International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

Review on Spring Boot and Spring Webflux for Reactive Web Development

Rakshith Rao R[1], Dr S R Swamy[2]

[1]Student, Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, Karnataka, India
[2]Professor, Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, Karnataka, India

---***---

Abstract - *Web applications constitute a major share of the applications powering today's businesses and technology. With the increase in userbase, robust applications must be built to serve the requests. Asynchronous and distributed systems serve to solve this problem effectively.*

In this paper, we provide a detailed review of Spring Boot, reactive programming and how reactive systems can be built using Spring Boot and Spring Webflux. We also discuss the support for coroutines provided by Kotlin. The paper also reviews how effective systems can be built by combining imperative and reactive paradigms.

Key Words: Spring boot, Spring Webflux, Reactive programming, Kotlin coroutines

1. INTRODUCTION

Modern enterprise web applications handle millions of requests each second. This requires development of robust backend services which scale effectively. Developing the applications as monoliths becomes a bottleneck while scaling and extending the applications. Because of this the modern applications are built using microservice architecture which enables easy scaling. This saves a lot of cost while deployment of applications. The individual services should also serve the requests asynchronously in order to avoid long delays.

Spring Boot together with Spring Webflux provides very good solution for development of these kind of applications. Spring Webflux is based on the reactive programming model which helps asynchronous request processing. Language support such as coroutines provided by Kotlin further eases reactive programming and helps to create robust applications.

2. Spring Boot review

Spring Boot enables development of stand-alone Spring based applications with production quality which can directly be run using the embedded tomcat server. This avoids the creation of unnecessary WAR files for deployment. Spring Boot comes with many default configurations and provides an opiniated view of a Spring application. It also includes some default third party JARs common to all Spring applications.

Spring Framework is based on JVM languages and gives complete infrastructure support for writing effective web applications. Spring provides more time for the development of the application and reduces the configuration.

Spring boot applications can be easily created using Spring Initializr, which is a bootstrapping tool provided by the Pivotal team. The dependencies required by the application can be chosen while generating the project. Further Spring boot doesn't provide web.xml for configuring the application and avoids the overhead induced in managing XML files. However, it provides support for modifying the configuration of the applications using annotations. Adding @EnableAutoConfiguration annotation or @SpringBootApplication to the main class, automatically configures the applications based on JAR dependencies [2].

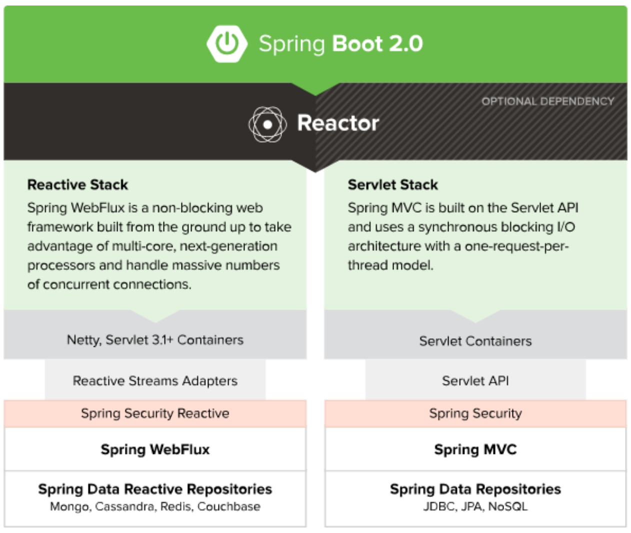
Spring Boot 2.0
Reactor
OPTIONAL DEPENDENCY
Reactive Stack
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.
Servlet Stack
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.
Netty, Servlet 3.1+ Containers
Servlet Containers
Reactive Streams Adapters
Servlet API
Spring Security Reactive
Spring Security
Spring WebFlux
Spring MVC
Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase
Spring Data Repositories
JDBC, JPA, NoSQL

Fig -1: Spring boot integration with Spring MVC and Spring Webflux [3]

Spring Boot provides an effective way to build microservices which implement a single business capability. Developing the whole application based on microservice architecture has many advantages such as,

1. High scalability
2. Loose coupling
3. Fault tolerance and fault isolation

4. Ability to use different technologies for different microservices.
5. Easier to understand a single microservice compared to the entire monolith.
6. Easier development and deployment of individual services.

Spring 5.0 framework supports both the core Spring MVC and Spring Webflux through Reactor core, thus enabling the use of Spring boot on top of Spring Webflux.

3. Spring Boot application architecture

Spring Boot has a layered architecture in which each layer communicates with the layer directly below and above it. The four layers in Spring Boot are as follows:

1. Presentation Layer: This layer processes the HTTP requests from client, converts JSON message body to objects and handles authentication of the request before transferring it to the business layer. It also handles presenting the views to the client.
2. Business Layer: This layer encompasses the business logic of the application. It performs the validation and authorization of the request and has service classes. These service classes then interact with the data access layer/ persistence layer.
3. Persistence Layer: This layer converts the objects in the application to database rows/objects using the storage logic.
4. Database Layer: This is the layer in which data required by the application is stored and on which CRUD (create, read, update, delete) operations are executed.

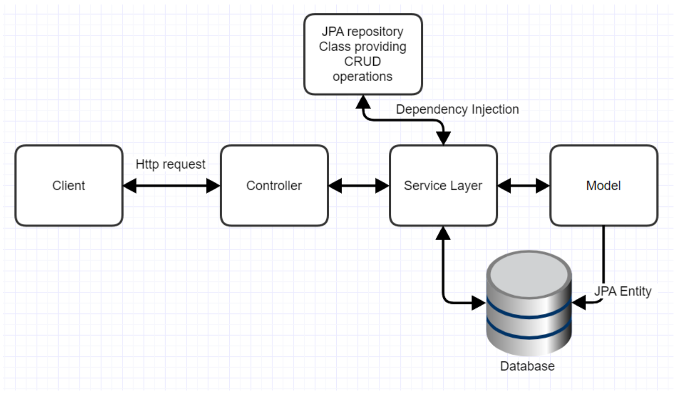

Fig -2: Architecture flow of spring boot Applications

Spring boot uses all the features of Spring like Spring MVC, Spring Data and JPA. A generic Spring Boot application consists of a controller which serves the clients HTTP request. This controller then interacts with the service layer which processes the request to modify the model and the database using the JPA repository through dependency injection. A view page is returned to the user if no error occurred. The architecture of a typical Spring boot application is presented in Fig 2.

Spring Framework 5 supports reactive application development by using Reactor internally and has tools for building reactive REST servers. The annotations can be used to implement the required configuration along with controller methods which handle HTTP requests with reactive components. It supports reactive streams for managing backpressure on asynchronous components. Spring 5 builds exposes APIs which can be implemented using libraries such RxJava or Reactor.

The webserver can be Tomcat, Jetty for Spring MVC or Netty for Webflux. Reactive streams are supported by Java in the form of java.util.concurrent.Flow.

4. Reactive programming

Reactive programming deals with developing non-blocking and event-driven applications built around asynchronous data streams [1]. This helps to scale them easily. Backpressure is an important concept of reactive systems which provides a mechanism so that producers will not overwhelm the consumers. This leads to developing the applications in a declarative style. Java 8 provides a similar solution called CompletableFuture to compose follow-up actions using lambdas.

In reactive programming, events, notification and HTTP requests are represented as data streams and applications are built to handle such streams making them asynchronous by nature. There are two types of data streams namely Hot and Cold streams.

Cold streams are lazy and start flowing only when a consumer demands them. They represent asynchronous actions which exhibit lazy execution upon need. Cold streams maybe asynchronous actions having lazy execution, lazy file download where data is not pulled until someone requires it. This stream is sent to the subscriber who demands it and not to all subscribers. Hot streams are active and are sent to all subscribers like a continuous stream when the data is generated. When a subscriber registers to the stream, it automatically receives the next measure. Understanding the nature of the stream is crucial to develop the systems consuming it.

Using reactive programming in certain components does not guarantee reactive systems. Each node must implement functions which are non-blocking and exhibit task-based concurrency. Reactive systems are essentially responsive distributed systems. Properties of a reactive system are,

1. A reactive application processes a request within a definite time period.
2. Reactive system components must interact with each other using asynchronous message passing.
3. Reactive systems must be resilient and must be responsive even in case of failures such as crash, timeout and 500 errors.

4. Reactive systems must be easily scalable to handle varying load with minimal resources.

Reactive programming helps creating robust and highly extendable applications. It makes the system more maintainable when processing data sent as streams in regular intervals.

5. Spring Webflux

Spring framework originally was developed to support Servlets through Spring Web MVC. Spring MVC uses servlet stack to process requests and produce response synchronously. Later Spring 5 introduced Spring Webflux for developing non-blocking asynchronous applications and to support reactive streams which run on Netty and Servlet 3 containers. These containers deliver high performance for reactive applications. Spring MVC and Spring Webflux are developed over their source modules and applications can be built using either or both. Such as having Web Clients implemented using Webflux and controllers through MVC.

Spring Webflux improves concurrency with small number of threads and minimal hardware because of asynchronous execution. It provides a common API using which applications can be run on any non-blocking runtime. Netty servers support concurrency through async and non-blocking functionalities.

Lambda expressions introduced in Java 8 enables use of functional APIs which are crucial for developing non-blocking and asynchronous applications. Spring Webflux makes excellent use of this feature to offer functional endpoints along with annotated controllers.

Another mechanism with reactive programming is non-blocking back pressure. In imperative programming synchronous calls act as back pressure and make the client wait for the response. Non-blocking programs require controlling the rate of events being delivered by producer so that the consumer is not overwhelmed. The consumer needs the control to alter rate at which he chooses to receive the data stream.

Spring Webflux uses Reactor as the library to achieve reactive programming by using types such as Mono and Flux which accommodate (0..1) and (0..N) data streams. Reactor supports non-blocking backpressure by default. Webflux provides HTTP abstractions, WebHandler API to support non-blocking contracts and adapters for reactive streams.

Spring WebFlux helps application development through:

1. Use of Annotated Controllers which are similar in their use in Spring MVC. Spring Webflux also supports the @ResuestBody parameters which are reactive. Reactive return types are supported by both Spring MVC and Spring Webflux.
2. Use of Functional Endpoints which is achieved through the development of applications using lambdas. Lambdas act as lightweight modules which help to transform or process the requests. This provides a declarative style of defining how to process requests as opposed to annotation-based controllers in which the application is in charge of request handling.

Both Spring MVC and Webflux can be used together to expand the range of options. Both these frameworks complement each other and provide solutions to the short comings from each side. Fig. 3 shows the similarities and distinction between these two frameworks.

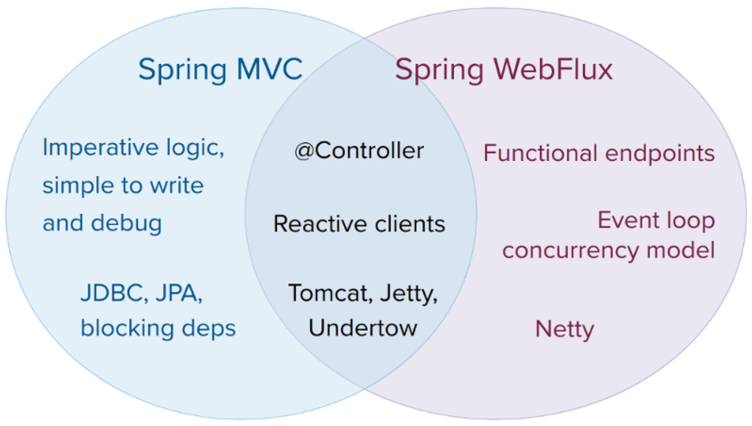

Fig -3: Spring MVC and Spring Webflux features and similarities

6. Kotlin for server-side application development and coroutines support

Kotlin serves as an excellent option to develop server-side applications. Kotlin provides a concise and expressive way to write code. It also allows the application being developed to seamlessly integrate with existing Java technologies. Some of the advantages using Kotlin are:

1. Kotlin is a highly concise and expressive language. It provides type safe builders and useful abstractions. Kotlin is very easy to learn for Java developers.
2. Kotlin is supported by many of the popular IDEs and some of them support for Spring framework and other frameworks.
3. Kotlin provides support for coroutines which helps developing highly scalable server-side applications.
4. Kotlin is completely interoperable with Java and Java-based frameworks. This provides the developer to use the framework of choice together with the benefits provided by Kotlin.

Spring 5.0 supports application development using Kotlin and makes effective use of Kotlin's features stated above to provide concise APIs. Kotlin provides Coroutine support which enables to develop asynchronous applications [4].

A coroutine is a simple light weight component which can be run concurrently with other program components. They can easily switch contexts and share the underlying hardware effectively.

Asynchronous applications are crucial to provide seamless user experience and achieve high scalability. Coroutines also provides additional features such as concurrency, actors and many more.

This makes Kotlin to seamlessly integrate with frameworks such Spring Webflux and develop reactive applications. The unnecessary verbose as in case of java can be completely avoided using Kotlin.

7. CONCLUSIONS

The increase in demand for reactive systems, applications must be developed to handle asynchronous requests and responses. However certain part of the systems can be developed efficiently only when they are synchronous. Spring framework 5.0 provides an excellent solution to use both imperative (Spring MVC) and reactive (Spring Webflux) paradigms in the same application.

Spring boot 2.0 provides many default configurations and makes the development of the application faster and easier. Kotlin provides inbuilt support for asynchronous programming using coroutines. Systems with reactive components can be developed effectively using these technologies.

This paper presents a view as to how the language support for coroutines presented by Kotlin, advantages of reactive programming using Webflux and ease of annotated controllers can be combined effectively to develop highly scalable and robust enterprise application.

REFERENCES

[1] K. Siva Prasad Reddy, "Reactive Programming Using Spring WebFlux", Beginning Spring Boot 2, pp. 159-132 DOI: 10.1007/978-1-4842-2931-6_12

[2] Joseph B. Ottinger, Andrew Lombardi, "Spring Boot", Beginning Spring 5, DOI: 10.1007/978-1-4842-4486-9_7

[3] https://spring.io/reactive

[4] https://kotlinlang.org/docs/reference/coroutines-overview.html