**Spring Boot Starter Project Options**

As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.

* spring-boot-starter-web-services: SOAP Web Services
* spring-boot-starter-web: Web and RESTful applications
* spring-boot-starter-test: Unit testing and Integration Testing
* spring-boot-starter-jdbc: Traditional JDBC
* spring-boot-starter-hateoas: Add HATEOAS features to your services
* spring-boot-starter-security: Authentication and Authorization using Spring Security
* spring-boot-starter-data-jpa: Spring Data JPA with Hibernate
* spring-boot-starter-cache: Enabling Spring Framework’s caching support
* spring-boot-starter-data-rest: Expose Simple REST Services using Spring Data REST

**Other Goals of Spring Boot**

There are a few starters for technical stuff as well

* spring-boot-starter-actuator: To use advanced features like monitoring and tracing to your application out of the box
* spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat: To pick your specific choice of Embedded Servlet Container
* spring-boot-starter-logging: For Logging using logback
* spring-boot-starter-log4j2: Logging using Log4j2

Spring Boot aims to enable production ready applications in quick time.

* Actuator: Enables Advanced Monitoring and Tracing of applications.
* Embedded Server Integrations: Since the server is integrated into the application, I would need to have a separate application server installed on the server.
* Default Error Handling

**Pros**

• Quick up and running of the standalone and container less Spring applications. You just assemble the jar artifact which comes with an embedded Tomcat, Jetty or Undertow application server and you are ready to go.

• Simplified dependency management through the starter POMs.

• Visibility into your application internals. Spring Boot provides the HTTP endpoints to get detailed metrics about the application inner working, health status, etc. SSH shell and the jtop are really awesome.

• For the Java purists and the haters of XML configuration files, the good news is there are no XML at all. The beans are initialized, configured and wired automatically.

• The Spring Initializer provides a project generator to make you productive with the certain technology stack from the beginning. You can create a skeleton project with web, data access (relational and NoSQL datastores), cloud, or messaging support.

**Cons**

• If you are new to Spring and want to learn how the dependency injection, AOP programming, and proxies work, starting with Spring Boot is not a good choice. Spring Boot hides the most of these details from you.

• I find the Java Config programming model is more intrusive with your application code base. If the IoC principle is about delegating the lifecycle and the instantiation of the classes to external entity (the container), then, the principle is being violated. It’s the programmer itself who is responsible for creating the required beans. Moreover, if the implementation of your class changes, you are forced to change it in your code. With XML oriented programming model it would be enough to update the bean definition. The introduction of Spring XML namespaces had greatly simplified the configuration files, so there is one less reason to hate them.

• If you are a control freak, I doubt Spring Boot would fit your needs.

• Spring Boot sticks good with microservices. The Spring Boot artifacts can be deployed directly into Docker containers. In a large and monolithic based applications, I wouldn’t encourage you to use Spring Boot.

• If you are not familiarized with other projects of the Spring ecosystem (Spring Integration, Spring AMQP, Spring Security, etc), using them from Spring Boot will make you miss a lot of concepts, which by the other hand you would acquire if you had started using them independently.