Contents

[SPRING BOOT 2](#_Toc78630694)

[WHAT IS SPRING BOOT 2](#_Toc78630695)

[BOOTING SPRING BOOT 2](#_Toc78630696)

[SPRING INITIALIZR 2](#_Toc78630697)

[SPRING CLI 2](#_Toc78630698)

[SPRING IDE 3](#_Toc78630699)

[SPRING BOM 3](#_Toc78630700)

[SAMLE SPRING BOOT POM.XML 3](#_Toc78630701)

[HOW SPRING WORKS 4](#_Toc78630702)

[SAMPLE SPRING BOOT MAIN CLASS 4](#_Toc78630703)

[SAMLE PROJECT STRUCTURE 4](#_Toc78630704)

[REST CONTROLLER 5](#_Toc78630705)

[SERVICE 5](#_Toc78630706)

[BEAN 6](#_Toc78630707)

[SPRING BOOT – BEHIND THE SCENE 7](#_Toc78630708)

[APPLICATION.PROPERTIES 7](#_Toc78630709)

[COMPONENT SCAN - LOOKUP 7](#_Toc78630710)

[SPRING BOOT STARTER WEB 7](#_Toc78630711)

[AUTO CONFIGURATION 8](#_Toc78630712)

[SPRING JPA 8](#_Toc78630713)

[CREATING SPRING JPA PROJECT 8](#_Toc78630714)

[DEPLOYING SPRING APP 8](#_Toc78630715)

[PACKAGING THE SPRING APP 8](#_Toc78630716)

[MONITORING SPRING APP 9](#_Toc78630717)

# SPRING BOOT

## 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 minimal Spring configuration.
* ***Spring boot has an embedded Tomcat – which is bootstrap by Spring boot and deploys the code to tomcat container.***

## SPRING BOOT FEATURES

* **AUTO CONFIGURATION**
  + Spring boot does the configuration for us the type of module we are using . For example – For MVC based application – it configures the Dispatcher Servlet for us. Even for ORM module – we no longer need a to configure Data Source or Transaction Manager – as we use to do in traditional Spring Application.
* **SPRING BOOT STARTERS**
  + This is the parent project of every Spring application and it manages the dependencies.
  + It makes sure that for a particular Spring version correct version of dependent module are downloaded – rather than developer worrying about it.
* **EMBEDDED SERVLET CONTAINER**
  + Default container is Tomcat but can be switched to another container like Jetty or Undertow
* **SPRING ACTUATORS**
  + For the” Health Check” of a Spring application. Spring boot has exposed it using multiple endpoints.

## BOOTING SPRING BOOT

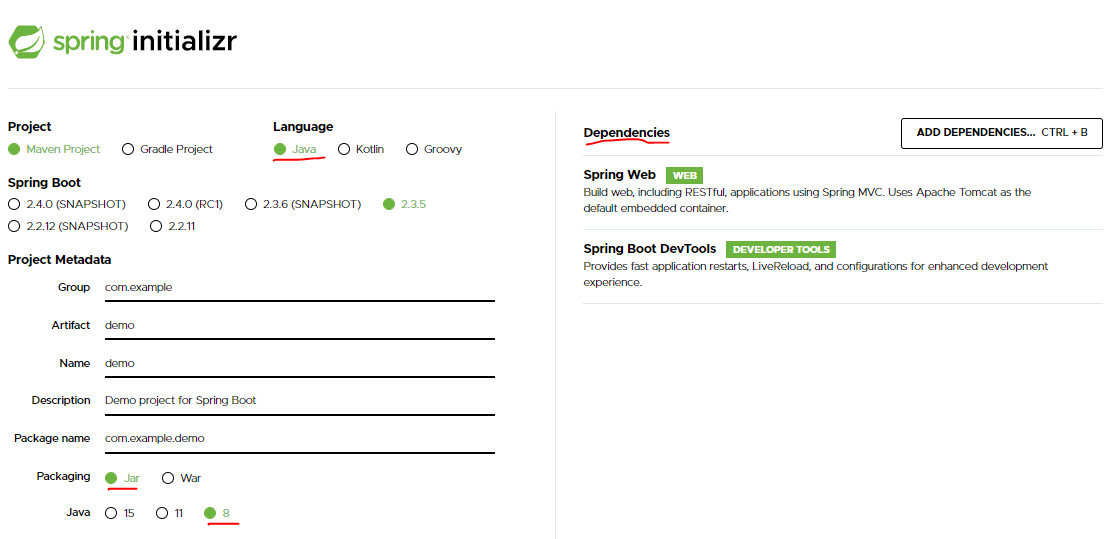
Spring boot can be created in 3 ways

### SPRING INITIALIZR

* The Spring initializr is an online UI on <https://start.spring.io/> which helps us to create a spring boot project.

The gives the configuration to select below items

* The Project Management Tool – Maven /Gradle
* Java Version (8+)
* Dependencies
  + The “Spring web” dependencies are used to create Spring MVC project.
  + “Spring boot Dev tools” helps in hot deployment of Spring App(changes deployed without restarting the server)
* Project specific properties like (Group Id, Artifacts, name)
* Packaging strategies “jar” or war



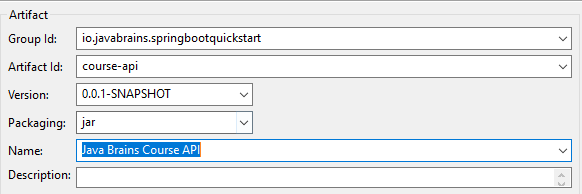
* After the above configuration we can download the zip files and import in a desired editor
* The downloaded zip project will have all the basic classes, configuration and required dependencies

### SPRING CLI

* Spring boot CLI [ <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started-installing-the-cli> ]

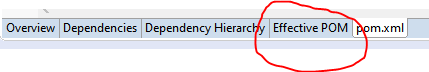
### SPRING IDE

1. File 🡪 New Maven Project 🡪 create a simple project (without archetype selection)
2. Enter the project details



## SPRING BOM

* In our project’s pom.xml we hardly define the dependency . All the default configuration and dependencies and inherited from the parent - **spring-boot-starter-parent**. The project pom just inherits the pom.xml.
* Spring has given “starter” project for each module like “spring-boot-starter-web” for Spring MVC application and ‘spring-boot-starter-data-jpa” for JPA(Hibernate) applications.
* Once these starter projects are added as dependencies – It transitively pulls all the dependent libraries. The list of Jars and its version are listed in the **spring-boot-starter-parent. This version list can be viewed in eclipse in the effective POM**



* **The spring-boot-starter-parent project – maintains the version of all the dependencies and its transitive dependencies. Hence Spring call it a BOM(Bills of Material)**

## SAMLE SPRING BOOT POM.XML

* Spring-boot-starter-parent is the parent of all Spring Boot applications.

|  |
| --- |
| <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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>io.javabrains.springbootquickstart</groupId>  <artifactId>course-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>Java Brains Course API</name>  <parent> 🡨 ***Inheritance from Parent POM***  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.2.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId> 🡨 **To support Spring MVC Project**  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId> 🡨 **Hot deployment**  <scope>runtime</scope>  <optional>true</optional>  </dependency>  </dependencies>  <properties>  <**java.version>1.8</java.version> 🡸 Java Version**  </properties>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

## SAMPLE SPRING BOOT MAIN CLASS

|  |
| --- |
| import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  **@SpringBootApplication**  public class CourseApiApp {  public static void main(String[] args) {  SpringApplication.run(CourseApiApp.class, args);  }  } |

The above code does below things for us internally

* **SET UP THE DEFAULT CONFIGURATION**
* **START THE SPRING APPLICATION CONTEXT** 
  + Application context act as a container of all spring-based components like Controller/ Service / Repositories
  + This manages the life cycle of those components
* **PERFORM THE CLASS PATH SCAN**
  + To manage the spring-based component it must scan the files look for the Annotation which a java can has
  + It manages the components the it has been annotated. For example – the components are marked with @Service annotation are treated as service and spring always treat it as a singleton class.
* **START THE TOMCAT SERVER.**
  + Spring boot has in built tomcat container.
  + Running the Spring boot Application “main” class starts the Tomcat container too.

## SAMLE PROJECT STRUCTURE

|  |  |
| --- | --- |
|  | * The Spring Test Suite are marked by @**SpringBootTest** annotation. * It searches of for an annotation **@SpringBootApplication** – and using this class create a container – with all the beans – so that it can execute all the test cases.   Note: Due to above bean creation steps . Test methods can able to autowire the beans into the test cases. |

### REST CONTROLLER

|  |
| --- |
| **package** io.javabrains.springbootstarter.topic;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestBody;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public** **class** TopicController {  @Autowired  **private** TopicService topicService;  @RequestMapping(path = "/topics")  **public** List<Topic> getAllTopics() {  **return** topicService.getAllTopics();  }    @RequestMapping(path = "/topics/{id}")  **public** Topic getTopics(@PathVariable String id) {  **return** topicService.getTopic(id);  }    @RequestMapping(path = "/topics",method = RequestMethod.***POST***)  **public** **void** addTopic(@RequestBody Topic topic) {  topicService.addTopic(topic);  }    @RequestMapping(path = "/topics/{id}",method = RequestMethod.***PUT***)  **public** **void** updateTopic(@RequestBody Topic topic, @PathVariable String id) {  topicService.updateTopic(topic, id);  }    @RequestMapping(path = "/topics/{id}",method = RequestMethod.***DELETE***)  **public** **void** deleteTopic(@RequestBody Topic topic, @PathVariable String id) {  topicService.deleteTopic(topic, id);  }  } |

### SERVICE

* If Spring Component is annotated as a service. Spring makes it a Singleton class and creates just one instance of the service which in turn used by multiple controller by auto wiring.

|  |
| --- |
| package io.javabrains.springbootstarter.topic;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import org.springframework.stereotype.Service;  @Service  public class TopicService {  private List<Topic> topics = new ArrayList<>(Arrays.asList(  new Topic("spring", "Spring Framework", "Spring Framework") ,  new Topic("Java", "Core Java", "Core Java"),  new Topic("springboot", "Spring Boot", "Spring Boot"))  );  public List<Topic> getAllTopics(){  return topics;  }  public Topic getTopic(String id) {  return topics.stream().filter(t -> t.getId().equals(id)).findFirst().get();  }  public void addTopic(Topic topic) {  topics.add(topic);  }  public void updateTopic(Topic topic, String id) {  for(int topicLength = 0 ; topicLength < topics.size() ; topicLength ++) {  Topic topic2 = topics.get(topicLength);  if(topic2.getId().equals(id)) {  topics.set(topicLength, topic);  }  }  }  public void deleteTopic(Topic topic, String id) {  for(int topicLength = 0 ; topicLength < topics.size() ; topicLength ++) {  Topic topic2 = topics.get(topicLength);  if(topic2.getId().equals(id)) {  topics.remove(topicLength);  }  }  }  } |

### BEAN

|  |
| --- |
| **package** io.javabrains.springbootstarter.topic;  **public** **class** Topic {  **private** String id;  **private** String name;  **private** String description;  **public** Topic() { }  **public** Topic(String id, String name, String description) {  **super**();  **this**.id = id;  **this**.name = name;  **this**.description = description;  }  **public** String getId() {  **return** id;  }  **public** **void** setId(String id) {  **this**.id = id;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getDescription() {  **return** description;  }  **public** **void** setDescription(String description) {  **this**.description = description;  }  } |

## SPRING BOOT – BEHIND THE SCENE

* **spring-boot-starter-parent:** 
  + The **Spring boot starter parent** project is parent project which gets inherited by our custom Spring boot project. It has some default configuration which will be inherited by the custom Spring boot app.
  + Due to this inheritance don’t have to do any configuration on a custom app level
* **spring-boot-starter-web**
  + The “**spring-boot-starter-web**” is dependencies we add as a dependencies of custom Spring boot Application . This will download all the web related dependencies. So now adding just one dependencies download all the related web dependencies unlike adding the dependencies individually.

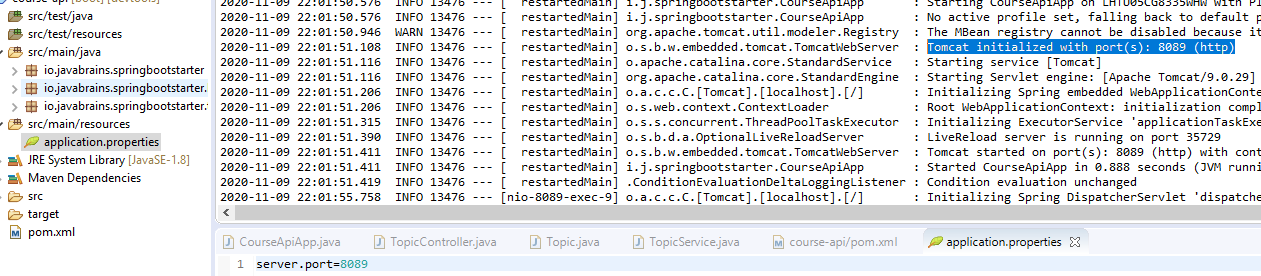
The “**spring-boot-starter-web” dependencies decide what dependency to download but the version of the of those dependencies depends upon the Spring version (Spring boot starter parent)**

* **EMBEDDED TOMCAT:**

## APPLICATION.PROPERTIES

Spring boot project comes with some default configuration. We still can override those default configuration using **application.properties**

**EXAMPLE: TO OVERRIDE THE SERVER PORT**



* This url will have all the keys/configuration which can be used in application.properties file : <https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html>

# COMPONENT SCAN - LOOKUP

|  |  |
| --- | --- |
| @SpringBootApplication  **@ComponentScan("com.javabrains")**  public class CourseApiApp {  public static void main(String[] args) {  SpringApplication.run(CourseApiApp.class, args);  }  } | * By default Spring scans the component ,service, controller, repository in its child package * If the Service & component do not reside in the child page. The Spring boot starter class must know the package when it must search. That can be done using @ComponentScan annotation |

## SPRING BOOT STARTER WEB

* The Spring boot started web is leverage to create Spring MVC project and Rest API
* The project has to inherit the starter web project - The inheritance is accomplished in pom.xml file using <parent> tag.

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.2.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent> |

## AUTO CONFIGURATION

1. Spring Interface Injection
2. Hot reloading in+6 Spring Boot -> Add dev tools module – spring-boot-devtools
3. Deploy the Spring boot to external web server – Package it as war

## SPRING JPA

* Spring Data JPA is a separate project which lets spring with ORM based application.

### CREATING SPRING JPA PROJECT

|  |  |
| --- | --- |
| * For demo application – We will be using a in memory DB – Apache Derby * Create Spring Starter Project and select below dependencies   + Spring Data JPA   + Apache Derby   + Spring Web |  |

#### INTERACTION BETWEEN LAYERS

|  |  |
| --- | --- |
|  |  |

## DEPLOYING SPRING APP

### PACKAGING THE SPRING APP

* The Spring application are standalone application which can run on its own
* **STEPS TO RUN SPRING BOOT APP**
  + Step 1: **mvn clean install** 🡪 This will create a jar/ war file in the target folder of the app
  + Step 2: java -jar <**JAR\_FILE\_NAME**>
* **PACKAGING SPRING APP**
  + The Spring app can be packaged as “jar” or “war” depending upon the pom.xml configuration

|  |  |
| --- | --- |
| **PACKAGING AS JAR** | **PACKAGING AS WAR** |
| **Packaging as jar can run from command line.** | **Unlike jar packaging as war need a container where the app can be deployed** |

## MONITORING SPRING APP