**Important Points on Spring Boot and Hibernate**

**What is the Spring Boot?**

Spring Boot is basically an extension of the Spring framework which eliminated the boilerplate configurations required for setting up a Spring application.  
  
Spring Boot is an opinionated framework that helps developers build Spring-based applications quickly and easily. **The main goal of Spring Boot is to quickly create Spring-based applications without requiring developers to write the same boilerplate configuration again and again.**

**Spring Boot Primary Goals**

* Provide a radically faster and widely accessible getting-started experience for all Spring development.
* Be opinionated out of the box but get out of the way quickly as requirements start to diverge from the defaults.
* Provide a range of non-functional features that are common to large classes of projects (such as embedded servers, security, metrics, health checks, and externalized configuration).
* Absolutely no code generation and no requirement for XML configuration.

**Key Spring Boot features**

Let me a list of a few key features of the Spring boot and we will discuss each key feature briefly.

1. Spring Boot starters
2. Spring Boot autoconfiguration
3. Elegant configuration management
4. Spring Boot actuator
5. Easy-to-use embedded servlet container support

**1. Spring Boot Starters**

Spring Boot offers many starter modules to get started quickly with many of the commonly used technologies, like SpringMVC, JPA, MongoDB, Spring Batch, SpringSecurity, Solr, ElasticSearch, etc. These starters are pre-configured with the most commonly used library dependencies so you don’t have to search for the compatible library versions and configure them manually.

For example, the *spring-boot-starter-data-jpa* starter module includes all the dependencies required to use Spring Data JPA, along with Hibernate library dependencies, as Hibernate is the most commonly used JPA implementation.

One more example, when we add the spring-boot-starter-web dependency, it will by default pull all the commonly used libraries while developing Spring MVC applications, such as spring-webmvc, jackson-json, validation-api, and tomcat.

Not only does the spring-boot-starter-web add all these libraries but it also configures the commonly registered beans like *DispatcherServlet*, *ResourceHandlers*, *MessageSource*, etc. with sensible defaults.

Read more about starters on [**Important Spring boot Starters with Examples**](http://www.javaguides.net/2018/09/important-spring-boot-starters-with-examples.html)

**2. Spring Boot Autoconfiguration**

Spring Boot addresses the problem that Spring applications need complex configuration by eliminating the need to manually set up the boilerplate configuration.

Spring Boot takes an opinionated view of the application and configures various components automatically, by registering beans based on various criteria. The criteria can be:

* Availability of a particular class in a classpath
* Presence or absence of a Spring bean
* Presence of a system property
* An absence of a configuration file

For example, if you have the *spring-webmvc* dependency in your classpath, Spring Boot assumes you are trying to build a SpringMVC-based web application and automatically tries to register *DispatcherServlet* if it is not already registered. If you have any embedded database drivers in the classpath, such as H2 or HSQL, and if you haven’t configured a *DataSource* bean explicitly, then Spring Boot will automatically register a *DataSource* bean using in-memory database settings.

You will learn more about the autoconfiguration on [**What is Spring Boot Auto Configuration?**](http://www.springboottutorial.com/spring-boot-auto-configuration)

**3. Elegant Configuration Management**

Spring supports externalizing configurable properties using the [**@PropertySource**](http://www.javaguides.net/2018/09/spring-propertysource-annotation-with-example.html) configuration. Spring Boot takes it even further by using the sensible defaults and powerful type-safe property binding to bean properties. Spring Boot supports having separate configuration files for different profiles without requiring many configurations.

Read more [**http://www.javaguides.net/2018/09/spring-propertysource-annotation-with-example.html**](http://www.javaguides.net/2018/09/spring-propertysource-annotation-with-example.html)

**4. Spring Boot Actuator**

Being able to get the various details of an application running in production is crucial to many applications. The Spring Boot actuator provides a wide variety of such production-ready features without requiring developers to write much code. Some of the Spring actuator features are:

* Can view the application bean configuration details
* Can view the application URL mappings, environment details, and configuration parameter values
* Can view the registered health check metrics

Read more about Spring Boot Actuator on [**Spring Boot Actuator**](https://www.baeldung.com/spring-boot-actuators)

**5. Easy-to-Use Embedded Servlet Container Support**

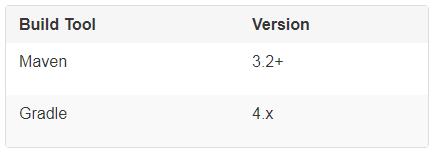
Traditionally, while building web applications, you need to create *WAR* type modules and then deploy them on external servers like *Tomcat*, *WildFly*, etc. But by using Spring Boot, you can create a *JAR* type module and embed the servlet container in the application very easily so that the application will be a self-contained deployment unit.

Also, during development, you can easily run the Spring Boot JAR type module as a Java application from the IDE or from the command-line using a build tool like [**Maven**](http://www.javaguides.net/p/maven.html) or Gradle.

**System Requirements**

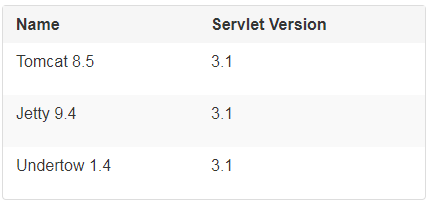
Spring Boot 2+ requires Java 8 or 9 and Spring Framework 5.1.0.RELEASE or above.

Explicit build support is provided for the following build tools:

**[](https://2.bp.blogspot.com/-kW0hSn1inuI/W7B8qDvCBxI/AAAAAAAAEA8/pp7Bm3gWNlgGxD3EL61LS3BkVsd8noR5ACLcBGAs/s1600/build-tool-support.PNG)**

**Servlet Containers**

Spring Boot supports the following embedded servlet containers:

**[](https://3.bp.blogspot.com/-kArn3r6WYlU/W7B8vX76_TI/AAAAAAAAEBA/BNh_hsHaJB8QnerlJD9erkkDMOXmdGl8wCLcBGAs/s1600/servlet-containers.PNG)**

You can also deploy Spring Boot applications to any Servlet 3.1+ compatible container.

<?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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>net.javaguides.springboot</groupId>

<artifactId>Springboot-helloworld-application</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Springboot-helloworld-application</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>2.0.5.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

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

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

From above pom.xml, let's understand a few important spring boot features.

**Spring Boot Maven plugin**

The Spring Boot Maven plugin provides many convenient features:

* It collects all the jars on the classpath and builds a single, runnable "über-jar", which makes it more convenient to execute and transport your service.
* It searches for the public static void *main()* method to flag as a runnable class.
* It provides a built-in dependency resolver that sets the version number to match Spring Boot dependencies. You can override any version you wish, but it will default to Boot’s chosen set of versions.

**spring-boot-starter-parent**

All Spring Boot projects typically use spring-boot-starter-parent as the parent in pom.xml.

<parent>

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

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

<version>2.0.5.RELEASE</version>

</parent>

Parent Poms allow you to manage the following things for multiple child projects and modules:

* Configuration - Java Version and Other Properties
* Dependency Management - Version of dependencies
* Default Plugin Configuration
* import java.util.concurrent.atomic.AtomicLong;
* import org.springframework.web.bind.annotation.RequestMapping;
* import org.springframework.web.bind.annotation.RequestParam;
* import org.springframework.web.bind.annotation.RestController;
* @RestController
* public class GreetingController {
* private static final String template = "Hello, %s!";
* private final AtomicLong counter = new AtomicLong();
* @RequestMapping("/greeting")
* public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
* return new Greeting(counter.incrementAndGet(), String.format(template, name));
* }
* }

Let's understand the above Controller with step by step.

* The *@RequestMapping* annotation ensures that HTTP requests to */greeting* are mapped to the *greeting()* method.
* The above example does not specify GET vs. PUT, POST, and so forth, because of *@RequestMapping* maps all HTTP operations by default. Use *@RequestMapping(method=GET)* to narrow this mapping.
* *@RequestParam* binds the value of the query string parameter name into the name parameter of the *greeting()* method. If the name parameter is absent in the request, the defaultValue of "World" is used.
* The implementation of the method body creates and returns a new *Greeting* object with id and content attributes based on the next value from the counter and formats the given name by using the greeting template.
* A key difference between a traditional MVC controller and the RESTful web service controller above is the way that the HTTP response body is created. Rather than relying on a view technology to perform server-side rendering of the greeting data to HTML, this RESTful web service controller simply populates and returns a Greeting object. The object data will be written directly to the HTTP response as JSON.
* This code uses Spring 4’s new *@RestController* annotation, which marks the class as a controller where every method returns a domain object instead of a view. It’s shorthand for @Controller and *@ResponseBody* rolled together.
* The *Greeting* object must be converted to JSON. Thanks to Spring’s HTTP message converter support, you don’t need to do this conversion manually. Because Jackson 2 is on the classpath, Spring’s *MappingJackson2HttpMessageConverter* is automatically chosen to convert the Greeting instance to JSON.

## Make the application executable - SpringbootHelloworldApplication.java

package net.javaguides.springboot.Springboothelloworldapplication;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringbootHelloworldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringbootHelloworldApplication.class, args);

}

}

[@SpringBootApplication](http://www.javaguides.net/2018/09/spring-boot-springbootapplication-annotation-with-example.html) is a convenience annotation that adds all of the following:

@Configuration tags the class as a source of bean definitions for the application context.

[@EnableAutoConfiguration](http://www.javaguides.net/2018/09/spring-boot-enableautoconfiguration-annotation-with-example.html) tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

Normally you would add @EnableWebMvc for a Spring MVC app, but Spring Boot adds it automatically when it sees *spring-webmvc* on the classpath. This flags the application as a web application and activates key behaviors such as setting up a DispatcherServlet.

[@ComponentScan](http://www.javaguides.net/2018/09/spring-configuration-annotation-with-example.html) tells Spring to look for other components, configurations, and services in the hello package, allowing it to find the controllers.

The *main()* method uses Spring Boot’s *SpringApplication.run()* method to launch an application. Did you notice that there wasn’t a single line of XML? No web.xml file either. This web application is 100% pure Java and you didn’t have to deal with configuring any plumbing or infrastructure.