### ****Q1.****[Spring](https://www.edureka.co/blog/spring-tutorial/)****vs Spring Boot****

|  |  |
| --- | --- |
| **Spring** | **Spring** Boot |
| A web application framework based on Java | A module of Spring |
| Provides tools and libraries to create customized web applications | Used to create a Spring application project which can just run/ execute |
| Spring is more complex than Spring Boot | Spring Boot is less complex than the Spring framework |
| Takes an unopinionated view | Takes an opinionated view of a platform |

### ****Q2. What is Spring Boot and mention the need for it?****

Spring Boot is a Spring module that aims to simplify the use of the Spring framework for Java development. It is used to create stand-alone Spring-based applications that you can just run. So, it basically removes a lot of configurations and dependencies. Aiming at the Rapid Application Development, Spring Boot framework comes with the auto-dependency resolution, embedded HTTP servers, auto-configuration, management endpoints, and [Spring Boot CLI](https://www.edureka.co/blog/spring-boot-setup-helloworld-microservices-example/).

So, if you ask me why should anybody use Spring Boot, then I would say, Spring Boot not only improves productivity but also provides a lot of conveniences to write your own business logic.

### ****Q3. Mention the advantages of Spring Boot****

The advantages of Spring Boot are as follows:

* Provides auto-configuration to load a set of default configuration for a quick start of the application
* Creates stand-alone applications with a range of non-functional features that are common to large classes of projects
* It comes with embedded tomcat, servlet containers jetty to avoid the usage of WAR files
* Spring Boot provides an opinionated view to reduce the developer effort and simplify maven configurations
* Provides CLI tool to develop and test applications
* Comes with Spring Boot starters to ensure dependency management and also provides various security metrics
* Consists of a wide range of APIs for monitoring and managing applications in dev and prod.
* Integrates with Spring Ecosystem like Spring [JDBC](https://www.edureka.co/blog/connect-mysql-database-in-java), Spring ORM, Spring Data, Spring Security easily by avoiding boilerplate code.

**Q4. Mention a few features of Spring Boot.**

Few important features of Spring Boot are as follows:

1. Spring CLI – Spring Boot CLI allows you to Groovy for writing Spring boot application and avoids boilerplate code.
2. Starter Dependency – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity
3. Auto-Configuration – The auto-configuration feature of Spring Boot helps in loading the default configurations according to the project you are working on. In this way, you can avoid any unnecessary WAR files.
4. Spring Initializer – This is basically a web application, which can create an internal project structure for you. So, you do not have to manually set up the structure of the project, instead, you can use this feature.
5. Spring Actuator –  This feature provides help while running Spring Boot applications.
6. Logging and Security – The logging and security feature of Spring Boot, ensures that all the applications made using Spring Boot are properly secured without any hassle.

**Q5. Explain how to create a Spring Boot application using Maven.**

Well, there are various approaches to [create a Spring Boot application](https://www.edureka.co/blog/microservices-with-spring-boot) using maven, but if I have to name a few, then following are the ways to create a Spring Boot project/ application using [maven](https://www.edureka.co/blog/maven-in-java/):

* Spring Boot CLI
* Spring Starter Project Wizard
* Spring Initializr
* Spring Maven Project

**Q6. Mention the possible sources of external configuration.**

There is no doubt in the fact that Spring Boot allows the developers to run the same application in different environments. Well, this is done with the support it provides for external configuration. It uses environment variables, properties files, command-line arguments, YAML files, and system properties to mention the required configuration properties. Also, the @value annotation is used to gain access to the properties. So, the most possible sources of external configuration are as follows:

* **Application Properties –** By default, Spring Boot searches for the application properties file or its YAML file in the current directory, classpath root or config directory to load the properties.
* **Command-line properties –** Spring Boot provides command-line arguments and converts these arguments to properties. Then it adds them to the set of environment properties.
* **Profile-specific properties –**  These properties are loaded from the application-{profile}.properties file or its YAML file. This file resides in the same location as that of the non-specific property files and the{profile} placeholder refers to an active profile.

**Q7. Can you explain what happens in the background when a Spring Boot Application is “Run as Java Application”?**

When a Spring Boot application is executed as “Run as Java application”, then it automatically launches up the tomcat server as soon as it sees, that you are developing a web application.

**Q8. What are the Spring Boot starters and what are available the starters?**

Spring Boot starters are a set of convenient dependency management providers that can be used in the application to enable dependencies. These starters, make development easy and rapid. All the available starters come under the org.springframework.boot group. Few of the popular starters are as follows:

* spring-boot-starter: – This is the core starter and includes logging, auto-configuration support, and YAML.
* spring-boot-starter-jdbc – This starter is used for HikariCP connection pool with JDBC
* spring-boot-starter-web – Is the starter for building web applications, including RESTful, applications using Spring MVC
* spring-boot-starter-data-jpa – Is the starter to use Spring Data JPA with Hibernate
* spring-boot-starter-security – Is the starter used for Spring Security
* spring-boot-starter-aop: This starter is used for aspect-oriented programming with AspectJ and  Spring AOP
* spring-boot-starter-test: Is the starter for testing Spring Boot applications

**Q9. Explain Spring Actuator and its advantages.**

Spring Actuator is a cool feature of Spring Boot with the help of which you can see what is happening inside a running application. So, whenever you want to debug your application, and need to analyze the logs you need to understand what is happening in the application right? In such a scenario, the Spring Actuator provides easy access to features such as identifying beans, CPU usage, etc. The Spring Actuator provides a very easy way to access the production-ready REST points and fetch all kinds of information from the web. These points are secured using Spring Security’s content negotiation strategy.

**Q10. What is Spring Boot dependency management?**

Spring Boot dependency management is basically used to manage dependencies and configuration automatically without you specifying the version for any of that dependencies.

**Q11. Mention the minimum requirements for a Spring boot System.**

Spring Boot 2.1.7.RELEASE requires

* Java 8 +
* Spring Framework 5.1.9 +

**Explicit build support**

* Maven 3.3+
* Gradle 4.4+

**Servlet Container Support**

* Tomcat 9.0 – Servlet Version 4.0
* Jetty 9.4 –  Servlet Version 3.1
* Undertow 2.0 – Servlet Version 4.0

**Q12. Explain what is thymeleaf and how to use thymeleaf?**

Thymeleaf is a server-side Java template engine used for web applications. It aims to bring natural template for your web application and can integrate well with Spring Framework and HTML5 Java web applications. To use Thymeleaf, you need to add the following code in the pom.xml file:

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

**Q13. Can we change the port of the embedded Tomcat server in Spring boot?**

Yes, we can change the port of the embedded tomcat server by using the application properties file. In this file, you have to add a property of “server.port” and assign it to any port you wish to. For example, if you want to assign it to 8081, then you have to mention server.port=8081. Once you mention the port number, the application properties file will be automatically loaded by Spring Boot and the required configurations will be applied on to the application.

**Q14. What is the need for Spring Boot DevTools?**

Spring Boot Dev Tools are an elaborated set of tools and aims to make the process of developing an application easier. If the application runs in the production, then this module is automatically disabled, repackaging of archives are also excluded by default. So, the Spring Boot Developer Tools applies properties to the respective development environments.  To include the DevTools, you just have to add the following dependency into the pom.xml file:

|  |
| --- |
| <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-devtools</artifactId>  </dependency> |

**Q15. Mention the steps to create a Spring Boot project using Spring  Initializer.**

Spring Initializr is a web tool provided by Spring. With the help of this tool, you can create Spring Boot projects by just providing project details. The following steps need to be followed to create a Spring Boot project using Spring Initializer:

* Choose the maven project and the required dependencies. Then, fill in the other required details like Group, Artifact, and then click on Generate Project.
* Once the project is downloaded, extract the project onto your system
* Next, you have to import this project using the import option on the Spring Tool Suite IDE
  + While importing the project, remember that you have to choose the project type to be Maven and the source project should contain the pom.xml file.

Once, all the above steps are followed you will see that the Spring Boot project is created with all the required dependencies.

**Spring** **Boot Interview Questions**

**Q16. Mention the steps to connect Spring Boot application to a database using JDBC.**

Spring Boot starter projects provide the required libraries to connect the application with JDBC. So, for example, if you just have to create an application  and connect it with [MySQL](https://www.edureka.co/blog/mysql-tutorial/) database, you can follow the below steps:

**Step 1:** Create a database in MySQL

|  |
| --- |
| CREATE DATABASE example; |

**Step 2:**Then you have to create a table inside this database.

|  |
| --- |
| CREATE TABLE customers(customerid INT PRIMARY KEY NOT NULL AUTO\_INCREMENT, customername VARCHAR(255)); |

**Step 3:** Now, create a Spring Boot project and provide the required details

**Step 4:** Add the JDBC, MySQL and web dependencies.

**Step 5:** Once the project is created, you have to configure the database into application properties

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/example  spring.datasource.username=root  spring.datasource.password=edureka  spring.jpa.hibernate.ddl-auto=create-drop |

**Step 6:** The main application.java class should have the following code:

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

**Step 7:** Next, you have to create a controller to handle the HTTP requests, by mentioning the following code:

|  |
| --- |
| package com.edureka;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jdbc.core.JdbcTemplate;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class JdbcController {  @Autowired  JdbcTemplate jdbc;  @RequestMapping("/insert")  public String index(){  jdbc.execute("insert into customers(name)values('Aryya')");  return "Data Entry Successful";  }  } |
|  |

**Step 8:** Finally, execute this project as a Java application.  
**Step 9:** Next, open the URL (localhost:8080/insert), and you will see the output as Data Entry Successful. You can also go forward and check if the data is entered into the table.

**Q17. How to enable HTTP/2 support in Spring Boot?**

You can enable the HTTP/2 support in Spring Boot by: server.http2.enabled=true

**Q18.  What are the @RequestMapping  and @RestController annotation in Spring Boot used for?**

|  |  |
| --- | --- |
| **@RequestMapping** | **@RestController** |
| This annotation is used to provide the routing information and tells to Spring that any HTTP request must be mapped to the respective method. | This annotation is used to add the @ResponseBody and @Controller annotation to the class |
| To use this annotation, you have to import org.springframework.web.  bind.annotation.RequestMapping; | To use this annotation, you have to import org.springframework.web.  bind.annotation.RestController; |

**Example:** Consider you have a method example() which should map with /example URL.

|  |
| --- |
| package com.edureka;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class SampleController {  @RequestMapping("/example")  public String example(){  return"Welcome To Edureka";  }  } |

**Q19. What is Spring Boot CLI and how to execute the Spring Boot project using boot CLI?**

Spring Boot CLI is a tool supported by the official [Spring Framework](https://www.edureka.co/blog/what-is-spring-framework/). The steps to execute a Spring Boot project are as follows:

* Download the CLI tool from the official site and extract the zip file. The bin folder present in the Spring setup is used to execute the Spring Boot application.
* Since Spring Boot CLI executes groovy files, you need to create a groovy file for Spring Boot application. So, to do that, open terminal and change the current directory to the bin folder. Now, open a groovy file (for example Sample.groovy)
* In this file create a controller as follows:

@RestController public class Sample {

@RequestMapping("/example")

String index(){

<h1>"Welcome To Edureka"</h1>;

} }

* Then execute the groovy file by mentioning:

|  |  |
| --- | --- |
| 1 | ./spring run Sample.groovy; |

Once, the project is executed go to the URL(localhost:8080:/example) and you will see the output as **Welcome To Edureka**

In case you are facing any challenges with these Spring Boot interview questions, please comment on your problems in the comment section below.

**Q20. Mention the differences between JPA and**[**Hibernate.**](https://www.edureka.co/blog/what-is-hibernate-in-java/)

|  |  |
| --- | --- |
| **JPA** | **Hibernate** |
| JPA is a Data Access Abstraction used to reduce the amount of boilerplate code | Hibernate is an implementation of Java Persistence API and offers benefits of loose coupling |

**Q21. How can we create a custom endpoint in Spring Boot Actuator?**

To create a custom endpoint in Spring Boot 2.x, you can use the @Endpoint annotation. Spring Boot also exposes endpoints using @WebEndpointor, @WebEndpointExtension over HTTP with the help of [Spring MVC](https://www.edureka.co/blog/spring-mvc-tutorial/), [Jersey](https://www.edureka.co/blog/java-web-services-tutorial/), etc.

**Q22. Explain Spring Data.**

Spring Data aims to make it easy for the developers to use relational and non-relational databases, cloud-based data services, and other data access technologies. So, basically, it makes it easy for data access and still retains the underlying data.

**Q23. What do you understand by auto-configuration in Spring Boot and how to disable the auto-configuration?**

Auto-configuration is used to automatically configure the required configuration for the application. For example, if you have a data source bean present in the classpath of the application, then it automatically configures the [JDBC template](https://www.edureka.co/blog/connect-mysql-database-in-java). With the help of auto-configuration, you can create a Java application in an easy way, as it automatically configures the required beans, controllers, etc.

To disable the auto-configuration property, you have to exclude attribute of @EnableAutoConfiguration, in the scenario where you do not want it to be applied.

|  |
| --- |
| @EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class}) |

If the class is not on the classpath, then to exclude the auto-configuration, you have to mention the following code:

|  |
| --- |
| @EnableAutoConfiguration(excludeName={Sample.class}) |

Apart from this, Spring Boot also provides the facility to exclude list of auto-configuration classes by using the spring.autoconfigure.exclude property. You can go forward, and add it either in the application.properties or add multiple classes with comma-separated.

**Q24. What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?**

|  |  |
| --- | --- |
| **@SpringBootApplication** | **@EnableAutoConfiguration** |
| Used in the main class or bootstrap class | Used to enable auto-configuration  and component scanning in your project |
| It is a combination of @Configuration, @ComponentScan and @EnableAutoConfiguration annotations. | It is a combination of @Configuration and @ComponentScan annotations |

**Q25. What are the steps to deploy Spring Boot web applications as JAR and WAR files?**

To deploy a Spring Boot web application, you just have to add the following plugin in the pom.xml file:

|  |
| --- |
| <plugin>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-maven-plugin</artifactId>  </plugin> |

By using the above plugin, you will get a JAR executing the package phase. This JAR will contain all the necessary libraries and dependencies required. It will also contain an embedded server. So, you can basically run the application like an ordinary JAR file.  
**Note:** The packaging element in the pom.xml file must be set to **jar** to build a JAR file as below:

|  |
| --- |
| <packaging>jar</packaging> |

Similarly, if you want to build a WAR file, then you will mention

|  |
| --- |
| <packaging>war</packaging> |

**Q.26 Can you give an example for ReadOnly as true in Transaction management?**

Example for ReadOnly as TRUE in transaction management could be as follows:

Consider a scenario, where you have to read data from the database. For example, let us say you have a customer database, and you want to read the customer details such as customerID, and customername. To do that, you will set**read-only on the transaction** as we do not want to check for the changes in the entities.

**Q27. Can you explain how to deploy to a different server with Spring Boot?**

To deploy a different server with Spring Boot, follow the below steps:

* Generate a WAR from the project
* Then, deploy the WAR file onto your favorite server

Note: The steps to deploy the WAR file on the server is dependent on the server you choose.

**Q28: What is the best way to expose custom application configuration with Spring Boot?**

One way to expose the custom application [configuration in Spring](https://www.edureka.co/blog/spring-tutorial/) Boot is by using the **@Value annotation**. But, the only problem with this annotation is that all the configuration values will be distributed throughout the application. Instead, you can use a centralized approach.

By centralized approach, I mean that you can define a configuration component using the @ConfigurationProperties as follows:

|  |
| --- |
| @Component  @ConfigurationProperties("example")  public class SampleConfiguration {  private int number;  private boolean value;  private String message; |

According to the above snippet, the values configured in application.properties will be as follows:

|  |
| --- |
| example.number: 100  example.value: true  example.message: Dynamic Message |

**Q29. Can we create a non-web application in Spring Boot?**

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.

**Q 30. What are the steps to connect an external database like MySQL or Oracle?**

To connect an external database, you have to follow the below steps:

* Start by adding the dependency for MySQL Connector to pom.xml
* Then remove H2 Dependency from pom.xml
* Now, set up your [MySQL database](https://www.edureka.co/blog/mysql-tutorial/) and configure your connection to the MySQL database
* Restart your project

**Spring** **Boot Interview Questions**

**Q31. Mention the advantages of the YAML file than Properties file and the different ways to load YAML file in Spring boot.**

The advantages of the YAML file than a properties file is that the data is stored in a hierarchical format. So, it becomes very easy for the developers to debug if there is an issue. The SpringApplication class supports the YAML file as an alternative to properties whenever you use the SnakeYAML library on your classpath. The different ways to load a YAML file in Spring Boot is as follows:

* Use YamlMapFactoryBean to load YAML as a Map
* Use YamlPropertiesFactoryBean to load YAML as Properties

**Q32. How is Hibernate chosen as the default implementation for JPA without any configuration?**

When we use the Spring Boot Auto Configuration, automatically the spring**-boot-starter-data-jpa**dependency gets added to the pom.xml file. Now, since this dependency has a transitive dependency on JPA and Hibernate, Spring Boot automatically auto-configures Hibernate as the default implementation for JPA, whenever it sees Hibernate in the classpath.

**Q33. What do you understand by Spring Data REST?**

Spring Data REST is used to expose the RESTful resources around Spring Data repositories. Consider the following example:

|  |
| --- |
| @RepositoryRestResource(collectionResourceRel = "sample", path = "sample")  public interface SampleRepository          extends CustomerRepository<sample, Long> { |

Now, to expose the REST services, you can use the POST method in the following way:

|  |
| --- |
| {  "customername": "Rohit"  } |

Response Content

|  |
| --- |
| {  "customername": "Rohit"  "\_links": {  "self": {  "href": "<a href="http://localhost:8080/sample/1">http://localhost:8080/sample/1</a>"  },  "sample": {  "href": "<a href="http://localhost:8080/sample/1">http://localhost:8080/sample/1</a>"  }  } |

Observe that the response content contains the href of the newly created resource.

**Q34. What is the difference between RequestMapping and GetMapping?**

The @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). Both these methods support the consumes. The consume options are :

consumes = “text/plain”  
consumes = {“text/plain”, “application/\*”}

**Q35. In which layer, should the boundary of a transaction start?**

The boundary of the transaction should start from the Service Layer since the logic for the business transaction is present in this layer itself.

**Q36.** **How does path=”sample”, collectionResourceRel=”sample” work with Spring Data Rest?**

|  |
| --- |
| @RepositoryRestResource(collectionResourceRel = "sample", path = "sample")  public interface SampleRepository extends  PagingAndSortingRepository<Sample, Long> |

* path – This section is used to mention the segment under which the resource is to be exported.
* collectionResourceRel – This value is used to generate links to the collection resource.

**Q37. Explain how to register a custom auto-configuration.**

In order to register an auto-configuration class, you have to mention the fully-qualified name under the @EnableAutoConfiguration key META-INF/spring. factories file. Also, if we build the with maven, then this file should be placed in the resources/META-INT directory.

**Q38. How do you Configure Log4j for logging?**

Since Spring Boot supports Log4j2 for logging a configuration, you have to exclude Logback and include Log4j2 for logging. This can be only done if you are using the starters project.

**Q39. Mention the differences between WAR and embedded containers**

|  |  |
| --- | --- |
| WAR | Embedded Containers |
| WAR benefits a considerable measure from Spring Boot | Only one component of Spring Boot and is utilized during improvements |

**Q40. What do you think is the need for Profiles?**

Profiles are used to provide a way to segregate the different parts of the application configuration and make it available for various environments. So, basically, any @Component or a @Configuration can be marked with a @Profile to limit as it is loaded. Consider you have multiple environments,

* Dev
* QA
* Stage
* Production

Now, let’s say, you want to have different application configuration in each of the environments, you can use profiles to have different application configurations for different environments. So, basically, Spring and Spring Boot provide features through which you can specify:

* The active profile for a specific environment
* The configuration of various environments for various profiles.

**Q41. What are the steps to add a custom JS code with Spring Boot?**

The steps to add a [custom JS code](https://www.edureka.co/blog/javascript-tutorial/) with Spring Boot are as follows:

* Now, create a folder and name it **static** under the resources folder
* In this folder, you can put the static content in that folder

**Note:** Just in case, the browser throws an unauthorized error, you either disable the security or search for the password in the log file, and eventually pass it in the request header.

**Q42. How to instruct an auto-configuration to back off when a bean exists?**

To instruct an auto-configuration class to back off when a bean exists, you have to use the @ConditionalOnMissingBean annotation. The attributes of this annotation are as follows:

* **value:** This attribute stores the type of beans to be checked
* **name:** This attribute stores the name of beans to be checked

**Q43. Why is Spring Data REST not recommended in real-world applications?**

Spring Data REST is not recommended in real-world applications as you are exposing your database entities directly as [REST Services](https://www.edureka.co/blog/what-is-rest-api/). While designing RESTful services, the two most important things that we consider is the domain model and the consumers. But, while using Spring Data REST, none of these parameters are considered. The entities are directly exposed. So, I would just say, you can use Spring Data REST, for the initial evolution of the project.

**Q44. What is the error you see if  H2 is not in the classpath?**

If H2 is not present in the classpath, then you see the following error:

Cannot determine embedded database driver class for database type NONE

To resolve this error, add H2 to the pom.xml file, and restart your server.  
The following code snippet can be added to add the dependency:

|  |
| --- |
| <dependency>      <groupId>com.h2database</groupId>      <artifactId>h2</artifactId>      <scope>runtime</scope>  </dependency> |

**Q45. What is the way to use profiles to configure the environment-specific configuration with Spring Boot?**

Since it is a known fact that a Profile is nothing but a key to identify an environment lets consider the following two profiles in the example:

* dev
* prod
* Consider the following properties present in the application properties file:

example.number: 100  
example.value: true  
example.message: Dynamic Message

Now, say you want to customize the application.properties for dev profile, then you need to create a file with name application-dev.properties and override the properties that you want to customize. You can mention the following code:

example.message: Dynamic Message in Dev

Similarly, if you want to customize the application.properties for prod profile, then you can mention the following code snippet:

example.message: Dynamic Message in Prod

Once you are done with the profile-specific configuration, you have to set the active profile in an environment. To do that, either you can

* Use -Dspring.profiles.active=prod in  arguments
* Use spring.profiles.active=prod in application.properties file

**Q46. Mention the dependencies needed to start up a JPA Application and connect to in-memory database H2 with Spring Boot?**

The dependencies are needed to start up a JPA Application and connect to in-memory database H2 with Spring Boot

* web starter
* h2
* data JPA starter
* To include the dependencies refer to the following code:

|  |
| --- |
| <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>      <groupId>com.h2database</groupId>      <artifactId>h2</artifactId>      <scope>runtime</scope>  </dependency>  <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency> |

**Q47. What do you understand by Spring Boot supports relaxed binding?**

Relaxed binding, is a way in which, the property name does not need to match the key of the environment property. In Spring Boot, relaxed binding is applicable to the type-safe binding of the configuration properties. For example, if a property in a bean class with the @ConfigurationPropertie annotation is used sampleProp, then it can be bounded to any of the following environment properties:

* sampleProp
* sample-Prop
* sample\_Prop
* SAMPLE\_PROP

**Q48.  Where is the database connection information specified and how does it automatically connect to H2?**

Well, the answer to this question is very simple. It is because of the Spring Boot auto-configuration that, configures the dependencies of the application. So, the database connection information, and automatically connecting the database to H2 is done by the auto-configuration property.

**Q49. What is the name of the default H2 database configured by Spring Boot?**

The name of the default H2 database is **testdb.  Refer below:**

spring.datasource.name=testdb # Name of the datasource.

**Note:** Just incase if you are using H2 in-memory database, then exactly that is the name of Spring Boot which is used to setup your H2 database.

**Spring** **Boot Interview Questions**

**Q50. Do you think, you can use jetty instead of tomcat in spring-boot-starter-web?**

Yes, we can use jetty instead of tomcat in spring-boot-starter-web, by removing the existing dependency and including the following:

|  |
| --- |
| <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-web</artifactId>      <exclusions>          <exclusion>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-tomcat</artifactId>          </exclusion>      </exclusions>  </dependency>  <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-jetty</artifactId>  </dependency> |

**1) What is Spring Boot?**

Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.

It is used to create stand alone spring based application that you can just run because it needs very little spring configuration.

**2) What are the advantages of Spring Boot?**

Create stand-alone Spring applications that can be started using java -jar.

Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.

It provides opinionated 'starter' POMs to simplify your Maven configuration.

It automatically configure Spring whenever possible.

**3) What are the features of Spring Boot?**

Web Development

SpringApplication

Application events and listeners

Admin features

**4) How to create Spring Boot application using Maven?**

There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.

Spring Maven Project

Spring Starter Project Wizard

Spring Initializr

Spring Boot CLI

For more information click here.

**5) How to create Spring Boot project using Spring Initializer?**

It is a web tool which is provided by Spring on official site. You can create Spring Boot project by providing project details.

For more information click here.

**6) How to create Spring Boot project using boot CLI?**

It is a tool which you can download from the official site of Spring Framework. Here, we are explaining steps.

Download the CLI tool from official site and For more information click here.

**7) How to create simple Spring Boot application?**

To create an application. We are using STS (Spring Tool Suite) IDE and it includes the various steps that are explaining in steps.

For more information click here.

**8) What are the Spring Boot Annotations?**

The @RestController is a stereotype annotation. It adds @Controller and @ResponseBody annotations to the class. We need to import org.springframework.web.bind.annotation package in our file, in order to implement it.

For more information click here.

**9) What is Spring Boot dependency management?**

Spring Boot manages dependencies and configuration automatically. You don't need to specify version for any of that dependencies.

Spring Boot upgrades all dependencies automatically when you upgrade Spring Boot.

For more information click here.

**10) What are the Spring Boot properties?**

Spring Boot provides various properties which can be specified inside our project's application.properties file. These properties have default values and you can set that inside the properties file. Properties are used to set values like: server-port number, database connection configuration etc.

For more information click here.

**11) What are the Spring Boot Starters?**

Starters are a set of convenient dependency descriptors which we can include in our application.

Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project.

For more information click here.

**12) What is Spring Boot Actuator?**

Spring Boot provides actuator to monitor and manage our application. Actuator is a tool which has HTTP endpoints. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

For more information click here.

**13) What is thymeleaf?**

It is a server side Java template engine for web application. It's main goal is to bring elegant natural templates to your web application.

It can be integrate with Spring Framework and ideal for HTML5 Java web applications.

For more information click here.

**14) How to use thymeleaf?**

In order to use Thymeleaf we must add it into our pom.xml file like:

<dependency>

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

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

</dependency>

For more information click here.

**15) How to connect Spring Boot to the database using JPA?**

Spring Boot provides spring-boot-starter-data-jpa starter to connect Spring application with relational database efficiently. You can use it into project POM (Project Object Model) file.

For more information click here.

16) How to connect Spring Boot application to database using JDBC?

Spring Boot provides starter and libraries for connecting to our application with JDBC. Here, we are creating an application which connects with Mysql database. It includes the following steps to create and setup JDBC with Spring Boot.

For more information click here.

**17) What is @RestController annotation in Spring Boot?**

The @RestController is a stereotype annotation. It adds @Controller and @ResponseBody annotations to the class. We need to import org.springframework.web.bind.annotation package in our file, in order to implement it.

For more information click here.

**18) What is @RequestMapping annotation in Spring Boot?**

The @RequestMapping annotation is used to provide routing information. It tells to the Spring that any HTTP request should map to the corresponding method. We need to import org.springframework.web.annotation package in our file.

For more information click here.

**19) How to create Spring Boot application using Spring Starter Project Wizard?**

There is one more way to create Spring Boot project in STS (Spring Tool Suite). Creating project by using IDE is always a convenient way. Follow the following steps in order to create a Spring Boot Application by using this wizard.

For more information click here.

**20) Spring Vs Spring Boot?**

Spring is a web application framework based on Java. It provides tools and libraries to create a complete cutomized web application.

Wheras Spring Boot is a spring module which is used to create spring application project that can just run.

**Q1. What is Spring Boot and What Are Its Main Features?**

Spring Boot is essentially a framework for rapid application development built on top of the Spring Framework. With its auto-configuration and embedded application server support, combined with the extensive documentation and community support it enjoys, Spring Boot is one of the most popular technologies in the Java ecosystem as of date.

Here are a few salient features:

Starters – a set of dependency descriptors to include relevant dependencies at a go

Auto-configuration – a way to automatically configure an application based on the dependencies present on the classpath

Actuator – to get production-ready features such as monitoring

Security

Logging

**Q2. What Are the Differences Between Spring and Spring Boot?**

The Spring Framework provides multiple features that make the development of web applications easier. These features include dependency injection, data binding, aspect-oriented programming, data access, and many more.

Over the years, Spring has been growing more and more complex, and the amount of configuration such application requires can be intimidating. This is where Spring Boot comes in handy – it makes configuring a Spring application a breeze.

Essentially, while Spring is unopinionated, Spring Boot takes an opinionated view of the platform and libraries, letting us get started quickly.

Here are two of the most important benefits Spring Boot brings in:

Auto-configure applications based on the artifacts it finds on the classpath

Provide non-functional features common to applications in production, such as security or health checks

Please check one of our other tutorials for a detailed comparison between vanilla Spring and Spring Boot.

**Q3. How Can We Set up a Spring Boot Application With Maven?**

We can include Spring Boot in a Maven project just like we would any other library. However, the best way is to inherit from the spring-boot-starter-parent project and declare dependencies to Spring Boot starters. Doing this lets our project reuse the default settings of Spring Boot.

Inheriting the spring-boot-starter-parent project is straightforward – we only need to specify a parent element in pom.xml:

<parent>

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

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

<version>2.4.0.RELEASE</version>

</parent>

We can find the latest version of spring-boot-starter-parent on Maven Central.

Using the starter parent project is convenient, but not always feasible. For instance, if our company requires all projects to inherit from a standard POM, we can still benefit from Spring Boot's dependency management using a custom parent.

**Q4. What is Spring Initializr?**

Spring Initializr is a convenient way to create a Spring Boot project.

We can go to the Spring Initializr site, choose a dependency management tool (either Maven or Gradle), a language (Java, Kotlin or Groovy), a packaging scheme (Jar or War), version and dependencies, and download the project.

This creates a skeleton project for us and saves setup time so that we can concentrate on adding business logic.

Even when we use our IDE's (such as STS or Eclipse with STS plugin) new project wizard to create a Spring Boot project, it uses Spring Initializr under the hood.

**Q5. What Spring Boot Starters Are Available out There?**

Each starter plays a role as a one-stop-shop for all the Spring technologies we need. Other required dependencies are then transitively pulled in and managed in a consistent way.

All starters are under the org.springframework.boot group and their names start with spring-boot-starter-. This naming pattern makes it easy to find starters, especially when working with IDEs that support searching dependencies by name.

At the time of this writing, there are more than 50 starters at our disposal. The most commonly used are:

spring-boot-starter: core starter, including auto-configuration support, logging, and YAML

spring-boot-starter-aop: starter for aspect-oriented programming with Spring AOP and AspectJ

spring-boot-starter-data-jpa: starter for using Spring Data JPA with Hibernate

spring-boot-starter-security: starter for using Spring Security

spring-boot-starter-test: starter for testing Spring Boot applications

spring-boot-starter-web: starter for building web, including RESTful, applications using Spring MVC

For a complete list of starters, please see this repository.

To find more information about Spring Boot starters, take a look at Intro to Spring Boot Starters.

**Q6. How to Disable a Specific Auto-Configuration?**

If we want to disable a specific auto-configuration, we can indicate it using the exclude attribute of the @EnableAutoConfiguration annotation. For instance, this code snippet neutralizes DataSourceAutoConfiguration:

// other annotations

@EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)

public class MyConfiguration { }

If we enabled auto-configuration with the @SpringBootApplication annotation — which has @EnableAutoConfiguration as a meta-annotation — we could disable auto-configuration with an attribute of the same name:

// other annotations

@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)

public class MyConfiguration { }

We can also disable an auto-configuration with the spring.autoconfigure.exclude environment property. This setting in the application.properties file does the same thing as before:

spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration

**Q7. How to Register a Custom Auto-Configuration?**

To register an auto-configuration class, we must have its fully-qualified name listed under the EnableAutoConfiguration key in the META-INF/spring.factories file:

org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.baeldung.autoconfigure.CustomAutoConfiguration

If we build a project with Maven, that file should be placed in the resources/META-INF directory, which will end up in the mentioned location during the package phase.

**Q8. How to Tell an Auto-Configuration to Back Away When a Bean Exists?**

To instruct an auto-configuration class to back off when a bean is already existent, we can use the @ConditionalOnMissingBean annotation. The most noticeable attributes of this annotation are:

value: The types of beans to be checked

name: The names of beans to be checked

When placed on a method adorned with @Bean, the target type defaults to the method's return type:

@Configuration

public class CustomConfiguration {

@Bean

@ConditionalOnMissingBean

public CustomService service() { ... }

}

**Q9. How to Deploy Spring Boot Web Applications as Jar and War Files?**

Traditionally, we package a web application as a WAR file, then deploy it into an external server. Doing this allows us to arrange multiple applications on the same server. During the time that CPU and memory were scarce, this was a great way to save resources.

However, things have changed. Computer hardware is fairly cheap now, and the attention has turned to server configuration. A small mistake in configuring the server during deployment may lead to catastrophic consequences.

Spring tackles this problem by providing a plugin, namely spring-boot-maven-plugin, to package a web application as an executable JAR. To include this plugin, just add a plugin element to pom.xml:

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

With this plugin in place, we'll get a fat JAR after executing the package phase. This JAR contains all the necessary dependencies, including an embedded server. Thus, we no longer need to worry about configuring an external server.

We can then run the application just like we would an ordinary executable JAR.

Notice that the packaging element in the pom.xml file must be set to jar to build a JAR file:

<packaging>jar</packaging>

If we don't include this element, it also defaults to jar.

In case we want to build a WAR file, change the packaging element to war:

<packaging>war</packaging>

And leave the container dependency off the packaged file:

<dependency>

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

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

<scope>provided</scope>

</dependency>

After executing the Maven package phase, we'll have a deployable WAR file.

**Q10. How to Use Spring Boot for Command Line Applications?**

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

@SpringBootApplication

public class MyApplication {

public static void main(String[] args) {

SpringApplication.run(MyApplication.class);

// other statements

}

}

The SpringApplication class then fires up a Spring container and auto-configures beans.

Notice we must pass a configuration class to the run method to work as the primary configuration source. By convention, this argument is the entry class itself.

After calling the run method, we can execute other statements as in a regular program.

**Q11. What Are Possible Sources of External Configuration?**

Spring Boot provides support for external configuration, allowing us to run the same application in various environments. We can use properties files, YAML files, environment variables, system properties, and command-line option arguments to specify configuration properties.

We can then gain access to those properties using the @Value annotation, a bound object via the @ConfigurationProperties annotation, or the Environment abstraction.

**Q12. What Does it Mean that Spring Boot Supports Relaxed Binding?**

Relaxed binding in Spring Boot is applicable to the type-safe binding of configuration properties.

With relaxed binding, the key of a property doesn't need to be an exact match of a property name. Such an environment property can be written in camelCase, kebab-case, snake\_case, or in uppercase with words separated by underscores.

For example, if a property in a bean class with the @ConfigurationProperties annotation is named myProp, it can be bound to any of these environment properties: myProp, my-prop, my\_prop, or MY\_PROP.

**Q13. What is Spring Boot Devtools Used For?**

Spring Boot Developer Tools, or DevTools, is a set of tools making the development process easier. To include these development-time features, we just need to add a dependency to the pom.xml file:

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

The spring-boot-devtools module is automatically disabled if the application runs in production. The repackaging of archives also excludes this module by default. Hence, it won't bring any overhead to our final product.

By default, DevTools applies properties suitable to a development environment. These properties disable template caching, enable debug logging for the web group, and so on. As a result, we have this sensible development-time configuration without setting any properties.

Applications using DevTools restart whenever a file on the classpath changes. This is a very helpful feature in development, as it gives quick feedback for modifications.

By default, static resources, including view templates, don't set off a restart. Instead, a resource change triggers a browser refresh. Notice this can only happen if the LiveReload extension is installed in the browser to interact with the embedded LiveReload server that DevTools contains.

For further information on this topic, please see Overview of Spring Boot DevTools.

**Q14. How to Write Integration Tests?**

When running integration tests for a Spring application, we must have an ApplicationContext.

To make our life easier, Spring Boot provides a special annotation for testing – @SpringBootTest. This annotation creates an ApplicationContext from configuration classes indicated by its classes attribute.

In case the classes attribute isn't set, Spring Boot searches for the primary configuration class. The search starts from the package containing the test up until it finds a class annotated with @SpringBootApplication or @SpringBootConfiguration.

For detailed instructions, check out our tutorial on testing in Spring Boot.

**Q15. What Is Spring Boot Actuator Used For?**

Essentially, Actuator brings Spring Boot applications to life by enabling production-ready features. These features allow us to monitor and manage applications when they're running in production.

Integrating Spring Boot Actuator into a project is very simple. All we need to do is to include the spring-boot-starter-actuator starter in the pom.xml file:

<dependency>

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

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

</dependency>

Spring Boot Actuator can expose operational information using either HTTP or JMX endpoints. Most applications go for HTTP, though, where the identity of an endpoint and the /actuator prefix form a URL path.

Here are some of the most common built-in endpoints Actuator provides:

env: Exposes environment properties

health: Shows application health information

httptrace: Displays HTTP trace information

info: Displays arbitrary application information

metrics: Shows metrics information

loggers: Shows and modifies the configuration of loggers in the application

mappings: Displays a list of all @RequestMapping paths

Please refer to our Spring Boot Actuator tutorial for a detailed rundown.

**Q16. Which Is a Better Way to Configure a Spring Boot Project – Using Properties or YAML?**

YAML offers many advantages over properties files, such as:

More clarity and better readability

Perfect for hierarchical configuration data, which is also represented in a better, more readable format

Support for maps, lists, and scalar types

Can include several profiles in the same file (since Spring Boot 2.4.0, this is possible for properties files too)

However, writing it can be a little difficult and error-prone due to its indentation rules.

For details and working samples, please refer to our Spring YAML vs Properties tutorial.

**Q17. What Are the Basic Annotations that Spring Boot Offers?**

The primary annotations that Spring Boot offers reside in its org.springframework.boot.autoconfigure and its sub-packages. Here are a couple of basic ones:

@EnableAutoConfiguration – to make Spring Boot look for auto-configuration beans on its classpath and automatically apply them.

@SpringBootApplication – used to denote the main class of a Boot Application. This annotation combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

Spring Boot Annotations offers more insight into the subject.

**Q18. How Can You Change the Default Port in Spring Boot?**

We can change the default port of a server embedded in Spring Boot using one of these ways:

using a properties file – we can define this in an application.properties (or application.yml) file using the property server.port

programmatically – in our main @SpringBootApplication class, we can set the server.port on the SpringApplication instance

using the command line – when running the application as a jar file, we can set the server.port as a java command argument:

java -jar -Dserver.port=8081 myspringproject.jar

Q19. Which Embedded Servers does Spring Boot Support, and How to Change the Default?

As of date, Spring MVC supports Tomcat, Jetty, and Undertow. Tomcat is the default application server supported by Spring Boot's web starter.

Spring WebFlux supports Reactor Netty, Tomcat, Jetty, and Undertow with Reactor Netty as default.

In Spring MVC, to change the default, let's say to Jetty, we need to exclude Tomcat and include Jetty in the dependencies:

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<dependency>

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

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

</dependency>

Similarly, to change the default in WebFlux to UnderTow, we need to exclude Reactor Netty and include UnderTow in the dependencies.

“Comparing embedded servlet contains in Spring Boot” contains more details on the different embedded servers we can use with Spring MVC.

**Q20. Why Do We Need Spring Profiles?**

When developing applications for the enterprise, we typically deal with multiple environments such as Dev, QA, and Prod. The configuration properties for these environments are different.

For example, we might be using an embedded H2 database for Dev, but Prod could have the proprietary Oracle or DB2. Even if the DBMS is the same across environments, the URLs would definitely be different.

To make this easy and clean, Spring has the provision of profiles, to help separate the configuration for each environment. So that instead of maintaining this programmatically, the properties can be kept in separate files such as application-dev.properties and application-prod.properties. The default application.properties points to the currently active profile using spring.profiles.active so that the correct configuration is picked up.

**1) Explain the term 'Spring Boot'.**

It is a Spring module that offers Rapid Application Development to Spring framework. Spring module is used to create an application based on Spring framework which requires to configure few Spring files.

**2) Mention some advantages of Spring Boot**

Here are some major advantages of using spring-boot:

Helps you to create a stand-alone application, which can be started using java.jar.

It offers pinpointed‘started' POMs to Maven configuration.

Allows you to Embed Undertow, Tomcat, or Jetty directly.

Helps you to configure spring whenever possible automatically.

**3) How to create a Spring Boot application using Spring Initializer?**

It is a web tool provided by Spring on its official website. However, you can also create Spring Boot project by entering project details.

**4) Name the features of using Spring Boot**

Features of using Spring Boot are:

Starter dependency

Auto-configuration

Spring initializer

**5) Explain different phases of RAD model.**

This is a frequently asked job interview. Various phases of RAD mode are:

**Business Modeling**: Based on the flow of information and distribution between various business channels, the product is designed.

**Data Modeling** : The information collected from business modeling is refined into a set of data objects that are significant for the business.

**Application Generation**: Automated tools are used for the construction of the software, to convert process and data models into prototypes.

**6) What is RAD model?**

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. RAD follow the iterative

SDLC RAD model has the following phases:

Business Modeling

Data Modeling

Process Modeling

Application Generation

Testing and Turnover