**Presentation Script Léa-Line – Scenario + pros and cons**

**Slide 10 - Common Use Cases of Spring Boot**

There are several **typical use cases** where Spring Boot provides real value.

First, it’s heavily used for **microservice architectures**, where each service is a small, independent Spring Boot application exposing a REST API.  
Second, it’s ideal for **web and backend APIs**, for example when building a server that handles requests from a mobile or front-end client.  
Third, Spring Boot is commonly used in **cloud-native environments**, because it runs easily inside Docker containers and supports deployment on cloud platforms like AWS or Azure.  
Another frequent use case is **event-driven or reactive systems**, where Spring Boot works with technologies such as Kafka or RabbitMQ for asynchronous communication.  
Finally, many companies still rely on Spring Boot for **enterprise applications**, such as batch processing, data pipelines, or internal integration systems.

In short, Spring Boot provides a versatile foundation that adapts to both lightweight microservices and larger enterprise-scale applications.

<https://spring.io/projects/spring-boot#overview>

<https://docs.spring.io/spring-boot/index.html>

<https://en.wikipedia.org/wiki/Spring_Boot>

<https://www.researchgate.net/publication/362747012_An_Analysis_of_the_Significance_of_Spring_Boot_in_The_Market>

**Slide 11 - Example: Order Service for an E-Commerce Platform**

To make this more concrete, imagine a simple online shop where customers place orders.  
Behind the scenes, the system is built from several small services — for example, one for products, one for users, and another for orders.

Let’s focus on the **Order Service**.  
It receives each order request, checks that the data is valid, verifies that the product is in stock, and saves the order in a database.

With **Spring Boot**, this service is quick to build:  
we can create REST endpoints using @RestController, connect easily to the database using Spring Data, and monitor the application through **Spring Boot Actuator**.

When it’s ready, we simply package it as an executable JAR file and run it directly — even inside a **Docker container**.  
This makes the application lightweight, portable, and easy to deploy as part of a larger e-commerce system.

Same ressources

**Slide 12 - Advantages of Using Spring Boot**

Spring Boot is widely appreciated for the way it simplifies Java development and accelerates productivity.

First, it allows **rapid development**. Thanks to its auto-configuration and starter dependencies, developers can launch a project with minimal setup and focus directly on business logic.

Second, Spring Boot applications are **standalone** — they include an **embedded server**, such as Tomcat, which means there’s no need to deploy to an external application server.

Third, it integrates seamlessly with the **Spring ecosystem**, like Spring Data, Spring Security, or Spring Cloud, providing a unified and consistent experience.

It also comes **production-ready**, with features like monitoring, metrics, and health checks through Spring Boot Actuator.

Finally, it benefits from a **large, active community** and excellent documentation, which makes problem-solving and learning much easier.

Altogether, these features make Spring Boot a powerful and developer-friendly framework for building modern, cloud-ready applications.

<https://docs.spring.io/spring-boot/reference/features/index.html>

<https://spring.io/projects/spring-boot#overview>

<https://www.researchgate.net/publication/362747012_An_Analysis_of_the_Significance_of_Spring_Boot_in_The_Market>

<https://en.wikipedia.org/wiki/Spring_Boot>

**Slide 13 - Limitations of Spring Boot**

Although Spring Boot offers many advantages, there are also some limitations and challenges to keep in mind.

First, **applications can be heavier** than minimal frameworks. Because Spring Boot includes an embedded server and many dependencies, startup time and memory usage may be higher.

Second, the framework relies on a lot of **auto-configuration and hidden behaviour**. While this saves time, it can make debugging or customization more difficult, since much happens behind the scenes.

Third, the use of **starter packages** can lead to dependency bloat — extra libraries being included even if they’re not actually used.

Another challenge appears **when scaling large systems**. As multiple modules and features are added — such as caching, security, and asynchronous processing — configuration can become complex.

Finally, Spring Boot may not always be the best choice for **very small or resource-constrained services**, where its overhead is not justified.

In short, understanding these trade-offs helps developers use Spring Boot effectively and decide when its power and convenience outweigh its cost.

Same ressources