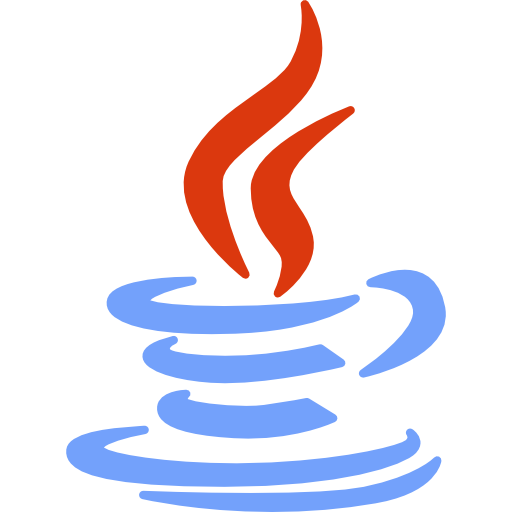
**ĐÀO TẠO FLUTTER**





**MỤC LỤC**

[I. Tạo dự án Spring Boot đầu tiên 3](#_Toc210564433)

[I.1. Chuẩn bị 3](#_Toc210564434)

[I.2. Spring initializr 3](#_Toc210564435)

[I.3. Hoàn tất 3](#_Toc210564436)

[II. JAVA CORE 4](#_Toc210564437)

[II.1. Dependency injection áp dụng vào Spring Boot 4](#_Toc210564438)

[II.2. Spring Bean 5](#_Toc210564439)

[II.2.1.1. Cách Tạo Bean 5](#_Toc210564440)

[II.2.1.2. Component Scan: 6](#_Toc210564441)

[II.2.1.3. Lifecycle của Bean 6](#_Toc210564442)

[Cơ chế Lazy: Bean sẽ chỉ được khởi tạo khi nó thực sự được sử dụng, thay vì tạo ra ngay khi container Spring start. 7](#_Toc210564443)

[II.3. luồng chạy của ứng dụng 7](#_Toc210564444)

[III. BÀI CODE MẪU 8](#_Toc210564445)

[III.1. JAVA SPRING BOOT + MYSQL + CRUD 8](#_Toc210564446)

[III.2. JAVA SPRING BOOT + REDIS 8](#_Toc210564447)

[III.3. JAVA SPRING BOOT + KAFKA 9](#_Toc210564448)

[III.4. JAVA SPRING BOOT + MINIO 10](#_Toc210564449)

[III.4.1. CẤU HÌNH 10](#_Toc210564450)

[2. MinIO Console (UI Web) 11](#_Toc210564451)

[III.5. JAVA SPRING BOOT + ELASTIC 11](#_Toc210564452)

[III.5.1. CẤU HÌNH 11](#_Toc210564453)

# Tạo dự án Spring Boot đầu tiên

## Chuẩn bị

Đối với Spring Boot một trong hai IDE là **Eclipse** (miễn phí) và ***IntelliJ IDEA*** Ultimate (bản Community không có hỗ trợ Spring). Tải xuống tại đây:

A screen shot of a phone

AI-generated content may be incorrect. IntelliJ IDEA: <https://www.jetbrains.com/idea/download/>

## Spring initializr

Spring Initializr có thể truy cập trên web tại <http://start.spring.io/>, hoặc với IntelliJ thì có tích hợp luôn vào khi tạo project luôn.

A screenshot of a computer

AI-generated content may be incorrect.

## Hoàn tất

Sau khi xong, các bạn nhấn nút Generate là xong. Một file zip chứa source ban đầu sẽ được tải về, chỉ cần giải nén và bắt đầu code.

Cấu trúc project được khởi tạo sẵn như sau.

A screenshot of a computer program

AI-generated content may be incorrect.

# JAVA CORE

## Dependency injection áp dụng vào Spring Boot

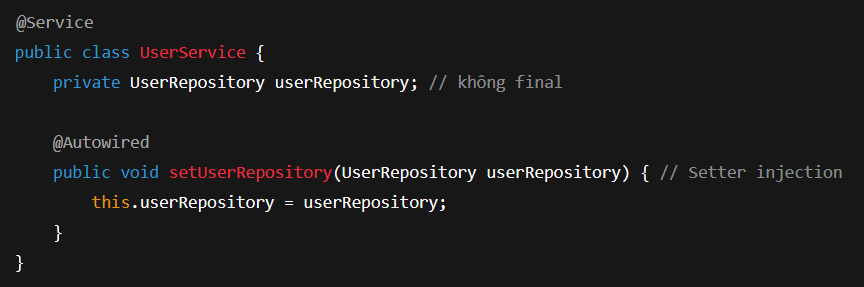
Có 3 loại chính:

* **Constructor-based injection:** Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field.
* **Setter-based injection:** Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.
* **@Autowired**: Sử dụng annotation @Autowired

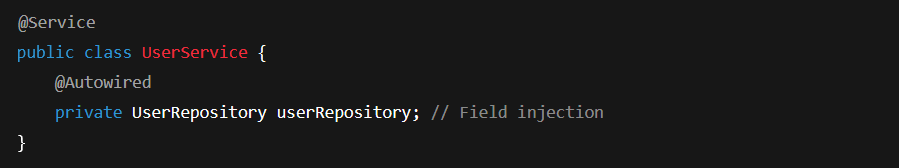
1. Constructor-based injection (tiêm qua constructor)



1. Setter-based injection (tiêm qua setter)



3. Field injection @Autowired



## Spring Bean

**Spring Bean** là một **đối tượng Java** do **Spring Container** quản lý vòng đời và các dependency của nó.

🡺 Những object mà được tạo và quản lý bởi **Spring Container** thì được gọi là Bean

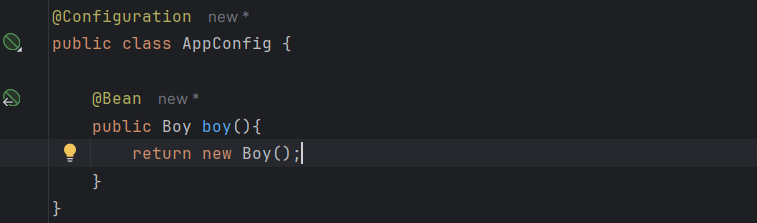
### Cách Tạo Bean

1. Dùng @Component, @Repository, @Service, @Controller
2. Dùng @Bean trong class có annotation @Configuration
   * **@Configuration**:
     + Là một Annotation đánh dấu trên một class, cho biết rằng lớp đó chứa các thông tin cấu hình cho ứng dụng.
     + Spring Boot sẽ tìm và quét các class được đánh dấu @Configuration để tạo và quản lý các beans.
   * **@Bean**:
     + Là một Annotation đánh dấu trên một method trong class được đánh dấu @Configuration.
     + Nó cho Spring Boot biết rằng method đó trả về một bean, và Spring Boot nên quản lý bean đó trong ứng dụng.

Ví dụ 1: @Component



* + **@Bean**:



***Ghi chú****: tại sao đã có cách 1 @Component mà lại xuất hiện cách 2 @Bean, vì 1 số trường hợp không phải lúc nào cũng dùng được @Component ví dụ ta muốn sử dụng 1 só thư viện bên ngoài ta không thể thực hiện @ Component đư*ợc , lúc đó dùng *@Bean*

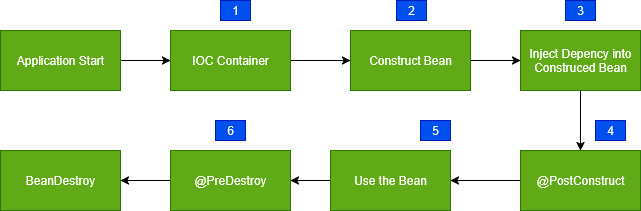
### Component Scan:

Khi ứng dụng Spring Boot bắt đầu chạy, thì nó sẽ tìm hết các class đánh dấu là bean trong chương trình và tạo bean. Quá trình tìm kiếm các bean này gọi là component scan.

@SpringBootApplication  
@ComponentScan(basePackages = {"com.example.demo", "com.example.demo.Orders"})  
public class DemoApplication {  
 public static void main(String[] args) {

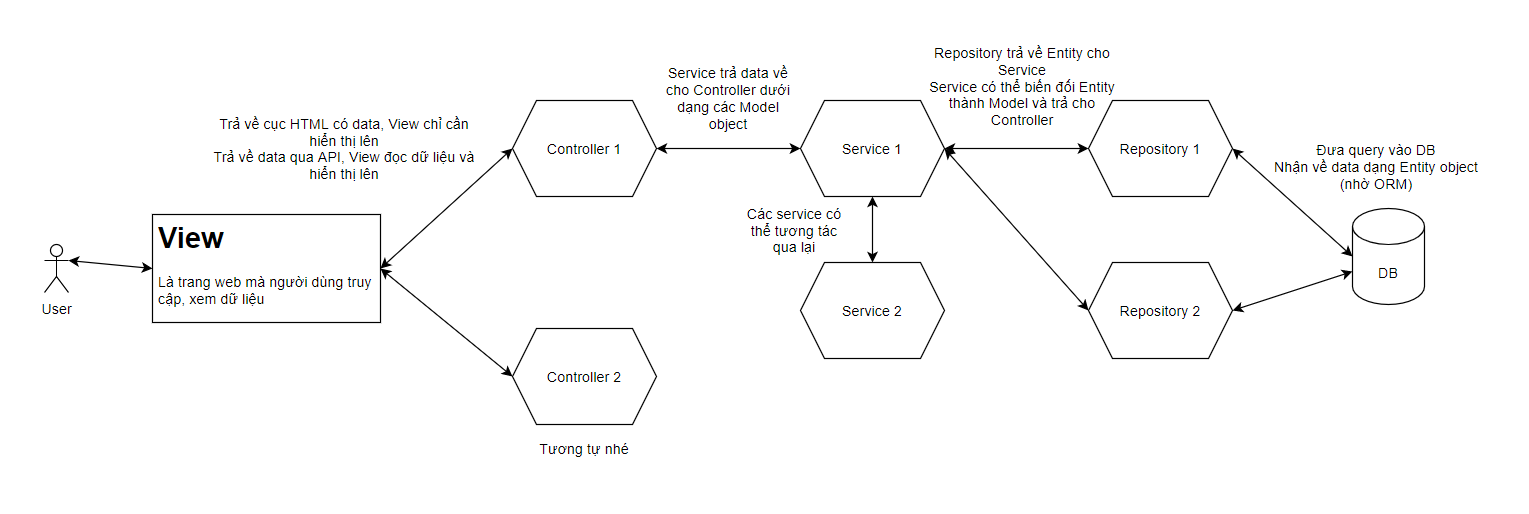
}

### Lifecycle của Bean



# Cơ chế Lazy: **Bean sẽ chỉ được khởi tạo khi nó thực sự được sử dụng**, thay vì tạo ra ngay khi container Spring start.

## luồng chạy của ứng dụng



# BÀI CODE MẪU

## JAVA SPRING BOOT + MYSQL + CRUD

**Cấu hình java**

* + **Framework chính: Spring Boot 3.5.6**
  + **Ngôn ngữ: Java 23**
  + **Build tool: Gradle Kotlin DSL**
  + **Server Port: 8085**

🐳 **Thông tin Docker Services**

*docker-compose up –build*

* + **Service name: mysql**
  + **Port: 3307 (host)**
  + **Database : demo\_db**
  + **User: demo**
  + **Password: demo123**
  + **Root: 123456**

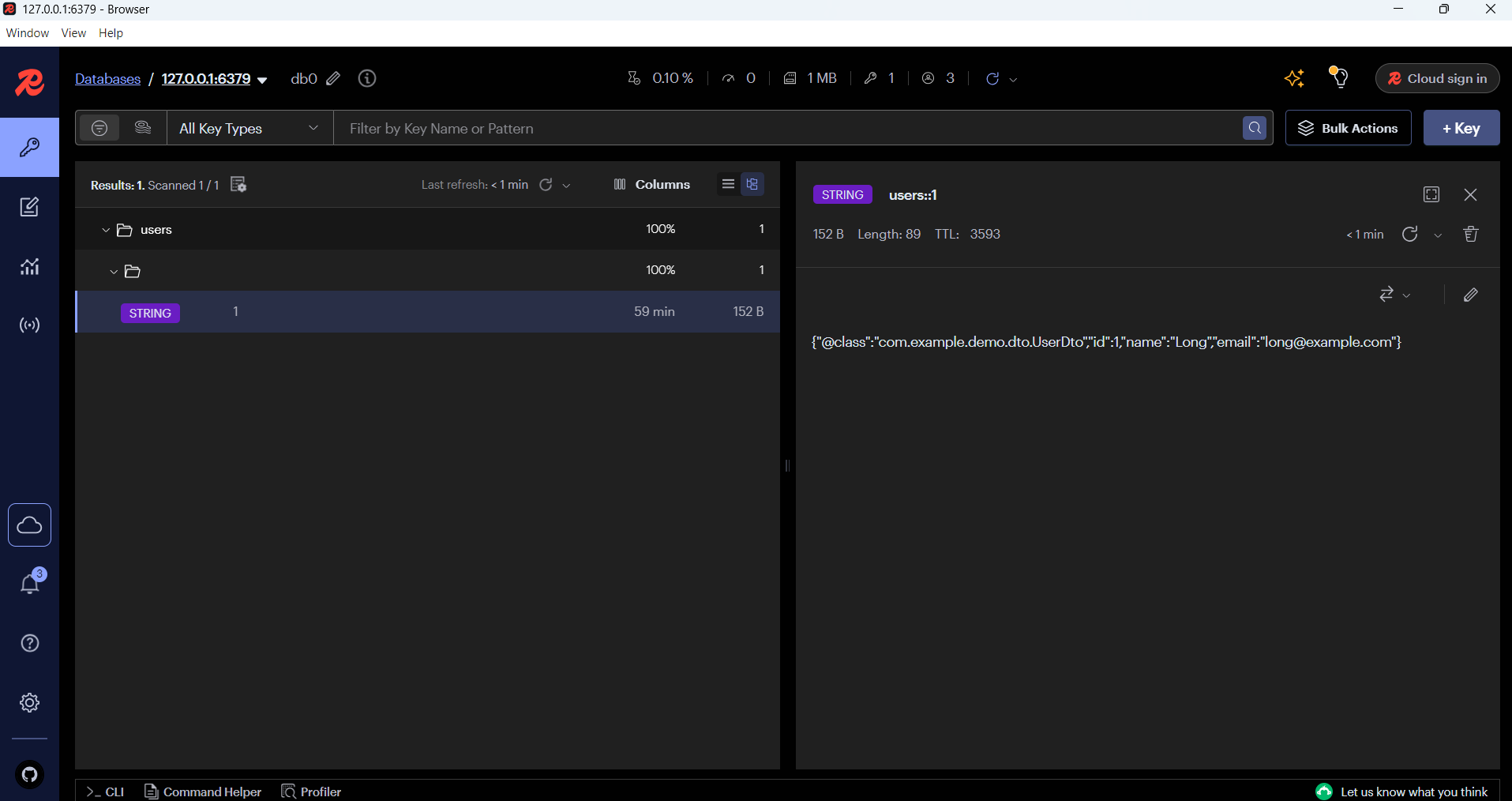
| **Method** | **URL** | **Body JSON** | **Mô tả** |
| --- | --- | --- | --- |
| GET | http://localhost:8085/api/users | - | Lấy danh sách user |
| POST | http://localhost:8085/api/users | { "name": "Long", "email": "long@example.com" } | Tạo user (gửi Kafka message) |
| PUT | http://localhost:8085/api/users/{id} | { "name": "Long (edit)", "email": "minh2@example.com" } | Cập nhật user |
| DELETE | http://localhost:8085/api/users/{id} | - | Xóa user |

## JAVA SPRING BOOT + REDIS

**Cấu hình java**

* + **Framework chính: Spring Boot 3.5.6**
  + **Ngôn ngữ: Java 23**
  + **Build tool: Gradle Kotlin DSL**
  + **Server Port: 8084**

🐳 **Thông tin Docker Services**



**Postman request:**

| **Method** | **URL** | **Body (JSON) ví dụ** | **Mục đích** |
| --- | --- | --- | --- |
| GET | http://localhost:8085/api/users | - | Lấy danh sách tất cả User |
| POST | http://localhost:8085/api/users | { "name": "long", "email": "long@example.com" } | Tạo User mới |
| PUT | http://localhost:8085/api/users/{id} | { "name": "long Updated", "email": "lonf@example.com" } | Cập nhật User theo id |
| DELETE | http://localhost:8085/api/users/{id} | - | Xóa User theo id |

## JAVA SPRING BOOT + KAFKA

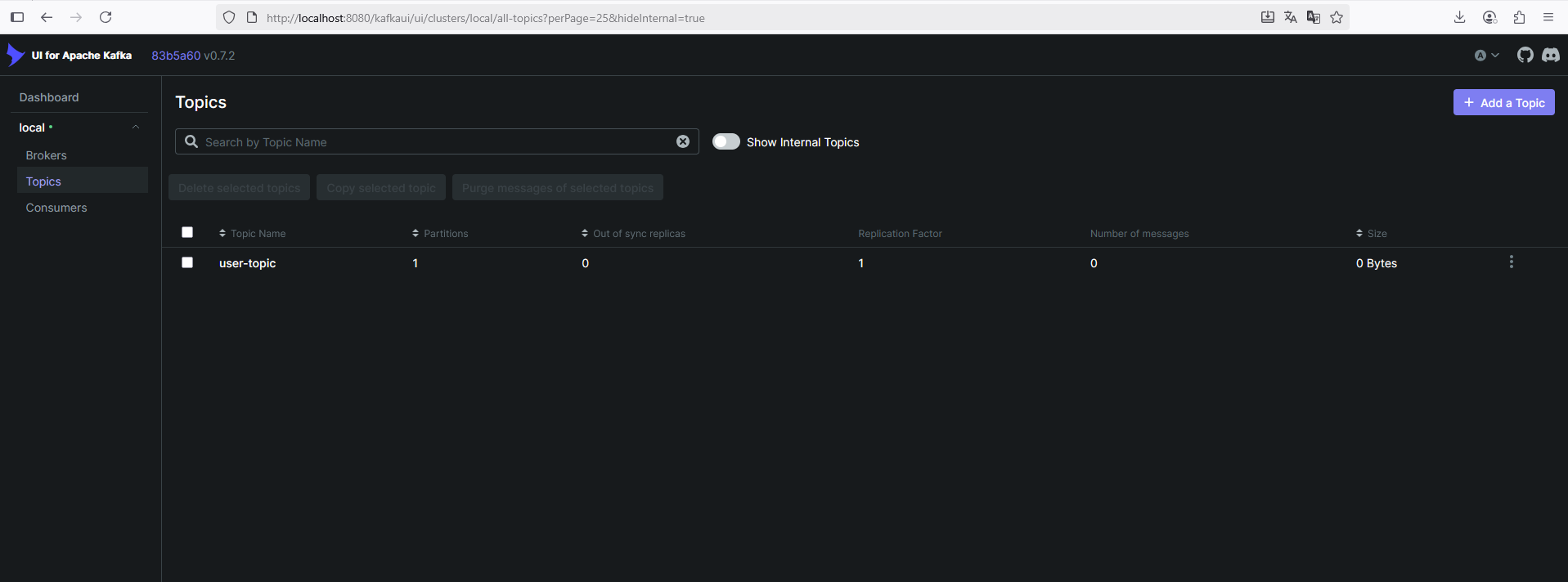
**Cấu hình**

* **Framework chính: Spring Boot 3.5.6**
* **Ngôn ngữ: Java 23**
* **Build tool: Gradle Kotlin DSL**
* **Server Port: 8085**

🐳 **Thông tin Docker Services**

📌 Kết nối cho Dev

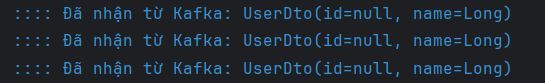
| **Component** | **Host** | **Port** | **Connection String** |
| --- | --- | --- | --- |
| Zookeeper | localhost | 2181 | localhost:2181 |
| Kafka Broker | localhost | 9092 | PLAINTEXT://localhost:9092 |
| Kafka UI (Console) | localhost | 8080 | http://localhost:8080/kafkaui |



**Postman request:**

| **Method** | **URL** | **Mô tả** |
| --- | --- | --- |
| POST | http://localhost:8085/api/users | Tạo User, send data lên kafka |

**Kết quả:**



## JAVA SPRING BOOT + MINIO

### CẤU HÌNH

* **Framework chính**: Spring Boot 3.5.6
* **Ngôn ngữ**: Java 23
* **Build tool**: Gradle Kotlin DSL
* **Server Port**: 8087

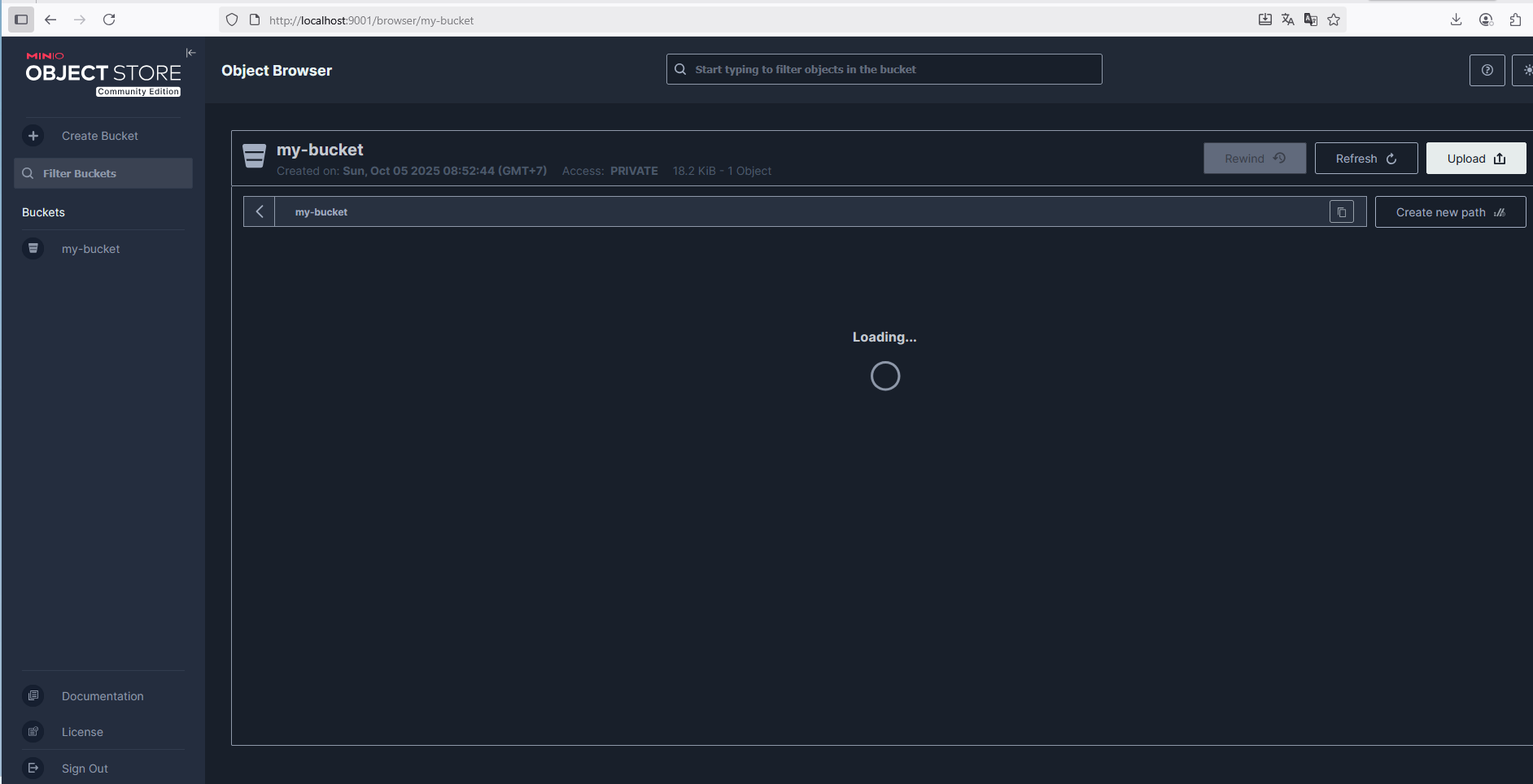
🐳 **Thông tin Docker Services**

### 2. ****MinIO Console (UI Web)****

* URL: <http://localhost:9001>

Login bằng:

* **Username**: minioadmin
* **Password**: minioadmin



**Postman request:**

| **Chức năng** | **Method** | **URL** | **Body (Postman)** | **Kết quả** |
| --- | --- | --- | --- | --- |
| Upload file | POST | http://localhost:8087/files/upload | - Body: **form-data**Key: file (type = File)Value: chọn file từ máy tính | File uploaded: <tên\_file> |
| Download file | GET | http://localhost:8087/files/download/{fileName} | Không có body | Tải về file (Postman → Save Response) |

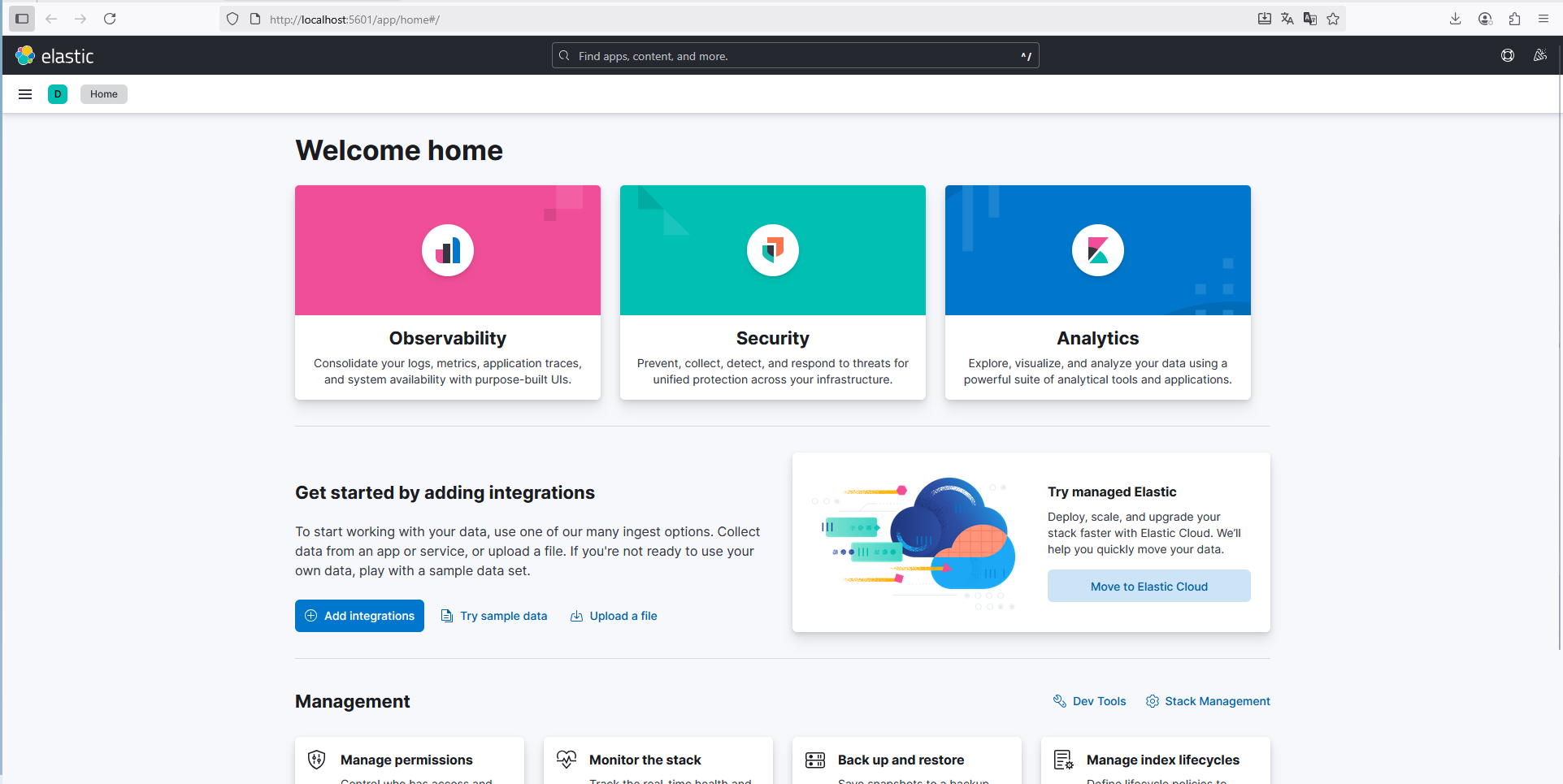
## JAVA SPRING BOOT + ELASTIC

### CẤU HÌNH

* **Framework chính**: Spring Boot 3.5.6
* **Ngôn ngữ**: Java 23
* **Build tool**: Gradle Kotlin DSL
* **Server Port**: 8086

🐳 **Thông tin Docker Services**

* *docker-compose up –build*
* *Giao diện web Kibana:* 
  + [*http://localhost:5601*](http://localhost:5601)
  + *http://localhost:5601/app/management*



**Postman request:**

| **Method** | **URL** | **Mô tả** |
| --- | --- | --- |
| POST | /api/users | Tạo User |
| GET | /api/users | Lấy tất cả Users |
| GET | /api/users/{id} | Lấy User theo ID |
| PUT | /api/users/{id} | Cập nhật User |
| DELETE | /api/users/{id} | Xóa User |
| GET | /api/users/search?email=abc@xyz.com | Tìm User theo Email |