What is Spring Boot? And, what is a Spring Framework? What are their goals? How can we compare them? There must be a lot of questions running through your mind. At the end of this blog, you will have the answers to all of these questions. In learning more about the Spring and Spring Boot frameworks, you will come to understand that each solve a different type of problem.

What Is Spring? What Are the Core Problems Spring Solves?
The Spring Framework is one of the most popular application development frameworks for Java. One of the best features in Spring is that it has the Dependency Injection (DI) or Inversion Of Control (IOC), which allows us to develop loosely coupled applications. And, loosely coupled applications can be easily unit-tested.
Example Without Dependency Injection

Consider the example below — MyController depends on MyService to perform a certain task. So, to get the instance of MyService, we will use:

```
MyService service = new MyService();
```

Now, we have created the instance for MyService , and we see both are tightly coupled. If I create a mock for MyService in a unit test for MyController , how do I make MyController use the mock? It's bit difficult — isn't it?

```
@RestController
public class MyController {
   private MyService service = new MyService();

   @RequestMapping("/welcome")
   public String welcome() {
      return service.retrieveWelcomeMessage();
   }
}
```

Example With a Dependency Injection
With the help of only two annotations, we can get the instance of MyService easily, which is not tightly coupled. The Spring Framework does all the hard work to make things simpler.

- @Component is simply used in the Spring Framework as a bean that you need to manage within your own BeanFactory (an implementation of the Factory pattern).
- @Autowired is simply used to in the Spring Framework to find the correct match for this specific type and autowire it.

So, Spring framework will create a bean for MyService and autowire it into MyController.

In a unit test, I can ask the Spring Framework to auto-wire the mock of MyService into MyController .

```
@Component
public class MyService {
   public String retrieveWelcomeMessage(){
     return "Welcome to InnovationM";
   }
}

@RestController
public class MyController {
   @Autowired
```

```
    private MyService service;

    @RequestMapping("/welcome")
    public String welcome() {
        return service.retrieveWelcomeMessage();
    }
}
```

The Spring Framework has many other features, which are divided into twenty modules to solve many common problems. Here are some of the more popular modules:

- Spring JDBC
- Spring MVC
- Spring AOP
- Spring ORM
- Spring JMS
- Spring Test
- Spring Expression Language (SpEL)

Aspect Oriented Programming(AOP) is another strong side of the Spring Framework. The key unit in object-oriented programming is the class, whereas, in AOP, the key unit is the aspect. For example, if you want to add the security in your project, logging, etc., you can just use the AOP and keep these as a cross-cutting concern away from your main business logic. You can perform any action after a method call, before a method call, after a method returns, or after the exception arises.

The Spring Framework does not have its own ORM, but it provides a very good integration with ORM, like Hibernate, Apache iBATIS, etc.

In short, we can say that the Spring Framework provides a decoupled way of developing web applications. Web application development becomes easy with the help of these concepts in Spring, like Dispatcher Servlet, ModelAndView, and View Resolver.

If Spring Can Solve so Many Problems, Why Do We Need Spring Boot?

Now, if you have already worked on Spring, think about the problem that you faced while developing a full-fledged Spring application with all functionalities. Not able to come up with one? Let me tell you — there was lot of difficulty to setup Hibernate Datasource, Entity Manager, Session Factory, and Transaction Management. It takes a lot of time for a developer to set up a basic project using Spring MVC with minimum functionality.

```
 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix">
         <value>/WEB-INF/views/</value>
      </property>

      <property name="suffix">
         <value>.jsp</value>
      </property>
 </bean>

 <mvc:resources mapping="/webjars/**" location="/webjars/"/>

<servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
```

```
    </servlet-class>

    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/my-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

When we use Hibernate, we have to configure these things like a datasource, EntityManager, etc.

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
      <property name="driverClass" value="${db.driver}" />
      <property name="jdbcUrl" value="${db.url}" />
      <property name="user" value="${db.username}" />
      <property name="password" value="${db.password}" />
  </bean>

  <jdbc:initialize-database data-source="dataSource">
    <jdbc:script location="classpath:config/schema.sql" />
    <jdbc:script location="classpath:config/data.sql" />
  </jdbc:initialize-database>

    <bean class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean" id="entityManager-
Factory">
      <property name="persistenceUnitName" value="hsql_pu" />
      <property name="dataSource" ref="dataSource" />
  </bean>

  <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
    <property name="dataSource" ref="dataSource" />
  </bean>

  <tx:annotation-driven transaction-manager="transactionManager"/>
```

How Does Spring Boot Solve This Problem?

- Spring Boot does all of those using AutoConfiguration and will take care of all the internal dependencies that your application needs — all you need to do is run your application. Spring Boot will auto-configure with the Dispatcher Servlet, if Spring jar is in the class path. It will auto-configue to the datasource, if Hibernate jar is in the class path. Spring Boot gives us a pre-configured set of Starter Projects to be added as a dependency in our project.
- During web-application development, we would need the jars that we want to use, which versions of the jars to use, and how to connect them together. All web applications have similar needs, for example, Spring MVC, Jackson Databind, Hibernate core, and Log4j (for logging). So, we had to choose the compatible versions of all these jars. In order to decrease the complexity, Spring Boot has introduced what we call Spring Boot Starters.

Dependency for Spring Web Project

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.2.2.RELEASE</version>
</dependency>

<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.5.3</version>
</dependency>

<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>5.0.2.Final</version>
</dependency>

<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.17</version>
</dependency>
```

Starters are a set of convenient dependencies that you can include in your Spring Boot application. For using Spring and Hibernate, we just have to include the spring-boot-starter-data-jpa dependency in the project.
Dependency for Spring Boot Starter Web

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

There are other packages that you will see. Once you add that starter dependency, the Spring Boot Starter Web comes pre-packaged with all of these. As a developer, we would not need to worry about these dependencies and their compatible versions.

Spring Boot Starter Project Options
These are few starter projects to help us get started quickly 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