Spring Boot is one of the Spring projects that uses new development model for quickly building Spring applications with very little configuration.

**But how does it work?**

Because it is,

**1. Opinionated**

Spring Boot forms opinions.  It means that Spring Boot has some sensible defaults which you can use to quickly build your application.  For example, Spring Boot uses embedded Tomcat as the default web container.

**2. Customizable**

Though Spring Boot has its defaults, you can easily customize it at any time during your development based on your need.  For example, if you prefer Jetty, then you can easily make <dependency> change in your POM file to replace the default Tomcat container.

**Spring Boot components**

* **Starter** - Manages the dependencies of the Spring based projects by combining the group of common or related dependencies.  Spring Boot forms the opinions based on starters.
* **Auto configure**  - Detects and automatically configures the Spring applications based on the added dependencies
* **CLI**- Command Line Interface to start, test and stop Spring Boot applications from command prompt (Not discussed in this course)
* **Actuator** - Enables enterprise features and gives the insight of the application

**how to develop a Spring Boot application.**

we use Spring Initializr and Spring Tool Suite IDE to develop Spring Boot applications.

# Spring Initializr

A web tool by Spring official web site from which you can download a pre-configured Spring Boot project, which you can then import into your IDE.

It lets you choose between Maven or Gradle project, the Boot version and the required dependencies for your project.

# Spring Tool Suite

**S**pring **T**ool **S**uite is an Eclipse based IDE dedicated for developing Spring based applications.

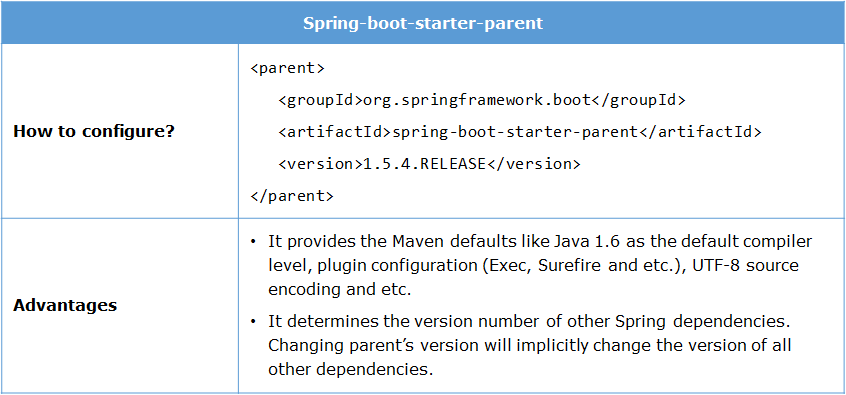
It comes with the pre-installed latest Eclipse release, Spring based components, server integration for Eclipse and other things required to generate an Enterprise application using Spring

Prerequisites for STS is we need java 1.8 version or higher

Building the application is taken care by Maven. Spring Initializr provides a ready-to-use  Maven pom.xml to package the application as Jar or War.

# Spring boot starter parent

Spring Boot provides various **starters**to add jars to your classpath easily. The spring-boot-starter-parent is a special starter defined in the <parent/> section of your pom.xml. The Maven build inherits information about dependencies from here.



The Maven defaults provided by spring-boot-starter-parent can be overridden. For example, if you want to change the Java version, you can easily add a java-version property in your pom.xml as below.

<properties>

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

</properties>

The starters combine all the related dependencies under single dependency. So, you get a one-stop-shop for all the Spring and related technologies without browsing through net and downloading them individually.

The core starter of Spring Boot is spring-boot-starter.  It informs Spring Boot that it is a Spring core application.

<dependency>

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

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

</dependency>

Note that the version number of spring-boot-starter is not defined here as it can be determined by spring-boot-starter-parent. By opening the Dependency Hierarchy tab of your pom.xml, you can see that this single starter POM combines four more dependencies in it.

# @SpringBootApplication

Every Spring Boot project created by Spring initializr will have a main class annotated with @SpringBootApplication.

This annotation is equivalent to declaring @Configuration, @EnableAutoConfiguration and @ComponentScan annotations with their defaults.

**@SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan**

# @Configuration

This annotation tags a class as a configuration class. This annotation is equivalent to defining beans in XML configuration file.

# @EnableAutoConfiguration

This annotation auto-configures all the beans based on class path dependencies.  For example, since you have added spring-core dependency in your classpath via spring-boot-starter, the auto-configuration will assume that you are developing a Spring core application and set up Spring accordingly.  Refer [Spring documentation](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html)to know more about this annotation.

# @ComponentScan

This annotation scans the base package (the package under which this class resides) and creates beans of other annotated java components like componenets, configurations, and services. This is equivalent to Spring's XML configuration <context:componenet-scan/>.

You can override the base package value by providing the **scanBasePackages**attribute as below:

@SpringBootApplication(scanBasePackages="com.infosys.irs")

public class InfyGoApplication {

// rest of the code

}

**How to start the spring boot application?**

The main class, which is annotated with @SpringBootApplication, is also a bootstrap class. It kick starts the application by invoking the SpringApplication.run() method.  You need to pass the .class file name of your main class to the run() method as shown below.

@SpringBootApplication(scanBasePackages="com.infosys.irs")

public class InfyGoApplication{

public static void main(String[] args) {

SpringApplication.run(InfyGoApplication.class, args);

//rest of the code

}

### **SpringApplication Class**

The SpringApplication, inside the main is a Spring Boot class. It is used to bootstrap our application. It calls static method run() which takes two arguments, one is class type and second is string array. It starts auto-configured tomcat web server when Spring application is started.

