1. Goal

The goal of this lesson is to introduce you to the context and reasons behind the need for reactive programming.

2. Lesson Notes

We've always had the need to handle large amounts of IO efficiently.

And, with microservices today, we have a lot more data is moving over the wire (as opposed to intra-process).

At a very high level - this is the problem we're trying to solve

2.1. Blocking IO / Async IO

In the Servlet model, we use thread-per-request model - for handling incoming requests.

And so, the thread is blocked until the response is ready.

The networking IO layer in the OS is asynchronous. It's only in the JDK we pretend that's not the case, and we're using simplified abstractions (InputStream, OutputStream).

This is the problem we're trying to solve.

2.2. Non-Blocking

Simply put, non-blocking means - the thread is never blocked - for which we need an event driven architecture.

The solution in the industry is **the reactive streams specification**.

The core aspects of that solution are: publisher and subscriber/consumer.

With the highly interesting ability that the consumer can signal to the producer - how much data it can handle - **backpressure**.

2.3. Reactive Programming

This this spec allows us to talk about data along these lines, at a low level.

We need higher-level abstractions.

And that’s what the Spring 5 reactive support is - building on top of the spec.

Before the Spring 5 implementation, we had the Reactive Extensions for .Net (Microsoft), RxJava (Netflix), Akka, VertX and Project Reactor, based on the reactive streams specification.

2.4. Reactive in Spring 5

Spring 5 introduced a new runtime - Spring Webflux, parallel to the standard Servlet stack.

This can still be deployed in a Servlet 3.1 environment, but it's really mean to run in a non-blocking runtime such as Netty.

Spring WebFlux takes the Publisher from reactive streams - and extends it with Mono or Flux.

Simply put, a Mono is a publisher - that will only produce one value, and a Flux is a publisher - that will produce multiple values.

Within the new reactive paradigm - processing can start only when a subscriber is available, and that subscriber can then control the flow of events (back-pressure).

Finally, **Spring WebFlux offers two programming models:**

- Annotations-based programming model
- Functional programming model

We're going to be exploring both of these models in this module.

3. Resources

- Spring 5 on Baeldung