**Spring Boot Introduction**

**()->{}**

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

## What is Micro Service?

Micro Service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.

### Advantages

Micro services offers the following advantages to its developers −

* Easy deployment
* Simple scalability
* Compatible with Containers
* Minimum configuration
* Lesser production time

## What is Spring Boot?

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**. You can get started with minimum configurations without the need for an entire Spring configuration setup.

### Advantages

Spring Boot offers the following advantages to its developers −

* Easy to understand and develop spring applications
* Increases productivity
* Reduces the development time

### Goals

Spring Boot is designed with the following goals −

* To avoid complex XML configuration in Spring
* To develop a production ready Spring applications in an easier way
* To reduce the development time and run the application independently
* Offer an easier way of getting started with the application

## Why Spring Boot?

You can choose Spring Boot because of the features and benefits it offers as given here −

It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.

It provides a powerful batch processing and manages REST endpoints.

In Spring Boot, everything is auto configured; no manual configurations are needed.

It offers annotation-based spring application

Eases dependency management

It includes Embedded Servlet Container

## How does it work?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

## Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include **spring-boot-starter-data-jpa** dependency in your project.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** \*, where \* indicates that it is a type of the application.

### Examples

Look at the following Spring Boot starters explained below for a better understanding −

**Spring Boot Starter Actuator dependency** is used to monitor and manage your application. Its code is shown below −

<dependency>

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

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

</dependency>

**Spring Boot Starter Security dependency** is used for Spring Security. Its code is shown below −

<dependency>

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

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

</dependency>

**Spring Boot Starter web dependency** is used to write a Rest Endpoints. Its code is shown below −

<dependency>

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

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

</dependency>

**Spring Boot Starter Thyme Leaf dependency** is used to create a web application. Its code is shown below −

<dependency>

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

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

</dependency>

**Spring Boot Starter Test dependency** is used for writing Test cases. Its code is shown below −

<dependency>

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

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

</dependency>

## Auto Configuration

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add **@EnableAutoConfiguration** annotation or **@SpringBootApplication** annotation to your main class file. Then, your Spring Boot application will be automatically configured.

Observe the following code for a better understanding −

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}}

## Spring Boot Application

The entry point of the Spring Boot Application is the class contains **@SpringBootApplication** annotation. This class should have the main method to run the Spring Boot application. **@SpringBootApplication** annotation includes Auto- Configuration, Component Scan, and Spring Boot Configuration.

If you added **@SpringBootApplication** annotation to the class, you do not need to add the **@EnableAutoConfiguration, @ComponentScan** and **@SpringBootConfiguration** annotation. The **@SpringBootApplication** annotation includes all other annotations.

Observe the following code for a better understanding −

import org.springframework.boot.SpringApplication;import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}}

## Component Scan

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the **@ComponentScan** annotation for your class file to scan your components added in your project.

Observe the following code for a better understanding −

import org.springframework.boot.SpringApplication;import org.springframework.context.annotation.ComponentScan;

@ComponentScan

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}}

# **Spring Boot - Quick Start**

Prerequisites

Your system need to have the following minimum requirements to create a Spring Boot application −

Java 7

Maven 3.2

Gradle 2.5

Spring Boot CLI

The Spring Boot CLI is a command line tool and it allows us to run the Groovy scripts. This is the easiest way to create a Spring Boot application by using the Spring Boot Command Line Interface. You can create, run and test the application in command prompt itself.

This section explains you the steps involved in manual installation of Spring Boot CLI. For further help, you can use the following link: [https://docs.spring.io/springboot/ docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-springboot](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/" \l "getting-started-installing-spring-boot" \t "https://www.tutorialspoint.com/spring_boot/_blank)

You can also download the Spring CLI distribution from the Spring Software repository at: [https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-manual-cli-installation](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/" \l "getting-started-manual-cli-installation" \t "https://www.tutorialspoint.com/spring_boot/_blank)

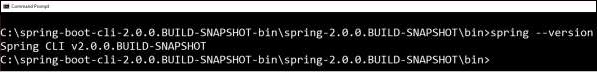
For manual installation, you need to use the following two folders −

spring-boot-cli-2.0.0.BUILD-SNAPSHOT-bin.zip

spring-boot-cli-2.0.0.BUILD-SNAPSHOT-bin.tar.gz

After the download, unpack the archive file and follow the steps given in the install.txt file. Not that it does not require any environment setup.

In Windows, go to the Spring Boot CLI bin directory in the command prompt and run the command spring –-version to make sure spring CLI is installed correctly. After executing the command, you can see the spring CLI version as shown below −



Run Hello World with Groovy

Create a simple groovy file which contains the Rest Endpoint script and run the groovy file with spring boot CLI. Observe the code shown here for this purpose −

@Controllerclass Example {

@RequestMapping("/")

@ResponseBody

public String hello() {

"Hello Spring Boot"

}}

Now, save the groovy file with the name hello.groovy. Note that in this example, we saved the groovy file inside the Spring Boot CLI bin directory. Now run the application by using the command spring run hello.groovy as shown in the screenshot given below −

IMG_257

Once you run the groovy file, required dependencies will download automatically and it will start the application in Tomcat 8080 port as shown in the screenshot given below −

IMG_258

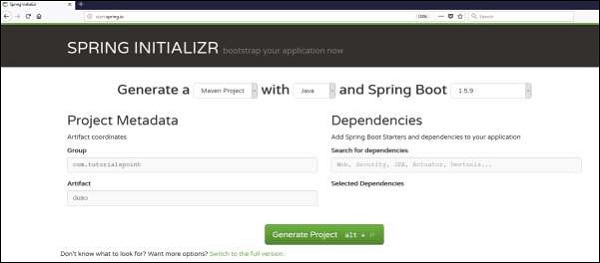
Once Tomcat starts, go to the web browser and hit the URL http://localhost:8080/ and you can see the output as shown.



## Spring Initializer

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page [www.start.spring.io](https://start.spring.io/" \t "https://www.tutorialspoint.com/spring_boot/_blank) and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.

Observe the following screenshot that shows an example where we added the **spring-boot-starter-web** dependency to write REST Endpoints.



Once you provided the Group, Artifact, Dependencies, Build Project, Platform and Version, click **Generate Project** button. The zip file will download and the files will be extracted.

This section explains you the examples by using both Maven and Gradle.

## Maven

After you download the project, unzip the file. Now, your **pom.xml** file looks as shown 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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.tutorialspoint</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>demo</name>

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

<parent>

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

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

<version>1.5.8.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>

## Gradle

Once you download the project, unzip the file. Now your **build.gradle** file looks as shown below −

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()}

dependencies {

compile('org.springframework.boot:spring-boot-starter-web')

testCompile('org.springframework.boot:spring-boot-starter-test')}

## Class Path Dependencies

Spring Boot provides a number of **Starters** to add the jars in our class path. For example, for writing a Rest Endpoint, we need to add the **spring-boot-starter-web** dependency in our class path. Observe the codes shown below for a better understanding −

### Maven dependency

<dependencies>

<dependency>

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

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

</dependency></dependencies>

### Gradle dependency

dependencies {

compile('org.springframework.boot:spring-boot-starter-web')}

## Main Method

The main method should be writing the Spring Boot Application class. This class should be annotated with **@SpringBootApplication**. This is the entry point of the spring boot application to start. You can find the main class file under **src/java/main** directories with the default package.

In this example, the main class file is located at the **src/java/main** directories with the default package **com.tutorialspoint.demo**. Observe the code shown here for a better understanding −

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplicationpublic class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}}

## Write a Rest Endpoint

To write a simple Hello World Rest Endpoint in the Spring Boot Application main class file itself, follow the steps shown below −

Firstly, add the **@RestController** annotation at the top of the class.

Now, write a Request URI method with **@RequestMapping** annotation.

Then, the Request URI method should return the **Hello World** string.

Now, your main Spring Boot Application class file will look like as shown in the code given below −

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.web.bind.annotation.RequestMapping;import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication@RestController

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String hello() {

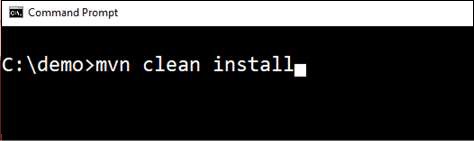
return "Hello World";

}}

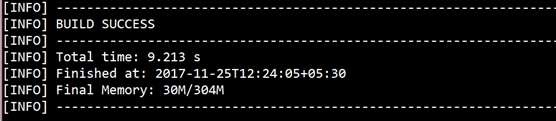
## Create an Executable JAR

Let us create an executable JAR file to run the Spring Boot application by using Maven and Gradle commands in the command prompt as shown below −

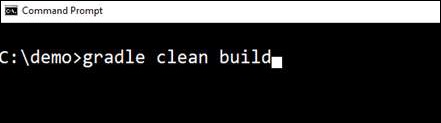
Use the Maven command mvn clean install as shown below −



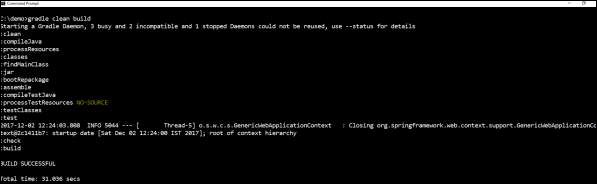
After executing the command, you can see the **BUILD SUCCESS** message at the command prompt as shown below −



Use the Gradle command **gradle clean build** as shown below −



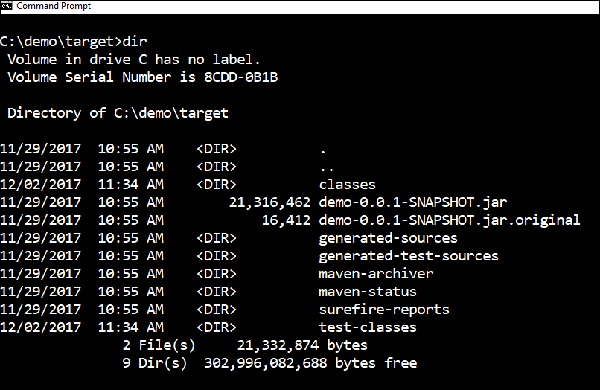
After executing the command, you can see the **BUILD SUCCESSFUL** message in the command prompt as shown below −



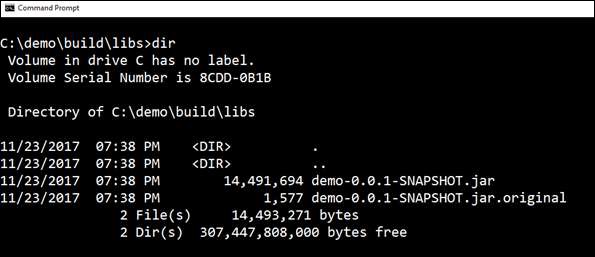
## Run Hello World with Java

Once you have created an executable JAR file, you can find it under the following directories.

For Maven, you can find the JAR file under the target directory as shown below −



For Gradle, you can find the JAR file under the **build/libs** directory as shown below −



Now, run the JAR file by using the command **java –jar <JARFILE>**. Observe that in the above example, the JAR file is named **demo-0.0.1-SNAPSHOT.jar**



Once you run the jar file, you can see the output in the console window as shown below −

IMG_264

Now, look at the console, Tomcat started on port 8080 (http). Now, go to the web browser and hit the URL **http://localhost:8080/** and you can see the output as shown below −

