Spring Boot Auto-configuration

Spring Boot auto-configuration automatically configures the Spring application based on the jar dependencies that we have added.

We can enable the auto-configuration feature by using the annotation **@EnableAutoConfiguration**.

* When we add the **spring-boot-starter-web** dependency in the project, Spring Boot auto-configuration looks for the Spring MVC is on the classpath. It auto-configures **dispatcherServlet**, a default **error page,** and Web JARs for managing the static dependencies and default embedded server is **Tomcat**.

Similarly, when we add the **spring-boot-starter-data-jpa** dependency, we see that Spring Boot Auto-configuration, auto-configures a **datasource** and an **Entity Manager**.

* **@Component:** This annotation is the most generic annotation for any Spring-managed component. It is used to mark a class as a Spring bean that will be managed by the Spring container.

**SpringBoot**

Springboot is a framework that simplifies the development of spring based applications.

It provides a number of features that make it easier to create stand-alone,production-grade applications, such as

Auto-configuration

Starter dependencies

Embedded server

Metrics and Health checks

1. **What are the advantages of using Spring Boot?**

**------------------------------------------------------------**

* Easy to understand and develop spring applications.
* Spring Boot is nothing but an existing framework with the addition of an embedded HTTP server and annotation configuration which makes it easier to understand and faster the process of development.
* Increases productivity and reduces development time.
* Minimum configuration.
* We don’t need to write any XML configuration, only a few annotations are required to do the configuration.

**How does Spring Boot works?**

**---------------------------------------------**

Spring Boot automatically configures your application based on the dependencies you have added to the project by using annotation. The entry point of the spring boot application is the class that contains @SpringBootApplication annotation and the main method.

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

**What does the @SpringBootApplication annotation do internally?**

@SpringBootApplication automatically configures the application based on the dependencies added during project creation and bootstraps the application by using run() method inside the main class of an application.

The **@SpringBootApplication** annotation com

bines three annotations. Those three annotations are: **@Configuration, @EnableAutoConfiguration,**and **@ComponentScan**.

* **@AutoConfiguration**: This annotation automatically configuring beans in the class path and automatically scans the dependencies according to the application need.
* **@ComponentScan** annotation is used to tell Spring to scan a package and automatically detect Spring components, configurations, and services to configure.
* **@Configuration:** This annotation is used to indicate that a class contains configuration methods for the application context. It is typically used in combination with @Bean annotations to define beans and their dependencies.

Spring Boot enables the developer to use a single annotation instead of using multiple.

But, as we know, Spring provided loosely coupled features that we can use for each annotation as per our project needs.

**. What is the purpose of using @ComponentScan in the class files?**

**------------------------------------------------------------------------------**

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 to your project.

**8. How does a spring boot application get started?**

**----------------------------------------------------------------**

Just like any other Java program, a Spring Boot application must have a main method. This method serves as an entry point, which invokes the SpringApplication#run method to bootstrap the application.

**What is spring-boot-starter-parent?**

It is a special starter which makes Gradle or [Maven](https://www.guru99.com/maven-tutorial.html) dependency-management easy by adding jars to your classpath.

**What are starter dependencies?**

**Spring Boot Starters** are a collection of pre-configured maven dependencies that makes it easier to develop particular types of applications. These starters include,

* Dependencies
* Version control
* Configuration needed to make certain features.

To use a **Spring Boot starter dependency**, we simply need to add it to our project’s **pom.xml** file.

For example, to add the Spring Boot starter web dependency, add the following dependency to the pom.xml file:

-> for creating a web application.

<dependency>  
 <groupId>org.springframework.boot</groupId>   
 <artifactId>spring-boot-starter-web</artifactId>   
</dependency>

**What is Spring Initializer?**

**--------------------------------------------**

Spring Initializer is a web application that helps you to create an initial spring boot project structure and provides a maven or gradle file to build your code. It solves the problem of setting up a framework when you are starting a project from scratch.

**What is Spring Boot dependency management?**

**---------------------------------------------------------------**

**Spring Boot dependency management** makes it easier to manage dependencies in a Spring Boot project. It makes sure that all necessary dependencies are appropriate for the current Spring Boot version and are compatible with it.

-> without you specifying the version for any of that dependencies.

*To create a web application, we can add the S****pring Boot starter web dependency****to our application.*

**. Can we create a non-web application in Spring Boot?**

Yes, we can create a non-web application in Spring Boot. Spring Boot is not just for web applications. Using Spring Boot, we can create applications like Microservices, Console applications, and batch applications.

Yes, we can create a non-web application by removing the web dependencies from the classpath along with changing the way Spring Boot creates the application context.

**Describe the flow of HTTPS requests through the Spring Boot application.**

* First client makes an **HTTP request**(**GET, POST, PUT, DELETE**) to the browser.
* After that the request will go to the controller, where all the requests will be mapped and handled.
* After this in Service layer, all the **business logic** will be performed. It performs the business logic on the data that is mapped to **JPA (Java Persistence API)**using model classes.
* In repository layer, all the **CRUD** operations are being done for the **REST APIs**.
* A **JSP page** is returned to the end users if no errors are there.

**Is it possible to change the port of the embedded Tomcat server in Spring Boot?**

**-----------------------------------------------------------------------------------**

Yes, it is possible to change the port of the embedded Tomcat server in a Spring Boot application.

The simple way is to set the **server. port** property in your application’s **application.properties** file.

For example, to set the port to 8081, add the following property to the application.properties file:

server.port=8081

**What is the default port of tomcat in spring boot?**

The default port of the embedded Tomcat server in Spring Boot is **8080**. We can change the default port by setting the **server.port** property in your application’s **application.properties** file.

**Can we override or replace the Embedded tomcat server in Spring Boot?**

**---------------------------------------------------------------------------------------**

Yes, we can replace the Embedded Tomcat server with any server by using the Starter dependency in the **pom.xml** file. Like you can use spring-boot-starter-jetty as a dependency for using a jetty server in your project.

**Can we disable the default web server in the Spring Boot application?**

Yes, we can disable the default web server in the Spring Boot application. To do this, we need to set the **server.port** property to “-1” in the application’s **application.properties**file.

**How to disable a specific auto-configuration class?**

To disable a specific auto-configuration class in a Spring Boot application, we can use the **@EnableAutoConfiguration** annotation with the “**exclude”** attribute.

@EnableAutoConfiguration(exclude = {//classname})

**Explain @RestController annotation in Spring boot?**

* **@RestController:**This annotation is used to define a RESTful web service controller. It is a specialized version of the @Controller annotation that includes the @ResponseBody annotation by default.
* **@Controller**: Marks the class as a request handler in the Spring MVC framework.
* **@ResponseBody**: It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

It enables us to Define endpoints for different **HTTP methods (GET, POST, PUT, DELETE),**return data in various formats (JSON, XML, etc.) and map the request parameters to method arguments.

**What is the difference between @RestController and @Controller in Spring Boot?**

**-------------------------------------------------------------------------------------**

@Controller Map of the model object to view or template and make it human readable but,

@RestController simply returns the object and object data is directly written in HTTP response as JSON or XML.

**What is the difference between RequestMapping and GetMapping?**

**@RequestMapping:**This annotation is used to map HTTP requests to a specific handler method in your controller classes. It can be applied at the class level to define a base URL for all methods in the class, or at the method level to specify a specific URL mapping.

It supports mapping by:

* HTTP method – GET, POST, PUT, DELETE
* URL path
* URL parameters
* Request headers

RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation.

Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

**What is the use of Profiles in spring boot?**

**------------------------------------------------------**

While developing the application we deal with multiple environments such as dev, QA, Prod, and each environment requires a different configuration. For eg., we might be using an embedded H2 database for dev but for prod, we might have proprietary Oracle or DB2. Even if DBMS is the same across the environment, the URLs will be different.

To make this easy and clean, Spring has the provision of Profiles to keep the separate configuration of environments.

**Spring Profiles** are like different scenarios for the application depending on the environment.

* You define sets of configurations (like database URLs) for different situations (development, testing, production).
* Use the **@Profile** annotation to clarify which config belongs to where.
* Activate profiles with **environment variables** or **command-line** options.

To use Spring Profiles, we simply need to define the **spring.profiles.active** property to specify which profile we want to use.

**What is Spring Boot Actuator?**

Spring Boot Actuator is a sub-project of the spring boot framework.

. It includes a number of additional features that help us to monitor and manage the Spring Boot application.

. If we want to get production-ready features in an application, we should use the S**pring Boot actuator.**

By using actuators :

-> We can find the health of application

->No of beans loaded in application

->what config props loaded in application

-> No of threads running in application

->No of URL endpoints in application

*Note: To use Spring Boot Actuator, we simply need to add the****spring-boot-starter-actuator****dependency to our project.*

Below are the steps to enable actuator in Spring Boot Application:

* Add Actuator dependency.
* Enable endpoints in application.properties.
* Run your Spring Boot app.

Now we can access Actuator endpoints at URLs on the management port.

Actuators provide below pre-defined endpoints to monitor our application -

* Info
* Beans-> It will give all the beans which are loaded by application
* Mappings-> It will give all URL mappings available in application
* Config props->It will give config properties loaded by application.
* Heapdump
* Threaddump
* Shutdown
* Health-> It will give health status of application (up or down).
* Metrics-> it will give metrics of application
* Loggers
* Conditions -> Autoconfiguration classes , positive and negative matches
* Env-> get the all properties from properties file

Management.endpoints.web.exposure.include=\*

By default all endpoints comes in default context path of the application, suffixed with /actuator

We have existing endpoints in application starting with /actuator then

We can customize the base path to something else

-> management.endpoints.web.base-path=/manage

now we will be able to access all actuator endpoints under a new URL.

-> /manage/health

Customize the management server port:

Management.server.port=8090

Configuration Properties

management.endpoints.web.exposure.include=\*  
management.endpoints.web.base-path=/manage  
management.server.port=8090

Create Custom Endpoint

@Component  
@Endpoint(id = "customEndpoint")  
public class CustomActuatorEndpoint {  
  
 @ReadOperation  
 public String endpointData(){  
 return "This data came from the custom actuator endpoint....";  
 }  
  
}

**How to get the list of all the beans in your Spring boot application?**

Spring Boot actuator “/Beans” is used to get the list of all the spring beans in your application.

* Using the **ApplicationContext**object in Spring Boot, we can retrieve a list of all the beans in our application.
* The ApplicationContext is responsible for managing the beans and their dependencies.

**How to check the environment properties in your Spring boot application?**

Spring Boot actuator “/env” returns the list of all the environment properties of running the spring boot application.

Yes, we can check the environment properties in our Spring Boot Application. The Environment object in a Spring Boot application can be used to check the environment’s properties.

Configuration settings for the application, includes:

* property files
* command-line arguments
* environment variables

We can get the Environment instance by calling the **getEnvironment()** method.

**How to enable debugging log in the spring boot application?**

Debugging logs can be enabled in three ways -

* We can start the application with --debug switch.
* We can set the logging.level.root=debug property in application.property file.
* We can set the logging level of the root logger to debug in the supplied logging configuration file.

**Where do we define properties in the Spring Boot application?**

You can define both application and Spring boot-related properties into a file called application.properties. You can create this file manually or use Spring Initializer to create this file. You don’t need to do any special configuration to instruct Spring Boot to load this file, If it exists in classpath then spring boot automatically loads it and configure itself and the application code accordingly.

**What is dependency Injection?**

**Dependency Injection** (DI) is a design pattern that enables us to produce loosely coupled components.

In DI, an object’s ability to complete a task depends on another object.

There three types of dependency Injections.

* Setter Injection: The IOC container will inject the dependent bean object into the target bean object by calling the setter method.
* Constructor Injection: The IOC container will inject the dependent bean object into the target bean object by calling the target bean constructor.
* Field Injection: The IOC container will inject the dependent bean object into the target bean object by Reflection API.

The process of injecting dependent bean objects into target bean objects is called dependency injection.

**What is an IOC container?**

IoC Container is a framework for implementing automatic dependency injection. It manages object creation and its life-time and also injects dependencies into the class.

**Mention the differences between WAR and embedded containers.**

**---------------------------------------------------------------------------**

**WAR:**

Contains all of the files needed to deploy a web application to a web server.

Requires external configuration files (e.g., web.xml, context.xml) to define the web application.

**Embedded containers:**

It is a web application server included in the same JAR file as the application code.

Uses configuration properties or annotations within the application code.

**Explain Spring MVC**

**------------------------------------**

**MVC**stands for**Model, View,**and**Controller. Spring MVC**is a web MVC framework built on top of the Spring Framework. It provides a comprehensive programming model for building web applications

**What is Spring Bean?**

**--------------------------------------**

An object that is managed by the Spring IoC container is referred to as a spring bean. A Spring bean can be any Java object.

**What is Bean Wiring?**

**Bean wiring**is a mechanism in Spring that is used to manage the dependencies between beans. It allows Spring to inject collaborating beans into each other.

There are two types of Bean Wiring:

* Autowiring
* Manual wiring

**Mention the steps to connect the Spring Boot application to a database using JDBC.**

To connect an external database like MySQL or Oracle to a Spring Boot application using JDBC, we need to follow below steps:

* Add the dependency for the JDBC driver of the database.
* Create an application.properties file.
* Configure the database connection properties.
* Create a JdbcTemplate bean.
* Use the JdbcTemplate bean to execute SQL queries and statements.

**Mention the advantages of the YAML file over than Properties file and the different ways to load the YAML file in Spring boot.**

Advantages of YAML file over Properties file:

* Easy to edit and modify.
* Conciseness
* Supports Complex data types.

Different ways to load YAML file in Spring Boot:

* Using the @ConfigurationProperties annotation
* Using the YamlPropertiesFactoryBean class

**How is Hibernate chosen as the default implementation for JPA without any configuration?**

Spring Boot automatically configures **Hibernate** as the default JPA implementation when we add the **spring-boot-starter-data-jpa** dependency to our project. This dependency includes the Hibernate JAR file as well as the Spring Boot auto-configuration for JPA.

**What is DevTools in Spring Boot?**

Spring Boot DevTools helps you to increase the productivity of the developer. So, you don’t require to redeploy your application every time you make the changes. It allows the developer to reload changes without the need of restarting of the server.

* *What is Spring Boot?*
* *What are the advantages of using Spring Boot?*
* *What are the features of Spring Boot?*
* *How to create a Spring Boot application?*
* *What is the difference between Spring Boot and Spring Framework?*
* *What are the starter dependencies in Spring Boot?*
* *What is the purpose of the @SpringBootApplication annotation?*
* *What is the purpose of the @Configuration annotation?*
* *What is the purpose of the @Bean annotation?*
* *What is the purpose of the @Autowired annotation?*
* *What is the purpose of the @Value annotation?*
* *What is the purpose of the @Profile annotation?*
* *What is the purpose of the @EnableAutoConfiguration annotation?*
* *What is the default port of the embedded Tomcat server in Spring Boot?*
* *How to change the port of the embedded Tomcat server in Spring Boot?*
* *How to enable actuator in Spring Boot?*
* *How to access actuator endpoints in Spring Boot?*
* *Basics of Spring Boot*
* *Components of Spring Boot*
* *Create a Spring Boot application*
* *Configure Spring Boot application*
* *Run a Spring Boot application*
* *Deploy a Spring Boot application*
* *Best practices for developing Spring Boot applications*
* *What are the different ways to start a Spring Boot application?*
* *What are the different ways to configure Spring Boot applications?*
* *How to use Spring Boot starters?*
* *How to use Spring Boot actuator?*
* *How to use Spring Boot CLI?*
* *How to use Spring Boot in a microservices architecture?*
* *How to secure a Spring Boot application?*
* *How to troubleshoot Spring Boot applications?*
* **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
* **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
* **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
* **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
* **@PatchMapping:** It maps the **HTTP PATCH**requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
* **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
* **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
* **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
* **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.**For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.