<http://www.dineshonjava.com/2016/06/introduction-to-spring-boot-a-spring-boot-complete-guide.html>

Spring Boot automatically configures required classes depending on the libraries on its classpath. Suppose your application want to interact with DB, if there are Spring Data libraries on class path then it automatically sets up connection to DB along with the Data Source class.

**Goals:**

1. no code generation and no requirement for XML configuration, to avoid XML Configuration completely
2. To avoid defining more Annotation Configuration
3. To reduce Development, Unit Test and Integration Test time by providing some defaults.

**Features:**

1. It is very easy to develop Spring Based applications with Java or Groovy.
2. It reduces lots of development time and increases productivity.
3. It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
4. It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
5. It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
6. It provides Embedded HTTP servers like **Tomcat**, **Jetty** etc. to develop and test our web applications very easily.
7. It provides CLI (**Command Line Interface**) tool to develop and test Spring Boot (Java or Groovy) Applications from command prompt very easily and quickly.
8. It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like **Maven** and **Gradle**

**Note:**

Spring Boot Framework Programming model is inspired by **Groovy** Programming model.

**POM**

<modelVersion>4.0.0</modelVersion>

<groupId>com.doj</groupId>

<artifactId>my-spring-boot-project</artifactId>

<version>0.0.1-SNAPSHOT </version>

<parent>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-starter-parent**</artifactId>

<version>1.3.5.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>**spring-boot-starter-web**</artifactId>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

<build>

<plugins>

<plugin>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-maven-plugin**</artifactId>

</plugin>

</plugins>

</build>

</project>

**Annotations:**

@Controller

@**EnableAutoConfiguration**

**Internals of Spring Boot:**

1. **Spring Boot Starters—**JPA/DAO/Web-MVC/databind etc
2. **Spring Boot AutoConfigurator—**forauto configuration
3. **Spring Boot CLI-** optional feature for Command line Impl
4. **Spring Boot Actuator-**what’s going on inside of a running Spring Boot application

**Note:**  properties from yaml or appl.prop file

@Component

@**ConfigurationProperties**(prefix="database")

public class DatabaseSettings {

private String dbname;

private String dburl;

// ... getters and setters

}

1. The**@EnableConfigurationProperties**annotation is automatically applied to your project so that any beans annotated with **@ConfigurationProperties** will be configured from the Environment properties. The **@ConfigurationProperties** annotation won’t work unless you’ve enabled it by adding **@EnableConfigurationProperties** in one of your Spring configuration classes. This style of configuration works particularly well with the **SpringApplication**external **YAML**configuration:  
   **application.yml**
2. It is also possible to shortcut the registration of**@ConfigurationProperties** bean definitions by simply listing the properties classes directly in the **@EnableConfigurationProperties** annotation:
3. @Configuration
4. @**EnableConfigurationProperties**(DatabaseSettings.**class**)
5. **public** **class** DBConfiguration {
6. }

**Profiling in spring Boot:**

1. @Configuration
2. @**Profile**("production")
3. **public** **class** ProductionConfiguration {
4. // ...
5. }

**In yaml file🡪application.yml**

1. spring:
2. profiles:
3. active: production

We can maintain specific to like ***application-{profile}.properties***

**Multi-profile YAML documents:**

We can specify multiple profile-specific YAML documents in a single file by using a ***spring.profiles*** key to indicate when the document applies. For example:

1. logging:
2. level:
3. root: INFO
4. ---
5. spring:
6. profiles: development
7. logging:
8. level:
9. root: DEBUG
10. ---
11. spring:
12. profiles: production
13. logging:
14. path: /opt/
15. file: MyApp.log
16. level:
17. root: WARN

Here **application.yml** file is divided into three sections by a set of triple hyphens (---)

**DB configuration:**

1. If you use the **spring-boot-starter-jdbc** or **spring-boot-starter-data-jpa** ‘**starters**’ you will automatically get a dependency to **tomcat-jdbc**.
2. We can configure the database properties in **application.properties** file so that **SpringBoot** will use those jdbc parameters to configure **DataSource**bean.
3. spring.datasource.driver-**class**-name=com.mysql.jdbc.Driver
4. spring.datasource.url=jdbc:mysql://localhost:3306/dojdb
5. spring.datasource.username=root
6. spring.datasource.password=root
7. for JNDI

spring.datasource.jndi-name=java:tomcat/datasources/users