

**What is Spring Boot?**

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

**Features**

* Create stand-alone Spring applications.
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files).
* Provide opinionated 'starter' POMs to simplify your Maven configuration.
* Automatically configure Spring whenever possible.
* Provide production-ready features such as metrics, health checks and externalized configuration.
* Absolutely no code generation and no requirement for XML configuration.

**To make application as Spring Boot application:**

1. **Spring Boot started parent:** provides version to spring jar’s you can override using properties tag.

* Dependency Versioning (to avoid dependency version conflicts.)
* Default Plugins
* Java Version

<parent>

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

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

<version>1.4.0.RELEASE</version>

</parent>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

1. **Spring Boot started web:** is used get lots of jar instead of adding one by one in maven. Tomcat is default embedded container.
2. **Spring Boot maven plugin:** is used to create war file and launch the application.
3. **Application Main Class:**

@SpringBootApplication

**public** **class** Application {

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

SpringApplication.*run*(Application.**class**, args);

}

}

**Spring Boot auto configure:** if jar’s is there in class path spring boot it will automatically register Dispatcher servlet, error page and other stuff.

To see all auto configuration add this in application.properties file.

logging.level.org.springframework=DEBUG

**Spring Boot starter web service:** is used to for SOAP web services.

**Spring Boot starter data rest:**

**Spring Boot vs Spring:**

**Applications with Spring Framework**

* Over the next few years, a number of applications were developed with Spring Framework
* Testable but
* Lot of configuration (XML and Java)
* Developing Spring Based application need configuration of a lot of beans!
* Integration with other frameworks need configuration as well!
* In the last few years, focus is moving from monolith applications to microservices. We need to be able to start project quickly. Minimum or Zero start up time
* Framework Setup
* Deployment - Configurability
* Logging, Transaction Management
* Monitoring
* Web Server Configuration

**Spring Boot**

* Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.
* We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss.

**Spring Boot vs Spring MVC:**

There is no relation.

Spring MVC provides decoupled way of developing web applications.

Spring Boot configures Dispatcher Servlet

**Configure Profile:**

Using -Dspring.profiles.active=prod in VM Arguments.

Or

spring.profiles.active=dev in application.properties file

spring.profiles.active=dev (if key is not available it will pick from default properties file)

application.properties (default)

application-dev.properties

application-qa.properties

application-prod.properties



@Profile is used to define bean in specific environment.

To exclude tomcat from spring boot.

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

**Spring Boot Starters:**

spring-boot-devtools: Is used to reflect the code changes automatically without server re start.

**Excluding Resources:**

Certain resources do not necessarily need to trigger a restart when they are changed.

spring.devtools.restart.exclude=static/\*\*,public/\*\*

If you do not like application.properties as the configuration file name, you can switch to another file name by specifying a spring.config.name environment property. You can also refer to an explicit location by using the spring.config.location environment property.

spring.config.name and spring.config.location are used very early to determine which files have to be loaded, so they must be defined as an environment property (typically an OS environment variable, a system property, or a command-line argument).

**Reading properties file key as bean:**

@Component

@ConfigurationProperties("prefix name of property")

public class BasicConfiguration {

}

Autowire this in any class and call getter methods.

<https://github.com/in28minutes/SpringBootForBeginners/blob/master/Step17.md>

spring.application.name=application name

server.port=2018

**What is @SpringBootApplication annotation in spring boot?**

* Many Spring Boot developers always have their main class annotated with @Configuration, @EnableAutoConfiguration and @ComponentScan. Since these annotations are so frequently used together (especially if you follow the best practices above), Spring Boot provides a convenient @SpringBootApplication alternative.
* The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes:
* If your other package hierarchies are below your main app with the @SpringBootApplication annotation, you’re covered by the implicit Component Scan.
* If there are beans/components in other packages that are not sub-packages of the main package, you should manually add them as @ComponentScan({"package1","package2"})

The following are the parameters accepted in the @SpringBootApplication annotation:

**exclude:** Exclude the list of classes from the auto configuration.

**excludeNames:** Exclude the list of fully qualified class names from the auto configuration. This parameter added since spring boot 1.3.0.

**scanBasePackageClasses:** Provide the list of classes that has to be applied for the @ComponentScan.

**scanBasePackages** Provide the list of packages that has to be applied for the @ComponentScan. This parameter added since spring boot 1.3.0.

**How to create war file**

* 1. Change in pom.xml <packaging>war</packaging>
  2. @Override

Protected SpringApplicationBuilder configure(SpringApplicationBuilder app) {

return application.sources(SpringBootWebApplication.class);

}

**Get JSON and XML from service:**

1. http://localhost:2018/hi.json and http://localhost:2018/hi.xml
2. http header accept application/json or application/xml
3. Annotate POJO with @XmlRootElement