**What is Spring Boot (SB)?**

**Misconceptions about Spring Boot.**

1. SB is not a code generation framework. It does not generate any code.
2. SB is neither an application server, nor it is web server. It just provides good integration with different application and web server.
3. SB does not implement any specific frameworks or specifications.

**Primary Goals of SB:**

1. Enable ready to run ground with Spring-based projects.
2. Be opinionated i.e. make default assumptions based on common uses and if required provide configuration option to handle or modify these default configurations.
3. Do not use code generation and avoid using a lot of XML configurations.

**Spring Boot Hello World:**

The following steps are involved in starting up with the Spring Boot application:

1. Configure [spring-boot-starter-parent] in pom.xml
2. Configure the pom.xml file with required starter projects
3. Configure [spring-boot-maven-plugin] to be able to run the application
4. Create Spring Boot launch class.

**Why we need [spring-boot-starter-parent]?**

1. A spring-boot-starter-parent dependency contains the default version of java to use,
2. the default versions of dependencies that SB uses
3. the default configuration of maven plugins
4. A spring-boot-starter-parent dependency is the parent POM providing dependency and plugin management for Spring Boot-based application

|  |
| --- |
| **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<version>1.5.21.RELEASE</version>**  **<relativePath/> <!-- lookup parent from repository -->**  **</parent>** |

**Note1:** As it provides default dependency management for all the dependencies that SB uses, if we want to override a specific version of a dependency, we can do that by <properties> tag with the right name in the pom.xml **of our application. For example**

|  |  |
| --- | --- |
| **<properties>**  **<java.version>1.8</java.version>**  **</properties>** | **<properties>**  **<mockito.version>1.10.20</mockito.version>**  **</properties>** |

The above snippet shows an example of configuration out application to use version java 1.8 and Mockito 1.10.20

**Note2-** Compatibility between different versions of framework is one of the major problems faced by the developers. How do I find the latest spring session version which is compatible with specific version of Spring? The usual answer read the documentation or search in google. However, if we use Spring Boot, this is made simple by [spring-boot-starter-parent]. If we want to upgrade to a newer Spring version, all that we need to do is to find the spring-boot-starter-parent dependency for the spring version. Once we upgrade our application with specific version of spring-boot-starter-parent, we would have all the other dependencies upgraded automatically.

**Configure pom.xml with the Required Starter Projects: Understanding starter Projects**

Whenever we want to build an application in SB, we need to have some starter project. These starters are nothing but dependency descriptors customized for different purposes.

For example [**spring-boot-starter-web**] is the starter for building web application including RESTful, using Spring MVC. It uses Tomcat as the default embedded container. So, if we want to develop a web application using Spring MVC then we would need to do is include [**spring-boot-starter-web**] in our dependencies and we get the following automatically pre-configured.

1. Spring MVC
2. Compatible versions of databinding and validators
3. [**spring-boot-starter-tomcat**] (starter project for Tomcat for running web application).
4. [**spring-boot-starter-test**] dependency provides the following test framework needed for unit testing: (Junit (basic unit test framework), Mockito (for Mocking), Hamcrest, AssertJ (for readable asserts)
5. Spring Test: Unit testing framework for spring application.

We now have our pom.xml file configured with starter parent and required starter project. Let’s add [spring-boot-maven-plugin] now which would enable us to run Spring boot applications.

**Configuring spring-boot-maven-plugin[spring-boot-maven-plugin]**

When we build applications using Spring Boot, there are a couple of situations are possible.

1. We would like to run the applications without building a JAR or WAR.
2. We would like to build a JAR and a WAR and deploy it later.

The [**spring-boot-maven-plugin]** dependency provide both the above capability

|  |
| --- |
| *<! -- Package as an executable jar -->*  <build>  <plugins>  <plugin>  <groupId>org.springframework.web. boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

The [spring-boot-maven-plugin] dependency provides several goals for SB application The most popular goal is run

**Creating First Spring Boot Launch Class:**

|  |
| --- |
| **package** com.student.springbootstudentsoapservice;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** SpringBootStudentSoapServiceApplication {  **public** **static** **void** main (String [] args) {  SpringApplication. *Run* (SpringBootStudentSoapServiceApplication.**class**, args);  }  } |

**It uses static run method of** SpringApplication class.

**SpringApplication**: This class can be used to Bootstrap and launch a spring application from a java main method.

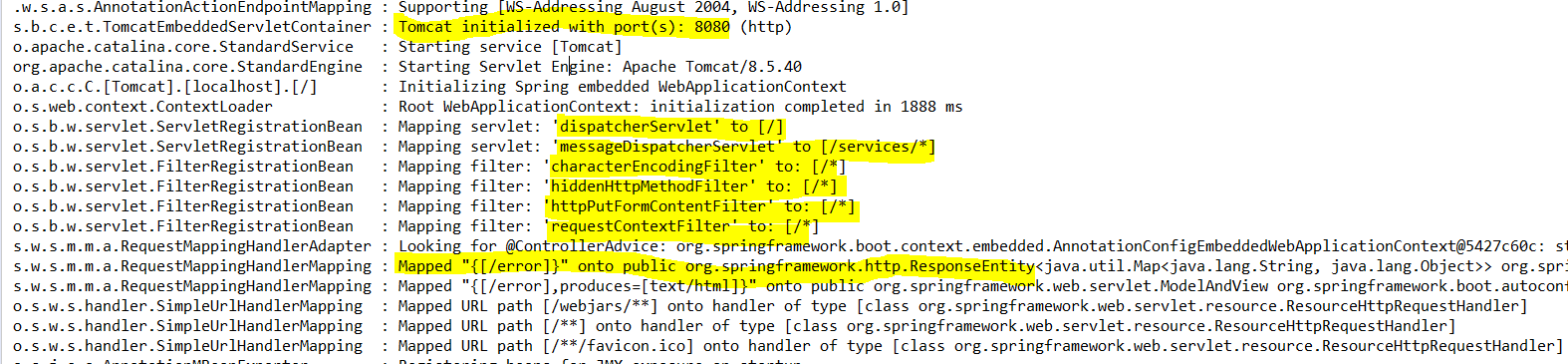
**@SpringBootApplication:** This annotation is the shortcut for three annotations:

1. @**Configuration**: Indicates that this a spring application context configuration file.
2. @**EnableAutoconfiguration**: Enable auto-configuration, an important feature of Spring Boot
3. @**ComponentScan**: Enables scanning for Spring beans in the package of this class and all its sub classes.

**Running our Spring Boot Application:**

**To run our SB application, in the Eclipse IDE, right click on the SB launch java class and run it as java Application.**

**The following are the key things to note:**



1. **Tomcat server is launched on port 8080**
2. **DispatcherServlet is configured. This means that Spring MVC framework is ready to accept request-** Mapping servlet: 'dispatcherServlet' to [/]
3. **MessageDispatcherServlet** is configured. This means service (REST/SAOP) is ready to accept request: Mapping servlet: 'messageDispatcherServlet' to [/services/\*]
4. **Four filters** characterEncodingFilter, hiddenHttpMethodFilter, httpPutFormContentFilter, requestContextFilter are enabled by default.
5. **Default error page is configured** Mapped "{[/error]}" onto public org. spring framework. http. ResponseEntity.
6. Similarly, WebJars for dependency management, & hander for http request are configured.

**Auto-configuration:**

To enable us to understand auto-configuration further let’s expand our Spring boot launcher class to include few more line of code:

|  |
| --- |
| **package** com.student.springbootstudentsoapservice;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.context.ApplicationContext;  @SpringBootApplication  **public** **class** SpringBootStudentSoapServiceApplication {  **public** **static** **void** main(String[] args) {  //SpringApplication.run(SpringBootStudentSoapServiceApplication.class, args);  **ApplicationContext ctx = SpringApplication.*run*(SpringBootStudentSoapServiceApplication.class, args);**  String[] beanNames = ctx.getBeanDefinitionNames();  System.***out***.println("All loaded Bean name is: -");  **for**(String beanName : beanNames) {  System.***out***.println(beanName);  }  }  } |

While run the above launcher class will print all the beans which are defined in the spring application context

|  |
| --- |
| All loaded Bean name is:-  org.springframework.context.annotation.internalConfigurationAnnotationProcessor  org.springframework.context.annotation.internalAutowiredAnnotationProcessor  org.springframework.context.annotation.internalRequiredAnnotationProcessor  org.springframework.context.annotation.internalCommonAnnotationProcessor  org.springframework.context.event.internalEventListenerProcessor  org.springframework.context.event.internalEventListenerFactory  springBootStudentSoapServiceApplication  org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory  org.springframework.boot.autoconfigure.AutoConfigurationPackages  org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration  org.springframework.boot.autoconfigure.condition.BeanTypeRegistry  propertySourcesPlaceholderConfigurer  org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$Jackson2ObjectMapperBuilderCustomizerConfiguration  standardJacksonObjectMapperBuilderCustomizer  spring.jackson-org.springframework.boot.autoconfigure.jackson.JacksonProperties  org.springframework.boot.context.properties.ConfigurationPropertiesBindingPostProcessor  org.springframework.boot.context.properties.ConfigurationPropertiesBindingPostProcessor.store  org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperBuilderConfiguration  jacksonObjectMapperBuilder  org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObjectMapperConfiguration  jacksonObjectMapper  org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration  jsonComponentModule  org.springframework.boot.autoconfigure.websocket.WebSocketAutoConfiguration$TomcatWebSocketConfiguration  websocketContainerCustomizer  org.springframework.boot.autoconfigure.websocket.WebSocketAutoConfiguration  org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration$EmbeddedTomcat  tomcatEmbeddedServletContainerFactory  org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration  embeddedServletContainerCustomizerBeanPostProcessor  errorPageRegistrarBeanPostProcessor  org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration$DispatcherServletConfiguration  dispatcherServlet  spring.mvc-org.springframework.boot.autoconfigure.web.WebMvcProperties  org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration$DispatcherServletRegistrationConfiguration  dispatcherServletRegistration  org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration  org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration  defaultValidator  methodValidationPostProcessor  org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration  error  beanNameViewResolver  org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$DefaultErrorViewResolverConfiguration  conventionErrorViewResolver  org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration  errorAttributes  basicErrorController  errorPageCustomizer  preserveErrorControllerTargetClassPostProcessor  spring.resources-org.springframework.boot.autoconfigure.web.ResourceProperties  org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$EnableWebMvcConfiguration  requestMappingHandlerAdapter  requestMappingHandlerMapping  mvcValidator  mvcContentNegotiationManager  mvcPathMatcher  mvcUrlPathHelper  viewControllerHandlerMapping  beanNameHandlerMapping  resourceHandlerMapping  mvcResourceUrlProvider  defaultServletHandlerMapping  mvcConversionService  mvcUriComponentsContributor  httpRequestHandlerAdapter  simpleControllerHandlerAdapter  handlerExceptionResolver  mvcViewResolver  mvcHandlerMappingIntrospector  org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter$FaviconConfiguration  faviconHandlerMapping  faviconRequestHandler  org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter  defaultViewResolver  viewResolver  welcomePageHandlerMapping  requestContextFilter  org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration  hiddenHttpMethodFilter  httpPutFormContentFilter  org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration  mbeanExporter  objectNamingStrategy  mbeanServer  org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration  org.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfiguration$StringHttpMessageConverterConfiguration  stringHttpMessageConverter  spring.http.encoding-org.springframework.boot.autoconfigure.web.HttpEncodingProperties  org.springframework.boot.autoconfigure.web.JacksonHttpMessageConvertersConfiguration$MappingJackson2HttpMessageConverterConfiguration  mappingJackson2HttpMessageConverter  org.springframework.boot.autoconfigure.web.JacksonHttpMessageConvertersConfiguration  org.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfiguration  messageConverters  org.springframework.boot.autoconfigure.info.ProjectInfoAutoConfiguration  spring.info-org.springframework.boot.autoconfigure.info.ProjectInfoProperties  org.springframework.boot.autoconfigure.web.HttpEncodingAutoConfiguration  characterEncodingFilter  localeCharsetMappingsCustomizer  org.springframework.boot.autoconfigure.web.MultipartAutoConfiguration  multipartConfigElement  multipartResolver  spring.http.multipart-org.springframework.boot.autoconfigure.web.MultipartProperties  org.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfiguration  serverProperties  duplicateServerPropertiesDetector  org.springframework.boot.autoconfigure.web.WebClientAutoConfiguration$RestTemplateConfiguration  restTemplateBuilder  org.springframework.boot.autoconfigure.web.WebClientAutoConfiguration  org.springframework.ws.config.annotation.DelegatingWsConfiguration  payloadRootAnnotationMethodEndpointMapping  soapActionAnnotationMethodEndpointMapping  annotationActionEndpointMapping  defaultMethodEndpointAdapter  soapFaultAnnotationExceptionResolver  simpleSoapExceptionResolver  org.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration$WsConfiguration  org.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration  messageDispatcherServlet  spring.webservices-org.springframework.boot.autoconfigure.webservices.WebServicesProperties |

Now the important question comes

1. Where are these beans are defined?
2. How these beans are created?

So that is the magic of Spring auto-configuration. Whenever we add a dependency in spring-boot-starter-web then following beans are auto-configured.

1. **DispatcherServlet**: front controller in Spring MVC
2. **viewResolver**: Resolve logical view name
3. **beanNameHandlerMapping**: Resolves paths to a handler
4. **mvcValidator**: supports validation of http request
5. **jacksonObjectMapper**: Translate object to JSON and JSON to object in REST service
6. **multipartResolver**: Support to upload files in web applications. …...and so on.

**Stater Projects and its descriptions**:

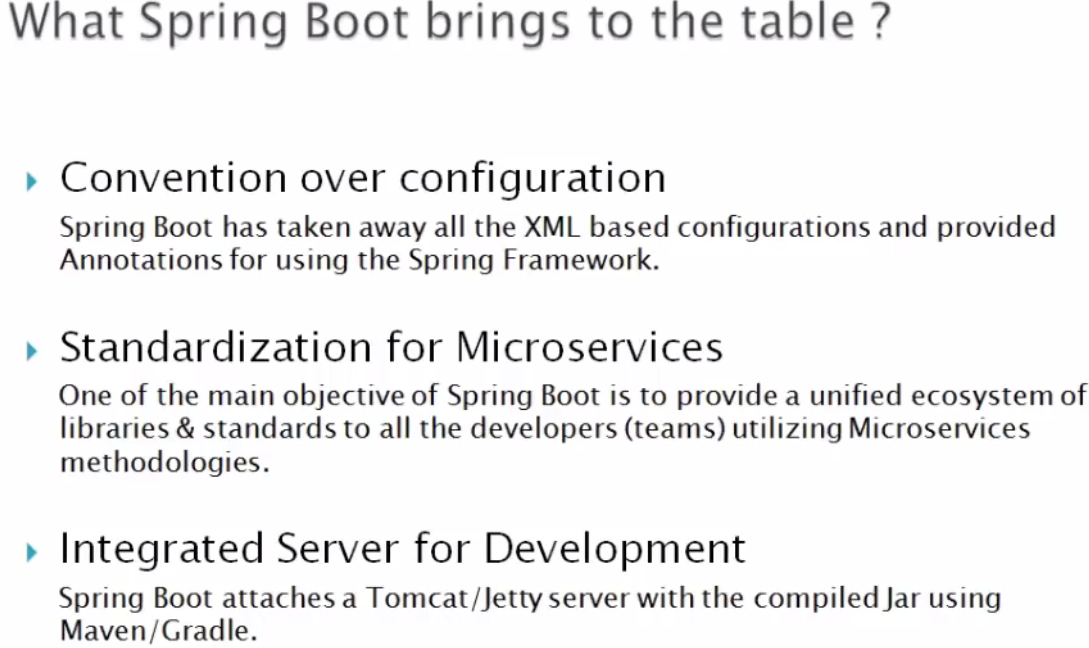
<https://www.javatpoint.com/spring-boot-starters>

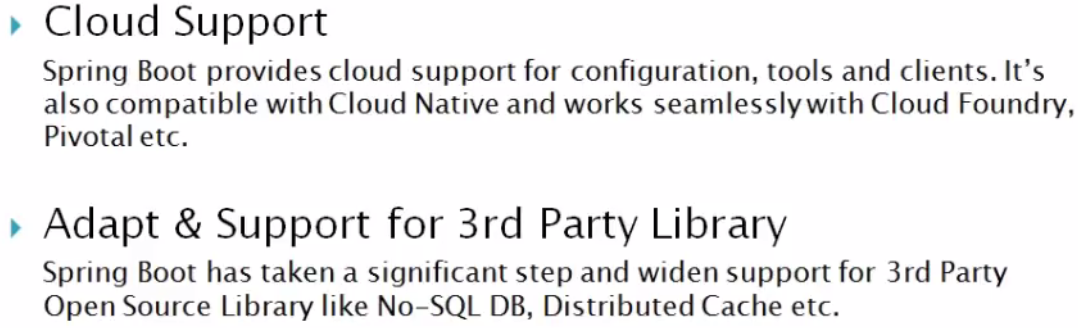
**Annotations used**

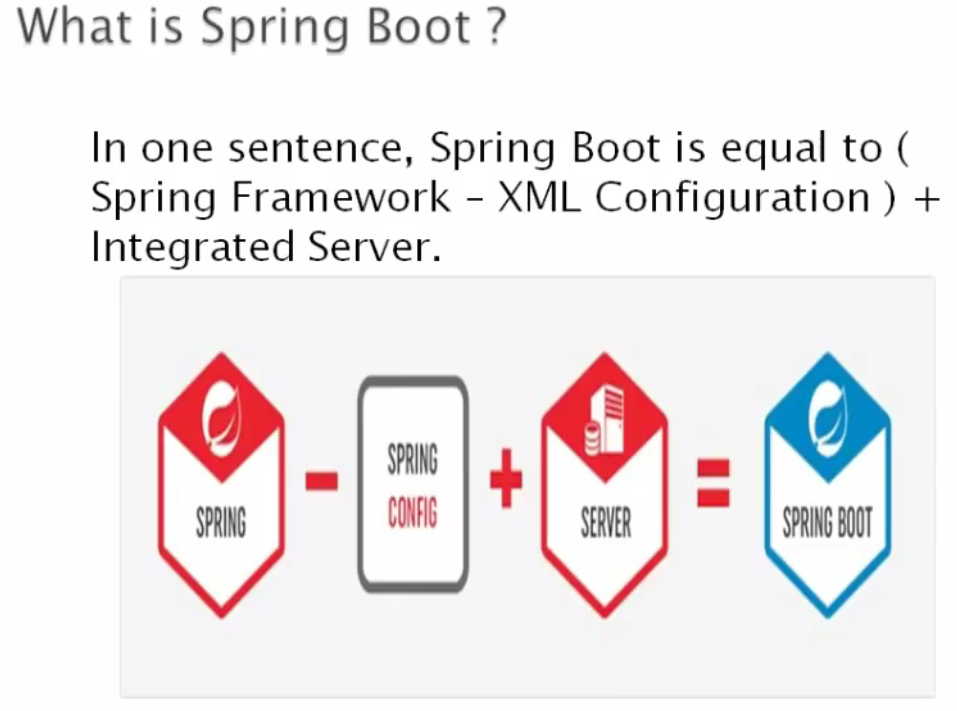
|  |  |
| --- | --- |
| @RestController | This annotation is the combination of @ResponseBody and @Controller annotation. This typically used to create REST controller in REST Service. |
| @GetMapping | This annotation is the shortcut of @RequestMapping (method=RequestMethod.GET) |
| @RunWith(SpringRunner.class) | SpringRunner is the shortcut to the SpringJUnit4ClassRunner annotation. This launches a simple spring context for unit testing. |
| @WebMvcTest(BasicController.class) | This annotation can be used along with SpringRunner to write test class for Spring MVC |

1. Even though Spring is a great framework, however it has some drawbacks. Spring XML based configuration is a nightmare in the world of annotation.

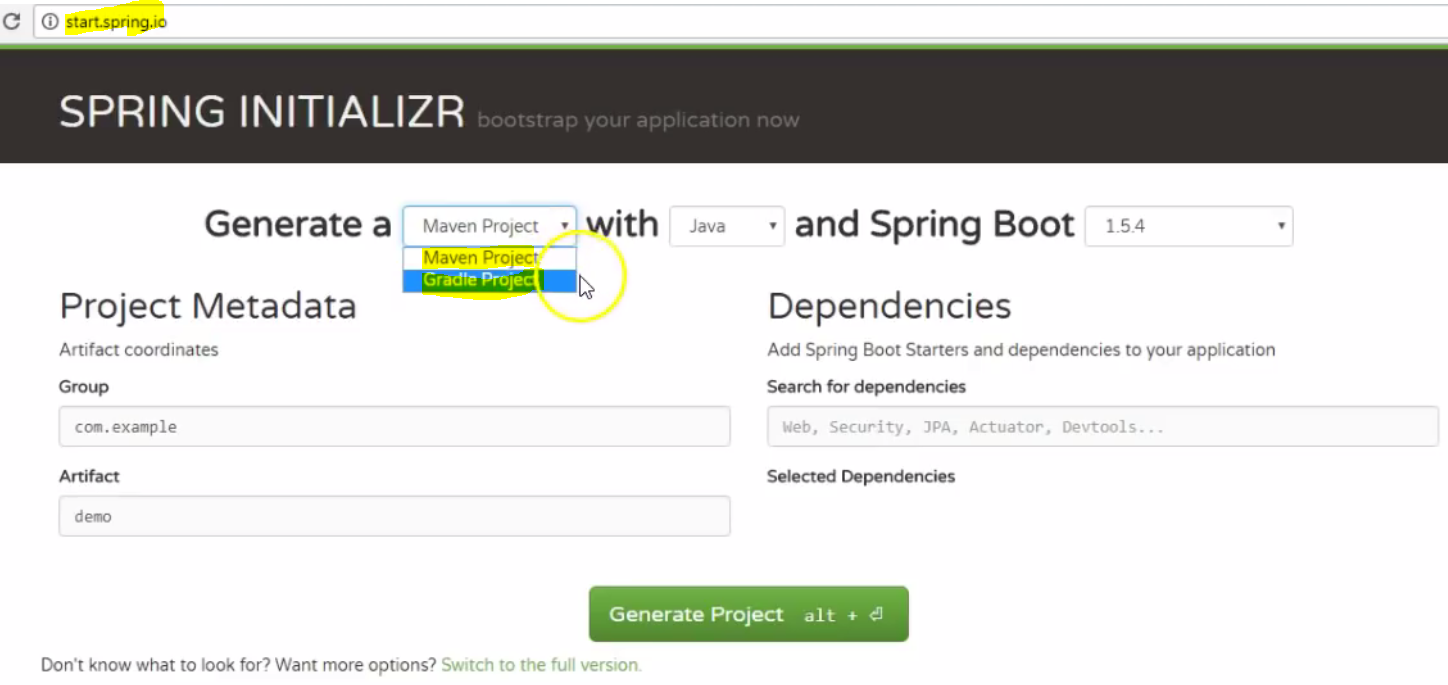
Spring boot support modern software architecture like microservices.







Using <https://start.spring.io> we can create spring boot project according to screen pasted below.



1. We can select the type of project (Maven or Gradle)
2. We can select the spring boot version.
3. We can add dependencies and module (web, JPA, MySQL).

