

Circular Dependencies in Spring Boot — and Why @Lazy is NOT the Real Fix

🔄 What is a Circular Dependency?
 In Spring Boot, a circular dependency occurs when two or more beans depend on each other for initialization.

⚠️ Why Not @Lazy?
Many developers love solving this quickly by using @Lazy to break the initialization cycle:
👉 This works, but it’s just a band-aid solution.
👉It hides the real design flaw in your code.
👉Makes debugging harder in large systems.
👉Can lead to runtime surprises when the lazy bean is accessed later.

💡 Better Approaches
>Instead of patching with @Lazy, fix the design issue:
>Refactor Dependencies → Extract common functionality into a third bean that both classes can depend on.
>Event-Driven Communication → Use Spring Events, Kafka, or RabbitMQ to decouple services.
>For example, instead of OrderService directly calling PaymentService, publish an event and let PayementService consume it asynchronously.
>This eliminates direct bean-to-bean dependency while improving scalability.

But personally, I prefer using asynchronous communication (Kafka or RabbitMQ) instead of relying on @Lazy.
It keeps the architecture clean, scalable, and easier to maintain.