## What is Spring boot?

[**Sprint boot**](https://www.interviewbit.com/java-interview-questions/) is a Java-based spring framework used for Rapid Application Development. It has extra support of auto-configuration and embedded application server like tomcat, jetty, etc.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

## Features of Spring Boot?

## Externalized Configuration

Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments.

We can use a variety of external configuration sources like Java properties files, YAML files, etc.

* **Profiles**

Spring Profiles provide a way to segregate parts of your application configuration and make it be available only in certain environments.

## Auto configuration

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

So if we want to implement jpa, we can just add the jpa dependencies using spring-boot-starter, all those configurations will be added automatically using the spring-boot-auto-configurer template.

## 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.

## Embedded server

If we see the traditional way of deploying the java application is like, we create the application, we create the war file of the entire application and deploy that war file in any of the application server or the web server.

But with spring boot that particular server will be embedded to that particular entire application. so we won’t be creating the war file, we will be creating the jar file and in that particular JAR file our server will be embedded. So we can directly run the JAR file in any of the environment. So it’s always production-ready.

## Dependency Management

Spring boot provides a way to group all the dependencies into a different starter templates that are required to do that particular task. Suppose if you want to work with the jdbc and there is a spring-boot-starter-jdbc template available that will include all the dependencies required to do that particular thing.

## Create stand-alone Spring applications

## Absolutely no code generation and no requirement for XML configuration

## Benefits of Spring Boot? Advantages of Spring Boot?

## Rapid application development

Spring boot provides us to do the Rapid application development. So whatever application you want to create, you can create using spring boot very easily.

## Dependency Management

Spring boot provides a way to group all the dependencies into a different starter templates that are required to do that particular task. Suppose if you want to work with the jdbc and there is a spring-boot-starter-jdbc template available that will include all the dependencies required to do that particular thing.

## Auto configuration

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

So if we want to implement jpa, we can just add the jpa dependencies using spring-boot-starter, all those configurations will be added automatically using the spring-boot-auto-configurer template.

## Embedded server

If we see the traditional way of deploying the java application is like, we create the application, we create the war file of the entire application and deploy that war file in any of the application server or the web server.

But with spring boot that particular server will be embedded to that particular entire application. so we won’t be creating the war file, we will be creating the jar file and in that particular JAR file our server will be embedded. So we can directly run the JAR file in any of the environment. So it’s always production-ready.

* Decreased boilerplate code
* Fast and easy development of Spring-based applications;
* The ability to create standalone applications;
* No need for XML configuration;
* Reduced amounts of source code;

## What are the Spring Boot key components?

* Spring Boot auto-configuration.
* Spring Boot CLI.
* Spring Boot starter POMs.
* Spring Boot Actuators.

## Why Spring Boot over Spring?

* **Starter POM**

Spring Boot comes with the concept of starter in pom.xml file that internally takes care of downloading the dependencies JARs based on Spring Boot Requirement.

* **Version Management**
* **Auto Configuration**
* **Component Scanning**
* **Embedded server**

Spring Boot offers embedded server such as Jetty and Tomcat, etc.

* **InMemory DB**

It offers several plugins for working with an embedded and in-memory database such as H2.

* **Actuators**
* It helps to create a stand-alone application(executed independently) with less configuration.
* It reduces boilerplate code.

## What is the starter dependency of the Spring boot module?

* spring-boot-starter-mail

It is used to support Java Mail and Spring Framework's email sending.

* spring-boot-starter-web

It is used for building the web application, including RESTful applications using Spring MVC. It uses Tomcat as the default embedded container.

* spring-boot-starter-test

It is used to test Spring Boot applications with libraries, including JUnit, Hamcrest, and Mockito.

* spring-boot-starter-jdbc

It is used for JDBC with the Tomcat JDBC connection pool.

* spring-boot-starter-validation

It is used for Java Bean Validation with Hibernate Validator.

* spring-boot-starter-security

It is used for Spring Security.

* spring-boot-starter-data-jpa

It is used for Spring Data JPA with Hibernate.

* spring-boot-starter-data-mongodb

It is used for MongoDB document-oriented database and Spring Data MongoDB.

## How does Spring Boot works?

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

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

**DAILY CODE BUFFER**

**What is Spring and spring boot?**

Spring is a java Framework built to create an enterprise ready application. When you create a java application there are a lot more things that you have to do. A lot of configurations, lot of property additions and lot of Technologies and a lot of packages and lot of modulus and JAR file that you need to add to create your applications. Spring is a Framework that allows us as a Java developer to do lot of the things. There are a lot of modules available for different kind of stuffs that we can use as a part of spring framework. That is Spring Core or spring MVC, spring web,  spring batch, spring data, spring data jpa, there are a lot of stuffs available in the spring framework.

We can use all those stuffs for our different activities. So we can only concentrate about actual coding that we have to do. But when you are creating the spring application, there is a lot of configurations that you have to do.

If you are working with spring framework and you want to include Hibernate and you have to do the configuration for that. If you want to include any caching mechanism then you have to do configuration for that. If you want to use any messaging queue you have to do the configuration for that as well.

So as a Java developer, as a spring user you have to do a lot of configuration to get ready with your application. After that once you create your application you have to deploy application to any of the application server or web server. So there are a lot of moving parts when you are working with the java application.

Spring framework  makes it easy for us to create all the application but that is a lot of configurations that we have to do. With this in mind spring developer thought of ok Let's create something easier so all the developers can really concentrate on the actual work. Only concentrate on the convention not the configuration part. For that they created the Spring boot.

Spring boot is just the extension of a spring framework. It’s not a different framework.

Spring boot provides key benefits for our Java developer,

Firstly the main thing is **Rapid application development**: Spring boot provides us to do the Rapid application development. So whatever application you want to create, you can create using spring boot very easily.

The other thing is **managing the dependencies**.When you are working with the spring framework, At that time also we have to add a lot of dependencies that you are using for your application.

Spring boot provides a way to group all those dependencies into a different starter templates. So spring boot provides different starter templates that includes all the dependencies that are required to do that particular task. Suppose if you want to work with the jdbc and there is a spring-boot-starter-jdbc template available that will include all the dependencies required to do that particular thing. If you want to work with JPA there is a spring-boot-starter-jpa template available that will include all the dependencies required to do that particular task. There is a spring-boot-starter-test available to do all our spring boot unit testing using the Junit and Mockito. So there are a lot of different templates available that you can directly used as a starter template and we can directly work with it.

The other thing spring boot provide is the **auto configuration**, so whenever we implement any of the things we have to do the configuration for all the stuffs. Suppose if we implement Hibernate we have to do the Hibernate configuration. Suppose if we implement any messaging queue we have to do the Hibernate configuration for that messaging queue.

If we implement any of the Other libraries there might be one or the other configurations  that we have to do to work with the spring framework. What spring boot does is, use auto configuration for all those dependencies, all those libraries that we can use.

So if we want to implement hibernate, just add the hibernate dependencies using spring-boot-starter, all those configurations will be added automatically using the spring-boot-auto-configurer template.

Next is the **Embedded server**: If we see the traditional way of deploying the java application is like, we create the application, we create the war file of the entire application and deploy that war file in any of the application server or the web server. Either JBoss, Tomcat, web sphere oranything. You will deploy war file to that particular server

 but with spring boot that particular server will be embedded to that particular entire application. so we won’tl be creating the war file, we will be creating the jar file and in that particular JAR file our server will be embedded. So we can directly run the JAR file in any of the environment. So it’s always production-ready. There are a lot more benefits to work with the spring application and the main thing is that market is moving towards the microservices architecture, rather than monolith.

So to create all the microservices using Java, Spring boot is the default option. So now with this it is pretty much clear that spring boot is the way to go.

**Dependency injection:**

Whenever we talk about the Spring Framework or spring boot, dependency injection is the topic that everyone discuss. Let’s understand why dependency injection is so hype over here. So if we take any of the programming languages or any of the frameworks for that particular programming languages dependency injection pattern is a default way they will go for creating the different objects in the application.

If we take the traditional example to create any of the class in your Java applications suppose student. now to create the object of the student what you will do student  s = new Student();  This means you yourself creating the object of that particular class over here. And when there is  a lot of hundreds and thousands of classes created and all those classes are interrelated to each other then it's not a good idea to use that approach to create the class manually

So in this type of scenarios inversion of control comes into picture.

what is **inversion of control**

 inversion of control is nothing but to give the control from yourself as a developer to the Framework that you are using Suppose if you are using Spring Framework that means we are giving the control to spring to create the object for us.

Now to implement that thing, dependency injection is the pattern that we use.

Dependency injection means suppose you have created one of the class that is student and in that particular class there is a another class that is subject available so you don't have to create that particular object for that dendency injection pattern will create the particular object for you whenever you want, so what will happen is when a spring boot application start, it have the factory that will create all the objects or all the beans of your application

Suppose if we have Hundreds and thousands of classes is in your application, all those particular classes object will be created when that particular spring boot context  is started. And all those particular beans will be stored in one of the springs container and whenever you want that particular bean or particular object, we will tell ok spring I want this particular class and spring will give you directly. You don't have to create the objects, spring already has all the objects you can directly use it.

So this is very important part to understand when we do the demo will be using it everywhere that the time I will show you how to do this but this is the theoretical part that you should understand what is IOC and what is dependency injection in which everything will be working all about inversion of control and penalties injection in Spring Framework now let's go ahead and start our applications

**Spring initializer:**

IF we go to start.spring.io, with this particular tool we can create the barebone spring boot project.

* Here we can choose Project build tools like Maven Project or Gradle Project.
* We can choose Languages like Java, Kotlin, Groovy.
* We can choose the different spring boot versions available. I always go with the latest stable version.
* We also have Packaging Jar, War. We can select the Java version 8, 11, 16.
* We also have project metadata like Group, Artifact(Project name), Name(Springboot application name), Description, Package name.
* Now we can add the different dependencies for example to create restful applications, we can add the spring web dependency.

**Spring boot IDE's**

We can open the project in any of the IDE's like STS, Intellij IDEA, Eclipse, VS Code, etc.

There are a lot of different plugins available for all the different editors. We can add those plugins and it will be very useful for java applications as well.

**Spring boot starters:**

In pom.xml we have dependencies like spring-boot-starter-parent which takes care of the versions of other dependencies like spring-boot-starter-web, spring-boot-starter-test, etc.

Inside spring-boot-starter-web, we have spring-boot-starter, spring-boot-starter-json,spring-boot-starter-tomcat,spring-web, etc.

Similarly for other dependencies.

**Creating Simple API:**

The file which has the main method is the starting point for our spring boot application.

and this class is Annotated with @SpringBootApplication. This is the key annotation for our spring boot application.

@SpringBootApplication annotation is equalent to

@SpringBootConfiguration, @EnableAutoConfiguration, @ComponentScan.

@SpringBootConfiguration: It tells the application that this is the main spring boot configuration file.

@EnableAutoConfiguration: All the autoconfigurations for our application will be added to it. Suppose if we want some of the classes or some of the configurations not add automatically, we can also exclude them.**@EnableAutoConfiguration(exclude={CLASS\_NAME})**

Suppose if we don't want to add the autoconfiguration for hibernate,we can exclude by adding the classes.

@ComponentScan:It will scan all the components available in your spring boot application and all those components will be added to the spring container when the application starts.

The SpringApplication.run method starts our application.

The application will run on Tomcat port 8080.

Go to localhost:8080, now we will get a Whitelabel Error Page because we have not created any of the API's now.

We can create a controller class inside controller package. To make this controller class as the component of spring so that whenever the application is started, this particular class is added to the spring container for that we need to annotate the controller class with @Component. But this controller class is not a simple component, its a Resouce, so we can use @Controller.

Since we are creating Restful APIs, this controller has to a Rest controller,for that we need to annotate the controller class with @RestController

@RestController defines it is a controller and also returns a responseBody

We can create method inside this controller.

Now whenever I hit an endpoint, this particular method should be executed for that we annotate with @RequestMapping(value="/", method = RequestMethos.GET)

Instead of using RequestMapping we can use GetMapping("/")

Now suppose if we want to run the application in port 8082, we can add the server.port in application.properties file.

We can run the application using mvn spring-boot:run

**Springboot Devtools:**

Whenever we are developing the application, we need to do a lot of changes and we need to start our servers. To stop and start the server every time is a big task. For that spring boot provides the spring boot devtools dependency that we can add into our project. Whenever there is any changes spring boot will detect the changes and it will restart the application.

We need to add the spring-boot-devtools dependency in pom.xml and Save and reload the maven configuration

**Architecture and example:**

From Frontend technologies any thing call this Restful APIs.

If we consider Department,

In Department controller class inside controller package,

This controller layer handles all our request,

Controller layer is just to get the request and send the response back. Just for the routing purpose.

We can create post api to save the department.

we can create an api to get all the departments,

we can create an api to get that particular department, we can create an api to delete the particulardepartment. we can create an api to update that department.

Service layer will be our business layer. All the business logic that we want to add to handle our department will be adding in our dept layer.

From the service layer we will be calling the DataAccess/Repository layer. This layer is responsible to interact with the database.

This layer will handle all the database operations.

For this layer we will be using spring-data-jpa dependency

We will also have a database.

**Adding Dependency H2 & JPA:**

Suppose if we want to use H2 database, we need to add h2 dependency in pom.xml and for JPA, we can add spring-boot-starter-data-jpa dependency and some configurations, spring.h2.console.enabled=true, spring.datasource.url=jdbc:h2:mem:dbname,

spring.datasource.driverClaassName=org.h2.Driver,

spring.datasource.username,

spring.datasource.password,

spring.datasource.platform=*org.hibernate.dialect.H2Dialect*(Hibernate uses "dialect" configuration to know which database you are using so that it can switch to the database specific SQL generator code wherever/whenever necessary.)

With this particular properties, our entire h2 database is configured for jpa and hibernate

We can go to localhost:8080/h2-console

**Creating components:**

entity:

Inside entity package we can create Department entity, Inside this Department class we can create properties: deptId, deptName, deptAdd,deptCode, etc. Create getters and setters for these properties. We can also create the no-args and all-args constructor and toString method.

Now to make this particular class as an entity so that it can interact with our database using jpa, we need to annotate this class with @Entity using the javax.persistance.Entity package.

Now this Department entity can interact with the database.

In the relational database all the tables will have a primary key. So for this entity also we need to create one. Here deptId will be the primary key, so to make this as primary key we need to annotate it with @Id.

To generate the primary key automatically, we add @GeneratedValue(strategy = GeneratedType.AUTO).

Controller:

Inside controller package we can create DepartmentController. We will annotate this class with @RestController. We can create Rest apis here.

Inside service package, we can create the DepartmentService interface and DepartmentServiceImpl class which implements DepartmentService. Annote the DepartmentServiceImpl class with @Service.

Inside repository package, we can create the DepartmentRepository interface and we annotate with @Repository.

This DepartmentRepository extends JpaRepository<Department, Long>()

For this JpaRepository, we need to pass the entity and primary key type.

This JpaRepository extends PagingAndSortingRepository and QueryByExampleExecutor.

This PagingAndSortingRepository extends CrudRepository.

**Department save API:**

Inside the DepartmentController, create a method to save the Department and annotate with @PostMapping and I need to call this particular post with "/departments".

As I am creating the rest api using the postmapping, ie, the post request that I will do from any of the rest client and I will be passing the RequestBody as well, so that will be entire json object. I want the entire json object to be coming over here and to convert the entire json object to my Department entity,for that we add @RequestBody

@RequestBody: Get the particular json and convert to my Department object.

Now we have to call the service layer to pass this data and in this service layer we do the business logic. For this we create private DepartmentService deptService and autowire it using @Autowired

Types of Autowiring/ dependency injection:

Constructor based and setter based

Property based

then we use deptService.saveDepartment(dept) to create an abstract method to save the department.

Now we have to implement this method in DepartmentServiceImpl. Now we have to call the repository layer to pass the data from Impl class.

For this we create private DepartmentRepository deptRepository and autowire it using @Autowired

then we use deptRepository.save(dept)

For testing this we can using any of the Rest client like Postman, Fiddler, Insomnia, etc..

Select POST, pass the endpoint url localhost:8082/departments

Pass json data and click on send

**Hibernate Validtion:**

Suppose if we have 4 fields coming as an input parameter to save or update the data. We can add validations like, I want the deptName to be available. Without deptName that particular json data, particular requestbody that we are getting from the client that should be invalid

For this,

We need to add the spring-boot-starter-validation,

above deptName field, add @NotBlank(message="Please Add Department Name")

Then in the particular controller, add @Valid from where request is coming.

Similary we have more validations,

@Length(max=5,min=1), @Size(max=10,min=0), @Email, @Positive, @Negative, @PositiveOrZero, @NegativeOrZero, @Future, @FutureOrPresent, @Past, @PastOrPresent

**Loggers in Springboot:**

Springboot comes with the slf4j logging library,

In all the class add private final Logger logger = LoggerFactory.getLogger(name: DeptController.class)

then in method add logger.info()

This loggers will be helpful for debugging our application.

**Project Lombok:**

As we are working with the java application, we tend to create a lot of POJOs, properties and for all those properties we will be creating getter setters, constructors, toString method. for 4 fields, we have lot of boilerplate codes, To get rid of this particular thing a library is available called Lombok

Lombok allows us to remove the boilerplatecode from our java application

Add lombok dependency,We have to tell maven plugin that we are using lombok , add lomok plugin.

Go to your entity and annotate with @Data(equivalent to @Getter, @Setter, @RequiredArgsConstructor, @ToString, @EqualsAndHashCode), @NoArgsConstructor, @AllArgsConstructor, @Builder(builder pattern will be implemented)

**Exception Handling:**

Inside error package, create a custom exception DepartmentNotFoundException, this will extend Exception and override methods inside that.

We will throw this custom exception when there is no department found

Whenever an exception is occuring at the controller layer, we have to identify that this is the exception thrown. And whenever that exception is there we have to send back the data as a response that we have to configure here.

For that we create RestResponseEntityExceptionHandler class, this will extend ResponseEntityExceptionHandler.

Now whatever the class you are creating to handle all your exceptions, that particular class should be annotated with @ControllerAdvice and we define this particular class will return the response status by annotating with @ResponseStatus.

Now inside this class we have to create one method, and that method will be responsible to handle that particular kind of exception and we have to annotate this method with @ExceptionHandler(ClassName.class)

For that create a class ErrorMessage, add properties: HttpStatus status, String message

**H2 to MySQL:**

spring.datasource.url=jdbc:mysql:localhost:3306/dbname,

spring.datasource.driverClaasName=com.mysql.jdbc.Driver, spring.datasource.username,

spring.datasource.password,

spring.jpa.hibernate.ddl-auto=update(to create tables)

spring.jpa.show-sql:true

Now add the mysql-connector-java dependency.

create the schema dbname

By default, JPA databases are automatically created only if you use an embedded database (H2, HSQL, or Derby)

You can explicitly configure JPA settings by using spring.jpa.\* properties. For example, to create and drop tables you can add the following line to your application.properties:

spring.jpa.hibernate.ddl-auto=create-drop

**Adding configurations in application.properties:**

Let me add welcome.message=Welcome home, to use this in our application,

Lets create a private String welcomeMessage , above this add @Value("${welcome.message}")

**application.yml file:**

This file is more human readable format and it reduces the duplicate values here ie, we can remove the redundant part.

Create a application.yml inside resources folder

server:

port: 8082

There are different plugins available to convert properties to yml.

**Springboot Profiles:**

Consider we have an application, now this application has to be deployed in our dev server,qa server, stage server, prod server for all the different environments.

For all these different environments we will be using different configuration properties.

For example:

The database configuration properties have to be different for all the environments.We can achieve this using profiles.

We will create different profiles for our dev, qa, stage and prod servers and we will add the different configurations for it.

When we deploy our application,we will tell spring that deploy my application using this particular profile.

Now in yml we can create multiple documents in single file itself.

For that add 3 hypens for separation between profiles and default one,

spring:

profiles:dev

In default one,

spring:

profiles:

active:dev

Otherwise we can create the different configuration files for different environments.

Suppose your profile name is dev,then we can create the application-dev.yml, similarly for others.

**Running springboot with multiple profiles:**

Change the version,

mvn clean install.

jar file will be created in the target folder.

java -jar jarFileName.jar --spring.profiles.active=prod

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

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

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan . Spring Boot enables the developer to use a single annotation instead of using multiple. But, as we know, Spring provided loosely coupled features that we can use for each annotation as per our project needs

@EnableAutoConfiguration – This enables Spring Boot's autoconfiguration mechanism.Auto-configuration refers to creating beans automatically by scanning the classpath.

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

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the @ComponentScan annotation for your class file to scan your components added to your project.

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

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

@SpringBootApplication

**public** **class** **MyApplication** {

**public** **static** **void** **main**(String[] args) {

SpringApplication.run(MyApplication.class);

// other statements

}

}

**What are starter dependencies?**

Spring boot starter is a maven template that contains a collection of all the relevant transitive dependencies that are needed to start a particular functionality.  
Like we need to import spring-boot-starter-web dependency for creating a web application.

<dependency>

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

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

</dependency>

**What is Spring Initializer?**

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

**What 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.

**What is Spring Boot dependency management?**

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

### **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.

### **Application context**

It represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans.

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

Yes, it is possible. By using the *server.port* in the *application.properties.*

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

The default port of the tomcat server-id 8080. It can be changed by adding sever.port properties in the application.property file.

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

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

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

Yes, we can use *application.properties* to configure the web application type i.e*spring.main.web-application-type*=none.

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

You can use exclude attribute of @EnableAutoConfiguration if you want auto-configuration not to apply to any specific class.

//use of exclude**@EnableAutoConfiguration(exclude={className})**

### **Explain @RestController annotation in Sprint boot?**

It is a combination of @Controller and @ResponseBody, used for creating Restful controllers. It converts the response to JSON or XML. It ensures that data returned by each method will be written straight into the response body instead of returning a template.

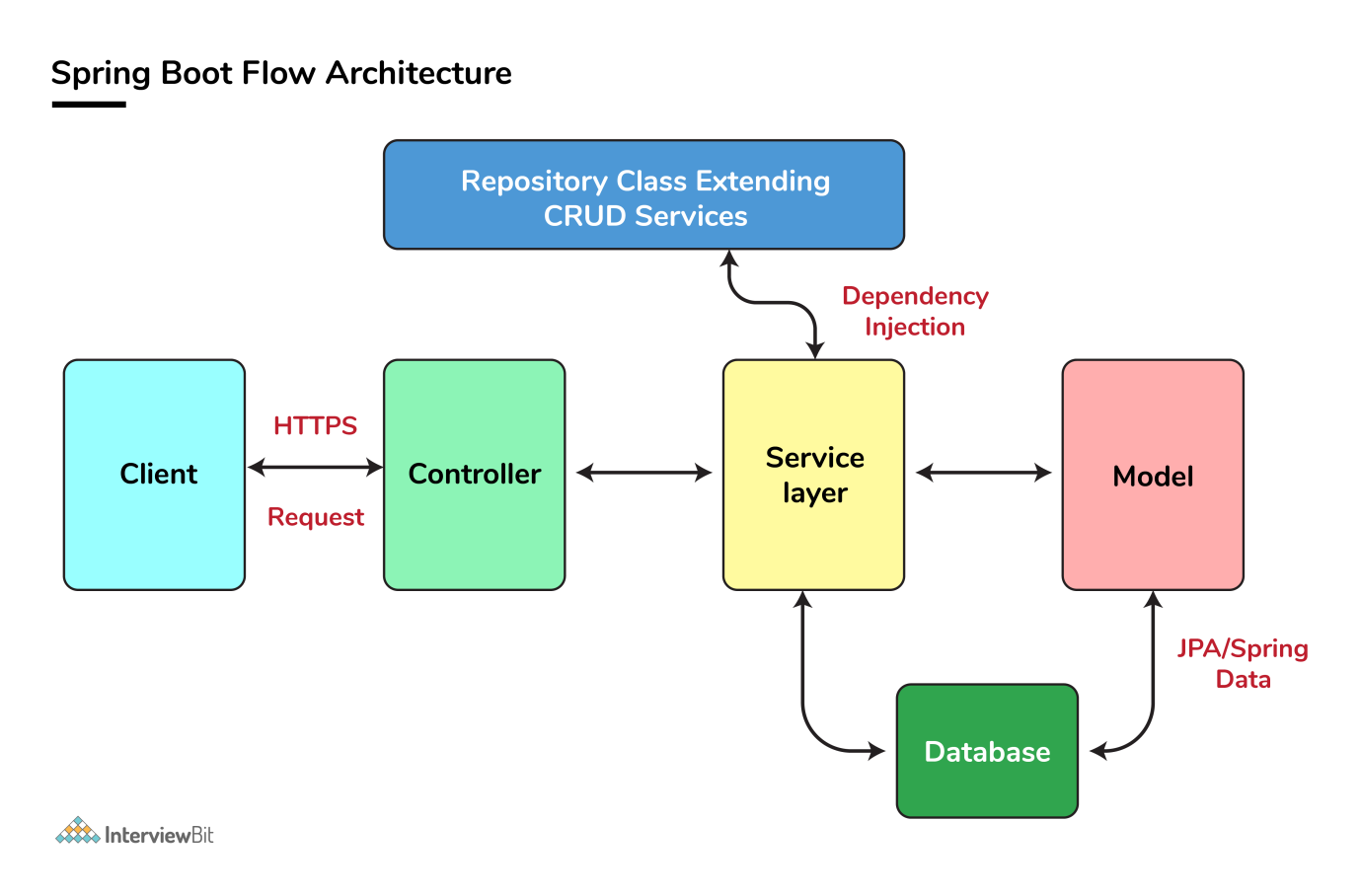
### **What is the difference between @RestController and @Controller in Spring Boot?**

@Controller is used to mark classes as Spring MVC Controller.

@RestController annotation is a special controller used in RESTful Web services, and it's the combination of @Controller and @ResponseBody annotation.

### **Describe the flow of HTTPS requests through the Spring Boot application?**

On a high-level spring boot application follow the MVC pattern which is depicted in the below flow diagram.



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

RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation. Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

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

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

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

### **What is Spring Actuator? What are its advantages?**

An actuator is an additional feature of Spring that helps you to monitor and manage your application when you push it to production. These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering, and many more that are automatically applied to your application.

### **How to enable Actuator in Spring boot application?**

To enable the spring actuator feature, we need to add the dependency of “spring-boot-starter-actuator” in pom.xml.

<dependency>

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

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

</dependency>

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

### **What are the actuator-provided endpoints used for monitoring the Spring boot application?**

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

* Health
* Info
* Beans
* Caches
* Conditions
* Configprops
* Env
* integrationgraph
* loggers
* Heapdump
* Threaddump
* Metrics
* scheduletasks
* Mappings
* Httptrace
* Shutdown

**/beans:**

 It displays a complete list of all the Spring beans in your application.

**/mappings:**

 This shows a collated list of all

@RequestMapping

 paths.

**/caches:**

 Exposes available caches.

**/loggers:**

 Lists down a list of all the configured loggers in your application with their corresponding log levels.

**/sessions:**

 Allows retrieval and deletion of user sessions from a Spring Session-backed session store.

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

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

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

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

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

Debugging logs can be enabled in three ways -

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

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

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

### **What is dependency Injection?**

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

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

### **What is an IOC container?**

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

### **Mention the 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 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.

**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

### **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.

### **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.

### **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

### **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.

### **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:

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

### **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.

### **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

|  |  |
| --- | --- |
| 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; |

### **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 |

### **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.

### **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). 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 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:

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

|  |  |
| --- | --- |
| 1 | <packaging>jar</packaging> |

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

|  |  |
| --- | --- |
| 1 | <packaging>war</packaging> |

### **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.

@Transactional(readOnly = true)

### **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.

### **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:

|  |  |
| --- | --- |
| 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 |

@Autowired

### **private** Example configuration;

Map map = **new** HashMap();

map.put("message", configuration.getMessage());

map.put("number", configuration.getNumber());

map.put("key", configuration.isValue());

### **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.

### **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

### **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 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:

|  |  |
| --- | --- |
| 1  2  3 | @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:

|  |  |
| --- | --- |
| 1  2  3 | {  "customername": "Rohit"  } |

Response Content

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | {  "customername": "Rohit"  "\_links": {  "self": {  "href": "<http://localhost:8080/sample/1>"  },  "sample": {  "href": "<http://localhost:8080/sample/1>"  }  } |

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

### **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 does path=”sample”, collectionResourceRel=”sample” work with Spring Data Rest?**

|  |  |
| --- | --- |
| 1  2  3 | @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.

### **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.

### **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.

### **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 |

### **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 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.

### **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

### **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.

### **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:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>com.h2database</groupId>      <artifactId>h2</artifactId>      <scope>runtime</scope>  </dependency> |

### **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

### **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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <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

### **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.

### **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.

### **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:

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

### **System Requirements Spring Boot 2.7.1**

Spring Boot 2.7.1 requires [Java 8](https://www.java.com/) and is compatible up to and including Java 18. [Spring Framework 5.3.21](https://docs.spring.io/spring-framework/docs/5.3.21/reference/html/) or above is also required.

Explicit build support is provided for the following build tools:

| **Build Tool** | **Version** |
| --- | --- |
| Maven | 3.5+ |
| Gradle | 6.8.x, 6.9.x, and 7.x |

#### 4.2.1. Servlet Containers

Spring Boot supports the following embedded servlet containers:

| **Name** | **Servlet Version** |
| --- | --- |
| Tomcat 9.0 | 4.0 |
| Jetty 9.4 | 3.1 |
| Jetty 10.0 | 4.0 |
| Undertow 2.0 | 4.0 |

You can also deploy Spring Boot applications to any servlet 3.1+ compatible container.