**Spring boot and Micro services questions**

### **[Spring](https://www.edureka.co/blog/spring-tutorial/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank)**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 |

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

It is a part of the broader Spring ecosystem and is specifically designed to simplify the process of creating, configuring, and deploying applications using the Spring framework.. 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. Traditionally, creating a Spring application required a fair amount of boilerplate code and configuration, which could be time-consuming and error-prone.  
Additionally, managing dependencies, integrating with databases, and setting up servers often involved numerous steps.

Spring Boot addresses these challenges by providing the following key benefits:

**Simplified Configuration**: Spring Boot uses sensible defaults and smart auto-configuration to minimize the amount of configuration needed. It automatically configures components based on the classpath and the dependencies present in your project. This reduces the burden on developers to specify every configuration detail explicitly.

**Embeddable Servers**: Spring Boot allows you to create standalone applications with embedded servers like Tomcat, Jetty, or Undertow. This means you don't need to deploy your application to a separate server during development and testing, making the development process more straightforward.

**Dependency Management**: Spring Boot uses a dependency management system that automatically manages and resolves the versions of dependencies used in your application. This ensures that you have compatible and up-to-date libraries, reducing the risk of version conflicts.

**Production-Ready Features**: Spring Boot provides various production-ready features out of the box, such as health checks, metrics, and externalized configuration. These features make it easier to monitor and manage your application in production environments.

**Rapid Development**: With Spring Boot, you can quickly start building applications without getting bogged down by complex setup and configuration. It allows you to focus on writing business logic and delivering features more efficiently.

**Mention the advantages of Spring Boot**

**Rapid Development**: With Spring Boot, you can quickly start building applications without getting bogged down by complex setup and configuration. It allows you to focus on writing business logic and delivering features more efficiently.

**Embeddable Servers/Standalone applications**: Spring Boot allows you to create standalone applications with embedded servers like Tomcat, Jetty, or Undertow. This means you don't need to deploy your application to a separate server during development and testing, making the development process more straightforward.

**Production-Ready Features**: Spring Boot provides various production-ready features out of the box, such as health checks, metrics, and externalized configuration. These features make it easier to monitor and manage your application in production environments.

**Dependency Managemen**t: Spring Boot uses a dependency management system that automatically manages and resolves the versions of dependencies used in your application. This ensures that you have compatible and up-to-date libraries, reducing the risk of version conflicts.

**Simplified/Auto Configuration**: Spring Boot uses sensible defaults and smart auto-configuration to minimize the amount of configuration needed. It automatically configures components based on the classpath and the dependencies present in your project. This reduces the burden on developers to specify every configuration detail explicitly.

**Easy Testing**: Spring Boot includes built-in support for testing, making it easier to write unit tests, integration tests, and end-to-end tests for your application.

**Security**: Spring Boot integrates well with Spring Security, allowing developers to easily implement security features, such as authentication and authorization, in their applications.

**Features of springboot**

**1 Spring CLI**  – Spring CLI (Command-Line Interface) is a command-line tool provided by the Spring framework that allows developers to interact with Spring projects and perform various tasks using a command-line interface. It simplifies the process of creating, testing, and running Spring applications without the need for an integrated development environment (IDE).

Some common tasks that can be performed using Spring CLI include:

**Creating a New Project**, **Running the Application**, **Managing Dependencies**, **Testing**, **Generating Code**(Spring CLI can generate code snippets and templates for various components, such as controllers, services, and repositories), **Profile Management**(You can use Spring CLI to manage application profiles, which allow you to define different configurations for different environments e.g., development, production).

**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 in managing and monitoring 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.

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

Spring Boot CLI

Spring Starter Project Wizard

Spring Initializr

Spring Maven Project

**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.

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

When you run a Spring Boot application as a "Java Application," several things happen in the background to start and execute your application:

**Initialization of ApplicationContext**: Spring Boot creates and initializes the ApplicationContext, which is the core container that manages the application's beans (components). The ApplicationContext is responsible for dependency injection and wiring of components.

**Auto-Configuration**: Spring Boot's auto-configuration mechanism automatically configures various components based on the dependencies and settings present in your application. It scans the classpath for specific libraries and provides sensible defaults for your application.

**Embedded Web Server Startup**: If your Spring Boot application includes web-related dependencies, such as spring-boot-starter-web, Spring Boot starts an embedded web server (Tomcat, Jetty, or Undertow) based on the dependencies it finds in the classpath.

**Component Scanning**: Spring Boot scans the specified packages (by default, the package where your main class is located) to find components like controllers, services, repositories, and other beans marked with annotations such as @Component, @Service, @Controller, etc.

**Dependency Injection**: Spring Boot performs dependency injection, injecting dependencies into beans and wiring them together based on the @Autowired and related annotations.

**Application Startup**: Spring Boot executes the main() method of your main class (the class annotated with @SpringBootApplication or @SpringBootConfiguration). This kicks off the application's lifecycle.

**Property Configuration**: Spring Boot reads properties from various sources (e.g., application.properties, application.yml, environment variables, etc.) and applies them to the application's configuration.

**Execution of Application Logic**: After the Spring context is fully initialized, the application's logic starts executing, including any code written in @PostConstruct methods or triggered by various events.

**Running the Web Server**: If your application is a web application, the embedded web server starts listening to incoming HTTP requests and routes them to the appropriate controllers.

**Graceful Shutdown**: When you stop the application, Spring Boot performs a graceful shutdown, releasing resources and stopping the web server, if applicable.

In summary, when you run a Spring Boot application as a "Java Application," Spring Boot leverages its auto-configuration and dependency management capabilities to quickly set up and start your application, allowing you to focus on writing business logic and features without worrying about tedious setup and configuration tasks.

**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-test: Is the starter for testing Spring Boot applications

**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.

**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:

|  |  |
| --- | --- |
| 1  2  3  4 | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency> |

**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

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

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.

### ****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/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank) database, you can follow the below steps:

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

|  |  |
| --- | --- |
| 1 | CREATE DATABASE example; |

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

|  |  |
| --- | --- |
| 1 | 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

|  |  |
| --- | --- |
| 1  2  3  4 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 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.

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

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

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; |

**What is @ResponseBody and @Controller**

**@ResponseBody**: The @ResponseBody annotation is used to indicate that the return value of a method should be bound to the web response body, instead of being rendered to a view.When a method is annotated with @ResponseBody, Spring will automatically serialize the return value to JSON or XML (based on the Accept header of the request) and include it in the HTTP response.

@RestController

public class UserController {

@GetMapping("/user")

public User getUser() {

User user = new User("John", 30);

return user;

}

}

In this example, when a GET request is sent to /user, the getUser() method will return a User object. With the @ResponseBody annotation, Spring will convert the User object to JSON (or XML) and include it in the HTTP response.

**@Controller**: The @Controller annotation is used to mark a class as a Spring MVC controller. It indicates that the class is responsible for handling HTTP requests and generating the HTTP responses. A controller typically contains methods (handler methods) that are mapped to specific HTTP endpoints (URLs).

@Controller

public class HomeController {

@GetMapping("/home")

public String home() {

return "index";

}

}

In this example, the HomeController class is marked with @Controller. The home() method is mapped to the /home URL, and it returns the string "index". Here, "index" refers to the name of a view template that will be resolved by Spring to generate the HTML response.

In summary, @ResponseBody is used to indicate that the return value should be directly included in the HTTP response body, while @Controller is used to mark a class as a Spring MVC controller responsible for handling HTTP requests and generating responses. These annotations are fundamental in building web applications using Spring Framework.

**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/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank). 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

**Mention the differences between JPA and [Hibernate.](https://www.edureka.co/blog/what-is-hibernate-in-java/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank)**

|  |  |
| --- | --- |
| JPA | Hibernate |
| JPA stands for Java Persistence API. It is a Java specification that defines a standard way to interact with databases in Java applications | Hibernate is an implementation of Java Persistence API and offers benefits of loose coupling |

**What is JPA**

JPA stands for Java Persistence API. It is a Java specification that defines a standard way to interact with databases in Java applications.

The main goal of JPA is to simplify the process of persisting (storing) Java objects (entities) into relational databases and retrieving them back when needed. JPA provides a set of annotations and APIs that allow developers to map Java classes to database tables and perform database operations using standard Java objects and methods.

**Entity**: A Java class that represents an object to be persisted in the database. The class is annotated with @Entity to indicate that it is an entity.

**EntityManager**: A central component in JPA that is responsible for managing entity instances, including saving, updating, and deleting them in the database.

**Persistence Unit**: A configuration that defines the persistence context, database connection details, and entity classes to be managed by JPA.

**Mapping Annotations**: JPA provides annotations like @Id, @Column, @ManyToOne, @OneToMany, etc., to map the properties of Java classes to database columns and establish relationships between entities.

**JPQL** (Java Persistence Query Language): A query language similar to SQL but specifically designed for querying Java objects and their relationships.

**WHAT IS HIBERNATE**

Hibernate is an open-source Java-based framework that provides Object-Relational Mapping (ORM) capabilities. ORM is a technique that allows developers to map Java objects (entities) to database tables and perform database operations using Java methods, without writing raw SQL queries. Hibernate simplifies the process of working with relational databases and makes it easier to interact with the database using object-oriented programming concepts.

**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/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank), [Jersey](https://www.edureka.co/blog/java-web-services-tutorial/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank), etc.

**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.

**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" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank). 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.

|  |  |
| --- | --- |
| 1 | @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:

|  |  |
| --- | --- |
| 1 | @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.

**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 |

**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/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank) 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @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:

|  |  |
| --- | --- |
| 1  2  3 | example.number: 100  example.value: true  example.message: Dynamic Message |

**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/" \t "https://www.edureka.co/blog/interview-questions/spring-boot-interview-questions/_blank) and configure your connection to the MySQL database

Restart your project

**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.

**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

**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.

**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/\*”}

**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.

**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.

**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.

**What is RAD model?**

[RAD](https://www.guru99.com/what-is-rad-rapid-software-development-model-advantages-disadvantages.html) 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:

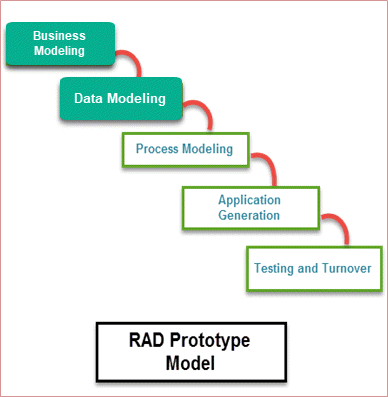
Business Modeling

Data Modeling

Process Modeling

Application Generation

Testing and Turnover



**Explain different phases of RAD model.**

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.

**What is Spring Boot starter? How is it useful?**

Spring Boot has many starters. They are a set of convenient dependency descriptors. Starter allows you to include these descriptors in your pom.xml.

For example, If you want to work with Spring MVC, you can include “spring–boot–starter–web” as a dependency in pom.xml.

**Can you use Spring Boot with applications which are not using Spring?**

No, it is not possible as Spring Boot is limited to Spring application only.

**What is the name of the configuration file which you can use in Spring Boot?**

The configuration file used in Spring Boot projects is called application.properties. It is an important file which allows you to override your default configurations.

**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 are the important features of Spring Boot?**

Important features of Spring Boot are:

Web Development

Spring Application

Application occasions and listeners

Admin highlights

YAML Support

Type-safe Configuration

Externalized Configuration

Properties Files

Logging and Security

**How are properties defined? Where?**

You can define properties in the application.properties file exists in the classpath.

Example: configure default DataSource bean

database.host=localhost

**What is a Spring Boot Actuator?**

Spring Boot Actuator allows you to monitor and manage your application when you want to push it for the production. It helps you to control your application by using HTTP endpoints.

**How can you access a value defined in the application? What is properties file in Spring Boot?**

Use the @Value annotation to access the properties which is defined in the application – properties file.

@Value("${custom.value}"

private String customVal;

**What is the primary difference between Spring and Spring Boot**

Spring is a web application development framework based on Java. On the other hand Spring Boot is an extension of the spring framework which eliminated the boilerplate configuration required for setup a Spring application.

**Explain Spring Boot Admin**

Spring Boot admin is a community project which helps you to manage and monitor your Spring Boot applications.

ow can you connect Spring Boot to the database using JPA?

Spring Boot supports spring-boot-data-JPA start, which helps you to connect spring application with a relational database.

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

The @RestController annotation helps you to add @ResponseBody and @Controller annotations to the class.

You can also import org.springframework.web.bind.annotation package in your file.

**Define the term Spring Initializer**

Spring initializer is a web application which can create an initial project structure for you.

**What are embedded containers support by Spring**

Spring Boot support the main three embedded containers:

1) Tomcat

2) Jetty

3) Undertow.

By default, it uses Tomcat as an embedded container.

**What is the main difference between JPA and Hibernate?**

The main difference between both of them is that JPA is a specification/Interface, whereas Hibernate is only JPA implementations.

**What is a shutdown in the actuator?**

A shutdown is an endpoint that helps application to be shut down properly. This feature is not enabled by default.

However, you can use it by setting command: management.endpoint.shutdown.enabled=true in your application.properties file.

**Is it possible to replace or override the Embedded Tomcat server in Spring Boot?**

Yes, it is possible to replace the Embedded Tomcat with any other servers by using the starter dependencies. For that, you can use spring-boot-starter-jetty or as a dependency for according you to your need.

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

Yes, we can disable the default web server by using application.properties to configure the web application type.

**How do you Add, Filter to an application?**

There are three methods to add filter to Spring Boot application:

By implementing Filter interface.

Using FilterRegistrationBean.

Using MVC controller.

**What is @pathVariable?**

@PathVariable annotation helps you to extract information from the URI directly.

**What is Swagger2?**

Swagger is used to describing the structure of APIs. Swagger 2 is an open-source service provided in Spring Boot which makes it easier for the machines to find out the structure of APIs like RESTful Web services.

**What is LiveReload in Spring Boot?**

LiveReload is a spring-boot-devtools module that includes LiveReload server to trigger a browser refresh when a resource is changed. LiveReload server extensions are available freeware for Firefox, Chrome, and Safari.

**What are the major benefits of spring Externalized Configuration?**

Externalized Configuration helps to work with the same code in different environments. Developers can use YAML files, properties files, command-line arguments, and environment variables to externalize configuration.

**What do you mean by hot-swapping in Spring Boot?**

It is a way to reload the changes without restarting Tomcat, or Jetty server. Eclipse and Many other IDEs support bytecode hot swapping. If you make any changes that don’t affect the method signature, it should reload without side effect.

**Explain Auto-Configuration in Spring Boot.**

Auto-configuration is used to configure Spring application automatically based on dependencies of classpath parameter. It makes development faster and easier.

**What is the meaning of Aspect-Oriented Programming (AOP)?**

Aspect-Oriented Programming supplements Object-Oriented Programming that aims to increase modularity. AOP breaks program logic into various parts, which are called concerns.

**How to enable logging in Spring Boot?**

In order to enable debug logging, you can specify –debug while starting the application from the command prompt.

**Explain overriding default properties in Spring Boot application.**

Spring Boot has lots of properties that can be easily overridden by specifying them in application.properties.

**Explain Docker in Spring Boot.**

It is a tool designed to create, deploy, and run a project by using containers.

**Define ELK stack.**

The ELK Stack is made of three open-source products: 1) Elasticsearch, 2) Logstash, and 3) Kibana.

Elasticsearch: It is a NoSQL database which is based on the open-source search engine called Lucene.

Logstash: It is a data processing pipeline tool which accepts inputs from sources, performs different transformations, and exports the data to targets.

Kibana: Kibana helps users to visualize data with graphs and chart in Elasticsearch.

**How to handle exception in Spring Boot.**

Spring Boot provides a very useful way to handle exceptions using @ControllerAdvice annotation.

**Explain caching.**

Caching is a memory are that temporary stores frequently accessed data that is otherwise expensive to get or compute.

**Explain different types of dependency injection.**

There are two types of dependency injection in Spring Boot. They are as follows:

Constructor based dependency injection: It is a technique in which one class object supplies the dependency of another object.

Setter-based dependency injection: It is a dependency injection in which the framework injects the primitive and string-based values using setter method.

**What are the advantages of micro service?**

Following are the major advantages of micro service:

It makes development fast and easy.

Compatible with all container.

Reduce production time.

It’s a lightweight model that supports a major business application.

**What is the default package in Spring Boot?**

A class without any package declaration is considered as a default package.

**Explain the difference between an embedded container and a WAR.**

The main difference between these two is:

Embedded containers help you to run Spring Boot application as a JAR from the command prompt without setting up any web server, while to run a WAR you need first to set up Tomcat.

**Explain Spring MVC.**

It is a traditional web application framework which helps you to build a web application. This framework is similar to the framework of Struts.

**What is the use of <set> tag?**

This tag is used to write to inject java set using XML.

**What is join point in Spring Boot?**

It is a program execution point like the handling of an exception or the execution of a method. In AOP, a join point is referred to as a method execution.

**How can you set active profile in Spring Boot?**

Follow the following methods to set an active profile in Spring Boot.

Pass this profile as an argument when you launch the Spring Boot application.

Set active the active profile in application.properties file.

**Is excluding package without using the basePackages filter is possible? How?**

Yes. It is possible to exclude package without using the basePackages filter by simply using the exclude attribute while using the @SpringBootApplication annotation.