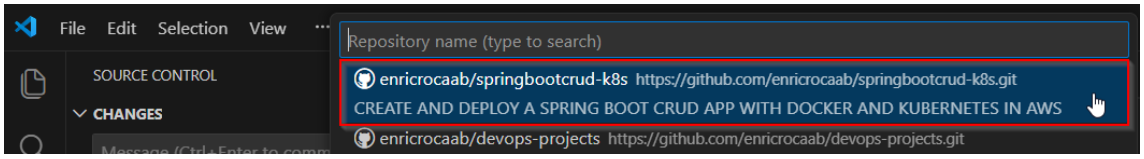
And now, before publishing the project to GitHub, we must add the remote repository. For that, press Ctrl+Shift+P to open the Command Palette, type **Git: Add Remote** and select the command:



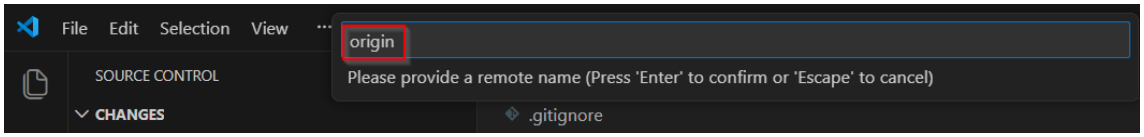
Click on "**Add remote from GitHub**":



Select our remote repository, or paste it if it doesn't appear:

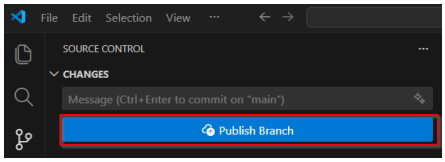


and provide the remote name **origin**:



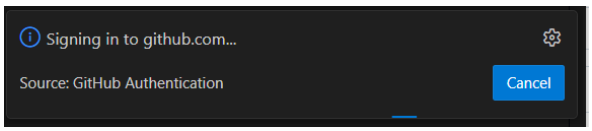
Press Enter. Our local repository is now linked to the new remote.

Now click on **Publish Branch** to publish to GitHub:

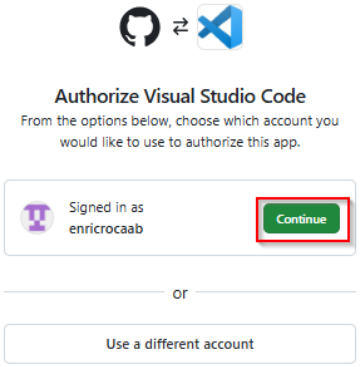


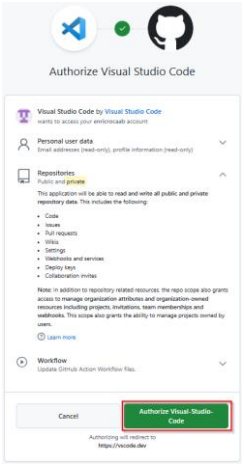
Depending if you have already signed in GitHub these screen may appear or not:

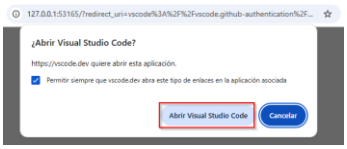
In the lower right corner of the screen appears:



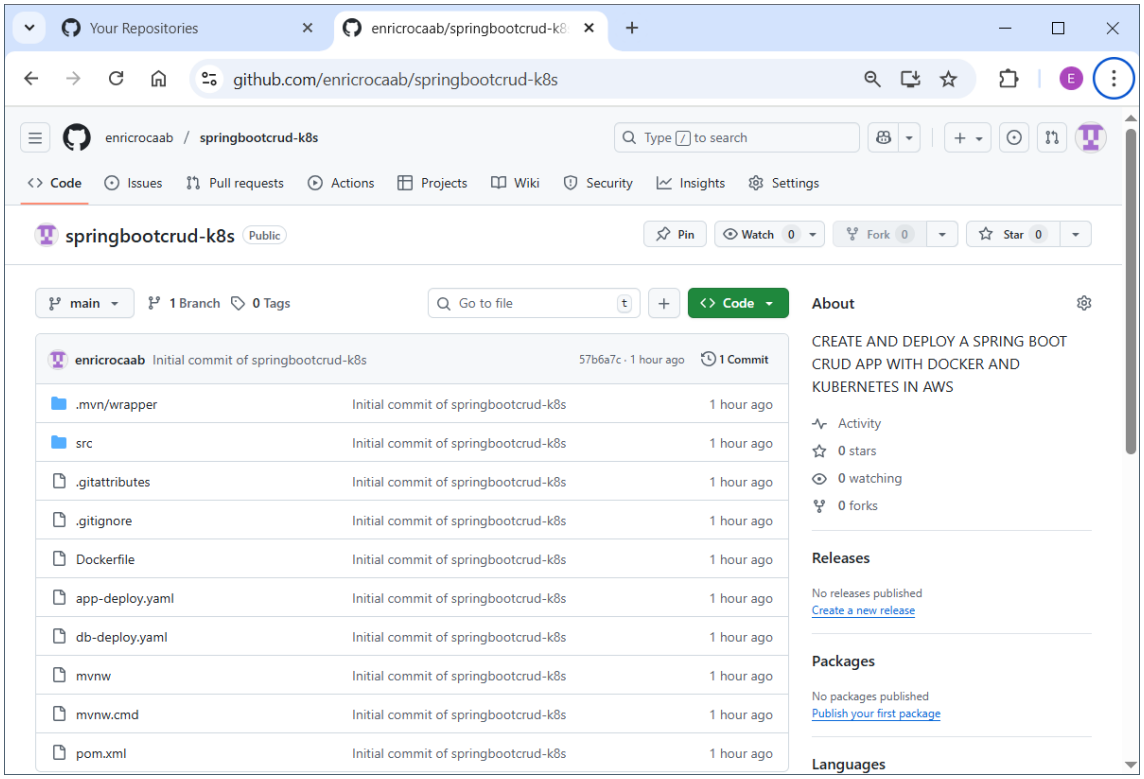
In a browser we can get messages to confirm and authorize to open this link:







Now we have our project files in the GitHub repository:

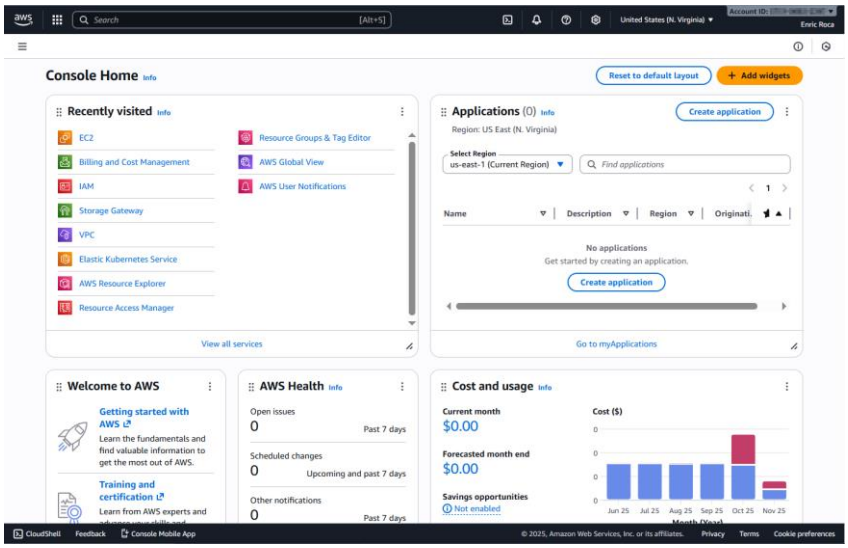


Part III: Launch an instance on AWS and install the software

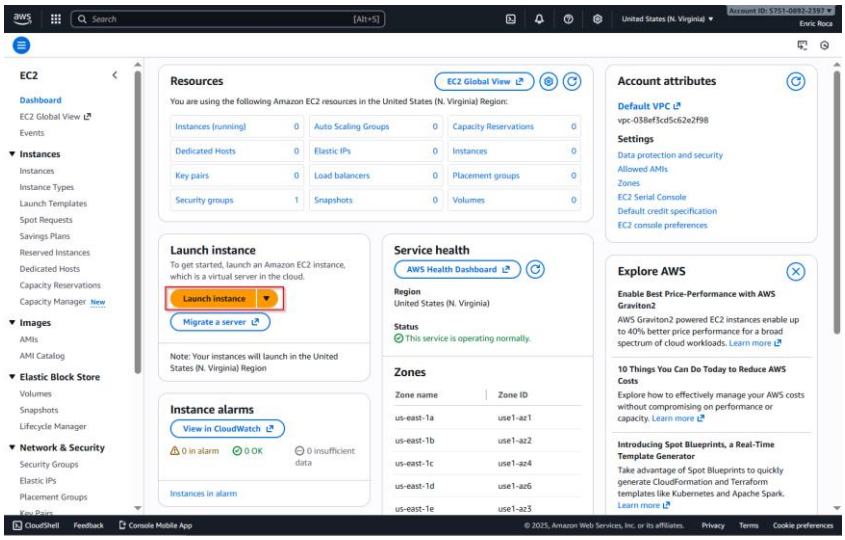
In the first part of the project, we have created the executable jar file to run our application by passing the data source parameters to connect to a MySQL database. In the second part, we have created the needed configuration files to deploy it. Now we will launch an instance on **AWS** and install the software needed to get a test open-source system for automating the deployment of our containerized App, scaling, and managing it. We will achieve this with **Minikube**, as we will see later.

III.1- Launch the Linux instance on AWS

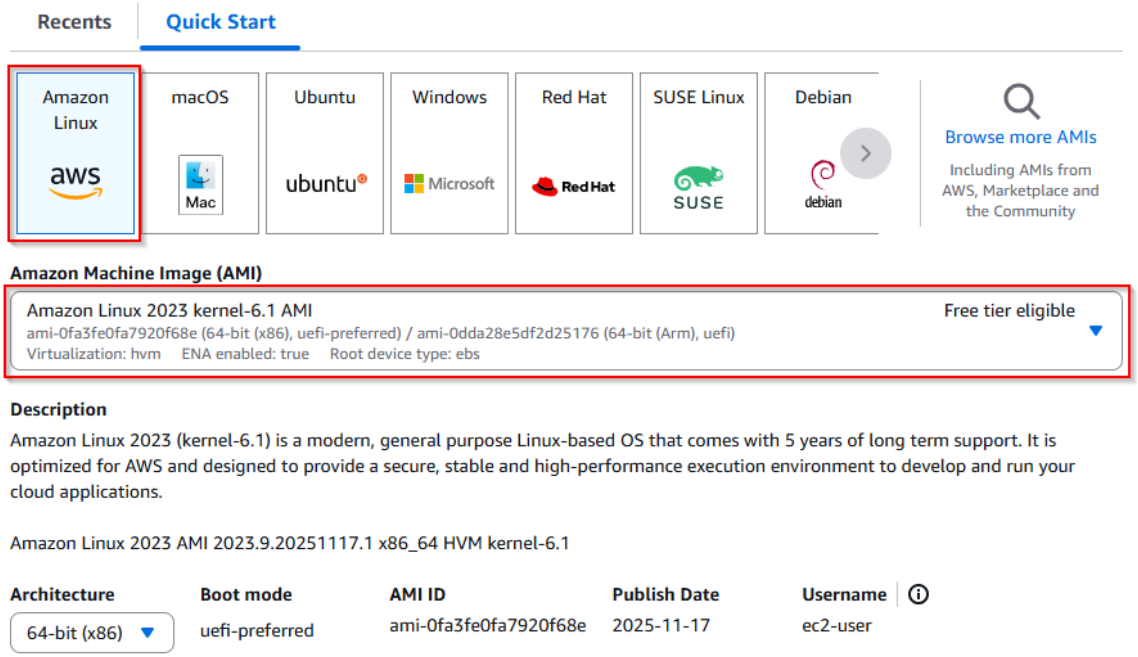
We log in to AWS using your own credentials:



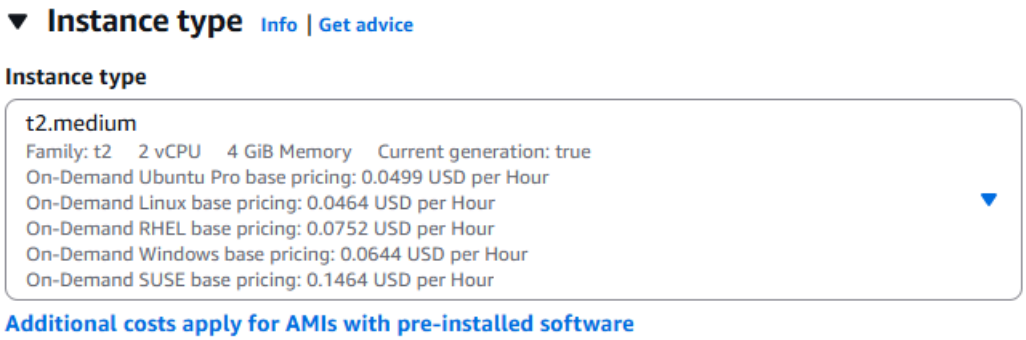
Go to EC2 and click on "**Launch an instance**":



Select the AMI (Amazon Machine Image) “**Amazon Linux 2023 Kernel-6.1 AMI**”:

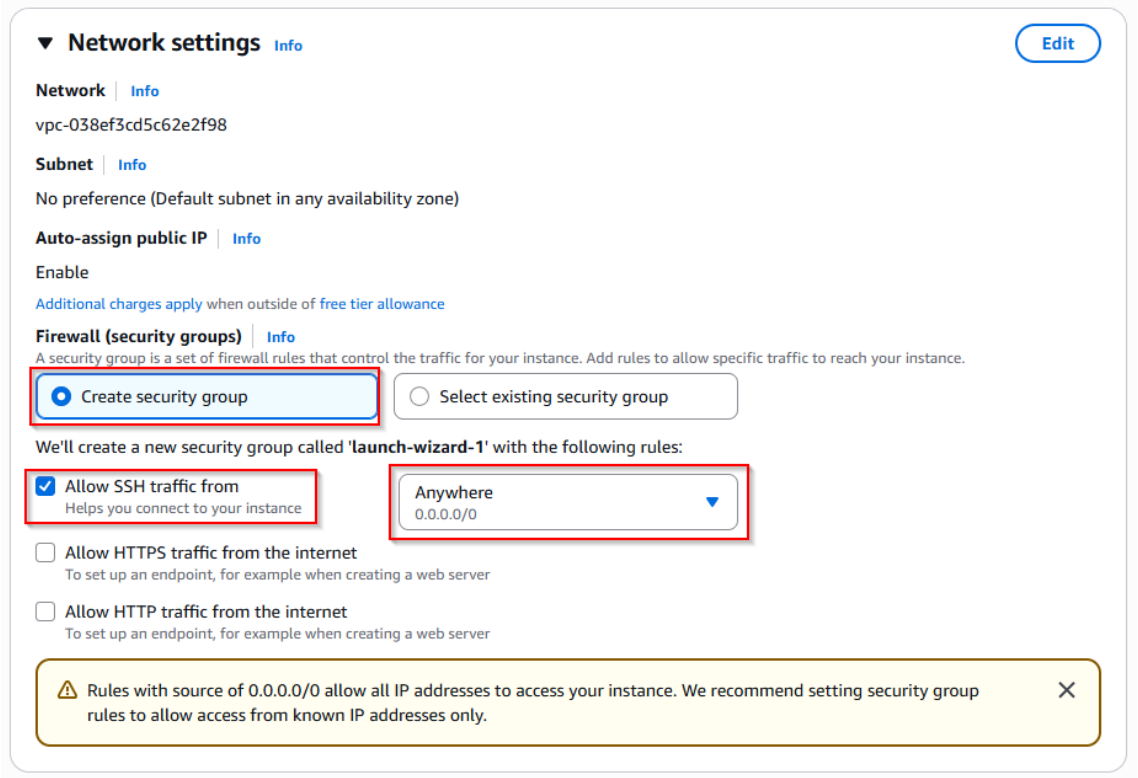


Choose **t2.medium** as the type of instance:

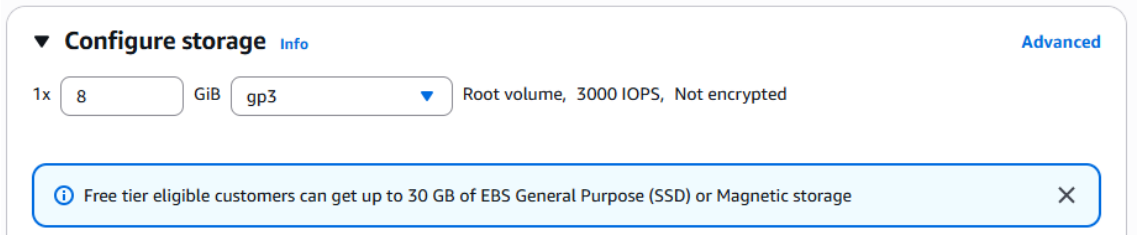


We don’t need to create a key pair.

In **Network settings** we create a security group and allow SSH traffic from anywhere (in a production environment we should specify a network or IP for security):

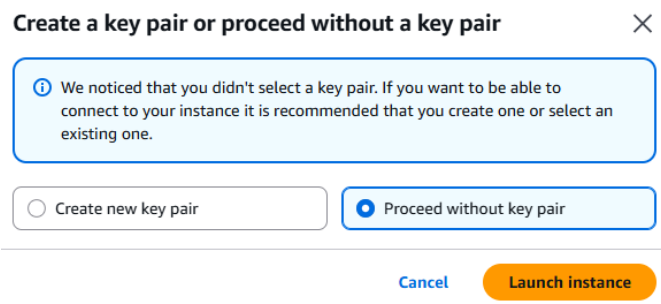


We configure an **8 GiB** of storage capacity:

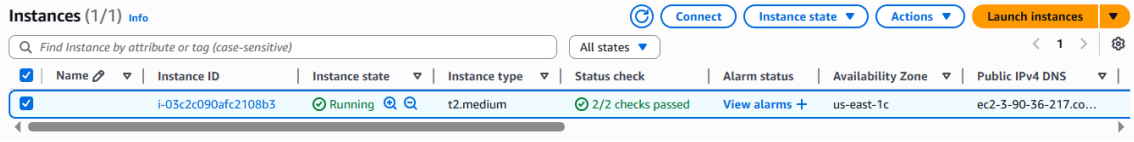


gp3 means **General Purpose SSD** (Solid State Drive) volumes, which is the current generation of general-purpose storage volumes offered by **Amazon Elastic Block Store** (EBS). These volumes are designed to provide a balance of price and performance for a wide variety of transactional workloads, such as virtual desktops, medium-sized databases (like MySQL and Oracle), development/test environments, and boot volumes.

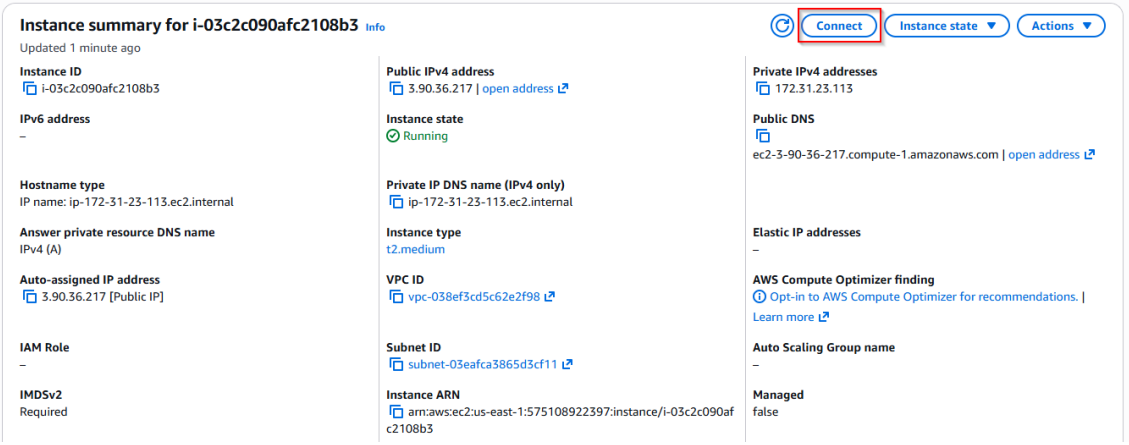
Click on "**Launch instance**", and confirm proceed without key pair:

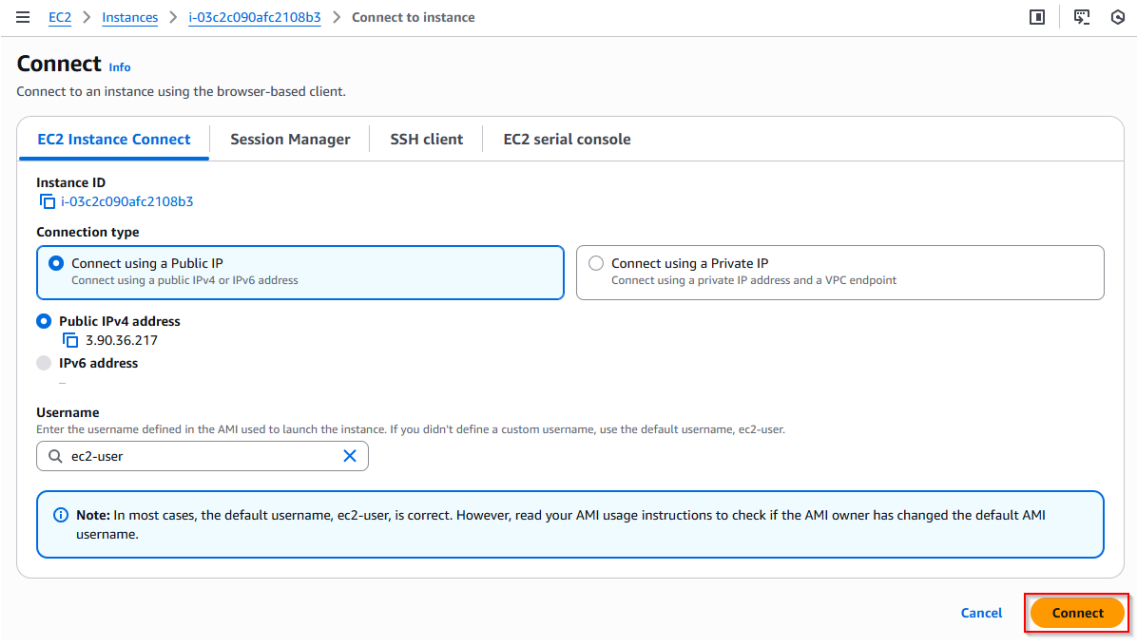


In few seconds we have our instance running. Wait until pass the two-check status:

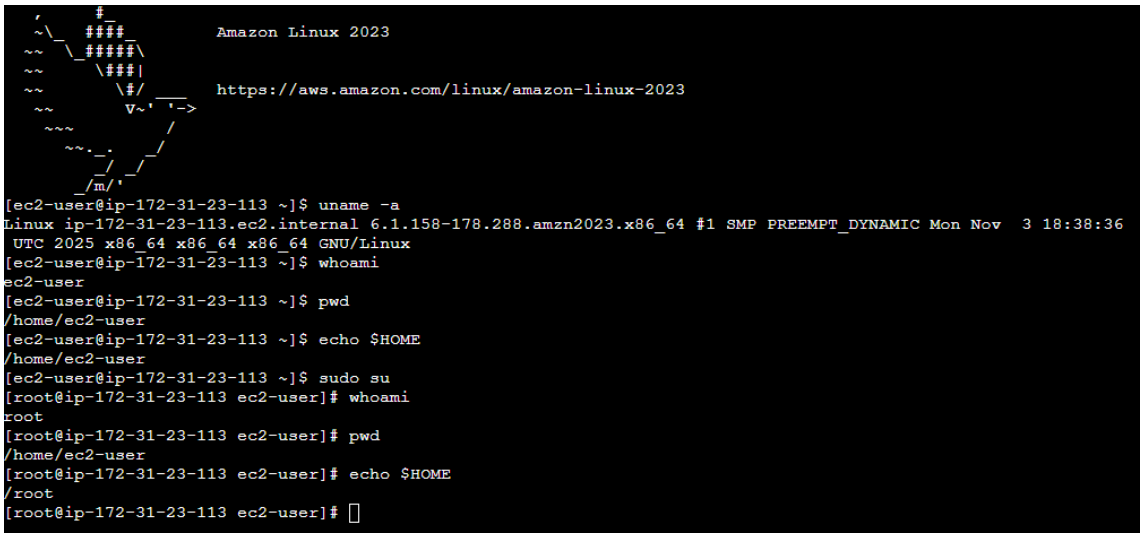


Select the instance to see all the details, and push **Connect**:





Once in the AWS terminal console, we check the system and user environment:



Commands:

- **uname -a** : It provides details about the system's kernel and architecture: Kernel Name = Linux; Network Node Hostname = ip-172-31-xx-yy.ec2.internal; Kernel Release, Version, etc.
- **whoami:** It confirms which user account we are currently logged in.
- **pwd** (**P**rint **W**orking **D**irectory): This command shows the absolute pathname of the current directory.
- **sudo su**: It allows a non-root user who has sudo privileges to switch to an interactive root shell session
  - **sudo** (**Su**peruser **Do**): It allows an authorized user (a "sudoer") to execute a single command as another user (by default, the root user). When you type sudo <command>, you are asked for your own password, not the root password.
  - **su** (**S**ubstitute **U**ser): This command, when run without a username, attempts to switch the current session to the root user. Normally, su requires the root password.

III.2- Install Docker, Minikube, kubectl, Git and Maven

Kubernetes is an open-source platform for container orchestration. It's a system designed to automate the deployment, scaling, and management of containerized applications (like those built with Docker) across a cluster of machines. It is an excellent choice for: Large-scale microservices architectures, Applications with high, fluctuating traffic that require dynamic scaling, and Organizations that need multi-cloud flexibility or desire strict control over their infrastructure.

For our simple App it wouldn't make sense the use of Kubernetes because of its complexity and infra requirements, but for learning purposes it’s highly interesting, and this is where **Minikube** comes in. It allows us to create a single-node Kubernetes cluster on our home workstation or on a single instance machine, which is our case now. Before installing Minikube, we must install **Docker**, as we will see now.

To install the software, as usual, first we must perform a fully system packages update. For Red Hat-based distributions (like CentOS, Fedora, and RHEL) and its derivatives like Amazon Linux, we use:

yum update -y

Where:

- **yum** (Yellowdog Updater Modified) is the primary package manager tool for installing, updating, and removing software packages from your system's repositories.
- **-y** is the flag that tells the yum command to automatically answer "yes" to any interactive prompts that come up during the update process

```
[ec2-user@ip-172-31-23-113 ~]$ sudo su
[root@ip-172-31-23-113 ec2-user]# whoami
root
[root@ip-172-31-23-113 ec2-user]# pwd
/home/ec2-user
[root@ip-172-31-23-113 ec2-user]# echo $HOME
/root
[root@ip-172-31-23-113 ec2-user]# yum update -y
Amazon Linux 2023 Kernel Livepatch repository       [===                                                ] ---  B/s
Amazon Linux 2023 Kernel Livepatch repository                                                            193 kB/s
|  29 kB     00:00
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-23-113 ec2-user]#
```

Now we will install the latest **Docker** Engine packages using the command:

yum install docker -y

```
[root@ip-172-31-23-113 ec2-user]# yum install docker -y
Last metadata expiration check: 0:01:56 ago on Thu Nov 20 11:14:19 2025.
Dependencies resolved.
================================================================================================================================
 Package                          Architecture          Version                                Repository                Size
================================================================================================================================
Installing:
 docker                           x86_64                25.0.13-1.amzn2023.0.2                 amazonlinux               46 M
Installing dependencies:
 container-selinux                noarch                4:2.242.0-1.amzn2023                   amazonlinux               58 k
 containerd                       x86_64                2.1.4-1.amzn2023.0.2                   amazonlinux               23 M
 iptables-libs                    x86_64                1.8.8-3.amzn2023.0.2                   amazonlinux              401 k
 iptables-nft                     x86_64                1.8.8-3.amzn2023.0.2                   amazonlinux              183 k
 libcgroup                        x86_64                3.0-1.amzn2023.0.1                     amazonlinux               75 k
 libnetfilter_conntrack           x86_64                1.0.8-2.amzn2023.0.2                   amazonlinux               58 k
 libnfnetlink                     x86_64                1.0.1-19.amzn2023.0.2                  amazonlinux               30 k
 libnftnl                         x86_64                1.2.2-2.amzn2023.0.2                   amazonlinux               84 k
 pigz                             x86_64                2.5-1.amzn2023.0.3                     amazonlinux               83 k
 runc                             x86_64                1.3.3-2.amzn2023.0.1                   amazonlinux              3.9 M

Transaction Summary
================================================================================================================================
Install  11 Packages
```

...

```
Installed:
  container-selinux-4:2.242.0-1.amzn2023.noarch      containerd-2.1.4-1.amzn2023.0.2.x86_64    docker-25.0.13-1.amzn2023.0.2.x86_64
  iptables-libs-1.8.8-3.amzn2023.0.2.x86_64          iptables-nft-1.8.8-3.amzn2023.0.2.x86_64  libcgroup-3.0-1.amzn2023.0.1.x86_64
  libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64 libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64 libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  pigz-2.5-1.amzn2023.0.3.x86_64                     runc-1.3.3-2.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-23-113 ec2-user]#
```

It installs the packages:

- **Main Package**: docker. This is the core Docker Engine package that provides the Docker Daemon, the Docker CLI, and the necessary tools for building, shipping, and running containers.
- **Dependency Packages**: These packages are automatically installed to ensure Docker has all the necessary components for its operations:
  - **Container Runtime and Management**: containerd (manage the complete container lifecycle), runc (container execution command-line tool) , libcgroup.
  - **Security**: container-selinux
  - **Networking and Firewall**: iptables-libs, iptables-nft, libnfnetlink, libnetfilter_conntrack (stateful firewall operations and NAT), libnftnl.
  - **Utility**: pigz (to compress and decompress container images efficiently).

Check the installation with **docker -v** :

```
[root@ip-172-31-23-113 ec2-user]# docker -v
Docker version 25.0.13, build 0bab007
[root@ip-172-31-23-113 ec2-user]#
```

Start the Docker service:

systemctl start docker

```
[root@ip-172-31-23-113 ec2-user]# systemctl start docker
[root@ip-172-31-23-113 ec2-user]#
```

and configure the Docker service to start automatically every time the operating system boots up:

systemctl enable docker

```
[root@ip-172-31-23-113 ec2-user]# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-23-113 ec2-user]#
```

We can see in the output that a symlink (symbolic link) is created. The symlink points from the directory /etc/systemd/system/multi-user.target.wants/ to the actual service file /usr/lib/systemd/system/docker.service. Any service symlinked into its .wants directory will be automatically started when the system reaches this target state during the boot process.

To check the Docker status:

systemctl status docker

```
[root@ip-172-31-23-113 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Thu 2025-11-20 12:03:31 UTC; 7min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 30736 (dockerd)
      Tasks: 9
     Memory: 28.8M
        CPU: 360ms
     CGroup: /system.slice/docker.service
             └─30736 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
```

For instance, we can change to the ec2-user session to avoid working more with the root one:

su ec2-user

```
[root@ip-172-31-23-113 ec2-user]# su ec2-user
[ec2-user@ip-172-31-23-113 ~]$ whoami
ec2-user
[ec2-user@ip-172-31-23-113 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-23-113 ~]$ echo $HOME
/home/ec2-user
[ec2-user@ip-172-31-23-113 ~]$
```

Now we can add the user ec2-user to the docker group, so this user can run Docker commands without using sudo:

sudo usermod -aG docker $USER && newgrp docker

This command is a combination of two separate shell commands:

- **sudo usermod -aG docker $USER** : It modifies the current user's account (ec2-user); -a (append): Ensures the user is added to the new group without removing them from their existing groups; -G docker: Specifies the supplementary group to add the user to, which is the docker group.
- **newgrp docker** : It changes the shell's current group ID; newgrp: Stands for "new group." This command is used to change the current group ID during a login session; docker: This specifies the new group you want to join.

```
[ec2-user@ip-172-31-23-113 ~]$ sudo usermod -aG docker $USER && newgrp docker
[ec2-user@ip-172-31-23-113 ~]$ groups ec2-user
ec2-user : ec2-user adm wheel systemd-journal docker
[ec2-user@ip-172-31-23-113 ~]$
```

To install Minikube, first download the installer:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm

This command is used in Linux environments that utilize the RPM Package Manager (like Fedora, CentOS, or RHEL) to download the latest 64-bit Minikube installer file:

- **curl**: This is a command-line tool for transferring data with URL syntax. It's widely used for downloading files from the internet.
- **-L**: This option tells curl to follow HTTP redirects. When you access a "latest" URL, the server often redirects you to the actual file location, and -L ensures curl follows that link to start the download.
- **-O**: This option tells curl to save the output to a local file named the same as the remote file. In this case, the downloaded file will be saved as minikube-latest.x86_64.rpm in your current directory.
- **https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm**: This is the URL (web address) of the file you are requesting to download.

```
[ec2-user@ip-172-31-23-113 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-23-113 ~]$ ls
[ec2-user@ip-172-31-23-113 ~]$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 50.5M  100 50.5M    0     0  90.5M      0 --:--:-- --:--:-- --:--:-- 90.5M
[ec2-user@ip-172-31-23-113 ~]$ ls
minikube-latest.x86_64.rpm
[ec2-user@ip-172-31-23-113 ~]$
```

Install Minikube:

sudo rpm -Uvh minikube-latest.x86_64.rpm

In detail:

- **sudo** (**S**uperuser **D**o): Executes the following command with root privileges. Installing or upgrading system-level software packages requires elevated permissions.
- **rpm** (**R**ed Hat **P**ackage **M**anager): This is the command-line utility for managing software packages distributed in the RPM format.
  - **-U** (Upgrade):This is the main action. It tells rpm to install the package if it's not already installed or upgrade it if an older version is present.
  - **-v** (Verbose): Displays more detailed information during the execution process.
  - **-h** (Hash): Displays hash marks (#) as the package archives are unpacked, providing a visual progress bar.

```
[ec2-user@ip-172-31-23-113 ~]$ sudo rpm -Uvh minikube-latest.x86_64.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:minikube-1.37.0-0                ################################# [100%]
[ec2-user@ip-172-31-23-113 ~]$
```

The next step is to install **kubectl** (Kubernetes command-line tool). It is the primary tool used to interact with and manage Kubernetes clusters and with the Minikube cluster.

Before to proceeding, it is important to confirm the version of kubectl that will be utilized. We will install kubectl using the most recent version, which aligns with the latest stable version of Kubernetes. Let's check the latest stable version of Kubernetes:

curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt

```
[ec2-user@ip-172-31-23-113 ~]$ curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt
v1.31.0[ec2-user@ip-172-31-23-113 ~]$
```

This command is used to retrieve the version number of the current stable Kubernetes release and print it directly to the console:

- **-s**: A flag that tells curl to operate in silent mode. This suppresses the progress meter, error messages, and generally keeps the output clean, making it ideal for piping the result into a variable or another command.

The latest stable version of Kubernetes when executing this command is v1.31.0 .

Now download this version of kubectl using:

curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.31.0/bin/linux/amd64/kubectl

```
[ec2-user@ip-172-31-23-113 ~]$ curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.31.0/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0  91.5M      0 --:--:-- --:--:-- --:--:-- 91.7M
[ec2-user@ip-172-31-23-113 ~]$ ls
kubectl  minikube-latest.x86_64.rpm
[ec2-user@ip-172-31-23-113 ~]$
```

After that, we must give the kubectl binary file permission to run as a program:

sudo chmod +x ./kubectl

The **chmod +x** command adds the x permission for all three categories (Owner, Group, and Others) unless specific targets are defined (e.g., u+x for just the owner).

```
[ec2-user@ip-172-31-23-113 ~]$ sudo chmod +x ./kubectl
[ec2-user@ip-172-31-23-113 ~]$ ls -l
total 106804
-rwxr-xr-x. 1 ec2-user ec2-user 56381592 Nov 21 10:24 kubectl
-rw-r--r--. 1 ec2-user ec2-user 52979185 Nov 21 09:49 minikube-latest.x86_64.rpm
[ec2-user@ip-172-31-23-113 ~]$
```

Now, to ensure that kubectl will be found when the user requests it, we must put it in a folder included in the $PATH environment variable, typically in /usr/local/bin:

```
[ec2-user@ip-172-31-23-113 ~]$ echo $PATH
/home/ec2-user/.local/bin:/home/ec2-user/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
[ec2-user@ip-172-31-23-113 ~]$
```

So, we move the file:

sudo mv ./kubectl /usr/local/bin/kubectl

```
[ec2-user@ip-172-31-23-113 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
[ec2-user@ip-172-31-23-113 ~]$ ls
minikube-latest.x86_64.rpm
[ec2-user@ip-172-31-23-113 ~]$ ls /usr/local/bin/k*
/usr/local/bin/kubectl
[ec2-user@ip-172-31-23-113 ~]$
```

Start Minikube:

minikube start

```
[ec2-user@ip-172-31-23-113 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-23-113 ~]$ minikube start
* minikube v1.37.0 on Amazon 2023.9.20251117 (xen/amd64)
* Automatically selected the docker driver. Other choices: none, ssh

X The requested memory allocation of 3072MiB does not leave room for system overhead (total system memory: 3904MiB). You may face stability issues.
* Suggestion: Start minikube with less memory allocated: 'minikube start --memory=3072mb'

* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.48 ...
* Downloading Kubernetes v1.34.0 preload ...
    > preloaded-images-k8s-v18-v1...:  337.07 MiB / 337.07 MiB  100.00% 72.52 M
    > gcr.io/k8s-minikube/kicbase...:  488.51 MiB / 488.52 MiB  100.00% 71.58 M
* Creating docker container (CPUs=2, Memory=3072MB) ...
* Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass

! /usr/local/bin/kubectl is version 1.31.0, which may have incompatibilities with Kubernetes 1.34.0.
  - Want kubectl v1.34.0? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
[ec2-user@ip-172-31-23-113 ~]$
```

Now, check that it is working by getting the pods (-A is for getting the pods of all namespaces, so it shows the pods of the system namespace, that is kube-system):

kubectl get pod -A

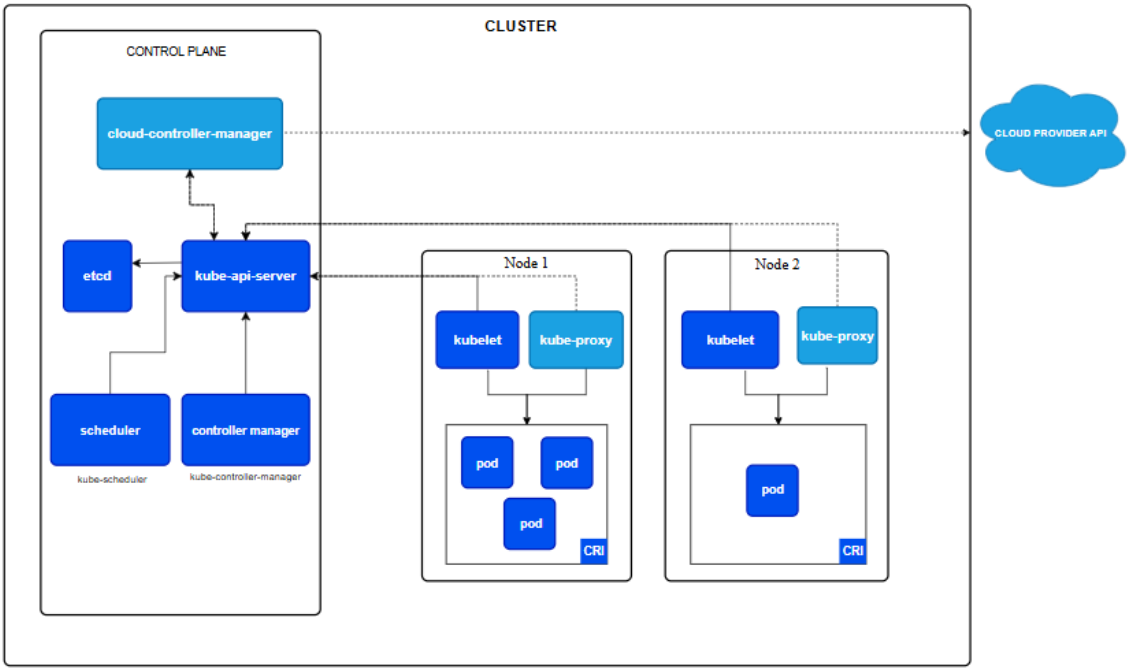
```
[ec2-user@ip-172-31-23-113 ~]$ kubectl get pod -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS      AGE
kube-system   coredns-66bc5c9577-4vznb           1/1     Running   0             2m5s
kube-system   etcd-minikube                      1/1     Running   0             2m12s
kube-system   kube-apiserver-minikube            1/1     Running   0             2m12s
kube-system   kube-controller-manager-minikube   1/1     Running   0             2m11s
kube-system   kube-proxy-f9r78                   1/1     Running   0             2m5s
kube-system   kube-scheduler-minikube            1/1     Running   0             2m11s
kube-system   storage-provisioner                1/1     Running   1 (95s ago)   2m9s
[ec2-user@ip-172-31-23-113 ~]$
```

This is good! Minikube (Kubernetes) and kubectl are working fine.

A Kubernetes cluster is made up of two main sets of components: the **Control Plane** and the **Worker Nodes**.

- **Control Plane Components**: The Control Plane makes global decisions about the cluster (like scheduling) and maintains its desired state.
  - **kube-apiserver**: The front door of the cluster. It exposes the Kubernetes API and is the central hub for all communication. All requests (from users, CLI tools like kubectl, or other components) must go through the API server.
  - **etcd**: The cluster memory. This is a consistent and highly available distributed key-value store that holds all the cluster's configuration data, state, and metadata.
  - **kube-scheduler**: It watches for newly created Pods and selects the best Node for them to run on. It considers factors like resource requirements, hardware constraints, and policies.
  - **kube-controller-manager**: It runs various controller processes (like the Node Controller, Replication Controller, etc.) that continuously monitor the actual state of the cluster and make changes to move it closer to the desired state.
  - **cloud-controller-manager** (Optional): This component is used when running Kubernetes on a public cloud. It embeds cloud-specific control logic, allowing the cluster to interact with the cloud provider's APIs to manage resources like load balancers and storage volumes.

- **Worker Node Components**: Worker Nodes are the machines (VMs or physical servers) that run your containerized applications (Pods).
  - **kubelet**: The node agent. It runs on every Node and is the primary communication channel with the Control Plane. It ensures that the containers are running and healthy on its Node.
  - **kube-proxy**: The network proxy. It maintains network rules on the Node, allowing communication to and from Pods, both within the cluster and to the outside world. It handles load balancing for Services.
  - **Container Runtime**: The engine. This is the software responsible for actually running the containers (e.g., containerd, CRI-O, or Docker Engine). It handles pulling images and executing the containers based on instructions from the Kubelet.



When we use Minikube, it runs a single-node Kubernetes cluster on one machine. So, we can see that all the main components of a Kubernetes cluster are running in this machine:

```
[ec2-user@ip-172-31-23-113 ~]$ kubectl get pod -A -o wide
NAMESPACE     NAME                                   READY   STATUS    RESTARTS      AGE   IP             NODE       NOMINATED NODE   READINESS GATES
kube-system   coredns-66bc5c9577-4vznb               1/1     Running   0             11m   10.244.0.2     minikube   <none>           <none>
kube-system   etcd-minikube                          1/1     Running   0             11m   192.168.49.2   minikube   <none>           <none>
kube-system   kube-apiserver-minikube                1/1     Running   0             11m   192.168.49.2   minikube   <none>           <none>
kube-system   kube-controller-manager-minikube       1/1     Running   0             11m   192.168.49.2   minikube   <none>           <none>
kube-system   kube-proxy-f9r78                       1/1     Running   0             11m   192.168.49.2   minikube   <none>           <none>
kube-system   kube-scheduler-minikube                1/1     Running   0             11m   192.168.49.2   minikube   <none>           <none>
kube-system   storage-provisioner                    1/1     Running   1 (11m ago)   11m   192.168.49.2   minikube   <none>           <none>
[ec2-user@ip-172-31-23-113 ~]$
```

Now, we will try to use the **Minikube Dashboard**. We can see it in the addons list:

minikube addons list

```
[ec2-user@ip-172-31-23-113 ~]$ minikube addons list
|-----------------------------|----------|----------|--------------------------------|
|         ADDON NAME          | PROFILE  |  STATUS  |           MAINTAINER           |
|-----------------------------|----------|----------|--------------------------------|
| ambassador                  | minikube | disabled | 3rd party (Ambassador)         |
| amd-gpu-device-plugin       | minikube | disabled | 3rd party (AMD)                |
| auto-pause                  | minikube | disabled | minikube                       |
| cloud-spanner               | minikube | disabled | Google                         |
| csi-hostpath-driver         | minikube | disabled | Kubernetes                     |
| dashboard                   | minikube | disabled | Kubernetes                     |
| default-storageclass        | minikube | enabled ✅| Kubernetes                     |
| efk                         | minikube | disabled | 3rd party (Elastic)            |
| freshpod                    | minikube | disabled | Google                         |
```

…

To enable the addon:

minikube addons enable dashboard

```
[ec2-user@ip-172-31-23-113 ~]$ minikube addons enable dashboard
* dashboard is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image docker.io/kubernetesui/dashboard:v2.7.0
  - Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
* Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server

* The 'dashboard' addon is enabled
[ec2-user@ip-172-31-23-113 ~]$
```

Create a second terminal session and execute (remember that the folder where kubectl is located, /usr/local/bin/, is in the system's path):

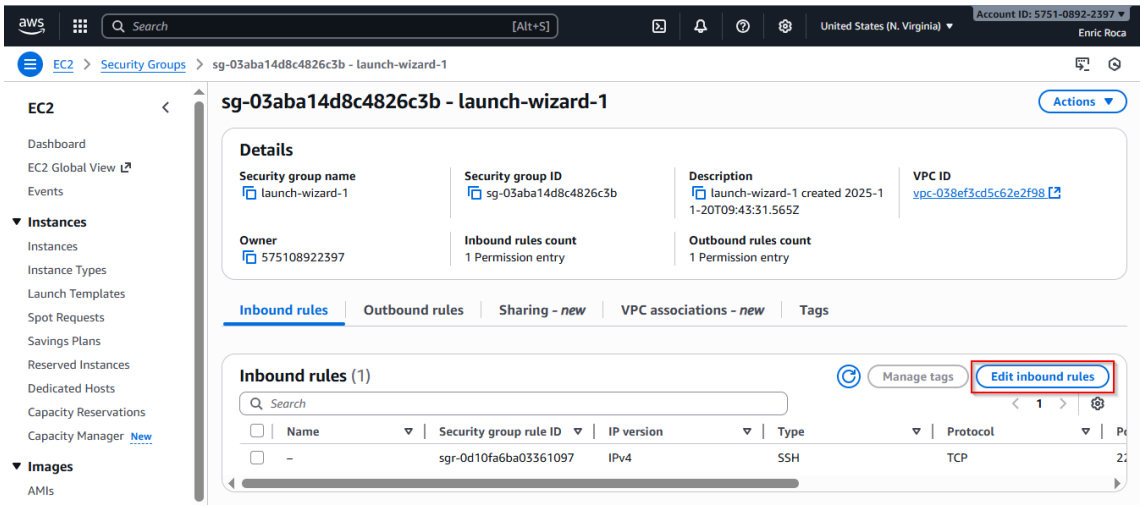
kubectl proxy --address='0.0.0.0' --accept-hosts='^*$'

🗐 This command is used to create a local HTTP proxy server that allows us to access the Kubernetes API server from outside the Minikube instance.

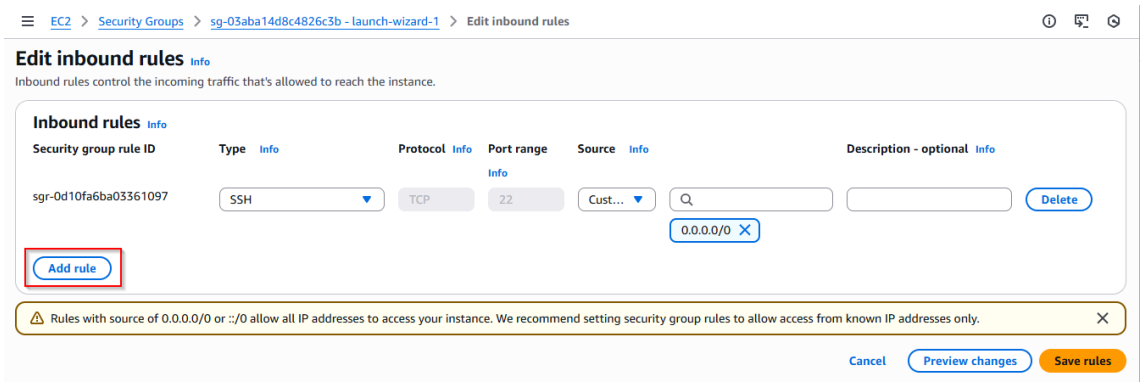
- **kubectl proxy**: It starts a proxy server on the Amazon Linux instance (usually on port **8001** by default, unless specified otherwise). This proxy handles the secure connection and authentication to the Kubernetes API server running inside the Minikube cluster, acting as a gateway.
- **--address='0.0.0.0'** : By default, kubectl proxy binds to 127.0.0.1 (localhost), which only allows access from within the instance itself. The --address='0.0.0.0' flag overrides this default and instructs the proxy to listen on all available network interfaces of the Amazon Linux 2023 instance. It allows us to connect to the proxy from our external machine (our personal computer) using the instance's public IP address or public DNS name.
- **--accept-hosts='^*$'** : The kubectl proxy includes a security feature that filters host headers in incoming requests. By default, it only accepts connections originating from localhost and 127.0.0.1. The --accept-hosts='^*$' flag provides a regular expression (^.*$) that matches any hostname or IP address. This is required because when you access the dashboard from your external machine, the hostname in the request header will be the public IP or DNS name of your EC2 instance, not localhost. This flag allows the proxy to accept these external requests.

```
[ec2-user@ip-172-31-23-113 ~]$ kubectl proxy --address='0.0.0.0' --accept-hosts='^*$'
Starting to serve on [::]:8001
```

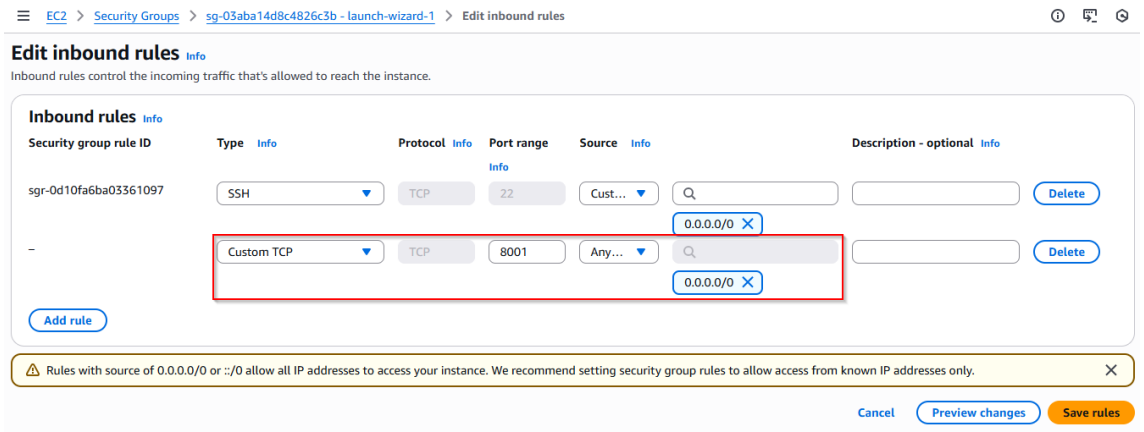
Now go to the **AWS Administration Console**, we are going to add an **inbound rule** to allow the **8001 TCP** traffic from anywhere (in production we should specify more for security): For that, select the security group and click on "**Edit inbound rules**":



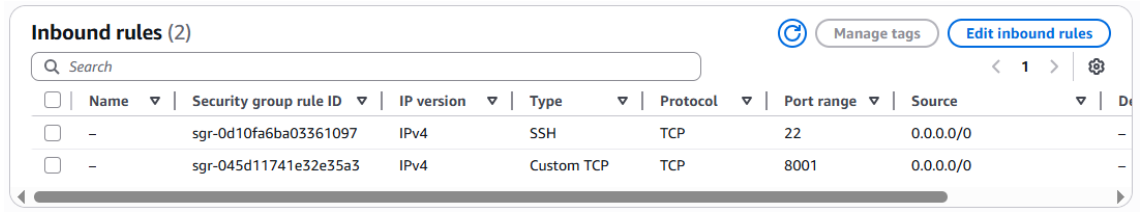
Push on “**Add rule**”:



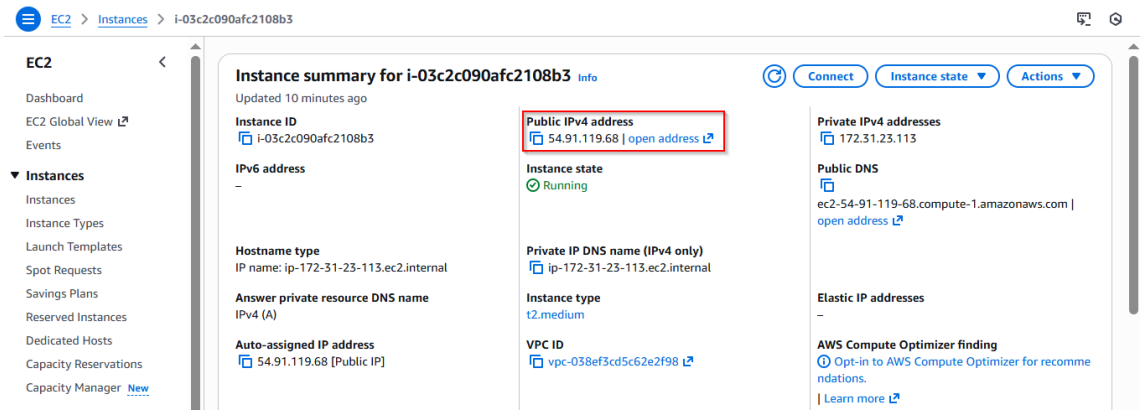
Chose type **Custom TCP**, Port range **8081** and source **Anywhere-IPv4**:



Click on “**Save rules**”.



Now, we need to know the public IP address of our EC2 instance, so we go to:

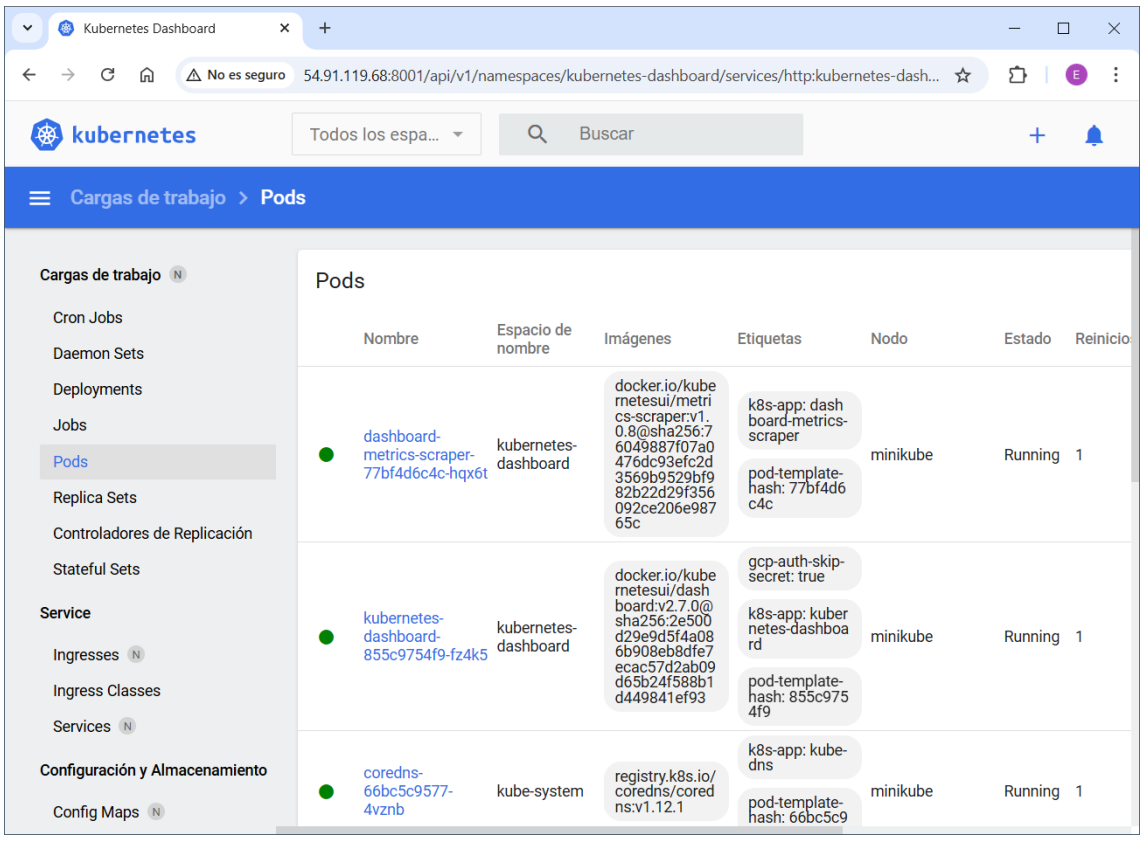


So, now the public IP is 54.91.119.68 .

Open a browser in your laptop and access the next URL specifying the public IP of your instance:

http://**54.91.119.68**:8001/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

And it appears the Dashboard:



Before finishing this part of the project, we are going to install **Git** and **Maven** on our EC2 instance. We will use Git to clone the GitHub repository and Maven to build the application's

executable JAR file, which will allow us to then create a Docker image of the application. For that, we execute:

sudo yum install git -y

```
[ec2-user@ip-172-31-23-113 ~]$ sudo yum install git -y
Last metadata expiration check: 1 day, 7:32:47 ago on Thu Nov 20 11:14:19 2025.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                 Repository      Size
================================================================================
Installing:
 git                     x86_64              2.50.1-1.amzn2023.0.1   amazonlinux     53 k
Installing dependencies:
 git-core                x86_64              2.50.1-1.amzn2023.0.1   amazonlinux    4.9 M
 git-core-doc            noarch              2.50.1-1.amzn2023.0.1   amazonlinux    2.8 M
 perl-Error              noarch              1:0.17029-5.amzn2023.0.2 amazonlinux    41 k
 perl-File-Find          noarch              1.37-477.amzn2023.0.7   amazonlinux     25 k
 perl-Git                noarch              2.50.1-1.amzn2023.0.1   amazonlinux     41 k
 perl-TermReadKey        x86_64              2.38-9.amzn2023.0.2     amazonlinux     36 k
 perl-lib                x86_64              0.65-477.amzn2023.0.7   amazonlinux     15 k
```

...

```
Installed:
  git-2.50.1-1.amzn2023.0.1.x86_64              git-core-2.50.1-1.amzn2023.0.1.x86_64          git-core-doc-2.50.1-1.amzn2023.0.1.noarch
  perl-Error-1:0.17029-5.amzn2023.0.2.noarch    perl-File-Find-1.37-477.amzn2023.0.7.noarch    perl-Git-2.50.1-1.amzn2023.0.1.noarch
  perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64   perl-lib-0.65-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-23-113 ~]$
```

Check it's working with **git -v** :

```
[ec2-user@ip-172-31-23-113 ~]$ git -v
git version 2.50.1
[ec2-user@ip-172-31-23-113 ~]$
```

sudo yum install maven -y

```
[ec2-user@ip-172-31-23-113 ~]$ sudo yum install maven -y
Last metadata expiration check: 1 day, 8:12:20 ago on Thu Nov 20 11:14:19 2025.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                 Repository      Size
================================================================================
Installing:
 maven                   noarch              1:3.8.4-3.amzn2023.0.5  amazonlinux     18 k
Installing dependencies:
 alsa-lib                x86_64              1.2.7.2-1.amzn2023.0.2  amazonlinux    504 k
 apache-commons-cli      noarch              1.5.0-3.amzn2023.0.3    amazonlinux     76 k
 apache-commons-codec    noarch              1.15-6.amzn2023.0.3     amazonlinux    303 k
 apache-commons-io       noarch              1:2.8.0-7.amzn2023.0.5  amazonlinux    283 k
 apache-commons-lang3    noarch              3.18.0-1.amzn2023.0.1   amazonlinux    655 k
 atinject                noarch              1.0.5-3.amzn2023.0.3    amazonlinux     23 k
 cairo                   x86_64              1.18.0-4.amzn2023.0.3   amazonlinux    717 k
 cdi-api                 noarch              2.0.2-6.amzn2023.0.3    amazonlinux     54 k
 dejavu-sans-fonts       noarch              2.37-16.amzn2023.0.2    amazonlinux    1.3 M
 dejavu-sans-mono-fonts  noarch              2.37-16.amzn2023.0.2    amazonlinux    467 k
 dejavu-serif-fonts      noarch              2.37-16.amzn2023.0.2    amazonlinux    1.0 M
```

...

```
Installed:
  alsa-lib-1.2.7.2-1.amzn2023.0.2.x86_64                apache-commons-cli-1.5.0-3.amzn2023.0.3.noarch
  apache-commons-codec-1.15-6.amzn2023.0.3.noarch       apache-commons-io-1:2.8.0-7.amzn2023.0.5.noarch
  apache-commons-lang3-3.18.0-1.amzn2023.0.1.noarch     atinject-1.0.5-3.amzn2023.0.3.noarch
  cairo-1.18.0-4.amzn2023.0.3.x86_64                    cdi-api-2.0.2-6.amzn2023.0.3.noarch
  dejavu-sans-fonts-2.37-16.amzn2023.0.2.noarch         dejavu-sans-mono-fonts-2.37-16.amzn2023.0.2.noarch
  dejavu-serif-fonts-2.37-16.amzn2023.0.2.noarch        fontconfig-2.13.94-2.amzn2023.0.2.x86_64
  fonts-filesystem-1:2.0.5-12.amzn2023.0.2.noarch       freetype-2.13.2-5.amzn2023.0.1.x86_64
  google-guice-4.2.3-8.amzn2023.0.6.noarch              google-noto-fonts-common-20240401-1.amzn2023.0.2.noarch
  google-noto-sans-vf-fonts-20240401-1.amzn2023.0.2.noarch graphite2-1.3.14-7.amzn2023.0.2.x86_64
  guava-31.0.1-3.amzn2023.0.6.noarch                    harfbuzz-7.0.0-2.amzn2023.0.2.x86_64
  httpcomponents-client-4.5.13-4.amzn2023.0.4.noarch    httpcomponents-core-4.4.13-6.amzn2023.0.3.noarch
  jakarta-annotations-1.3.5-13.amzn2023.0.3.noarch      jansi-2.4.0-3.amzn2023.0.3.x86_64
  java-17-amazon-corretto-devel-1:17.0.17+10-1.amzn2023.1.x86_64 java-17-amazon-corretto-headless-1:17.0.17+10-1.amzn2023.1.x86_64
  javapackages-filesystem-6.0.0-7.amzn2023.0.6.noarch   jcl-over-slf4j-1.7.32-3.amzn2023.0.4.noarch
  jsoup-1.16.1-4.amzn2023.0.2.noarch                    jsr-305-3.0.2-5.amzn2023.0.4.noarch
  langpacks-core-font-en-3.0-21.amzn2023.0.4.noarch     libX11-1.8.10-2.amzn2023.0.1.x86_64
  libX11-common-1.8.10-2.amzn2023.0.1.noarch            libXau-1.0.11-6.amzn2023.0.1.x86_64
  libXext-1.3.6-1.amzn2023.0.1.x86_64                   libXrender-0.9.11-6.amzn2023.0.1.x86_64
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64                 libjpeg-turbo-2.1.4-2.amzn2023.0.5.x86_64
  libpng-2:1.6.37-10.amzn2023.0.6.x86_64                libxcb-1.17.0-1.amzn2023.0.1.x86_64
  maven-1:3.8.4-3.amzn2023.0.5.noarch                   maven-amazon-corretto17-1:3.8.4-3.amzn2023.0.5.noarch
  maven-lib-1:3.8.4-3.amzn2023.0.5.noarch               maven-resolver-1:1.7.3-3.amzn2023.0.4.noarch
  maven-shared-utils-3.3.4-4.amzn2023.0.4.noarch        maven-wagon-3.4.2-6.amzn2023.0.4.noarch
  pixman-0.43.4-1.amzn2023.0.4.x86_64                   plexus-cipher-1.8-3.amzn2023.0.3.noarch
  plexus-classworlds-2.6.0-10.amzn2023.0.4.noarch       plexus-containers-component-annotations-2.1.0-9.amzn2023.0.4.noarch
  plexus-interpolation-1.26-10.amzn2023.0.4.noarch      plexus-sec-dispatcher-2.0-3.amzn2023.0.3.noarch
  plexus-utils-3.3.0-9.amzn2023.0.4.noarch              publicsuffix-list-20240212-61.amzn2023.noarch
  sisu-1:0.3.4-9.amzn2023.0.4.noarch                    slf4j-1.7.32-3.amzn2023.0.4.noarch
  xml-common-0.6.3-56.amzn2023.0.2.noarch

Complete!
[ec2-user@ip-172-31-23-113 ~]$
```

Check it works:

mvn -v

```
[ec2-user@ip-172-31-23-113 ~]$ mvn -v
Apache Maven 3.8.4 (Red Hat 3.8.4-3.amzn2023.0.5)
Maven home: /usr/share/maven
Java version: 17.0.17, vendor: Amazon.com Inc., runtime: /usr/lib/jvm/java-17-amazon-corretto.x86_64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.1.158-178.288.amzn2023.x86_64", arch: "amd64", family: "unix"
[ec2-user@ip-172-31-23-113 ~]$ ^C
[ec2-user@ip-172-31-23-113 ~]$
```

Part IV: Download the files and deploy the solution

In the previous part we created the EC2 AWS instance and installed in it all the software needed to deploy our application. Here, we are going to download the files of our GitHub repository, create the Docker image, apply the yaml deployment files and, finally, configure connectivity on AWS.

IV.1- Download the files from GitHub to the instance

In a terminal ec2-user session of our EC2 instance, clone our GitHub repo, where we pushed the Dockerfile and the two yaml deployment files in part II:

git clone https://github.com/enricrocaab/springbootcrud-k8s.git

If the GitHub repository is public, this command doesn't ask for the username and the password, this is my case now:

```
[ec2-user@ip-172-31-23-113 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-23-113 ~]$ ls
minikube-latest.x86_64.rpm
[ec2-user@ip-172-31-23-113 ~]$ git clone https://github.com/enricrocaab/springbootcrud-k8s.git
Cloning into 'springbootcrud-k8s'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 41 (delta 1), reused 41 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (41/41), 13.99 KiB | 13.99 MiB/s, done.
Resolving deltas: 100% (1/1), done.
[ec2-user@ip-172-31-23-113 ~]$
```

If I look in the folder:

```
[ec2-user@ip-172-31-23-113 ~]$ ls
minikube-latest.x86_64.rpm  springbootcrud-k8s
[ec2-user@ip-172-31-23-113 ~]$ ls -lrta
total 51772
-rw-r--r--.  1 ec2-user ec2-user      492 Jan 28  2023 .bashrc
-rw-r--r--.  1 ec2-user ec2-user      141 Jan 28  2023 .bash_profile
-rw-r--r--.  1 ec2-user ec2-user       18 Jan 28  2023 .bash_logout
drwxr-xr-x.  3 root     root           22 Nov 20 10:15 ..
drwx------.  2 ec2-user ec2-user       29 Nov 20 10:15 .ssh
drwxr-xr-x.  3 ec2-user ec2-user       33 Nov 21 10:46 .kube
drwxr-xr-x. 10 ec2-user ec2-user    16384 Nov 21 16:25 .minikube
-rw-------.  1 ec2-user docker       1439 Nov 21 20:17 .bash_history
-rw-r--r--.  1 ec2-user ec2-user 52979185 Nov 26 22:02 minikube-latest.x86_64.rpm
drwx------.  6 ec2-user ec2-user      185 Nov 26 22:03 .
drwxr-xr-x.  5 ec2-user ec2-user      187 Nov 26 22:03 springbootcrud-k8s
[ec2-user@ip-172-31-23-113 ~]$
```

The command **ls -lrta** is a common combination of options used with the ls command in Unix-like operating to list files and directories with detailed information, sorted by modification time, and including hidden files.

- **l** (Long format): Displays files in a detailed format, showing permissions, owner, group, size, and last modified date/time.
- **r** (Reverse order): Reverses the sorting order. Since the default sort is alphabetical, and the -t flag sorts by time (newest first), combining -r with -t will make the oldest files appear first.
- **t** (Time sort): Sorts the files by their last modification time, with the newest files appearing first (if -r is not used).
- **a** (All files): Includes entries that start with a dot (.), which are normally hidden files and directories (e.g., .bashrc, .minikube).

```
[ec2-user@ip-172-31-23-113 ~]$ cd sp*
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ pwd
/home/ec2-user/springbootcrud-k8s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ ls -lrta
total 48
drwx------. 6 ec2-user ec2-user   185 Nov 26 22:03 ..
drwxr-xr-x. 4 ec2-user ec2-user    30 Nov 26 22:03 src
-rw-r--r--. 1 ec2-user ec2-user  2042 Nov 26 22:03 pom.xml
-rw-r--r--. 1 ec2-user ec2-user  8481 Nov 26 22:03 mvnw.cmd
-rw-r--r--. 1 ec2-user ec2-user 11790 Nov 26 22:03 mvnw
-rw-r--r--. 1 ec2-user ec2-user  3219 Nov 26 22:03 db-deploy.yaml
-rw-r--r--. 1 ec2-user ec2-user  2259 Nov 26 22:03 app-deploy.yaml
-rw-r--r--. 1 ec2-user ec2-user   140 Nov 26 22:03 Dockerfile
drwxr-xr-x. 3 ec2-user ec2-user    21 Nov 26 22:03 .mvn
-rw-r--r--. 1 ec2-user ec2-user   442 Nov 26 22:03 .gitignore
-rw-r--r--. 1 ec2-user ec2-user    38 Nov 26 22:03 .gitattributes
drwxr-xr-x. 7 ec2-user ec2-user   147 Nov 26 22:03 .git
drwxr-xr-x. 5 ec2-user ec2-user   187 Nov 26 22:03 .
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

IV.2- Re-built the executable jar file in the pipeline server

As we explained before, in a CI/CD pipeline, the pipeline server runs the command, e.g., “mvn clean install”. The JAR file is re-built from the source code checked out from GitHub. So, to reproduce this process, we go to re-create the jar file in our EC2 instance.

We can execute the building from the project’s folder. By default, Maven expects the following folder structure relative to the project directory that contains the pom.xml file:

Folder Path	Purpose
src/main/java	**Primary Application Source Code** (Java classes)
src/main/resources	**Primary Application Resources** (Configuration files, property files, XML, etc.)
src/test/java	**Test Source Code** (Unit and integration tests)
src/test/resources	**Test Resources** (Test-specific configuration)
target	**Build Output** (Compiled classes, JAR/WAR files, etc.)

Notes:

- Maven automatically scans and uses all Java source code files found within the src/main/java directory and all its sub-folders. The use of sub-folders is not just allowed, it is the standard and mandatory convention for organizing your Java code based on its package structure.
- Every Maven project inherits configuration from a parent POM called the Super POM. If you want to change his settings, you can override it in your pon.xml file.

So, in our EC2 instance, go to the folder **/home/ec2-user/springbootcrud-k8s** and run the command:

mvn clean install

In detail:

- **clean**: This lifecycle phase removes the target directory, ensuring that you are starting with a clean build. This prevents issues from previous builds.
- **install**: It executes all preceding phases (like validate, compile, test, and package) and then copies the final artifact (our JAR file) to the local Maven repository for use as a dependency in other projects.

...

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/junit/platform/junit-platform-launch
er/6.0.1/junit-platform-launcher-6.0.1.jar (246 kB at 1.9 MB/s)
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.mycompany.crud.CrudApplicationTests
22:10:04.652 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -
- Could not detect default configuration classes for test class [com.mycompany.crud.CrudApplicationTes
ts]: CrudApplicationTests does not declare any static, non-private, non-final, nested classes annotate
d with @Configuration.
22:10:04.820 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Fo
und @SpringBootConfiguration com.mycompany.crud.CrudApplication for test class com.mycompany.crud.Crud
ApplicationTests

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v4.0.0)

2025-11-26T22:10:05.435Z  INFO 3414 --- [crud] [           main] c.mycompany.crud.CrudApplicationTests
    : Starting CrudApplicationTests using Java 17.0.17 with PID 3414 (started by ec2-user in /home/ec2
-user/springbootcrud-k8s)
2025-11-26T22:10:05.445Z  INFO 3414 --- [crud] [           main] c.mycompany.crud.CrudApplicationTests
    : The following 1 profile is active: "test"
```

...

```
[INFO] --- maven-install-plugin:3.1.4:install (default-install) @ crud ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolve
r-util/1.9.22/maven-resolver-util-1.9.22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver
-util/1.9.22/maven-resolver-util-1.9.22.pom (2.2 kB at 16 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolve
r/1.9.22/maven-resolver-1.9.22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver
/1.9.22/maven-resolver-1.9.22.pom (23 kB at 203 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolve
r-api/1.9.22/maven-resolver-api-1.9.22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver
-api/1.9.22/maven-resolver-api-1.9.22.pom (2.2 kB at 7.9 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolve
r-util/1.9.22/maven-resolver-util-1.9.22.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolve
r-api/1.9.22/maven-resolver-api-1.9.22.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver
-api/1.9.22/maven-resolver-api-1.9.22.jar (157 kB at 2.7 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver
-util/1.9.22/maven-resolver-util-1.9.22.jar (196 kB at 771 kB/s)
[INFO] Installing /home/ec2-user/springbootcrud-k8s/pom.xml to /home/ec2-user/.m2/repository/com/mycom
pany/crud/0.0.1-SNAPSHOT/crud-0.0.1-SNAPSHOT.pom
[INFO] Installing /home/ec2-user/springbootcrud-k8s/target/crud-0.0.1-SNAPSHOT.jar to /home/ec2-user/.
m2/repository/com/mycompany/crud/0.0.1-SNAPSHOT/crud-0.0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  48.930 s
[INFO] Finished at: 2025-11-26T22:10:17Z
[INFO] ------------------------------------------------------------------------
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Build has worked fine, and we can see that the target folder has been created:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ ls -lrta
total 96
drwxr-xr-x. 4 ec2-user ec2-user    30 Nov 26 22:03 src
-rw-r--r--. 1 ec2-user ec2-user  2042 Nov 26 22:03 pom.xml
-rw-r--r--. 1 ec2-user ec2-user  8481 Nov 26 22:03 mvnw.cmd
-rw-r--r--. 1 ec2-user ec2-user 11790 Nov 26 22:03 mvnw
-rw-r--r--. 1 ec2-user ec2-user  3219 Nov 26 22:03 db-deploy.yaml
-rw-r--r--. 1 ec2-user ec2-user  2259 Nov 26 22:03 app-deploy.yaml
-rw-r--r--. 1 ec2-user ec2-user   140 Nov 26 22:03 Dockerfile
drwxr-xr-x. 3 ec2-user ec2-user    21 Nov 26 22:03 .mvn
-rw-r--r--. 1 ec2-user ec2-user   442 Nov 26 22:03 .gitignore
-rw-r--r--. 1 ec2-user ec2-user    38 Nov 26 22:03 .gitattributes
drwxr-xr-x. 7 ec2-user ec2-user   147 Nov 26 22:03 .git
drwx------. 7 ec2-user ec2-user 16384 Nov 26 22:09 ..
drwxr-xr-x. 6 ec2-user ec2-user 16384 Nov 26 22:10 .
drwxr-xr-x. 9 ec2-user ec2-user 16384 Nov 26 22:10 target
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

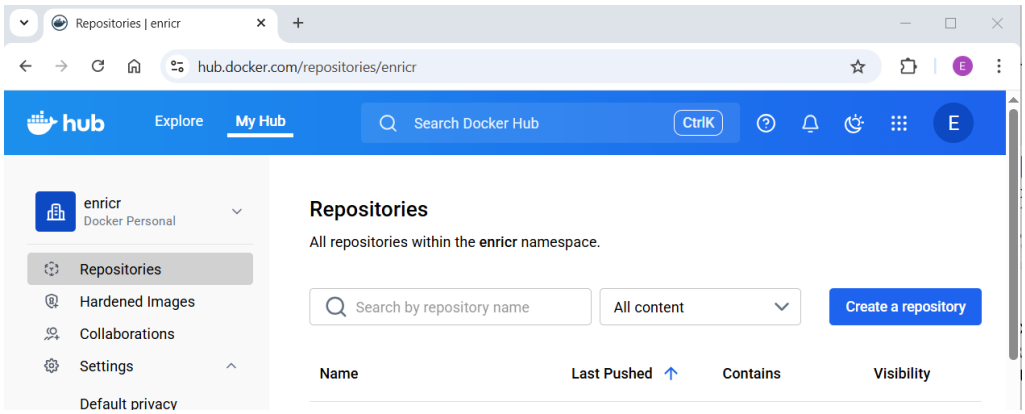
And inside of it we can find the executable jar file **crud-0.0.1-SNAPSHOT.jar**:

```
[ec2-user@ip-172-31-23-113 target]$ ls -lrta
total 55472
drwxr-xr-x. 3 ec2-user ec2-user       35 Nov 26 22:09 maven-status
drwxr-xr-x. 3 ec2-user ec2-user       25 Nov 26 22:09 generated-sources
drwxr-xr-x. 4 ec2-user ec2-user       64 Nov 26 22:10 classes
drwxr-xr-x. 3 ec2-user ec2-user       30 Nov 26 22:10 generated-test-sources
drwxr-xr-x. 3 ec2-user ec2-user       52 Nov 26 22:10 test-classes
drwxr-xr-x. 6 ec2-user ec2-user    16384 Nov 26 22:10 ..
drwxr-xr-x. 2 ec2-user ec2-user      113 Nov 26 22:10 surefire-reports
drwxr-xr-x. 2 ec2-user ec2-user       28 Nov 26 22:10 maven-archiver
-rw-r--r--. 1 ec2-user ec2-user     7907 Nov 26 22:10 crud-0.0.1-SNAPSHOT.jar.original
drwxr-xr-x. 9 ec2-user ec2-user    16384 Nov 26 22:10 .
-rw-r--r--. 1 ec2-user ec2-user 56759922 Nov 26 22:10 crud-0.0.1-SNAPSHOT.jar
[ec2-user@ip-172-31-23-113 target]$
```

IV.3- Create the Docker image and upload it to Docker Hub

Now we are going to use a standard repository to store our Docker image, so the image will be available when we need to create containers from this image. This repository is named **Docker Hub**, can access it from https://hub.docker.com/. Create an account if you don't have and sign in and take a little time to see how it works.

My Docker Hub username is enricr, you should have your own account:



Remember that in the Dockerfile we have the code to copy **target/crud-0.0.1-SNAPSHOT.jar** from the machine where the Docker build is executed to the new Docker Image:

```
FROM openjdk:8
EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","/crud-0.0.1-SNAPSHOT.jar"]
```

In the terminal of the EC2 instance, and located in the /home/ec2-user/springbootcrud-k8s folder, execute **docker images** to see the images that we have in the EC2 instance:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker images
REPOSITORY                     TAG       IMAGE ID       CREATED         SIZE
gcr.io/k8s-minikube/kicbase    v0.0.48   c6b5532e987b   2 months ago    1.31GB
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now create the image with (instead of enricr, use your own Docker Hub account):

docker build -t enricr/springbootcrud-k8s:1.0 .

In detail:

- **docker build**: It instructs the Docker daemon to build a new Docker image.
- **-t** (or **--tag**): This flag is used to name and optionally tag the resulting image in the format [NAME]:[TAG].
  - **enricr/springbootcrud-k8s**: This is the image name (or repository name). It follows the common pattern of [DOCKER_HUB_USERNAME]/[REPOSITORY_NAME]. This suggests the image is intended to be hosted on Docker Hub under the enricr account.
  - **:1.0**: This is the tag applied to the image. Tags are used to denote specific versions, changes, or variants of an image (e.g., 1.0, latest, dev).
- **.** (Dot): This is the build context path. It tells Docker where to look for the Dockerfile and any supporting files required for the build.

I got an error because it failed to solve openjdk:8. The best practice for running a Java 8 application on an Amazon Linux 2023 (AL2023) instance is to use Amazon Corretto 8 on the official Amazon Linux 2023 base container image. So, I changed the Dockerfile editing it with **vim Dockerfile**, now it is:

```
# 1. Use the official, minimal AL2023 base image
FROM public.ecr.aws/amazonlinux/amazonlinux:2023

# 2. Install Amazon Corretto 8 (the AL2023-compatible Java 8 JDK)
# We use 'dnf install -y' because AL2023 uses DNF as its package manager.
RUN dnf install -y java-1.8.0-amazon-corretto-devel \
    && dnf clean all

# 3. Set the JAVA_HOME environment variable for the application
ENV JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto-x86_64

# 4. Continue with your application steps

EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","/crud-0.0.1-SNAPSHOT.jar"]
~
```

```
# 1. Use the official, minimal AL2023 base image
FROM public.ecr.aws/amazonlinux/amazonlinux:2023

# 2. Install Amazon Corretto 8 (the AL2023-compatible Java 8 JDK)
# We use 'dnf install -y' because AL2023 uses DNF as its package manager.
RUN dnf install -y java-1.8.0-amazon-corretto-devel \
  && dnf clean all

# 3. Set the JAVA_HOME environment variable for the application
ENV JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto-x86_64

# 4. Continue with your application steps
EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Save the Dockerfile. (In my GitHub repo you can find this file as Dockerfile2.)

Executing the building again, now it works:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker build -t enricr/springbootcrud-k8s:1.0 .
[+] Building 57.3s (8/8) FINISHED                                                      docker:default
 => [internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 707B                                                             0.0s
 => [internal] load metadata for public.ecr.aws/amazonlinux/amazonlinux:2023                     0.3s
 => [internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                  0.0s
 => [1/3] FROM public.ecr.aws/amazonlinux/amazonlinux:2023@sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d  4.3s
 => => resolve public.ecr.aws/amazonlinux/amazonlinux:2023@sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d  0.0s
 => => sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d 770B / 770B       0.0s
 => => sha256:fb74284364222bd16641b0f0d36df624bf1d81758c1ad50a8b5764ab1ef9ebb0 528B / 528B       0.0s
 => => sha256:ac90a73983be7974c18062a0a56d2c4ad6b9369b4eb2b4090141462a2ef428b9 662B / 662B       0.0s
 => => sha256:1c7de4eb5ced9ea3f72366a34ec955a53e9b0f4ac53d332a155de21eb808d732 53.97MB / 53.97MB 0.6s
 => => extracting sha256:1c7de4eb5ced9ea3f72366a34ec955a53e9b0f4ac53d332a155de21eb808d732        3.5s
 => [internal] load build context                                                                0.8s
 => => transferring context: 56.77MB                                                             0.8s
 => [2/3] RUN dnf install -y java-1.8.0-amazon-corretto-devel     && dnf clean all              46.8s
 => [3/3] ADD target/crud-0.0.1-SNAPSHOT.jar crud-0.0.1-SNAPSHOT.jar                             0.2s
 => exporting to image                                                                           5.6s
 => => exporting layers                                                                          5.6s
 => => writing image sha256:472a95a3b48504d7277da9fe4b2f28f348fbc5212189511da33dd3905dfaa702     0.0s
 => => naming to docker.io/enricr/springbootcrud-k8s:1.0                                         0.0s
```

If we check the images in the EC2 instance with **docker images**, we can see it:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
enricr/springbootcrud-k8s     1.0       472a95a3b485   53 seconds ago   1.06GB
gcr.io/k8s-minikube/kicbase   v0.0.48   c6b5532e987b   2 months ago     1.31GB
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now log in to your Docker Hub account with **docker login** and introduce your username and password:

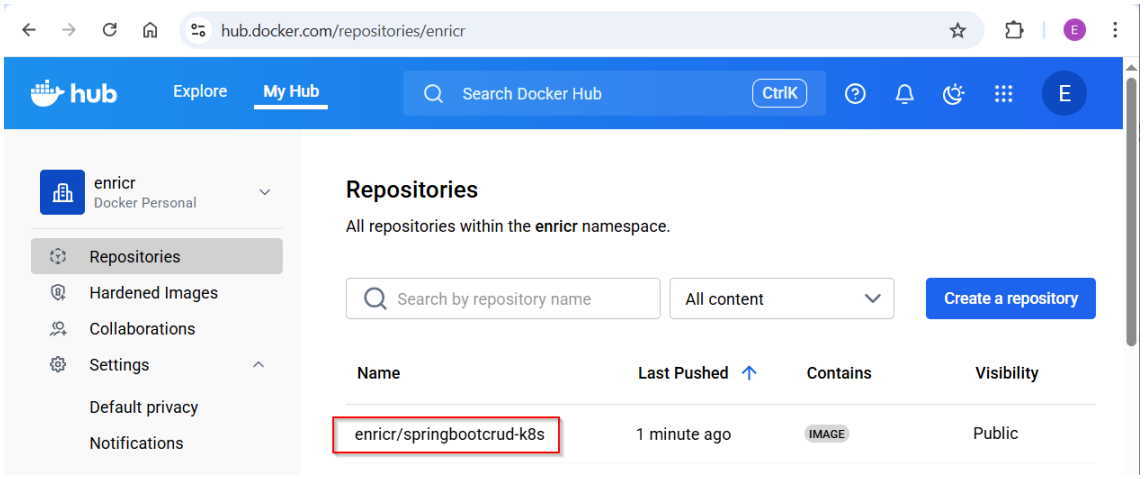
```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docke
r.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizat
ions using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: enricr
Password:
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The next is to upload the recently created image to Docker Hub with the command (instead of enricr, use your own Docker Hub account):

docker image push enricr/springbootcrud-k8s:1.0

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker image push enricr/springbootcrud-k8s:1.0
The push refers to repository [docker.io/enricr/springbootcrud-k8s]
a75fe2a70652: Pushed
87834d6d5c78: Pushed
0e7c6314d998: Pushed
1.0: digest: sha256:b2da72eb95e40b99897b16c9fc85b65f218fa49d0b7f326bfaf0cf98ab3eb719 size: 954
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

And we can see it in the Docker Hub web interface:



IV.4- Apply the db-deploy.yaml file

Now it is already time to deploy our solution. We go to apply the db-deploy.yaml file to create a pod with the MySQL container and the database that must be inside it.

The initial db-deploy.yaml and Dockerfile files contain some problems that needed to be resolved. The process of troubleshooting and implementing the workarounds to fix these issues is interesting, so I've included them in this documentation. But, if you simply need a working solution, please use the corrected files: db-deploy4.yaml and Dockerfile3.

Before, be sure that Minikube is running. Remember that to start it we must execute: **minikube start** . To check, get the pods of all namespaces with **kubectl get pod -A** .

In a terminal of our EC2 instance, located in the /**home/ec2-user/springbootcrud-k8s** folder, execute the command:

kubectl apply -f db-deploy.yaml

In detail:

- **kubectl**: Thi is command-line tool for interacting with the Kubernetes API server.
- **apply**: It instructs Kubernetes to create the resource(s) if they don't exist, or to update the resource(s) if they already exist, based on the configuration file. This is the preferred declarative method for managing resources.
- **-f:** This flag tells kubectl to apply that the configuration for the desired state is provided in the file that follows.
- **db-deploy.yaml**: This is the YAML file that contains the definition (manifest) of the Kubernetes resources to be created or updated. In this case, you know that it defines a **ConfigMap** (database name), a **Secret** (MySQL username and password), a **PersistentVolumeClaim**, a **Deployment** with the definition of a pod with one MySQL container and its database, and finally a **Service** to expose the MySQL service.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ ls -lrta
total 96
drwxr-xr-x. 4 ec2-user ec2-user    30 Nov 26 22:03 src
-rw-r--r--. 1 ec2-user ec2-user  2042 Nov 26 22:03 pom.xml
-rw-r--r--. 1 ec2-user ec2-user  8481 Nov 26 22:03 mvnw.cmd
-rw-r--r--. 1 ec2-user ec2-user 11790 Nov 26 22:03 mvnw
-rw-r--r--. 1 ec2-user ec2-user  3219 Nov 26 22:03 db-deploy.yaml
-rw-r--r--. 1 ec2-user ec2-user  2259 Nov 26 22:03 app-deploy.yaml
drwxr-xr-x. 3 ec2-user ec2-user    21 Nov 26 22:03 .mvn
-rw-r--r--. 1 ec2-user ec2-user   442 Nov 26 22:03 .gitignore
-rw-r--r--. 1 ec2-user ec2-user    38 Nov 26 22:03 .gitattributes
drwxr-xr-x. 7 ec2-user ec2-user   147 Nov 26 22:03 .git
drwxr-xr-x. 9 ec2-user ec2-user 16384 Nov 26 22:10 target
-rw-r--r--. 1 ec2-user ec2-user   609 Nov 26 23:25 Dockerfile
drwx------. 8 ec2-user ec2-user 16384 Nov 26 23:25 ..
drwxr-xr-x. 6 ec2-user ec2-user 16384 Nov 26 23:25 .
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f db-deploy.yaml
configmap/mysql-config created
secret/mysql-secret created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/mysql created
Warning: spec.SessionAffinity is ignored for headless services
service/mysql created
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

We can see that all the resources have been created. The warning is generated because Session Affinity only makes sense for Services that perform load balancing through a single ClusterIP. Since our service is Headless (**clusterIP: None**), it delegates the connection and routing logic to the client (which gets the list of Pod IPs directly from DNS). The Kubernetes Service proxy (kube-proxy) is not actively load-balancing the connections, so it cannot enforce Session Affinity. So, the setting is ignored. No modification is required if we didn't explicitly set sessionAffinity in our YAML. The warning is simply informing us that the default (or any potential) session affinity logic is disabled by the Headless configuration. Since our manifest is clean, we can safely ignore this warning, as it does not prevent our resources from being created or functioning as a Headless Service.

Now we can check the running pods with **kubectl get pods -A** :

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pods -A
NAMESPACE              NAME                                         READY   STATUS             RESTARTS        AGE
default                mysql-988f59d9c-j7t4x                        0/1     CrashLoopBackOff   8 (4m54s ago)   21m
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running            2 (5d18h ago)   6d1h
kube-system            etcd-minikube                                1/1     Running            2 (5d18h ago)   6d1h
kube-system            kube-apiserver-minikube                      1/1     Running            2 (24m ago)     6d1h
kube-system            kube-controller-manager-minikube             1/1     Running            2 (5d18h ago)   6d1h
kube-system            kube-proxy-f9r78                             1/1     Running            2 (5d18h ago)   6d1h
kube-system            kube-scheduler-minikube                      1/1     Running            2 (5d18h ago)   6d1h
kube-system            storage-provisioner                          1/1     Running            5 (23m ago)     6d1h
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running            2 (5d18h ago)   5d22h
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running            3 (23m ago)     5d22h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

We can see our pod created with the name **mysql-988f59d9c-j7t4x**, but his container is not running (READY = 0/1), and it shows the status **CrashLoopBackOff**.

The format of the READY column is always: **Ready Containers** (Number of containers within the Pod that have passed their readiness probe) / **Total Required Containers** (Total number of containers defined in the Pod's specification that must be running and ready)**.**

The CrashLoopBackOff status means that a container inside our Pod is starting, failing, and restarting repeatedly. It is not the error itself, but an indicator that our application is failing to start successfully.

To try to solve, we can inspect the Pod Description and Events:

kubectl describe pod mysql-988f59d9c-j7t4x

...

```
Events:
  Type     Reason            Age                    From               Message
  ----     ------            ----                   ----               -------
  Warning  FailedScheduling  45m (x4 over 45m)      default-scheduler  0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. not found
  Normal   Scheduled         45m                    default-scheduler  Successfully assigned default/mysql-988f59d9c-j7t4x to minikube
  Normal   Pulling           45m                    kubelet            Pulling image "mysql:5.7"
  Normal   Pulled            45m                    kubelet            Successfully pulled image "mysql:5.7" in 14.782s (14.782s including waiting). Image size: 501
392011 bytes.
  Normal   Created           28m (x9 over 44m)      kubelet            Created container: mysql
  Normal   Started           28m (x9 over 44m)      kubelet            Started container mysql
  Warning  BackOff           4m49s (x185 over 44m)  kubelet            Back-off restarting failed container mysql in pod mysql-988f59d9c-j7t4x_default(60c3aaa5-d31d
-4ae2-8caa-534f654349ca)
  Normal   Pulled            3m31s (x13 over 44m)   kubelet            Container image "mysql:5.7" already present on machine
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

We can see the message: "0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. not found". It seems that the reason for the problem could be that the Pod's manifest specifies a Volume that points to a PVC (claimName: mysql-pv-claim) which is currently in the Pending state. The scheduler cannot place the Pod until the required storage is ready.

To check this, we use **kubectl get pvc -A** and **kubectl get pv -A** ,but we can see that the PersistenVolumeClan and the PersistentVolum are created and bound:

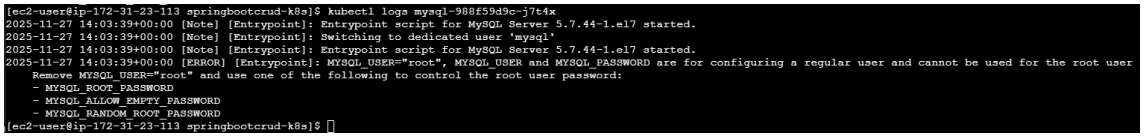
```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
NAMESPACE   NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
default     mysql-pv-claim   Bound    pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b   1Gi        RWO            standard       <unset>                 72m
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b   1Gi        RWO            Delete           Bound    default/mysql-pv-claim   standard       <unset>                          72m
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

So, the pvc and pv are working and the error of pvc not found is a consequence of another problem that we must investigate more in detail.

To check the logs of the pod:

kubectl logs mysql-988f59d9c-j7t4x

This is the problem:



[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl logs mysql-988f59d9c-j7t4x
2025-11-27 14:03:39+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.44-1.el7 started.
2025-11-27 14:03:39+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2025-11-27 14:03:39+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.44-1.el7 started.
2025-11-27 14:03:39+00:00 [ERROR] [Entrypoint]: MYSQL_USER="root", MYSQL_USER and MYSQL_PASSWORD are for configuring a regular user and cannot be used for the root user
Remove MYSQL_USER="root" and use one of the following to control the root user password:
- MYSQL_ROOT_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD
- MYSQL_RANDOM_ROOT_PASSWORD
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$

The cause of the problem is that in **mysql:5.7** Docker image's entrypoint script follows specific rules for setting up user accounts during initialization. MYSQL_USER and MYSQL_PASSWORD (or variables derived from them, like those in our Secret) are designed to create a new, non-root user account inside the database. MYSQL_ROOT_PASSWORD is used specifically to set the password for the default root user.

Our configuration is trying to use the MYSQL_USER variable (which is intended for a regular user) and setting its value to "root". The entrypoint script detects this conflict: we are asking it to create a regular user named root, while simultaneously trying to manage the default root user with MYSQL_ROOT_PASSWORD. The script rejects this ambiguous configuration and crashes.

To solve the issue, we must remove the MYSQL_USER variable in the db-deploy.yaml file. For that, edit the file doing **vim db-deploy.yaml** and write "#" at the beginning of each line that must not be executed (In my GitHub repo you can find the corrected file as db-deploy2.yaml.):

```
  spec:
    containers:
      - image: mysql:5.7      # image from docker-hub
        args:
          - "--ignore-db-dir=lost+found" # Workaround for https://github.com/docker-library/mysql/issues/186
        name: mysql
        env:
          # Get database name from the ConfigMap
          - name: MYSQL_DATABASE
            valueFrom:
              configMapKeyRef:
                name: mysql-config
                key: MYSQL_DATABASE
          # Get username from the Secret (Sensitive)
          #- name: MYSQL_USER
          #  valueFrom:
          #    secretKeyRef:
          #      name: mysql-secret
          #      key: MYSQL_USER
          # Get root password from the Secret (Sensitive)
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_ROOT_PASSWORD
```

Save and exit (In vim do ":" and "x").

Before to try the deployment again we must delete it:

kubectl delete deployment mysql

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deployment -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       0/1     1            0           4h37m
kube-system            coredns                     1/1     1            1           6d5h
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           6d2h
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           6d2h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete deployment mysql
deployment.apps "mysql" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deployment -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
kube-system            coredns                     1/1     1            1           6d5h
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           6d2h
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           6d2h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

delete the PersistentVolumeClaim:

kubectl delete pvc mysql-pv-claim

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
NAMESPACE   NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
default     mysql-pv-claim   Bound    pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b   1Gi        RWO            standard       <unset>                 4h39m
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete pvc mysql-pv-claim
persistentvolumeclaim "mysql-pv-claim" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
No resources found
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and delete the PersistentVolume:

kubectl delete pv pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM                   STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b   1Gi        RWO            Delete           Released   default/mysql-pv-claim   standard       <unset>                          4h40m
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete pv pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b
persistentvolume "pvc-c7250e84-0475-47e2-b94a-6ea2632c6f4b" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
No resources found
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now, apply for the deployment again:

kubectl apply -f db-deploy.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f db-deploy.yaml
configmap/mysql-config unchanged
secret/mysql-secret unchanged
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/mysql created
service/mysql unchanged
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and if we check the pods:

kubectl get pod -A

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS        AGE
default                mysql-79c9d95747-bvr8d                       1/1     Running   0               46s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running   3 (142m ago)    6d5h
kube-system            etcd-minikube                                1/1     Running   3 (142m ago)    6d5h
kube-system            kube-apiserver-minikube                      1/1     Running   3 (142m ago)    6d5h
kube-system            kube-controller-manager-minikube             1/1     Running   3 (142m ago)    6d5h
kube-system            kube-proxy-f9r78                             1/1     Running   3 (142m ago)    6d5h
kube-system            kube-scheduler-minikube                      1/1     Running   3 (142m ago)    6d5h
kube-system            storage-provisioner                          1/1     Running   7 (138m ago)    6d5h
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running   3 (142m ago)    6d3h
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running   5 (138m ago)    6d3h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now it has worked!

We also have the service created and it is listening on port 3306/tcp :

kubectl get svc

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP    6d5h
mysql        ClusterIP   None         <none>        3306/TCP   4h47m
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

To check that the MySQL database has been created and works, we must execute commands inside the container:

kubectl exec -it mysql-79c9d95747-bvr8d -- /bin/bash

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl exec -it mysql-79c9d95747-bvr8d -- /bin/bash
bash-4.2#
```

mysql -u root -p

Type the password = mysqlpass

```
bash-4.2# mysql -u root -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
```

But we get the message error: "ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)". So, it seems that the password stored in the Kubernetes Secret does not decode to "mysqlpass". To check, type "**exit**" to get out the container, and execute:

kubectl get secret mysql-secret -o yaml

MYSQL_ROOT_PASSWORD: bXlzcWxwb3Nz

Now, use a Base64 decoder (like the one available in the Linux shell) to see the password the MySQL container actually received:

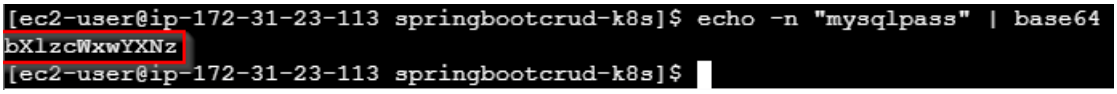
echo "bXlzcWxwb3Nz" | base64 --decode

we get: mysqlposs

So, there is a writing error, it should be “mysqlpass” and not “mysqlposs”.

To generate the correct Base64 string:

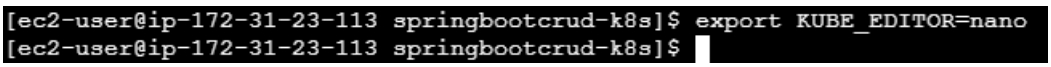
echo -n "mysqlpass" | base64



bXlzcWxwYXNz

To correct the password, we could edit the secret with “**kubectl edit secret mysql-secret**”, but it is very difficult or almost impossible to manage the vi editor in this environment. I suggest changing de kubectl editor to **nano** with:

export KUBE_EDITOR=nano



and now:

kubectl edit secret mysql-secret

and change the encoded password. Once done:

```
odifiede edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  MYSQL_ROOT_PASSWORD: bXlzcWxwYXNz
  MYSQL_USER: cm9vdA==
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"MYSQL_ROOT_PASSWORD":"bXlzcWxwb3Nz","MYSQL_USER":"cm9vdA=="},"kind":"Secret","metada
tacreationTimestamp: "2025-11-27T11:31:07Z"l","tier":"database"},"name":"mysql-
  labels:
    app: mysql
    tier: database
  name: mysql-secret
  namespace: default
  resourceVersion: "19754"
  uid: f604ccf9-0a34-47e9-b3eb-8a721581e67d
type: Opaque
```

Save (Control key and the letter 'o', Enter). It shows the temporary file where it is saved:

```
^G Help          ^O Write Out     ^F Where Is      ^K Cut
-A Set Mark      M-] To Bracket   M-B Previous
^X Exit          ^R Read File     ^\ Replace       ^U Paste

B Where Was                                  M-D DOS Format
File Name to Write: /tmp/kubectl-edit-986699683.yaml
```

and exit (Control key and the letter 'x').

The next step is to delete the old pod. The MySQL container only uses the environment variable MYSQL_ROOT_PASSWORD during its initial startup and database initialization. If you change the Secret while the Pod is running, the running MySQL process does not see the change.

kubectl delete pod mysql-79c9d95747-bvr8d

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete pod mysql-79c9d95747-bvr8d
pod "mysql-79c9d95747-bvr8d" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The Deployment will **automatically** create a new Pod with a new name. Wait for the new Pod to reach Running with 1/1 status.

kubectl get pod -A

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS        AGE
default                mysql-79c9d95747-kb6g8                       1/1     Running   0               51s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running   4 (15h ago)     6d20h
kube-system            etcd-minikube                                1/1     Running   4 (15h ago)     6d20h
kube-system            kube-apiserver-minikube                      1/1     Running   4 (15h ago)     6d20h
kube-system            kube-controller-manager-minikube             1/1     Running   4 (15h ago)     6d20h
kube-system            kube-proxy-f9r78                             1/1     Running   4 (15h ago)     6d20h
kube-system            kube-scheduler-minikube                      1/1     Running   4 (15h ago)     6d20h
kube-system            storage-provisioner                          1/1     Running   9 (98m ago)     6d20h
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running   4 (15h ago)     6d18h
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running   7 (98m ago)     6d18h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Once the new Pod is running (in my case: mysql-79c9d95747-kb6g8), access its shell again and try logging in with the correct password:

kubectl exec -it mysql-79c9d95747-kb6g8 -- /bin/bash

mysql -u root -p

Enter password: mysqlpass

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl exec -it mysql-79c9d95747-kb6g8 -- /bin/bash
bash-4.2# mysql -u root -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
bash-4.2#
```

We already have the problem. When the container starts and sees an existing data directory (from our persistent volume claim), it skips initialization. If the old, corrupt, or originally wrong password was set during the very first failed initialization, the server will continue to use that old password, ignoring the new Secret value. So, we are going to delete the deployment, the pv and the pvc and create the deployment again:

kubectl delete deployment mysql

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       1/1     1            1           16h
kube-system            coredns                     1/1     1            1           6d21h
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           6d19h
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           6d19h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete deployment mysql
deployment.apps "mysql" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

kubectl delete pvc mysql-pv-claim

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
NAMESPACE   NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   V
OLUMEATTRIBUTESCLASS   AGE
default     mysql-pv-claim   Bound    pvc-6c9da02c-d9ac-4bee-92b4-002ca7aa8582   1Gi        RWO            standard       <
unset>                 16h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete pvc mysql-pv-claim
persistentvolumeclaim "mysql-pv-claim" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
No resources found
```

delete the pv:

kubectl delete pv pvc-6c9da02c-d9ac-4bee-92b4-002ca7aa8582

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM                   S
TORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-6c9da02c-d9ac-4bee-92b4-002ca7aa8582   1Gi        RWO            Delete           Released   default/mysql-pv-claim   s
tandard       <unset>                          16h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete pv pvc-6c9da02c-d9ac-4bee-92b4-002ca7aa8582
persistentvolume "pvc-6c9da02c-d9ac-4bee-92b4-002ca7aa8582" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
No resources found
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and delete the secret:

kubectl delete secret mysql-secret

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get secret -A
NAMESPACE              NAME                              TYPE     DATA   AGE
default                mysql-secret                      Opaque   2      21h
kubernetes-dashboard   kubernetes-dashboard-certs        Opaque   0      6d19h
kubernetes-dashboard   kubernetes-dashboard-csrf         Opaque   1      6d19h
kubernetes-dashboard   kubernetes-dashboard-key-holder   Opaque   2      6d19h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete secret mysql-secret
secret "mysql-secret" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get secret -A
NAMESPACE              NAME                              TYPE     DATA   AGE
kubernetes-dashboard   kubernetes-dashboard-certs        Opaque   0      6d19h
kubernetes-dashboard   kubernetes-dashboard-csrf         Opaque   1      6d19h
kubernetes-dashboard   kubernetes-dashboard-key-holder   Opaque   2      6d19h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Modify the db-deploy.yaml file changing the encoded password to “bXlzcWxwYXNz” (In my GitHub repo you can find the corrected file as db-deploy3.yaml):

nano db-deploy.yaml

```
  GNU nano 8.3                                  db-deploy.yaml                                  Modified
# Define the ConfigMap for the non-sensitive username
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
  labels:
    app: mysql
    tier: database
data:
  MYSQL_DATABASE: springCrud            # The name of the database
---
# Define the Secret
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  labels:
    app: mysql
    tier: database
type: Opaque
data:
  MYSQL_USER: cm9vdA==                  # Base64 encoded 'root'
  MYSQL_ROOT_PASSWORD: bXlzcWxwYXNz     # Base64 encoded 'mysqlpass'
---
# Define a 'Persistent Volume Claim'(PVC) for Mysql Storage, dynamically provisioned by cluster
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim  # name of PVC essential for identifying the storage data
  labels:
    app: mysql

^G Help        ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Once corrected, save and exit. You can confirm the change with **cat db-deploy.yaml** .

Create the deployment again:

kubectl apply -f db-deploy.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f db-deploy.yaml
configmap/mysql-config unchanged
secret/mysql-secret created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/mysql created
service/mysql unchanged
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Check:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS       AGE
default                mysql-79c9d95747-7l7rv                       1/1     Running   0              38s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running   4 (16h ago)    6d21h
kube-system            etcd-minikube                                1/1     Running   4 (16h ago)    6d21h
kube-system            kube-apiserver-minikube                      1/1     Running   4 (16h ago)    6d21h
kube-system            kube-controller-manager-minikube             1/1     Running   4 (16h ago)    6d21h
kube-system            kube-proxy-f9r78                             1/1     Running   4 (16h ago)    6d21h
kube-system            kube-scheduler-minikube                      1/1     Running   4 (16h ago)    6d21h
kube-system            storage-provisioner                          1/1     Running   9 (159m ago)   6d21h
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running   4 (16h ago)    6d19h
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running   7 (159m ago)   6d19h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pvc -A
NAMESPACE   NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   V
OLUMEATTRIBUTESCLASS   AGE
default     mysql-pv-claim   Bound    pvc-c09f1005-8918-477d-bfe0-84c4d9cca099   1Gi        RWO            standard       <
unset>                 56s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STO
RAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-c09f1005-8918-477d-bfe0-84c4d9cca099   1Gi        RWO            Delete           Bound    default/mysql-pv-claim   sta
ndard       <unset>                          62s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The mysql pod is now: mysql-79c9d95747-7l7rv . Try again the mysql log in:

kubectl exec -it mysql-79c9d95747-7l7rv -- /bin/bash

mysql -u root -p

Enter password: mysqlpass

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl exec -it mysql-79c9d95747-7l7rv -- /bin/bash
bash-4.2# mysql -u root -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
```

If I check the environment variable in the mysql container:

echo $MYSQL_ROOT_PASSWORD

```
bash-4.2# echo $MYSQL_ROOT_PASSWORD
mysqlpass
bash-4.2#
```

It is correct.

Finally, I solved the problem putting double quotation marks (“) at the beginning and the end of the password in the secret in the db-deploy.yaml file (I store in my GitHub repository like db-deploy4.yaml). The reason for testing this is that it avoids potential problems with special characters such as line breaks. Then:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ cat db-deploy.yaml
# Define the ConfigMap for the non-sensitive username
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
  labels:
    app: mysql
    tier: database
data:
  MYSQL_DATABASE: springCrud            # The name of the database
---
# Define the Secret
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  labels:
    app: mysql
    tier: database
type: Opaque
data:
  MYSQL_USER: cm9vdA==                  # Base64 encoded 'root'
  MYSQL_ROOT_PASSWORD: "bXlzcWxwYXNz"   # Base64 encoded 'mysqlpass'
---
# Define a 'Persistent Volume Claim'(PVC) for Mysql Storage, dynamically provisioned by cluster
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim  # name of PVC essential for identifying the storage data
  labels:
    app: mysql
    tier: database
spec:
  accessModes:
    - ReadWriteOnce     #This specifies the mode of the claim that we are trying to create.
  resources:
    requests:
      storage: 1Gi              #This will tell kubernetes about the amount of space we are trying to claim.
```

kubectl exec -it mysql-79c9d95747-khqxv -- /bin/bash

And luckily it worked:

mysql -u root -p

Enter password: **mysqlpass**

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl exec -it mysql-79c9d95747-khqxv -- /bin/bash
bash-4.2#
bash-4.2# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Check that the database **springCrud** has been created:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| springCrud         |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

And logically the database is empty, without tables as we don't have initiated the application:

USE springCrud;

```
mysql> USE springCrud;
Database changed
mysql>
```

SHOW TABLES;

```
mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql>
```

IV.5- Apply the app-deploy.yaml file

Once we have our MySQL database running and with the **springCrud** database created and empty, we are going to deploy our application applying the app-deploy.yaml file.

Before proceeding, with the experience that we had with db-deploy.yaml file, we are going to correct the encoded string of DB_PASSWORD and put in between double quotes. To make it clearer, we are going to create the app-deploy2.yaml file with these changes. For that, in the terminal of our EC2 instance, copy the app-deploy.yaml and name it app-deploy2.yaml :

cp app-deploy.yaml app-deploy2.yaml

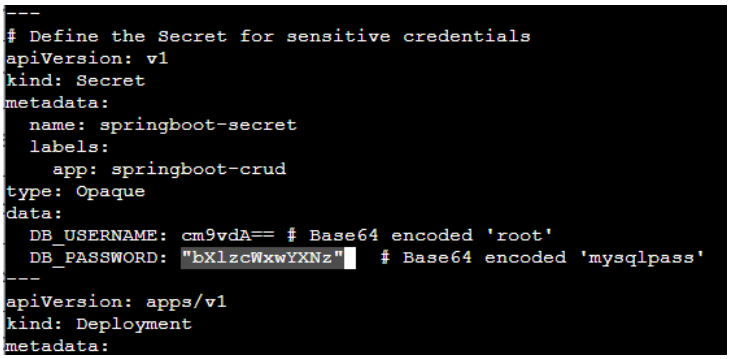
```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ pwd
/home/ec2-user/springbootcrud-k8s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ cp app-deploy.yaml app-deploy2.yaml
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now edit the app-deploy2.yaml file with nano, vim or vi and do the change. If you use nano:

nano app-deploy2.yaml

When corrected, save (Control key and the letter 'o', Enter) and exit (Control key and the letter 'x').

Once done:



The code of **app-deploy2.yaml** is now:

```
# Define the ConfigMap for non-sensitive configuration
apiVersion: v1
kind: ConfigMap
metadata:
 name: springboot-config
 labels:
  app: springboot-crud
data:
 DB_HOST: mysql                          # Hostname of the MySQL Service
 DB_NAME: springCrud                     # Name of the database
---
# Define the Secret for sensitive credentials
apiVersion: v1
kind: Secret
metadata:
 name: springboot-secret
 labels:
  app: springboot-crud
type: Opaque
data:
 DB_USERNAME: cm9vdA==                   # Base64 encoded 'root'
 DB_PASSWORD: "bXlzcWxwYXNz" # Base64 encoded 'mysqlpass'
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: springboot-crud-deploy
spec:
 selector:
  matchLabels:
   app: springboot-crud
 replicas: 3
 template:
  metadata:
   labels:
    app: springboot-crud
  spec:
   containers:
    - name: springboot-crud-container
      image: enricr/springbootcrud-k8s:1.0
```

```
      ports:
        - containerPort: 8080
      env:              # Setting Environmental Variables using ConfigMap and Secret
        # Get DB_HOST from ConfigMap
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: springboot-config
              key: DB_HOST
        # Get DB_NAME from ConfigMap
        - name: DB_NAME
          valueFrom:
            configMapKeyRef:
              name: springboot-config
              key: DB_NAME
        # Get DB_USERNAME from Secret
        - name: DB_USERNAME
          valueFrom:
            secretKeyRef:
              name: springboot-secret
              key: DB_USERNAME
        # Get DB_PASSWORD from Secret
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: springboot-secret
              key: DB_PASSWORD
---
apiVersion: v1                  # Kubernetes API version
kind: Service                   # Kubernetes resource kind we are creating
metadata:                       # Metadata of the resource kind we are creating
 name: springboot-crud-svc
spec:
 selector:
   app: springboot-crud
 ports:
   - protocol: "TCP"
     port: 8080                 # Port that the service is running on in the cluster
     targetPort: 8080           # The port exposed by the service
 type: NodePort                 # type of the service.
```

Now, to deploy our application execute:

kubectl apply -f app-deploy2.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f app-deploy2.yaml
configmap/springboot-config created
secret/springboot-secret created
deployment.apps/springboot-crud-deploy created
service/springboot-crud-svc created
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Check the pods:

kubectl get pod -A

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS             RESTARTS        AGE
default                mysql-79c9d95747-khqxv                       1/1     Running            1 (2d5h ago)    2d7h
default                springboot-crud-deploy-7d8c6b544c-6bwbn      0/1     ImagePullBackOff   0               13s
default                springboot-crud-deploy-7d8c6b544c-hrgqd      0/1     ErrImagePull       0               13s
default                springboot-crud-deploy-7d8c6b544c-shbl6      0/1     ImagePullBackOff   0               13s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running            5 (2d5h ago)    9d
kube-system            etcd-minikube                                1/1     Running            5 (2d5h ago)    9d
kube-system            kube-apiserver-minikube                      1/1     Running            5 (2d5h ago)    9d
kube-system            kube-controller-manager-minikube             1/1     Running            5 (2d5h ago)    9d
kube-system            kube-proxy-f9r78                             1/1     Running            5 (2d5h ago)    9d
kube-system            kube-scheduler-minikube                      1/1     Running            5 (2d5h ago)    9d
kube-system            storage-provisioner                          1/1     Running            11 (30m ago)    9d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running            5 (2d5h ago)    9d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running            9 (30m ago)     9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The instance has the status **ErrImagePull** and **ImagePullBackOff**. This indicates that Kubernetes environment could not successfully download the specified Docker Image.
To get the exact error message, we use these Kubernetes commands:

kubectl describe pod springboot-crud-deploy-7d8c6b544c-6bwbn

```
Events:
  Type     Reason          Age                  From               Message
  ----     ------          ----                 ----               -------
  Normal   Scheduled       20m                  default-scheduler  Successfully assigned default/springboot-crud-deploy-7d8c6b544c-6bwbn to minikube
  Warning  Failed          19m                  kubelet            Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlo
b2807541384: no space left on device
  Normal   SandboxChanged  19m                  kubelet            Pod sandbox changed, it will be killed and re-created.
  Warning  Failed          19m                  kubelet            Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlo
b1387085943: no space left on device
  Warning  Failed          19m                  kubelet            Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlo
b2672338364: no space left on device
  Warning  Failed          18m                  kubelet            Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlo
b1806491175: no space left on device
  Normal   Pulling         16m (x5 over 19m)    kubelet            Pulling image "enricr/springbootcrud-k8s:1.0"
  Warning  Failed          16m (x5 over 19m)    kubelet            Error: ErrImagePull
  Warning  Failed          16m                  kubelet            Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlo
b1437417324: no space left on device
  Normal   BackOff         4m47s (x65 over 19m)  kubelet           Back-off pulling image "enricr/springbootcrud-k8s:1.0"
  Warning  Failed          4m47s (x65 over 19m)  kubelet           Error: ImagePullBackOff
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Failed to pull image "enricr/springbootcrud-k8s:1.0": write /var/lib/docker/tmp/GetImageBlob1437417324: no space left on device

The disk on our Kubernetes node (Minikube VM) is full. When Kubernetes attempts to download the container image, the underlying container runtime (Docker in this case) cannot write the image layers to the disk because there is no free storage space.

Since we are running on Minikube, the simplest and most effective way to free up disk space is to use the dedicated Minikube commands to clean up the environment.

Before checking Minikube, we must ensure the host EC2 instance isn't full (especially the location where the Minikube driver stores its files). Execute the command:

df -h

Details:

- **df** (Disk Free): It reports the amount of disk space free on the entire filesystem level.
- **-h** (Human-readable): This option makes the output much easier to read by showing sizes in units like Kilobytes (K), Megabytes (M), Gigabytes (G), and Terabytes (T), instead of the default 1-Kilobyte blocks.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           781M  556K  781M   1% /run
/dev/xvda1      8.0G  7.8G  201M  98% /
tmpfs           2.0G     0  2.0G   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs           391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The use of the filesystem mounted on / is 98 %. So, first we should check if there are some files that we can remove.

To show the size of all directories in the root (/) and sort them by size to help us locate the biggest consumers:

sudo du -sh /* | sort -hr

Details:

- **du** (Disk Usage): Estimates file space usage.
  - **-s** (summary): Tells du to display only a grand total for each specified argument, rather than listing every file and subdirectory.
  - **-h** (human-readable): Prints the sizes in powers of 1024 (e.g., K, M, G) for easier reading.
  - **/*** (arguments): This is a wildcard that expands to every file and directory immediately inside the root directory (/).
- The pipe character (**|**) connects the output of the command on the left (sudo du -sh /*) to the input of the command on the right (sort -hr).
- **sort**: Sorts lines of text files or input data.
  - **-h** (human-numeric-sort): Tells sort to compare the human-readable sizes (like 2.5G, 480M) correctly, rather than just treating them as normal text strings.
  - **-r** (reverse): Reverses the result of comparisons, meaning it sorts from largest to smallest.

```
du: cannot access '/proc/29280/task/29280/fdinfo/3': No such file or directory
du: cannot access '/proc/29280/fd/3': No such file or directory
du: cannot access '/proc/29280/fdinfo/3': No such file or directory
5.7G    /var
2.1G    /usr
1.1G    /home
46M     /boot
22M     /etc
556K    /run
120K    /opt
24K     /root
0       /tmp
```

sudo du -sh /var/* | sort -hr

```
[root@ip-172-31-23-113 moby-detect-rro2833986498]# sudo du -sh /var/* | sort -hr
5.6G    /var/lib
112M    /var/cache
25M     /var/log
32K     /var/tmp
0       /var/yp
```

sudo du -sh /var/lib/* | sort -hr

```
[root@ip-172-31-23-113 moby-detect-rro2833986498]# sudo du -sh /var/lib/* | sort -hr
5.6G    /var/lib/docker
18M     /var/lib/selinux
18M     /var/lib/rpm
3.2M    /var/lib/dnf
1.3M    /var/lib/sss
236K    /var/lib/systemd
236K    /var/lib/cloud
120K    /var/lib/containerd
56K     /var/lib/alternatives
12K     /var/lib/amazon
4.0K    /var/lib/logrotate
4.0K    /var/lib/chrony
0       /var/lib/yum
```

The folder /var/lib/docker is using 5.6G of disc space.

sudo du -sh /var/lib/docker/* | sort -hr

```
[root@ip-172-31-23-113 moby-detect-rro2833986498]# sudo du -sh /var/lib/docker/* | sort -hr
3.6G    /var/lib/docker/overlay2
2.0G    /var/lib/docker/volumes
2.2M    /var/lib/docker/image
1.1M    /var/lib/docker/buildkit
272K    /var/lib/docker/containers
52K     /var/lib/docker/network
4.0K    /var/lib/docker/engine-id
0       /var/lib/docker/tmp
```

The /var/lib/docker/overlay2 folder on our Amazon Linux 2023 EC2 instance contains the filesystem layers used by the Docker OverlayFS storage driver for all our Docker images and containers. This is where the actual on-disk data for the container images and the writable layers of running or stopped containers are stored. It is possible and often necessary to reduce the size of this folder, as it is the most common cause of disk space issues on Docker hosts. For that we can use **docker system prune**. Possible options:

Command	Description
docker system prune	Removes **dangling** (unused and untagged) images, stopped containers, unused networks, and dangling build cache.
docker system prune -a	Removes **all unused** images (not just dangling ones) and everything listed above. This is much more aggressive and may remove images you intended to keep for later use.
docker system prune --volumes	Removes dangling **volumes** in addition to the items above. Volumes are usually stored outside /overlay2 but are often part of the cleanup process.

The images that we have before executing this command are:

```
[ec2-user@ip-172-31-23-113 ~]$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
enricr/springbootcrud-k8s     1.0       472a95a3b485   4 days ago      1.06GB
gcr.io/k8s-minikube/kicbase   v0.0.48   c6b5532e987b   2 months ago    1.31GB
[ec2-user@ip-172-31-23-113 ~]$
```

First, we try the less aggressive option:

docker system prune

```
[ec2-user@ip-172-31-23-113 ~]$ docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - unused build cache

Are you sure you want to continue? [y/N] y
Deleted build cache objects:
pac6mw3k108l3xq3fku17a7xm
j5i7dze4lj0hlu235uqhkshjb
zk4wwfy9oeio2zivjivgm42g9

Total reclaimed space: 56.76MB
[ec2-user@ip-172-31-23-113 ~]$
```

```
[ec2-user@ip-172-31-23-113 ~]$ sudo du -sh /var/lib/docker/overlay2
3.6G    /var/lib/docker/overlay2
[ec2-user@ip-172-31-23-113 ~]$
```

```
[ec2-user@ip-172-31-23-113 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           781M  556K  781M   1% /run
/dev/xvda1      8.0G  7.7G  255M  97% /
tmpfs           2.0G     0  2.0G   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs           391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 ~]$
```

We have recovered very little disc space.

Now try:

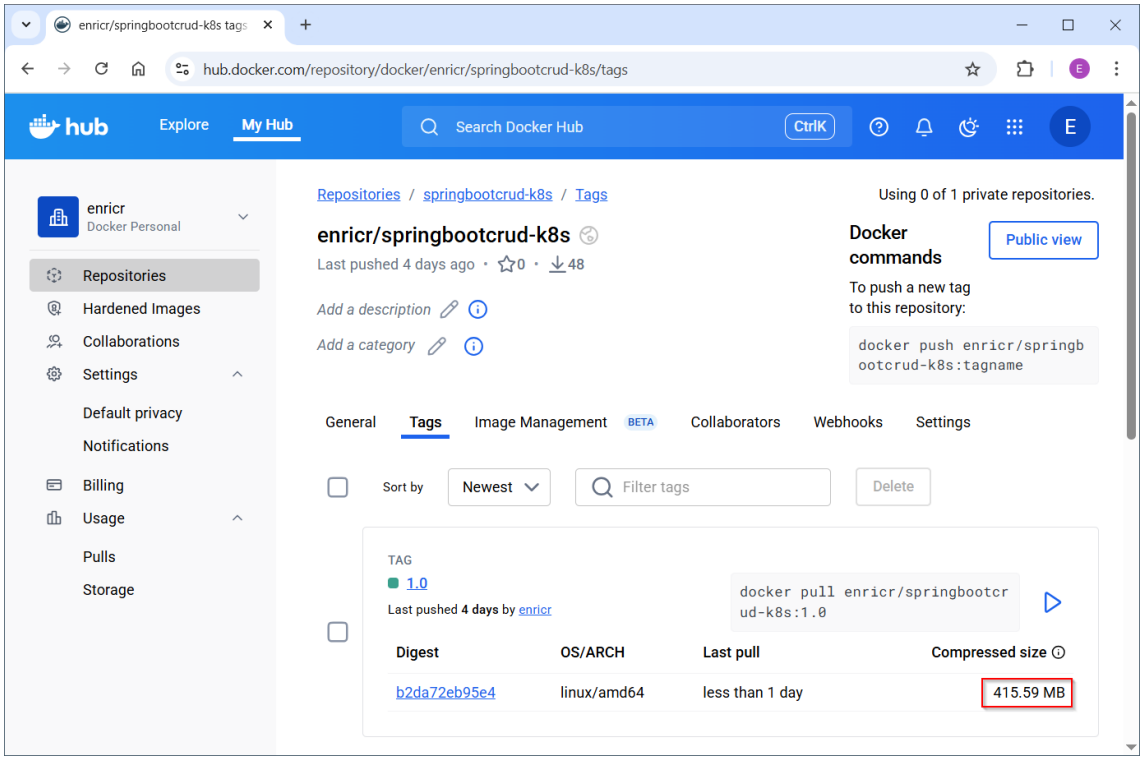
docker system prune -a

```
[ec2-user@ip-172-31-23-113 ~]$ docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: enricr/springbootcrud-k8s:1.0
untagged: enricr/springbootcrud-k8s@sha256:b2da72eb95e40b99897b16c9fc85b65f218fa49d0b7f326bfaf0cf98ab3eb719
deleted: sha256:472a95a3b48504d7277da9fe4b2f28f348fbc5212189511da33dd3905dfaa702

Deleted build cache objects:
a7nu9otupxsj6g3jyc6clq7qq
whr3158wz9mddt5buaoab0w69
74y9be33kzwuw7xciubvcx9wo

Total reclaimed space: 914.9MB
[ec2-user@ip-172-31-23-113 ~]$
```

```
[ec2-user@ip-172-31-23-113 ~]$ sudo du -sh /var/lib/docker/overlay2
2.5G    /var/lib/docker/overlay2
[ec2-user@ip-172-31-23-113 ~]$
```

```
[ec2-user@ip-172-31-23-113 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           781M  556K  781M   1% /run
/dev/xvda1      8.0G  6.7G  1.3G  84% /
tmpfs           2.0G     0  2.0G   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs           391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 ~]$
```

In Docker Hub we can see that the Docker image is 415.59 MB in compressed size:



Now delete the app deployment:

kubectl delete deploy springboot-crud-deploy

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       1/1     1            1           2d21h
default                springboot-crud-deploy      0/3     3            0           14h
kube-system            coredns                     1/1     1            1           9d
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           9d
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete deploy springboot-crud-deploy
deployment.apps "springboot-crud-deploy" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       1/1     1            1           2d21h
kube-system            coredns                     1/1     1            1           9d
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           9d
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS        AGE
default                mysql-79c9d95747-khqxv                       1/1     Running   2 (13h ago)     2d21h
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running   6 (13h ago)     9d
kube-system            etcd-minikube                                1/1     Running   6 (13h ago)     9d
kube-system            kube-apiserver-minikube                      1/1     Running   6 (13h ago)     9d
kube-system            kube-controller-manager-minikube             1/1     Running   6 (13h ago)     9d
kube-system            kube-proxy-f9r78                             1/1     Running   6 (13h ago)     9d
kube-system            kube-scheduler-minikube                      1/1     Running   6 (13h ago)     9d
kube-system            storage-provisioner                          1/1     Running   13 (25m ago)    9d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running   6 (13h ago)     9d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running   11 (25m ago)    9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and apply again the app deployment:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f app-deploy2.yaml
configmap/springboot-config unchanged
secret/springboot-secret unchanged
deployment.apps/springboot-crud-deploy created
service/springboot-crud-svc unchanged
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

But the problem persists:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS              RESTARTS       AGE
default                mysql-79c9d95747-khqxv                       1/1     Running             2 (13h ago)    2d21h
default                springboot-crud-deploy-7d8c6b544c-4m6gc      0/1     ImagePullBackOff    0              38s
default                springboot-crud-deploy-7d8c6b544c-x6nml      0/1     ContainerCreating   0              38s
default                springboot-crud-deploy-7d8c6b544c-z7qts      0/1     ContainerCreating   0              38s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running             6 (13h ago)    9d
kube-system            etcd-minikube                                1/1     Running             6 (13h ago)    9d
kube-system            kube-apiserver-minikube                      1/1     Running             6 (13h ago)    9d
kube-system            kube-controller-manager-minikube             1/1     Running             6 (13h ago)    9d
kube-system            kube-proxy-f9r78                             1/1     Running             6 (13h ago)    9d
kube-system            kube-scheduler-minikube                      1/1     Running             6 (13h ago)    9d
kube-system            storage-provisioner                          1/1     Running             13 (28m ago)   9d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running             6 (13h ago)    9d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running             11 (28m ago)   9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

To get a high-level summary of the space used by the four main Docker object types:

docker system df

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          1         1         1.308GB   0B (0%)
Containers      1         1         3.049MB   0B (0%)
Local Volumes   1         1         2.799GB   0B (0%)
Build Cache     0         0         0B        0B
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

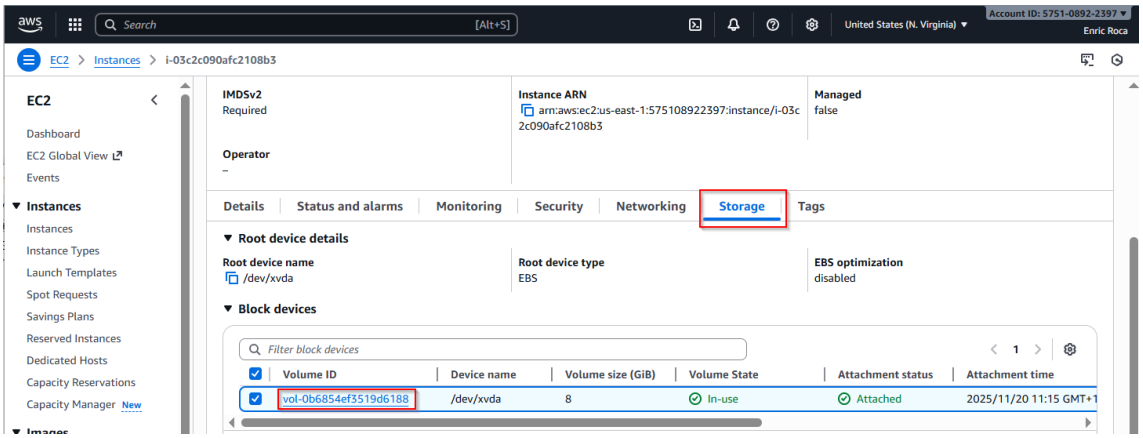
The RECLAIMABLE column gives us the amount of space that can be recovered by running a prune command like **docker system prune** or **docker system prune -a**, that is 0 B.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           781M  556K  781M   1% /run
/dev/xvda1      8.0G  7.8G  205M  98% /
tmpfs           2.0G     0  2.0G   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs           391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

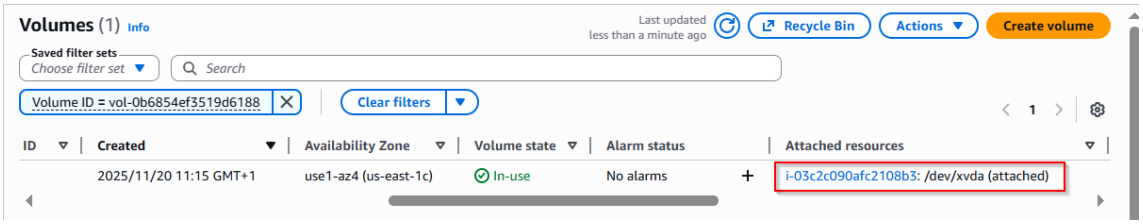
So, there is not enough free space in the filesystem **/dev/xvda1** mounted on "/".

Our 8 GiB root volume is insufficient for modern OS running complex tools like Minikube and Docker. We must increase the size of your EBS root volume in the AWS console. This is a three-step process: Modify the EBS volume size on AWS, extend the partition on the instance, and finally extend the filesystem.

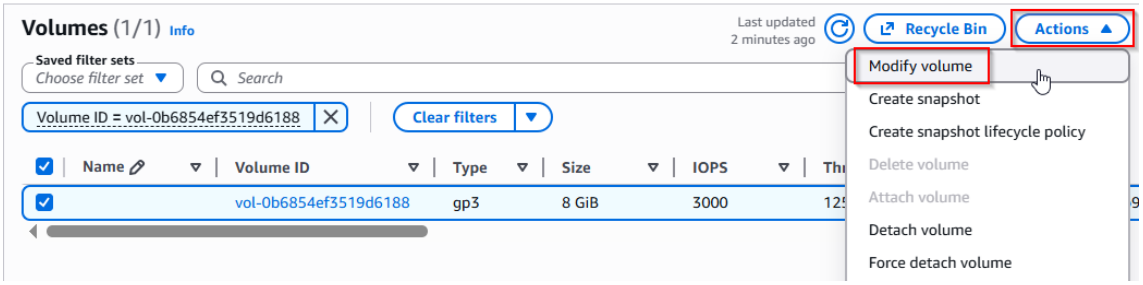
To modify the volume size, we go to the **AWS EC2 Console**, in our instance select the **Storage** tab and click on the **Volum ID**:



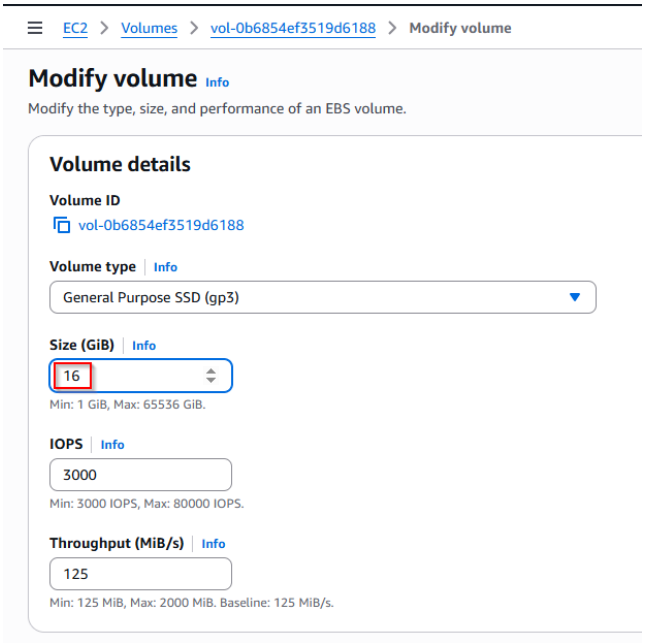
We can see that this volume is attached to **/dev/xvda**:



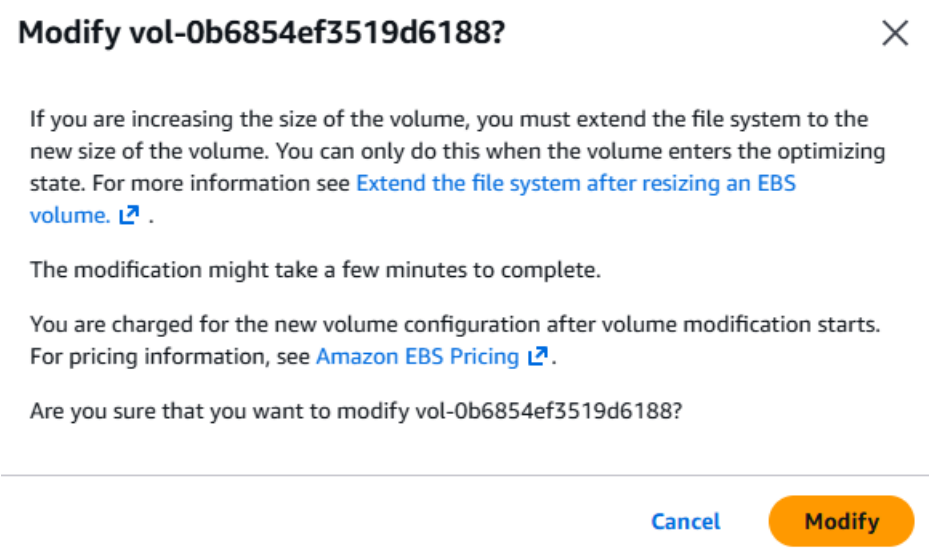
Go to **Actions** and **Modify volume**:



Change the size from 8 GiB to 16 GiB:



Click on **Modify**.



Press **Modify**.

Now we must **extend the partition**. Connected to the EC2 instance, execute:

lsblk

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0  16G  0 disk
├─xvda1   202:1    0   8G  0 part /
├─xvda127 259:0    0   1M  0 part
└─xvda128 259:1    0  10M  0 part /boot/efi
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

We can see that the volume **xvda** has changed from 8 to 16 G, but the partition xvda1 is still 8 G.

To **extend the partition** to fill the new volume size:

sudo growpart /dev/xvda 1

Details:

- **/dev/xvda**: This specifies the disk device containing the partition you want to resize
- **1**: This specifies the partition number on the disk /dev/xvda that we want to expand. In this case, it refers to the first partition, which is likely /dev/xvda1.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ sudo growpart /dev/xvda 1
CHANGED: partition=1 start=24576 old: size=16752607 end=16777183 new: size=33529823 end=33554399
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Now, we **extend the XFS filesystem** (typical on Amazon Linux 2023) to use the expanded partition.

Before, check the disk space usage for all mounted filesystems :

df -hT

Details:

- **df** (Disk Free): It reports the amount of disk space free on the entire filesystem level.
- **-h** (Human-readable): This option makes the output much easier to read by showing sizes in units like Kilobytes (K), Megabytes (M), Gigabytes (G), and Terabytes (T), instead of the default 1-Kilobyte blocks.
- -**T** (Print type): This option adds a column to the output that displays the filesystem type (e.g., ext4, xfs, tmpfs).

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
devtmpfs       devtmpfs  4.0M     0  4.0M   0% /dev
tmpfs          tmpfs     2.0G     0  2.0G   0% /dev/shm
tmpfs          tmpfs     781M  556K  781M   1% /run
/dev/xvda1     xfs       8.0G  6.7G  1.3G  84% /
tmpfs          tmpfs     2.0G     0  2.0G   0% /tmp
/dev/xvda128   vfat       10M  1.3M  8.7M  13% /boot/efi
tmpfs          tmpfs     391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

To expand a XFS filesystem to fill the entire capacity of the partition it resides on:

sudo xfs_growfs -d /

Details:

- **xfs_growfs**: Utility designed to resize XFS filesystems.
- **-d**: This option stands for "data section" and is what tells the utility to grow the main data area of the filesystem to the maximum size allowed by the device (partition) it is on.

- /: This specifies the mount point of the filesystem you want to expand. In this case, it is the root directory of your Linux system. This means you are resizing the primary filesystem where the operating system files are located.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ sudo xfs_growfs -d /
meta-data=/dev/xvda1              isize=512    agcount=2, agsize=1047040 blks
         =                        sectsz=4096  attr=2, projid32bit=1
         =                        crc=1        finobt=1, sparse=1, rmapbt=0
         =                        reflink=1    bigtime=1 inobtcount=1 nrext64=0
         =                        exchange=0
data     =                        bsize=4096   blocks=2094075, imaxpct=25
         =                        sunit=128    swidth=128 blks
naming   =version 2               bsize=16384  ascii-ci=0, ftype=1, parent=0
log      =internal log            bsize=4096   blocks=16384, version=2
         =                        sectsz=4096  sunit=4 blks, lazy-count=1
realtime =none                    extsz=4096   blocks=0, rtextents=0
data blocks changed from 2094075 to 4191227
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Verify the space:

df -h

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           781M  556K  781M   1% /run
/dev/xvda1       16G  7.8G  8.2G  49% /
tmpfs           2.0G     0  2.0G   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs           391M     0  391M   0% /run/user/1000
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

We see that "/" is 16 G total disc size and it has 8.2 G available size.

Now we can continue with our application deployment. Delete the app deploy:

kubectl delete deploy springboot-crud-deploy

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS             RESTARTS        AGE
default                mysql-79c9d95747-khqxv                       1/1     Running            2 (14h ago)     2d22h
default                springboot-crud-deploy-7d8c6b544c-4m6gc      0/1     CrashLoopBackOff   4 (40s ago)     65m
default                springboot-crud-deploy-7d8c6b544c-x6nml      0/1     CrashLoopBackOff   4 (45s ago)     65m
default                springboot-crud-deploy-7d8c6b544c-z7qts      0/1     CrashLoopBackOff   4 (39s ago)     65m
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running            6 (14h ago)     9d
kube-system            etcd-minikube                                1/1     Running            6 (14h ago)     9d
kube-system            kube-apiserver-minikube                      1/1     Running            6 (14h ago)     9d
kube-system            kube-controller-manager-minikube             1/1     Running            6 (14h ago)     9d
kube-system            kube-proxy-f9r78                             1/1     Running            6 (14h ago)     9d
kube-system            kube-scheduler-minikube                      1/1     Running            6 (14h ago)     9d
kube-system            storage-provisioner                          1/1     Running            13 (92m ago)    9d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running            6 (14h ago)     9d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running            11 (92m ago)    9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       1/1     1            1           2d22h
default                springboot-crud-deploy      0/3     3            0           65m
kube-system            coredns                     1/1     1            1           9d
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           9d
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete deploy springboot-crud-deploy
deployment.apps "springboot-crud-deploy" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and apply it again:

kubectl apply -f app-deploy2.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f app-deploy2.yaml
configmap/springboot-config unchanged
secret/springboot-secret unchanged
deployment.apps/springboot-crud-deploy created
service/springboot-crud-svc unchanged
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS             RESTARTS        AGE
default                mysql-79c9d95747-khqxv                       1/1     Running            2 (14h ago)     2d22h
default                springboot-crud-deploy-7d8c6b544c-49hvk      0/1     CrashLoopBackOff   3 (18s ago)     63s
default                springboot-crud-deploy-7d8c6b544c-s6xmn      0/1     CrashLoopBackOff   3 (14s ago)     63s
default                springboot-crud-deploy-7d8c6b544c-wrgtw      0/1     CrashLoopBackOff   3 (18s ago)     63s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running            6 (14h ago)     9d
kube-system            etcd-minikube                                1/1     Running            6 (14h ago)     9d
kube-system            kube-apiserver-minikube                      1/1     Running            6 (14h ago)     9d
kube-system            kube-controller-manager-minikube             1/1     Running            6 (14h ago)     9d
kube-system            kube-proxy-f9r78                             1/1     Running            6 (14h ago)     9d
kube-system            kube-scheduler-minikube                      1/1     Running            6 (14h ago)     9d
kube-system            storage-provisioner                          1/1     Running            13 (98m ago)    9d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running            6 (14h ago)     9d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running            11 (98m ago)    9d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

The problem with ImagePullBackOff has been solved, but now it appears the error **CrashLoopBackOff**. This means the container started, crashed, and Kubernetes is repeatedly trying to restart it. The error is now within our application's runtime, not the Kubernetes infrastructure.

kubectl describe pod springboot-crud-deploy-7d8c6b544c-49hvk

```
Events:
  Type     Reason     Age                  From               Message
  ----     ------     ----                 ----               -------
  Normal   Scheduled  3m19s                default-scheduler  Successfully assigned default/springboot-crud-deploy-7d8c6b544c-49hvk to minikube
  Normal   Pulled     22s (x6 over 3m19s)  kubelet            Container image "enricr/springbootcrud-k8s:1.0" already present on machine
  Normal   Created    22s (x6 over 3m19s)  kubelet            Created container: springboot-crud-container
  Normal   Started    22s (x6 over 3m18s)  kubelet            Started container springboot-crud-container
  Warning  BackOff    6s (x16 over 3m17s)  kubelet            Back-off restarting failed container springboot-crud-container in pod springboot-crud-deploy-7d8c
6b544c-49hvk_default(80e139aa-7d27-4aa0-b599-e4dfe7be1ad4)
```

To see the logs of the crashing container:

kubectl logs springboot-crud-deploy-7d8c6b544c-49hvk

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl logs springboot-crud-deploy-7d8c6b544c-49hvk
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/springframework/boot/loader/launch/JarLauncher has been
compiled by a more recent version of the Java Runtime (class file version 61.0), this version of the Java Runtime only recogniz
es class file versions up to 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:756)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:473)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:74)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:369)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:363)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:362)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:352)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:621)
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Error: A JNI error has occurred, please check your installation and try again

Exception in thread "main" java.lang.UnsupportedClassVersionError:

“org/springframework/boot/loader/launch/JarLauncher has been compiled by a more recent version of the Java Runtime (class file version 61.0), this version of the Java Runtime only recognizes class file versions up to 52.0”

The logs clearly identify the cause of the CrashLoopBackOff: a Java version mismatch between how our Spring Boot application was compiled and the Java Runtime Environment (JRE) being used inside the container.
Our Spring Boot application was compiled with Java 17, that corresponds to the class file version 61.0, and the JRE inside our container is only Java 8 that corresponds to the class file version 52.0. Our application requires a minimum of Java 17 to run, but the container is trying to execute it with Java 8, causing the application to fail immediately upon startup.

To solve the problem, we must change the **Docker image** used in our deployment to one that includes a compatible Java Runtime Environment (JRE), which means Java 17 or newer. For that, we must rebuild our Docker image changing the base image to use Java 17. So, we are going to create a new Dockerfile, that I store in my GitHub repository like **Dockerfile3**, with the code:

```
# 1. Use the official, minimal AL2023 base image
FROM public.ecr.aws/amazonlinux/amazonlinux:2023

# 2. Install Amazon Corretto 17 (the AL2023-compatible Java 17 JDK)
# We use the 'headless' package to install the smallest JRE necessary for server applications
RUN dnf install -y java-17-amazon-corretto-headless \
  && dnf clean all

# 3. Set the JAVA_HOME environment variable for the application
# Note the path is now '/usr/lib/jvm/java-17-amazon-corretto'
ENV JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto

# 4. Continue with your application steps
EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Now copy the code of Dockerfile3 and paste in the Dockerfile of the **/home/ec2-user/springbootcrud-k8s** folder of our instance, deleting its content before. You can use the editor that you want, for instance nano:

```
  GNU nano 8.3                                                    Dockerfile
# 1. Use the official, minimal AL2023 base image
FROM public.ecr.aws/amazonlinux/amazonlinux:2023

# 2. Install Amazon Corretto 17 (the AL2023-compatible Java 17 JDK)
# We use the 'headless' package to install the smallest JRE necessary for server applications
RUN dnf install -y java-17-amazon-corretto-headless \
    && dnf clean all

# 3. Set the JAVA_HOME environment variable for the application
# Note the path is now '/usr/lib/jvm/java-17-amazon-corretto'
ENV JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto

# 4. Continue with your application steps
EXPOSE 8080
ADD target/crud-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Save (CTRL+o, Enter), and exit (CTRL+x).

To make our deployment history more transparent we are going to change the **Docker Image tag** to **2.0** as a clearer way to handle a significant change (like a major Java version upgrade).

Build the new image with Java 17 (as specified in the Dockerfile):

docker build -t enricr/springbootcrud-k8s:2.0 .

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker build -t enricr/springbootcrud-k8s:2.0 .
[+] Building 30.9s (8/8) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                          0.0s
 => => transferring dockerfile: 750B                                                                          0.0s
 => [internal] load metadata for public.ecr.aws/amazonlinux/amazonlinux:2023                                  0.4s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 2B                                                                               0.0s
 => [1/3] FROM public.ecr.aws/amazonlinux/amazonlinux:2023@sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d  4.7s
 => => resolve public.ecr.aws/amazonlinux/amazonlinux:2023@sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d  0.0s
 => => sha256:ac90a73983be7974c18062a0a56d2c4ad6b9369b4eb2b4090141462a2ef428b9 662B / 662B                    0.0s
 => => sha256:1c7de4eb5ced9ea3f72366a34ec955a53e9b0f4ac53d332a155de21eb808d732 53.97MB / 53.97MB              0.6s
 => => sha256:f5ca6cafc706233c641ad838e738047389943afd3585cb1df7eedfc4d9b1799d 770B / 770B                    0.0s
 => => sha256:fb74284364222bd16641b0f0d36df624bf1d81758c1ad50a8b5764ab1ef9ebb0 528B / 528B                    0.0s
 => => extracting sha256:1c7de4eb5ced9ea3f72366a34ec955a53e9b0f4ac53d332a155de21eb808d732                     3.9s
 => [internal] load build context                                                                             0.9s
 => => transferring context: 56.77MB                                                                          0.8s
 => [2/3] RUN dnf install -y java-17-amazon-corretto-headless     && dnf clean all                           23.7s
 => [3/3] ADD target/crud-0.0.1-SNAPSHOT.jar app.jar                                                          0.2s
 => exporting to image                                                                                        1.7s
 => => exporting layers                                                                                       1.7s
 => => writing image sha256:f8f576e1d9baceb604f7a77d8ea807e7630d4c70ef7f3f99ecebe9054fb2506e                  0.0s
 => => naming to docker.io/enricr/springbootcrud-k8s:2.0                                                      0.0s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Check with **docker images** :

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED              SIZE
enricr/springbootcrud-k8s       2.0       f8f576e1d9ba   About a minute ago   484MB
gcr.io/k8s-minikube/kicbase     v0.0.48   c6b5532e987b   2 months ago         1.31GB
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Push the new image to the Docker Hub:

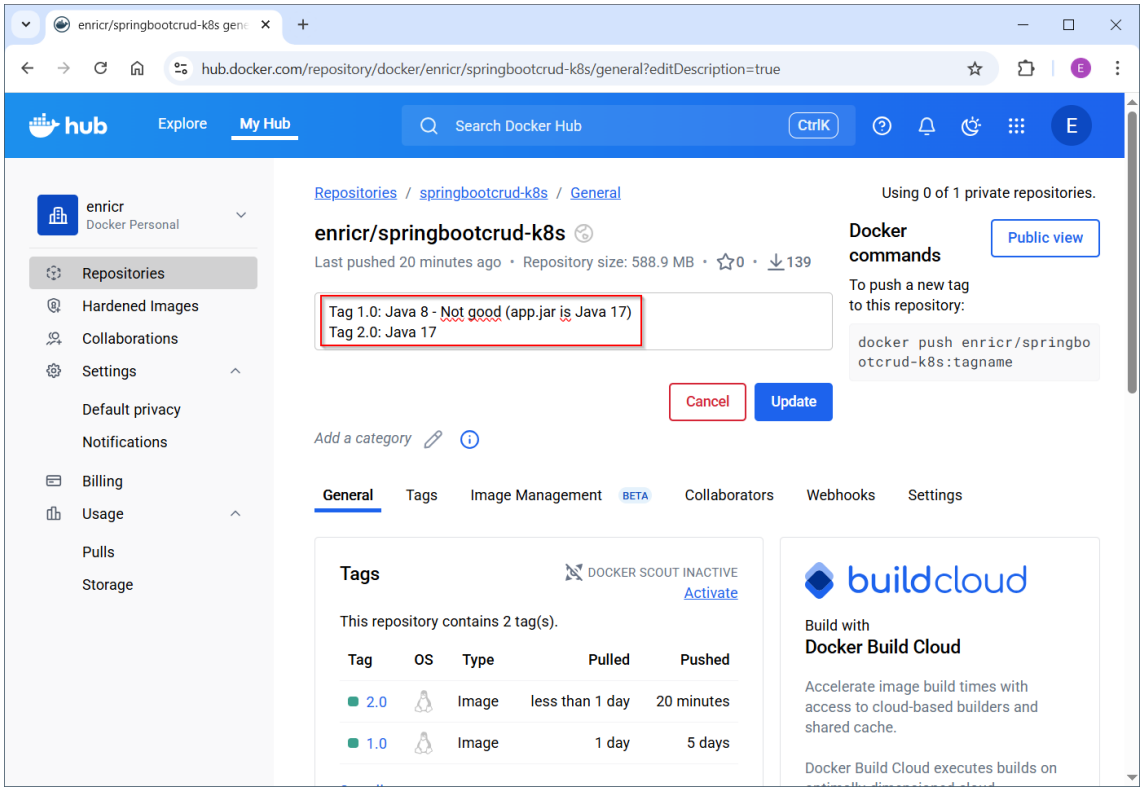
docker push enricr/springbootcrud-k8s:2.0

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ docker push enricr/springbootcrud-k8s:2.0
The push refers to repository [docker.io/enricr/springbootcrud-k8s]
c3f03e060da6: Pushed
9a779290f1ec: Pushed
0e7c6314d998: Layer already exists
2.0: digest: sha256:e0fd0446a11e9ae18b937db21ac729fa6f2a92fd8a07656c3a9a7e0a5b49be39 size: 954
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

To keep things organized, add a description in the **Docker Hub** repository about the Java version of each tag. For instance, description:

Tag 1.0: Java 8 - Not good (app.jar is Java 17)

Tag 2.0: Java 17



Note: We could have kept the 1.0 tag and replaced the image without changing the Dockerfile and later YAML file, but this way we've done more practice and the project's evolution is documented. We can delete the 1.0 tag when we need space on Docker Hub.

Now, we are going to change the image in the yaml deployment file. First copy the app-deploy2.yaml as app-deploy3.yaml

cp deploy2.yaml deploy3.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ cp app-deploy2.yaml app-deploy3.yaml
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Change the image from enricr/springbootcrud-k8s:1.0 to enricr/springbootcrud-k8s:2.0 (using nano, for example).

Once done:

```
  GNU nano 8.3                                          app-deploy3.yaml
  DB_PASSWORD: "bXlzcWxwYXNz"   # Base64 encoded 'mysqlpass'
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: springboot-crud-deploy
spec:
  selector:
    matchLabels:
      app: springboot-crud
  replicas: 3
  template:
    metadata:
      labels:
        app: springboot-crud
    spec:
      containers:
        - name: springboot-crud-container
          image: enricr/springbootcrud-k8s:2.0
          ports:
            - containerPort: 8080
          env:  # Setting Environmental Variables using ConfigMap and Secret
            # Get DB_HOST from ConfigMap
            - name: DB_HOST
              valueFrom:
                configMapKeyRef:
                  name: springboot-config
                  key: DB_HOST
            # Get DB_NAME from ConfigMap
            - name: DB_NAME
              valueFrom:
                configMapKeyRef:
                  name: springboot-config
                  key: DB_NAME

^G Help          ^O Write Out     ^F Where Is      ^K Cut           ^T Execute       ^C Location
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line
```

Delete the springboot-crud-deploy deployment:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy -A
NAMESPACE              NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
default                mysql                       1/1     1            1           3d23h
default                springboot-crud-deploy      0/3     3            0           24h
kube-system            coredns                     1/1     1            1           11d
kubernetes-dashboard   dashboard-metrics-scraper   1/1     1            1           10d
kubernetes-dashboard   kubernetes-dashboard        1/1     1            1           10d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl delete deploy springboot-crud-deploy
deployment.apps "springboot-crud-deploy" deleted
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and apply app-deploy3.yaml :

kubectl apply -f app-deploy3.yaml

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl apply -f app-deploy3.yaml
configmap/springboot-config unchanged
secret/springboot-secret unchanged
deployment.apps/springboot-crud-deploy created
service/springboot-crud-svc unchanged
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

and check the pods:

kubectl get pod -A

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS       AGE
default                mysql-79c9d95747-khqxv                       1/1     Running   3 (22h ago)    3d23h
default                springboot-crud-deploy-7bfdf5f7b-2rffh       1/1     Running   0              39s
default                springboot-crud-deploy-7bfdf5f7b-gjtkw       1/1     Running   0              39s
default                springboot-crud-deploy-7bfdf5f7b-kps4v       1/1     Running   0              39s
kube-system            coredns-66bc5c9577-4vznb                     1/1     Running   7 (22h ago)    11d
kube-system            etcd-minikube                                1/1     Running   7 (22h ago)    11d
kube-system            kube-apiserver-minikube                      1/1     Running   7 (22h ago)    11d
kube-system            kube-controller-manager-minikube             1/1     Running   7 (22h ago)    11d
kube-system            kube-proxy-f9r78                             1/1     Running   7 (22h ago)    11d
kube-system            kube-scheduler-minikube                      1/1     Running   7 (22h ago)    11d
kube-system            storage-provisioner                          1/1     Running   15 (55m ago)   11d
kubernetes-dashboard   dashboard-metrics-scraper-77bf4d6c4c-hqx6t   1/1     Running   7 (22h ago)    10d
kubernetes-dashboard   kubernetes-dashboard-855c9754f9-fz4k5        1/1     Running   13 (55m ago)   10d
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

Great! At last, we have our pods and their containers in running status.

All the resources that we have created are in the **default** namespace, so we don't need to specify -A (all namespaces):

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get pod
NAME                                    READY   STATUS    RESTARTS      AGE
mysql-79c9d95747-khqxv                  1/1     Running   3 (22h ago)   3d23h
springboot-crud-deploy-7bfdf5f7b-2rffh  1/1     Running   0             3m44s
springboot-crud-deploy-7bfdf5f7b-gjtkw  1/1     Running   0             3m44s
springboot-crud-deploy-7bfdf5f7b-kps4v  1/1     Running   0             3m44s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get deploy
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
mysql                  1/1     1            1           3d23h
springboot-crud-deploy 3/3     3            3           3m55s
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      11d
mysql-config       1      4d23h
springboot-config  2      40h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get secret
NAME               TYPE     DATA   AGE
mysql-secret       Opaque   2      3d23h
springboot-secret  Opaque   2      40h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get service
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.96.0.1        <none>        443/TCP          11d
mysql                ClusterIP   None             <none>        3306/TCP         4d23h
springboot-crud-svc  NodePort    10.102.130.160   <none>        8080:30463/TCP   40h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

IV.6- Configure connectivity

Now that we have our application running in a Minikube cluster with 3 pods, we have to get connectivity to the application from Internet (our laptop).

The core problem is that our Spring Boot application is running inside a Minikube cluster, which itself is running inside our EC2 instance. Our **laptop**, the **EC2 instance** (host), and the **Minikube cluster** (guest) are on different network layers.

We can get the public and private IPs of our EC2 from the AWS console:

Public IPv4 address = 98.93.19.40 (This value can change when restart the instance)
Private IPv4 addresses = 172.31.23.113

To retrieve the IP address of the **Minikube cluster's virtual machine (VM)** or **host** where the Kubernetes components are running:

minikube ip

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ minikube ip
192.168.49.2
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

And we can see the NodePort port of the springboot-crud-svc that exposes our app with:

kubectl get service springboot-crud-svc

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl get service springboot-crud-svc
NAME                  TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
springboot-crud-svc   NodePort   10.102.130.160   <none>        8080:30463/TCP   41h
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$
```

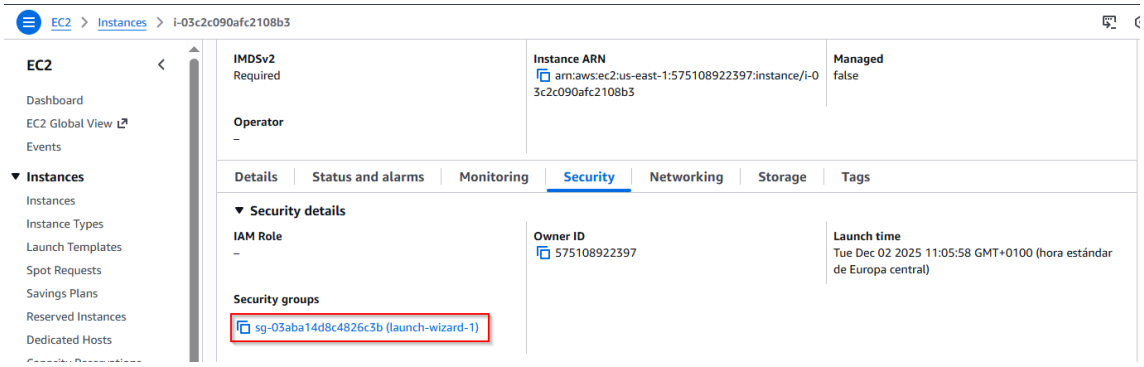
So, we have:

Component	IP Address	Purpose
Our Laptop	(External)	Origin of the web browser request.
EC2 Public IP	98.93.19.40	External entry point to the EC2 host.
EC2 Private IP	172.31.23.113	Internal IP of the EC2 host.
Minikube IP	192.168.49.2	The IP of the Minikube VM/container inside the EC2 host.
K8s Service	10.102.130.160 (ClusterIP)	Internal IP of the Kubernetes service.
NodePort	30463	The port Minikube's Node is listening on for the service.
App Port	8080	The port our Spring Boot app is listening on.

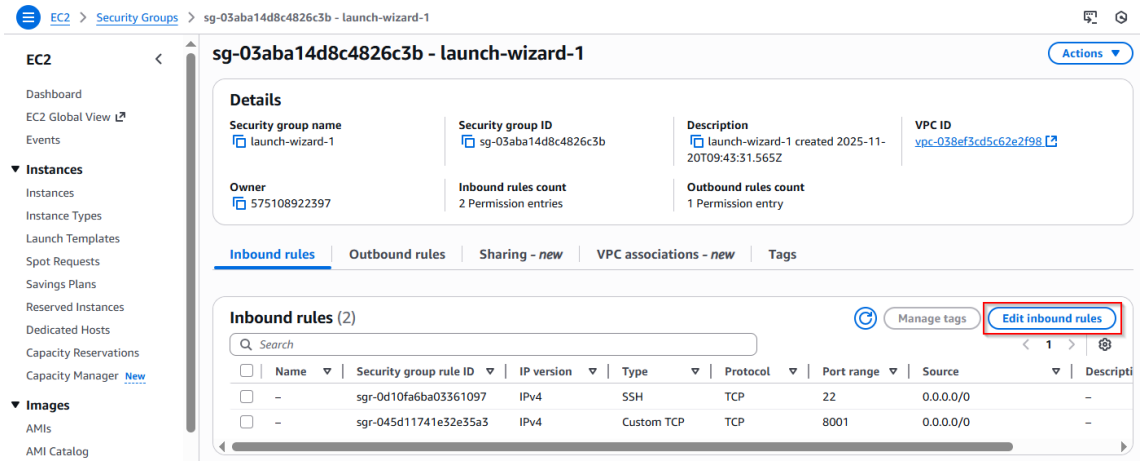
To solve the problem, we must do two things:

1. Create an **Inbound rule** in the **AWS Security Group** to allow all the traffic from Internet to the EC2 instance's public IP by port 8080.
2. Create an **Internal Bridge** that tunnels traffic from the EC2 instance's host interfaces to the Spring Boot application's service inside the Minikube cluster.

1) To create an Inbound rule in the AWS Console, go to our instance, select the **Security** tab and click on the security group:



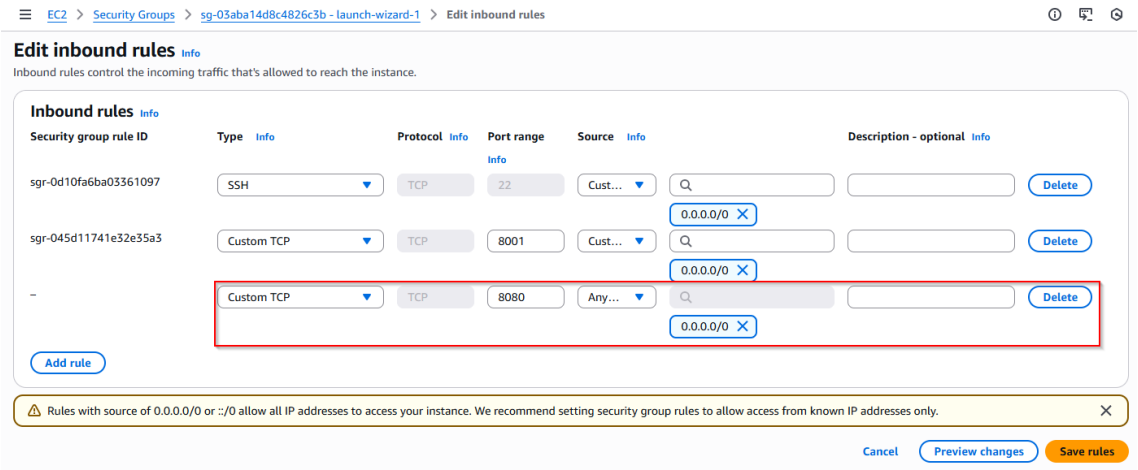
And press in **Edit Inbound rues**:



Select **Add rule** and set the configuration:

Type: Custom TCP
Port range: 8080
Source: Anywhere-IPv4

Once done:



Save rules.

2) To create an **Internal Bridge** we can execute in the terminal the command:

kubectl port-forward --address 0.0.0.0 svc/springboot-crud-svc 8080:8080 &

Details:

- **kubectl port-forward**: The Kubernetes command used to forward traffic from a local port (on the host where it's run) to a port on a Pod or Service in the cluster.
- **--address 0.0.0.0**: By default, kubectl port-forward binds only to the localhost interface (127.0.0.1) of the machine where it runs (the EC2 instance). Using --address 0.0.0.0 tells the command to bind to all available network interfaces on the EC2 instance, including the one associated with the EC2's private IP (172.31.23.113) and the public IP (98.93.19.40). This makes the forwarded port accessible from outside the EC2 instance.
- **svc/springboot-crud-svc**: This specifies the Kubernetes resource to forward to, which is our deployed service.
- **8080:8080**: This defines the port mapping:
  - The first 8080 is the local port on the EC2 instance (the host) that the traffic will be forwarded to.
  - The second 8080 is the target port of the service inside the Minikube cluster.
- **&**: This runs the command in the background, allowing you to close the terminal session or continue working on the EC2 instance while the port forwarding remains active.

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl port-forward --address 0.0.0.0 svc/springboot-crud-svc 8080:8080 &
[1] 47518
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ Forwarding from 0.0.0.0:8080 -> 8080
```

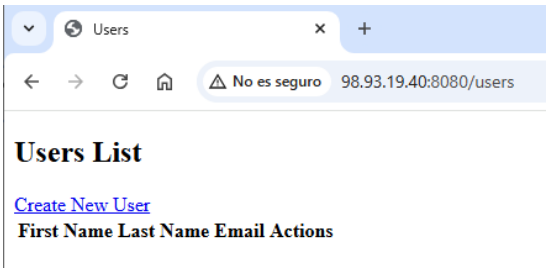
Part V: Check that the solution works

Finally, we are going to check that the solution is correctly deployed and that our application works.

V.1- Test the App and values in DB

To access our application from our laptop's web browser we must use the **EC2's Public IPv4** address and the forwarded port **8080**:

http://98.93.19.40:8080/users



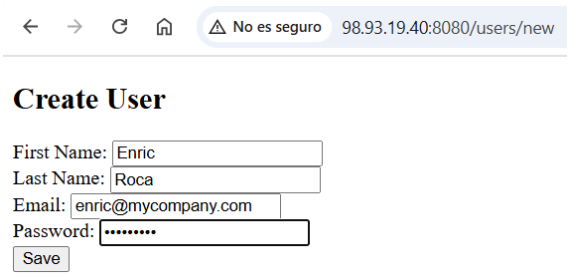
So, it works!

We can see a line with the connection in the terminal:

```
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ kubectl port-forward --address 0.0.0.0 svc/springboot-crud-svc 8080:8080 &
[1] 47518
[ec2-user@ip-172-31-23-113 springbootcrud-k8s]$ Forwarding from 0.0.0.0:8080 -> 8080
Handling connection for 8080
```

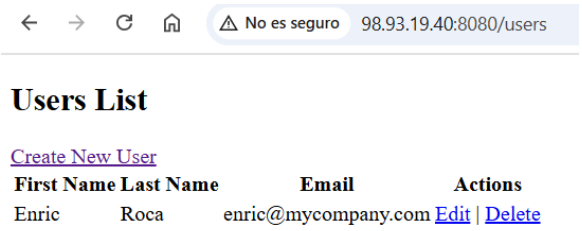
Click on **Create New User**:

Type the details of our first user:



As password I have write password1.

Click on Save and it lists the user:



To check that the data has been inserted in the MySQL DB, open a new terminal in the AWS console to connect to our instance, and access to the mysql container shell:

kubectl exec -it mysql-79c9d95747-khqxv -- /bin/bash

```
[ec2-user@ip-172-31-23-113 ~]$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS      AGE
mysql-79c9d95747-khqxv                   1/1     Running   3 (24h ago)   4d1h
springboot-crud-deploy-7bfdf5f7b-2rffh   1/1     Running   0             133m
springboot-crud-deploy-7bfdf5f7b-gjtkw   1/1     Running   0             133m
springboot-crud-deploy-7bfdf5f7b-kps4v   1/1     Running   0             133m
[ec2-user@ip-172-31-23-113 ~]$ kubectl exec -it mysql-79c9d95747-khqxv -- /bin/bash
bash-4.2#
```

mysql -u root -p

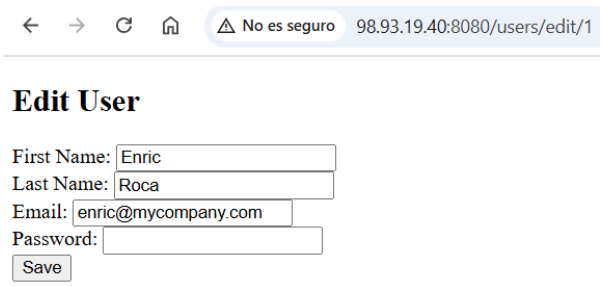
Type the password = mysqlpass

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| springCrud         |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> USE springCrud;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+----------------------+
| Tables_in_springCrud |
+----------------------+
| users                |
+----------------------+
1 row in set (0.00 sec)

mysql>
```

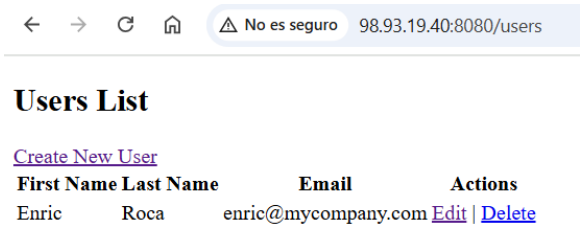
```
mysql> DESCRIBE users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| email      | varchar(255) | YES  | UNI | NULL    |                |
| first_name | varchar(255) | YES  |     | NULL    |                |
| last_name  | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM users;
+----+---------------------+------------+-----------+-----------+
| id | email               | first_name | last_name | password  |
+----+---------------------+------------+-----------+-----------+
|  1 | enric@mycompany.com | Enric      | Roca      | password1 |
+----+---------------------+------------+-----------+-----------+
1 row in set (0.00 sec)

mysql>
```

Now select Edit in the Uer List page in the browser to edit the user Enric.



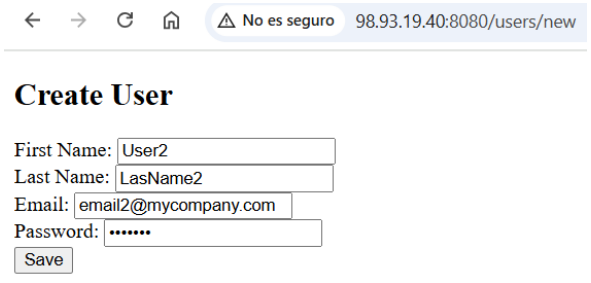
Write a different password, for instance password2, and press Save.



In the database we see that the password has changed:

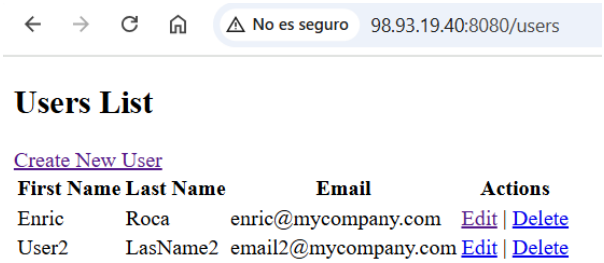
```
mysql> SELECT * FROM users;
+----+---------------------+------------+-----------+-----------+
| id | email               | first_name | last_name | password  |
+----+---------------------+------------+-----------+-----------+
|  1 | enric@mycompany.com | Enric      | Roca      | password2 |
+----+---------------------+------------+-----------+-----------+
1 row in set (0.00 sec)
```

Now, click on Create New User and introduce, for example:

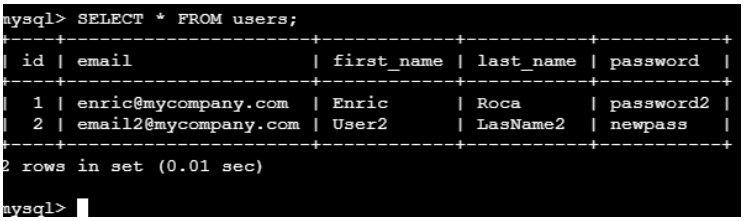


Password: newpass

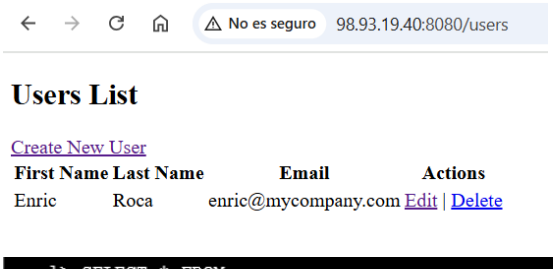
Press **Save**.

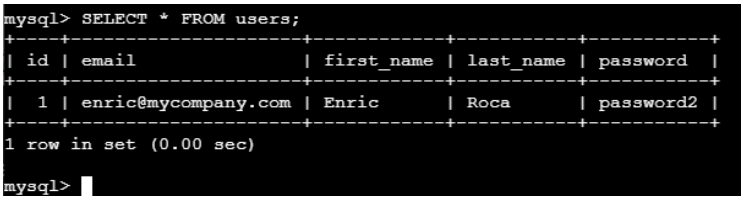


In the DB:



Now click on **Delete** to delete User2.





So, our application works fine. The four functions of CRUD (**Create**, **Read**, **Update**, and **Delete)** persistent storage are implemented without errors.

V.2- Check Minikube Dashboard

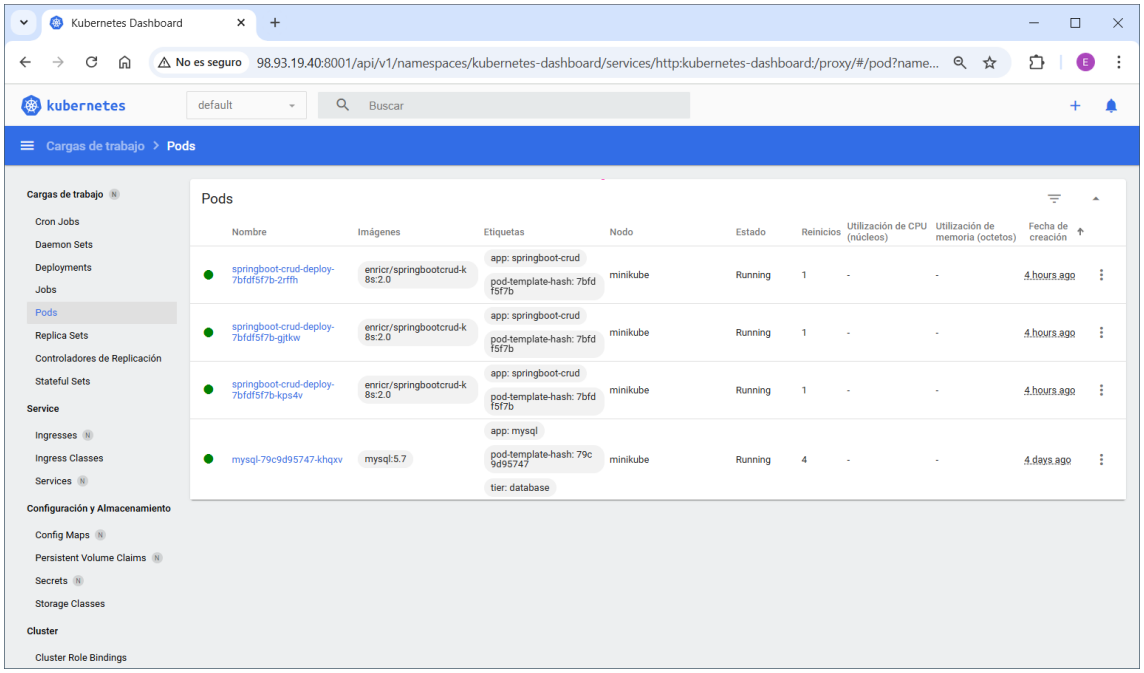
Remember that to use Minikube Dashboard we must first run a command to create an HTTP proxy server. If it's not currently running, execute the following:

kubectl proxy --address='0.0.0.0' --accept-hosts='^*$'

and leave this terminal listening.

Now, we can access with a browser from our laptop using the next command. Be in mind that now the public IP address of our EC2 instance is 98.93.19.40 and not 54.91.119.68.

http://**98.93.19.40**:8001/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/



We can select the different options on the left-hand menu to view all the resources we've created which are in the **default** namespace. We can also view resources from all namespaces using the drop-down list at the top.

V.3- Summary and Conclusions

This project was developed as a comprehensive tutorial, designed to allow users to practice the employed technologies and tools by following the detailed steps. All the necessary code is included within this documentation, allowing for easy copy-pasting, and is also available in the GitHub repository at https://github.com/enricrocaab/springbootcrud-k8s.

During development, we encountered and resolved various challenges, requiring careful error analysis to determine the root cause and apply appropriate solutions in each instance.

In Part I, we focused on local development, using **Visual Studio Code** for the **Java** application, **Docker Desktop** to run a local **MySQL** database container for development, and **Maven** to build the executable JAR file. We then created the necessary **Dockerfile** and **Kubernetes** YAML deployment files for both the database and the application (as detailed in Part II), which were subsequently pushed to a new **GitHub** repository.

The next phase (Part III) involved setting up an **EC2** instance on **AWS** and installing essential tools: **Docker**, **Minikube**, **kubectl**, **Git**, and **Maven**.

In Part IV, with the instance prepared, we cloned the GitHub repository. To correctly simulate a Continuous Integration/Continuous Deployment (**CI/CD**) **pipeline server**, we rebuilt the application code directly on the instance using Maven to generate the executable JAR. This JAR was then used with the prepared Dockerfile to create the **Docker image**, which was uploaded to **Docker Hub**.

We then applied the Kubernetes YAML deployment files for the MySQL and application components to **Minikube**, a step that involved significant troubleshooting.

For example, the MySQL pod initially entered a **CrashLoopBackOff** status because the root user was mistakenly treated as a standard user. Once that was corrected, further issues arose with the encoded password in the YAML file, which were also successfully resolved.

The application pods presented their own set of problems:

- Encoded Password: The encoded password issue needed correction here as well.
- Insufficient Disk Space: The App pods initially showed an **ImagePullBackOff** status due to a lack of free disk space on the instance. This required increasing the **EBS volume** storage resource on AWS and extending the instance partition.
- Incorrect Java Version: Once the storage issue was fixed, the pods defaulted to a **CrashLoopBackOff** status caused by an incompatible Java version within the Docker image.
- Connectivity Issues: Finally, after resolving connectivity issues, the web application was successfully deployed and functioned as intended.

In conclusion, completing all parts of this project served as an excellent practical exercise, providing hands-on experience with key DevOps technologies and tools.

Possible improvements could include migrating from Minikube to a multi-node Kubernetes cluster, implementing a formal CI/CD pipeline using tools like Jenkins, or adding DevSecOps functionalities. DevSecOps integration could involve tools such as Trivy for scanning dependencies and artifacts for known vulnerabilities, or SonarQube for identifying issues within the source code itself.

I hope this project proves valuable to many people. For me, it served as a rewarding exercise in practicing and reviewing numerous concepts and tools.

Acknowledgements:

I extend my gratitude to **Umar Rahman** for his insightful YouTube tutorial, "Spring Boot CRUD Operations with MySQL in VS Code," which I utilized for the initial development of this application.

Special thanks are due to **Praveen Singampalli** for his clear examples, explanations, and invaluable support in resolving complex DevOps challenges.