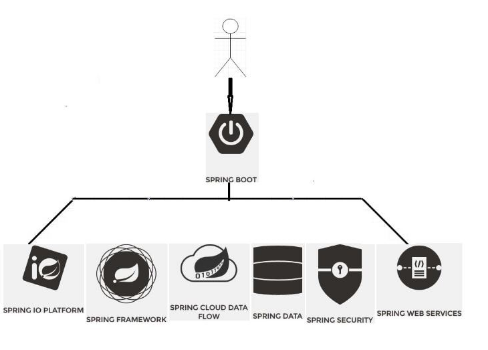
**Spring Boot**

If one has to start a new spring project we have to add build path or add maven dependencies, configure application server, add spring configuration . **So a lot of effort is required to start a new spring project as we have to currently do everything from scratch. Spring Boot is the solution to this problem**. Spring boot has been built on top of existing spring framework. Using spring boot we avoid all the boilerplate code and configurations that we had to do previously. Spring boot thus helps us use the existing Spring functionalities more robustly and with minimum efforts.



Features of Spring boot-

* **Auto-Configuration** - No need to manually configure dispatcher servlet, static resource mappings, property source loader, message converters etc.
* **Dependency Management** - The different versions of commonly used libraries are pre-selected and grouped in different starter POMs that we can include in your project. By selecting one Spring Boot version we are implicitly selecting dozens of dependencies that we would have to otherwise select and harmonize ourself. Example-
* **Advanced Externalized Configuration** - There is a large list of bean properties that can be configured through application.properties file without touching java or xml config.
* **Production support**- We get health checking, application and jvm metrics, jmx via http and a few more things for free.
* **Runnable Jars** - We can package your application as a runnable jar with embedded tomcat included so it presents a self-contained deployment unit.
* **InBuild**-**server:-** Spring Boot comes with inbuilt server, we no longer have to use any external servers like *Tomcat*, *Glass-fish* or anything else, so don’t need to deploy WAR files

So we can develop project using Spring boot in less time without any need of xml configuration but disadvantages of this Spring Boot is little tough to migrate the existing Spring enterprises application to Spring Boot application.

Spring boot provides the Command line Interface tools to develop and test the application.

What is Spring Boot?

Spring boot is abstraction layer on the Spring framework which provided the RAD(Rapid application development) feature on Spring framework. it is convention over the Configuration.

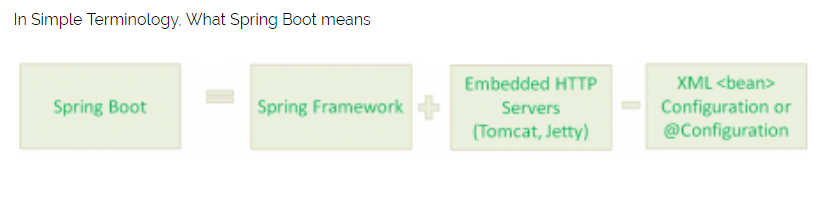
It provides the following.

a.)Basic configuration

b.)class path scanning

c.)Convention over convention.

Spring boot means



That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) – XML or Annotations Configurations.  
Here minus means we don’t need to write any XML Configuration and few Annotations only.

Main Goal of Spring Boot:

The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.

* To avoid XML Configuration completely
* To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)
* To avoid writing lots of import statements
* To provide some defaults to quick start new projects within no time.
* To provide Opinionated Development approach.

By providing or avoiding these things, Spring Boot Framework reduces Development time, Developer Effort and increases productivity.

We can create a Spring Boot Application by three ways

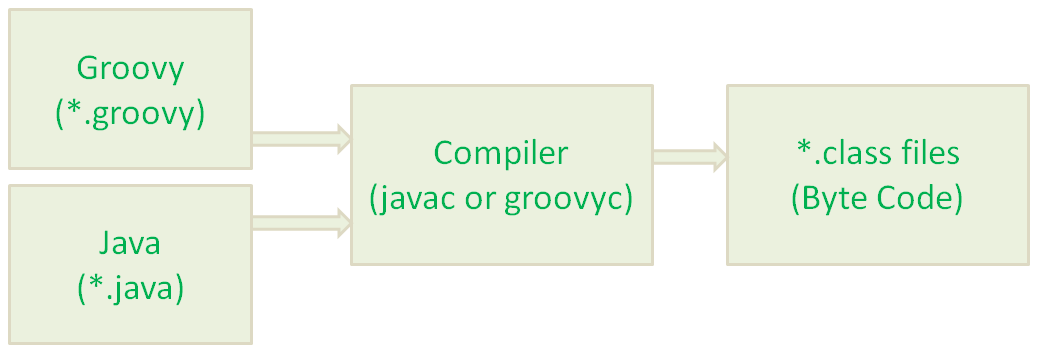
* Using Spring Boot CLI Tool
* Using Spring STS IDE
* Using Spring Initializr Website

We can find Spring Initializr Website at: http://start.spring.io/  
We can develop two flavors of Spring-Based Applications using Spring Boot

* Java-Based Applications
* Groovy Applications.

We can use Spring Boot CLI ,Spring Boot Initializer or Spring STS to develop Spring Boot based application using Groovy and we can use Spring Boot Initializer to develop Spring boot application using java.

Anyhow, Groovy is also JVM language almost similar to Java Language. We can combine both Groovy and Java into one Project. Because like Java files, Groovy files are finally compiled into \*.class files only. Both \*.groovy and \*.java files are converted to \*.class file (Same byte code format).



# Spring Boot Features

* Web Development
* SpringApplication
* Application events and listeners
* Admin features
* Externalized Configuration
* Properties Files
* YAML Support
* Type-safe Configuration
* Logging
* Security.

# Web Development

# It is well suited Spring module for web application development. We can easily create a self-contained HTTP server using embedded Tomcat, Jetty or Undertow. We can use the spring-boot- starter-web module to start and running application quickly.

# SpringApplication

# It is a class which provides the convenient way to bootstrap a spring application which can be started from main method. You can call start your application just by calling a static run() method.

# public static void main(String[] args){

# SpringApplication.run(className.class, args);

# }

# Application Events and Listeners

# Spring Boot uses events to handle variety of tasks. It allows us to create factories file that are used to add listeners. we can refer it by using ApplicationListener key.

# Always create factories file in META-INF folder like: META-INF/spring.factories

# Admin Support

# Spring Boot provides the facility to enable admin related features for the application. It is used to access and manage application remotely. We can enable it by simply using spring.application.admin.enabled property.

# Externalized Configuration

# Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments. Application use YAML files to externalize configuration.

# Properties Files

# Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: server-port = 8082 and many others. It helps to organize application properties.

# YAML Support

# It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.

# Type-safe Configuration

# Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

# Logging

# Spring Boot uses Common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies, if there is no required customization is needed.

# Security

# Spring Boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of Endpoints are available for develop a secure Spring Boot application.

Why Spring Boot?

Before Spring ,

we have to configure the dependency of class using xml file or annotation or java config.

For example using xml file

<bean id="userService" class="com.sivalabs.myapp.service.UserService">

    <property name="userDao" ref="userDao"/>

</bean>

<bean id="userDao" class="com.sivalabs.myapp.dao.JdbcUserDao">

    <property name="dataSource" ref="dataSource"/>

</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>

    <property name="url" value="jdbc:<mysql://localhost:3306/test>"/>

    <property name="username" value="root"/>

    <property name="password" value="secret"/>

</bean>

Or using annotation

|  |  |
| --- | --- |
| @Service  public class UserService  {      private UserDao userDao;        @Autowired      public UserService(UserDao dao){          this.userDao = dao;      }      ...      ...  } | |
|  | @Repository  public class JdbcUserDao  {      private DataSource dataSource;        @Autowired      public JdbcUserDao(DataSource dataSource){          this.dataSource = dataSource;      }      ...      ...  } |

Or using JavaConfig based configuration

Configuration

public class AppConfig

{

    @Bean

    public UserService userService(UserDao dao){

        return new UserService(dao);

    }

    @Bean

    public UserDao userDao(DataSource dataSource){

        return new JdbcUserDao(dataSource);

    }

    @Bean

    public DataSource dataSource(){

        BasicDataSource dataSource = new BasicDataSource();

        dataSource.setDriverClassName("com.mysql.jdbc.Driver");

        dataSource.setUrl("jdbc:<mysql://localhost:3306/test>");

        dataSource.setUsername("root");

        dataSource.setPassword("secret");

        return dataSource;

    }

}

We have to configure the server in order to run the application.

we have to give the all dependency of jar file to maven file also.

Like given below

?xml version="1.0" encoding="UTF-8"?>

<project xmlns="<http://maven.apache.org/POM/4.0.0>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0>

<http://maven.apache.org/maven-v4_0_0.xsd>">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.sivalabs</groupId>

    <artifactId>springmvc-jpa-demo</artifactId>

    <packaging>war</packaging>

    <version>1.0-SNAPSHOT</version>

    <name>springmvc-jpa-demo</name>

    <properties>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <maven.compiler.source>1.8</maven.compiler.source>

        <maven.compiler.target>1.8</maven.compiler.target>

        <failOnMissingWebXml>false</failOnMissingWebXml>

    </properties>

    <dependencies>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>4.2.4.RELEASE</version>

        </dependency>

        <dependency>

            <groupId>org.springframework.data</groupId>

            <artifactId>spring-data-jpa</artifactId>

            <version>1.9.2.RELEASE</version>

        </dependency>

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>jcl-over-slf4j</artifactId>

            <version>1.7.13</version>

        </dependency>

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>slf4j-api</artifactId>

            <version>1.7.13</version>

        </dependency>

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>slf4j-log4j12</artifactId>

            <version>1.7.13</version>

        </dependency>

        <dependency>

            <groupId>log4j</groupId>

            <artifactId>log4j</artifactId>

            <version>1.2.17</version>

        </dependency>

        <dependency>

            <groupId>com.h2database</groupId>

            <artifactId>h2</artifactId>

            <version>1.4.190</version>

        </dependency>

        <dependency>

            <groupId>commons-dbcp</groupId>

            <artifactId>commons-dbcp</artifactId>

            <version>1.4</version>

        </dependency>

        <dependency>

            <groupId>mysql</groupId>

            <artifactId>mysql-connector-java</artifactId>

            <version>5.1.38</version>

        </dependency>

        <dependency>

            <groupId>org.hibernate</groupId>

            <artifactId>hibernate-entitymanager</artifactId>

            <version>4.3.11.Final</version>

        </dependency>

        <dependency>

            <groupId>javax.servlet</groupId>

            <artifactId>javax.servlet-api</artifactId>

            <version>3.1.0</version>

            <scope>provided</scope>

        </dependency>

        <dependency>

            <groupId>org.thymeleaf</groupId>

            <artifactId>thymeleaf-spring4</artifactId>

            <version>2.1.4.RELEASE</version>

        </dependency>

    </dependencies>

</project>

Spring framework takes to much time to do the configuration of pom file and xml file.

To overcome this problem,Spring Boot developed

In Spring boot application ,we are only configured pom.xml file with basic dependency given by the spring boot framework which provided the basic dependency configuration for the application.

Example is given below

<dependencies>

        <dependency>

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

            <artifactId>spring-boot-starter-test</artifactId>

        </dependency>

        <dependency>

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

            <artifactId>spring-boot-starter-data-jpa</artifactId>

        </dependency>

        <dependency>

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

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

        </dependency>

        <dependency>

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

            <artifactId>spring-boot-starter-thymeleaf</artifactId>

        </dependency>

        <dependency>

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

            <artifactId>spring-boot-devtools</artifactId>

        </dependency>

        <dependency>

            <groupId>mysql</groupId>

            <artifactId>mysql-connector-java</artifactId>

        </dependency>

    </dependencies>

If look into theh pom.xml file of Spring boot application then we can draw some conclusion where we are providing the entry for dependency of bean and also spring boot fetch all the required jar file for the application.

For example

Spring-boot-starter-web which will fetch all the dependent jar file and also it will configured the dependency of bean automatically. For example it will fetch the spring-web,spring-webmvc,tomcat,jacson-json etc jar files will be fetched. And

registered beans like DispatcherServlet, ResourceHandlers, MessageSource etc beans with sensible defaults.

We have added spring-boot-starter-data-jpa dependency. This pulls all the spring-data-jpadependencies and also adds Hibernate libraries because majority of the applications use Hibernate as JPA implementation.

We haven’t defined any of the DataSource, EntityManagerFactory, TransactionManager etc beans but they are automatically gets created. How?If we have any in-memory database drivers like H2 or HSQL in our classpath then SpringBoot will automatically creates an in-memory DataSource and then registers EntityManagerFactory, TransactionManager beans automatically with sensible defaults.But we are using MySQL, so we need to explicitly provide MySQL connection details. We have configured those MySQL connection details in application.properties file and SpringBoot creates a DataSource using these properties.

The most important and surprising thing is we have created a simple Java class annotated with some magical annotation @SpringBootApplication having a main method and by running that main we are able to run the application and access it at http://localhost:8080/.

Where is the servlet container comes from?   
We have added spring-boot-starter-web which pull the spring-boot-starter-tomcat automatically and when we run the main() method it started tomcat as an embedded container so that we don’t have to deploy our application on any externally installed tomcat server.

By the way have you observe that our packaging type in pom.xml is ‘jar’ not ‘war’. Wonderful!

Ok, but what if I want to use Jetty server instead of tomcat?   
Simple, exclude spring-bootstarter-tomcat from spring-boot-starter-web and include spring-boot-starter-jetty.