**SPRING WEBFLUX**

**What is Reactive Programming?**

If we have one million user transactions happening in our application. Next year, it is going to increase to 10 million, so we need to scale it. The traditional method of doing this is to add enough servers (horizontal scaling). Instead of doing horizontal scaling, what if we get an option to scale with the same servers? Reactive programming will help us to do that. Reactive programming is all about non-blocking applications that are asynchronous and event-driven, and it doesn't require a lot of threads to scale vertically rather than horizontally (through clustering).

**What is Reactive Streams?**

Java8 introduced Reactive Core, implements Reactive programming model and built on top of Reactive streams specification.

Reactive streams is a push model whereas Java8 Streams focuses on pulling.

Reactive Stream has 4 Java interfaces - Publisher, Subscriber, Subscription, Processor

Publisher will publish a stream of data items to the Subscribers that are registered with the Publisher.

Using an executor, the Publisher publishes the items to the Subscriber. Also, Publisher makes sure that the Subscriber method

invocations for each subscription is strictly ordered.

Subscriber consumes items only when requested. You can cancel the receiving process any time by using Subscription.

Subscription behaves as a message mediator between the Publisher and the Subscriber.

Processor represents a processing stage, which can include both Publisher and the Subscriber. Processor can initiate back pressure and cancel the subscription as well.

**What is Spring WebFlux?**

Spring 5 introduced a Reactive Web Framework called Spring WebFlux. Spring WebFlux helps to build reactive RESTFul APIs. It is a non-blocking web framework which uses Reactor to support Reactive Streams API. It is used in mobile applications and microservices. It provides best fix for scenarios with many clients and uneven workloads. Spring WebFlux internally uses Project Reactor and its publisher implementations – ***Flux and Mono.***

***Imperative Programming -***

REST calls are generally web requests with IO. For each incoming web request a new servlet thread is assigned. When this web request is processing it interacts with DB and corresponding thread waits for DB driver to respond with data. If we have limited size of a servlet thread pool, then new concurrent web requests are blocked till waiting threads (for DB operation) are released when DB driver responds with data.

1) A web request is processed by a servlet thread

2) A Servlet thread calls a DB operation and waits for DB operation to complete

***Reactive Programming -***

We need a better mechanism to serve multiple concurrent requests and thus improve scalability of an application.

1) A web request is processed by a servlet thread

2) A Servlet thread calls a DB operation

3) A Servlet thread do not wait for DB operation to complete. Instead, it tells DB driver to call back once ready with data

4) A Servlet thread is free to serve other incoming requests

5) Db driver once ready with data, calls back (event driven mechanism) thread

6) Thread sends response back to client

With this approach, there are no thread switches, more efficient use of CPU and thus improves scalability. One disadvantage of this approach is it increases complexity due to request/response/IO event handling.

This complexity is handled by Spring WebFlux. Spring WebFlux is a framework which handles end to end reactive and event loop based mechanism so that we can use less number of threads, more efficient use of CPU and thus improve scalability.

**What programming models are supported by Spring WebFlux?**

The Spring WebFlux framework supports two programming models:

1) Annotation-based reactive components - Use RestController and WebClient to publish and consume reactive streams respectively.

2) Functional routing and handling - Use RouterFunction and HandlerFunction to route and handle requests respectively. WebClient can be used for consumption.

WebFlux framework also supports reactive WebSocket and corresponding WebSocketClient for socket style streaming of Reactive Streams.

**What is Mono and Flux in Spring WebFlux?**

Mono/Flux in Spring WebFlux corresponds to CompletableFuture in Java8. CompletableFuture allows asynchronous, non-blocking operations.

***Mono*** --> publisher focused on emitting 0 to 1 elements (Single value)

***Flux*** --> publisher focused on emitting 0 to n elements (List of values)

Flux is particularly used for hot/live sources of data. (Live List of values). Live stream of data ensures continuous data responses over an open connection between client and webserver. Whenever new data is inserted in DB, it is pushed to Spring Webflux. Spring webflux (Flux) allocates thread when data is available and pushes further to client over an open connection. With Flux, client do not have to continuously poll data but rather framework pushes data as and when it is available.

In short, for a non-blocking, asynchronous web request there are two components --

1) Spring WebFlux - Mono/Flux

2) Reactive DB driver repository

**How Mono/Flux works with web request?**

When working with responses, Mono/Flux allows asynchronous, non-blocking operations and returns Single value, List of values or live List of values. Similarly Mono/Flux works with web request too.

Instead of sending data back, we can get data from the user with asynchronous, non-blocking operations.

In a Post request with request-body, server needs to extract values from request-body. This IO operation also needs to be asynchronous and non-blocking.

Saving live data feed from client over an open connection is also possible using Flux. Spring Webflux (Flux) allocates thread when request data is available and pushes to reactive DB driver.

**When to use Spring WebFlux over typical Spring Boot ?**

Spring WebFlux is preferred over typical Spring Boot in cases below --

a) Scalability

1) Need better scalability when there are more IO bound operations (microservices)

2) When efficient CPU utilization is needed

3) Data locality and less context-switches

b) Streaming use cases (Live source of data)

c) Backpressure - If client or server is slow in processing the data then it can press back data source to slow down

**What are disadvantages of Spring WebFlux?**

1) Increase complexity

2) Different programming model - When data returned from DB needs further processing then we need to use API data processing functionality provided by Mono/Flux

3) End to end reactive required. If any point in request/IO/response flow, there is a blocking operation then it defeats the whole purpose.

4) Not too useful for CPU bound flows

5) Hard to debug

6) Hard to write test cases

**What is WebClient in Spring 5?**

WebClient is introduced as a new feature in Spring 5 along with Spring Webflux. A RESTful web service is created with Spring Webflux and it can be consumed by WebClient. The Spring MVC RestTemplate class is, by nature, blocking. Consequently, we don’t want to use it in a reactive application. For reactive applications, Spring offers the WebClient class, which is non-blocking. WebClient can be used to communicate with non-reactive, blocking services, too.

The WebClient uses reactive features, in the form of a Mono to hold the content of the URI in ClientResponse and a function (in the getResult method) to turn that content into a string. If we had different requirements, we might turn it into something other than a string.

**public class GreetingWebClient {**

**private WebClient client = WebClient.create("http://localhost:8080");**

**private Mono<ClientResponse> result = client.get()**

**.uri("/hello")**

**.accept(MediaType.TEXT\_PLAIN)**

**.exchange();**

**public String getResult() {**

**return ">> result = " + result.flatMap(res -> res.bodyToMono(String.class)).block();**

**}**

**}**

**What is spring-boot-starter-webflux dependency ?**

The spring-boot-starter-webflux dependency, which actually pulls in all other required dependencies:

1) spring-boot and spring-boot-starter for basic Spring Boot application setup

2) spring-webflux framework

3) reactor-core that we need for reactive streams and also reactor-netty