List of Spring Boot Interview Questions and Answers

Q1. What is the Spring Boot?

Spring Boot is basically an extension of the Spring framework which eliminated the boilerplate configurations required for setting up a Spring application.

Spring Boot is an opinionated framework that helps developers build Spring-based applications quickly and easily. **The main goal of Spring Boot is to quickly create Spring-based applications without requiring developers to write the same boilerplate configuration again and again.**

Read more about Spring Boot at **Getting Started with Spring Boot**

Q2. Explain a few important Spring Boot Key features?

Let me a list of a few key features of the Spring boot and we will discuss each key feature briefly.

1. Spring Boot starters
2. Spring Boot autoconfiguration
3. Elegant configuration management
4. Spring Boot actuator
5. Easy-to-use embedded servlet container support

1. Spring Boot Starters

Spring Boot offers many starter modules to get started quickly with many of the commonly used technologies, like SpringMVC, JPA, MongoDB, Spring Batch, SpringSecurity, Solr, ElasticSearch, etc. These starters are pre-configured with the most commonly used library dependencies so you don't have to search for the compatible library versions and configure them manually.

For example, the `spring-boot-starter-data-jpa` starter module includes all the dependencies required to use Spring Data JPA, along with Hibernate library dependencies, as Hibernate is the most commonly used JPA implementation.

One more example, when we add the `spring-boot-starter-web` dependency, it will by default pull all the commonly used libraries while developing Spring MVC applications, such as `spring-webmvc`, `jackson-json`, `validation-api`, and `tomcat`.

Not only does the spring-boot-starter-web add all these libraries but it also configures the commonly registered beans like `DispatcherServlet`, `ResourceHandlers`, `MessageSource`, etc. with sensible defaults.

Read more about starters on **Important Spring boot Starters with Examples**

2. Spring Boot Autoconfiguration

Spring Boot addresses the problem that Spring applications need complex configuration by eliminating the need to manually set up the boilerplate configuration.

Spring Boot takes an opinionated view of the application and configures various components automatically, by registering beans based on various criteria. The criteria can be:

- Availability of a particular class in a classpath
- Presence or absence of a Spring bean
- Presence of a system property

- An absence of a configuration file

For example, if you have the `spring-webmvc` dependency in your classpath, Spring Boot assumes you are trying to build a SpringMVC-based web application and automatically tries to register `DispatcherServlet` if it is not already registered.

If you have any embedded database drivers in the classpath, such as H2 or HSQL, and if you haven't configured a `DataSource` bean explicitly, then Spring Boot will automatically register a `DataSource` bean using in-memory database settings.

You will learn more about the autoconfiguration on **What is Spring Boot Auto Configuration?**

3. Elegant Configuration Management

Spring supports externalizing configurable properties using the **@PropertySource** configuration. Spring Boot takes it even further by using the sensible defaults and powerful type-safe property binding to bean properties. Spring Boot supports having separate configuration files for different profiles without requiring many configurations.

Read more **http://www.javaguides.net/2018/09/spring-propertysource-annotation-with-example.html**

4. Spring Boot Actuator

Being able to get the various details of an application running in production is crucial to many applications. The Spring Boot actuator provides a wide variety of such production-ready features without requiring developers to write much code. Some of the Spring actuator features are:

- Can view the application bean configuration details
- Can view the application URL mappings, environment details, and configuration parameter values
- Can view the registered health check metrics

Read more about Spring Boot Actuator on **Spring Boot Actuator**

5. Easy-to-Use Embedded Servlet Container Support

Traditionally, while building web applications, you need to create `WAR` type modules and then deploy them on external servers like `Tomcat` , `WildFly` , etc. But by using Spring Boot, you can create a `JAR` type module and embed the servlet container in the application very easily so that the application will be a self-contained deployment unit.

Also, during development, you can easily run the Spring Boot JAR type module as a Java application from the IDE or from the command-line using a build tool like **Maven** or Gradle.

Q3. What is Spring Boot Auto-configuration?

Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added.

Why do we need Spring Boot Auto Configuration?
-> Spring based applications have a lot of configuration.
-> When we use Spring MVC, we need to configure
- Component scan,
- Dispatcher Servlet
- View resolver
- Web jars(for delivering static content) among other things.

-> When we use Hibernate/JPA, we would need to configure a
- data source
- entity manager factory/session factory
- transaction manager among a host of other things.

-> When you use cache
- Cache configuration

-> When you use Message Queue
- Message queue configuration

-> When you use NoSQL database
- NoSQL database configuration

Spring Boot: Can we think differently?
Spring Boot brings in a new thought process around this:
-> Can we bring more intelligence into this? When a spring MVC jar is added to an application, can we auto-configure some beans automatically?
-> How about auto-configuring a Data Source if Hibernate jar is on the classpath?
-> How about auto-configuring a Dispatcher Servlet if the Spring MVC jar is on the classpath?

One more example, if HSQLDB is present on your classpath and you have not configured any database manually, Spring will auto-configure an in-memory database for you.

The Spring Boot auto-configuration feature tries to automatically configure your Spring application based upon the JAR dependency you have added in the classpath.

Learn more about Spring Boot Auto Configuration at **Spring Boot Auto Configuration | Example**

Q4. How Spring boot internally works or Explain the `run()` method in Spring boot?

The below 10 steps show the internal working of the run() method:

1. Spring boot application execution will start from the main() method
2. The main() method internally call SpringApplication.run() method
3. SpringApplication.run() method performs bootstrapping for our spring boot application
4. Starts StopWatch to identify the time taken to bootstrap the spring boot application

5. Prepares environment to run our spring boot application (dev, prod, qa, uat)
6. Print banner (Spring Boot Logo prints on console)
7. Start the IOC container (ApplicationContext) based on the classpath (default, Web servlet/ Reactive)
8. Refresh context
9. Trigger Runners (ApplicationRunner or CommandLineRunner)
10. Return ApplicationContext reference (Spring IOC)

Learn more at **How Spring Boot Application Internally Works | Let's Debug and Understand run() Method Step by Step**

Q5. What are different ways to create a Spring boot application?

Different ways to create Spring boot project:
1. Using Spring Initializr - Create Spring boot project using Spring Initializr and import in any IDE - Eclipse STS, Eclipse, IntelliJ idea, VSCode, Netbeans

2. Using Spring Starter Project in STS (Eclipse) - You can directly create a Spring boot project in STS using Spring Starter Project option.

3. Spring Boot CLI - The Spring Boot CLI is a command-line tool that you can use if you want to quickly develop a Spring application.

Q6. Explain @SpringBootApplication, @Configuration and @ComponentScan annotations

The **@SpringBootApplication** annotation indicates a configuration class that declares one or more **@Bean** methods and also triggers auto-configuration and component scanning. This is a convenience annotation that is equivalent to declaring below three annotations:

@Configuration, **@EnableAutoConfiguration**, and **@ComponentScan**.

Read more about **@EnableAutoConfiguration** annotation with an example at **Spring Boot @EnableAutoConfiguration Annotation with Example**

`@Configuration` annotation is used to create a Java-based Spring configuration class that contains `@Bean` definition methods. So Spring container can process the class and generate Spring Beans to be used in the application.

Read more about `@Configuration` annotation at **Spring @Configuration Annotation with Example**

The `@ComponentScan` annotation is used with the `@Configuration` annotation to tell Spring the packages to scan for annotated components.

Learn more about **@SpringBootApplication** annotation at **@SpringBootApplication Annotation**

Q7. What is Spring boot starters and name few important Spring boot starter dependencies?

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy-paste loads of dependency descriptors.

For example, while developing the REST service or web application; we can use libraries like Spring MVC, Tomcat, and Jackson – a lot of dependencies for a single application. **spring-boot-starter-web** starter can help to reduce the number of manually added dependencies just by adding **spring-boot-starter-web** dependency.

So instead of manually specifying the dependencies just add one **spring-boot-starter-web** starter as in the following example:

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

Few commonly used Spring boot starters:

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

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

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

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

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

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

`spring-boot-starter-data-mongodb` : Starter for using MongoDB document-oriented database and Spring Data MongoDB

`spring-boot-starter-data-rest` : Starter for exposing Spring Data repositories over REST using Spring Data REST

`spring-boot-starter-webflux` : Starter for building WebFlux applications using Spring Framework's Reactive Web support

You can find all the Spring Boot Starters at **Important Spring Boot Starters with Examples**

Q8. How does Spring Enable Creating Production-Ready Applications in Quick Time?

Spring Boot aims to enable production-ready applications in a quick time. Spring Boot provides a few non-functional features out of the box like caching, logging, monitoring, and embedded servers.

1. **spring-boot-starter-actuator** - To use advanced features like monitoring & tracing to your application out of the box
2. **spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat** - To pick your specific choice of Embedded Servlet Container
3. **spring-boot-starter-logging** - For Logging using logback
4. **spring-boot-starter-cache** - Enabling Spring Framework's caching support

Q9. What Is the Minimum Baseline Java Version for Spring Boot 2 and Spring 5?

Spring Boot 2.0 requires Java 8 or later. Java 6 and 7 are no longer supported. It also requires Spring Framework 5.0.

Recommended Reading - **https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0.0-M1-Release-Notes**.

Q10. What are Different Ways of Running Spring Boot Application?

Spring boot offers several ways of running Spring boot applications. I would like to suggest five ways we can run Spring Boot Application

1. Running from an IDE
2. Running as a Packaged Application
3. Using the Maven Plugin
4. Using External Tomcat
5. Using the Gradle Plugin

Read more at **http://www.javaguides.net/2018/09/different-ways-of-running-spring-boot-appilcation.html**

Q11. Name all Spring Boot Annotations?

Read more about each Spring Boot annotation at **http://www.javaguides.net/2018/10/spring-boot-annotations.html**

Q12. What Is the Difference Between @SpringBootApplication and @EnableAutoConfiguration Annotation?

@EnableAutoConfiguration is to enable the automatic configuration feature of the Spring Boot application which automatically configures things if certain classes are present in Classpath. For example, it can configure `Thymeleaf`, `TemplateResolver`, and `ViewResolver` if **Thymeleaf** is present in the classpath.

@EnableAutoConfiguration also combines **@Configuration** and **@ComponentScan** annotations to enable Java-based configuration and component scanning in your project

On the other hand, **@SpringBootApplication** annotation indicates a configuration class that declares one or more **@Bean** methods and also triggers auto-configuration and component scanning. This is a convenience annotation that is equivalent to
declaring **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**.

Read more about **@EnableAutoConfiguration** annotation with an example at **Spring Boot @EnableAutoConfiguration Annotation with Example**

Read more about **@SpringBootApplication** annotation with an example at **Spring Boot @SpringBootApplication Annotation with Example**

Q13. Why do we need a spring-boot-maven plugin?

The Spring Boot Maven plugin provides many convenient features:

- It collects all the jars on the classpath and builds a single, runnable "über-jar", which makes it more convenient to execute and transport your service.
- It searches for the public static void `main()` method to flag as a runnable class.

- It provides a built-in dependency resolver that sets the version number to match Spring Boot dependencies. You can override any version you wish, but it will default to Boot's chosen set of versions.

The Spring Boot Plugin has the following goals.

- `spring-boot:run` runs your Spring Boot application.
- `spring-boot:repackage` repackages your jar/war to be executable.
- `spring-boot:start` and `spring-boot:stop` to manage the lifecycle of your Spring Boot application (i.e. for integration tests).
- `spring-boot:build-info` generates build information that can be used by the Actuator.

Read more about Spring Boot Plugin at **https://docs.spring.io/spring-boot/docs/current/maven-plugin**

Q14. What is the Spring Boot Actuator and its Features?

Spring Boot Actuator includes a number of additional features to help you monitor and manage your application when it's pushed to production. You can choose to manage and monitor your application using HTTP or JMX endpoints. Auditing, health, and metrics gathering can be automatically applied to your application.

Enabling the Actuator

The simplest way to enable the features is to add a dependency to the **spring-boot-starter-actuator** 'Starter'. To add the actuator to a Maven-based project, add the following 'Starter' dependency:

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

For Gradle, use the following declaration:

```
dependencies {
    compile("org.springframework.boot:spring-boot-starter-actuator")
}
```

Features

Endpoints: Actuator endpoints allow you to monitor and interact with your application. Spring Boot includes a number of built-in endpoints and you can also add your own.

For example, the health endpoint provides basic application health information. Run up a basic application and look at `/actuator/health` .

Metrics: Spring Boot Actuator provides dimensional metrics by integrating with Micrometer.

Audit: Spring Boot Actuator has a flexible audit framework that will publish events to an `AuditEventRepository` . Once Spring Security is in play it automatically publishes authentication events by default. This can be very useful for reporting, and also to implementing a lock-out policy based on authentication failures.

Read more at **Spring Boot Actuator(Official Doc)**.

Q15. How to Use Jetty Instead of Tomcat in Spring-Boot-Starter-Web?

Remove the existing default tomcat dependency from `spring-boot-starter-web` and add the `spring-boot-starter-jetty` dependency:

```
<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>
```

Q16. How to generate a WAR file with Spring Boot?

I suggest below three steps to generate and deploy the Spring Boot WAR file.

1. Change the packaging type.

```
<packaging>war</packaging>
```

2. Add **spring-boot-starter-tomcat** as the **provided** scope

```
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
  </dependency>
```

3. Spring Boot Application or `Main` class extends `SpringBootServletInitializer`

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class Springboot2WebappJspApplication extends SpringBootServletInitializer{

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Springboot2WebappJspApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(Springboot2WebappJspApplication.class, args);
    }
}
```

Learn with a complete example at **Spring Boot 2 Deploy WAR file to External Tomcat**.

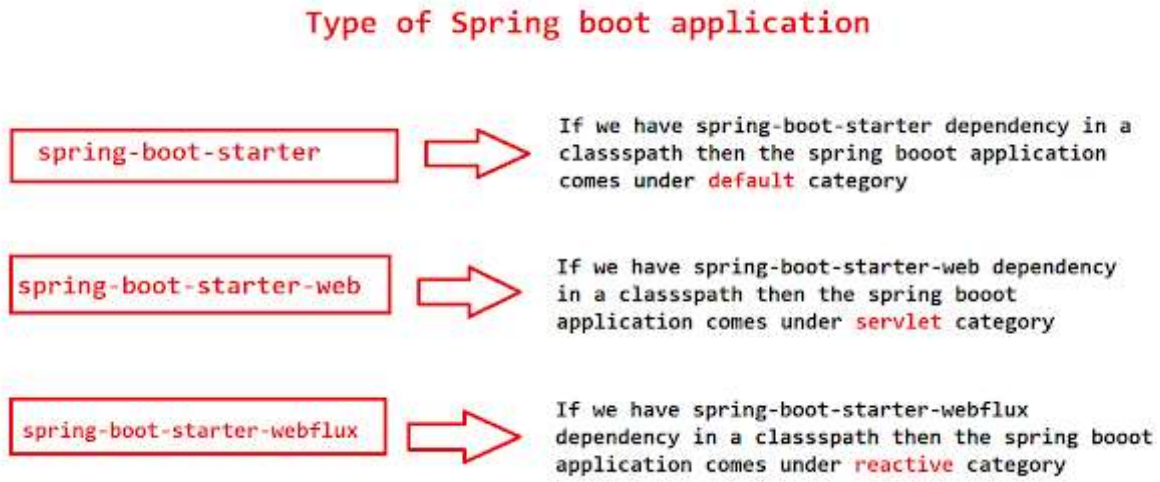
Q17. How many types of projects we can create using Spring boot?

We can create 3 types of projects using Spring boot starter dependencies.

3 types of Spring boot applications:

1. If `we have a spring-boot-starter` dependency in a classpath then the spring boot application comes under the **default** category.

2. If we have `spring-boot-starter-web` dependency in a classpath then the spring boot application comes under the **servlet** category.
3. If we have a `spring-boot-starter-webflux` dependency in a classpath then the spring boot application comes under the **reactive** category.



Q18. How to Change Default Embedded Tomcat Server Port and Context Path in Spring Boot Application?

By default, the embedded tomcat server starts on port 8080 and by default, the context path is "/". Now let's change the default port and context path by defining properties in an **application.properties** file -

`/src/main/resources/application.properties`

```
server.port=8080
server.servlet.context-path=/springboot2webapp
```

Read more at **Spring Boot How to Change Port and Context Path**.

Q19. What Embedded servers does Spring Boot support?

Spring Boot supports three embedded containers: Tomcat, Jetty, and Undertow.

By default, it uses Tomcat as embedded containers but you can change it to Jetty or Undertow.

Spring Boot 2+ supports the following embedded servlet containers:

Q20. How to use logging with Spring Boot?

We can use logging with Spring Boot by specifying log levels on the **application.properties** file. Spring Boot loads this file when it exists in the classpath and it can be used to configure both Spring Boot and application code.

Spring Boot, by default, includes **spring-boot-starter-logging** as a transitive dependency for the **spring-boot-starter** module. By default, Spring Boot includes `SLF4J` along with `Logback` implementations.

If `Logback` is available, Spring Boot will choose it as the logging handler. You can easily configure logging levels within the application.properties file without having to create logging provider-specific configuration files such as `logback.xml` or `log4j.properties` .

```
logging.level.org.springframework.web=INFO
logging.level.org.hibernate=ERROR
logging.level.net.guides=DEBUG
```

Read more at **Spring Boot 2 Logging SLF4j Logback and LOG4j2 Example**.

Q21. What is the Spring Boot Starter Parent and How to Use it?

All Spring Boot projects typically use `spring-boot-starter-parent` as the parent in `pom.xml` .

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.4.RELEASE</version>
    </parent>
```

`spring-boot-starter-parent` allows us to manage the following things for multiple child projects and modules:

- Configuration - Java Version and Other Properties
- Dependency Management - Version of dependencies
- Default Plugin Configuration

We should need to specify only the Spring Boot version number on this dependency. If you import additional starters, you can safely omit the version number.

Read more about spring-boot-starter-parent at **Overview of Spring Boot Starter Parent**.

Q22. How to Implement Security for Spring Boot Application?

Spring boot provided auto-configuration of spring security for a quick start. Adding the Spring Security Starter (spring-boot-starter-security) to a Spring Boot application will:

- Enable HTTP basic security
- Register the `AuthenticationManager` bean with an in-memory store and a single user
- Ignore paths for commonly used static resource locations (such as /css/, **/js/**, /images/**, etc.)
- Enable common low-level features such as `XSS` , `CSRF` , caching, etc.

Add below dependencies to the `pom.xml` file

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

Now if you run the application and access **http://localhost:8080**, you will be prompted to enter the user credentials. The default user is `user` and the password is auto-generated. You can find it in the console log.

Using default security password: `78fa095d-3f4c-48b1-ad50-e24c31d5cf35`

You can change the default user credentials in `application.properties` as follows:

```
security.user.name=admin
security.user.password=secret
security.user.role=USER,ADMIN
```

Download PDF of this article at spring boot interview questions and answers pdf.

References