



# Section 3 Part 1- How does the Spring boot works?

🌱 Starting point of the application:

```
@SpringBootApplication
//this is the shortcut to three annotations those are
public class JavabrainSpringBootApplication {

    public static void main(String[] args)
    /*This is the entry point for a Spring Boot application.
    Let's break it down line by line and understand what happens when you run
    */
    {
        SpringApplication.run(JavabrainSpringBootApplication.class, args)
    }

}
```


## 🌱 @springBootApplication:


- This is the combination of three different annotations. @configuration, @EnableAutoConfigurtion @ComponentScan


@Configuration: This tells the spring boot that the class with this specific annotation is the configuration class where we will define all the beans. {beans are nothing but objects that are automatically taken care by spring boot. This class has the manual configuration in it.


Lets see it in details:

# Why we need @configuration when spring boot does all the auto configuration for us?

 We want to be more flexible and also need an advantage of customization. Even though spring boot configures everything for us there can be some scenarios where we do not need to go with the default configuration that spring has to provide to us.

 Auto-Configuration is generic but sometimes our needs can be specific.

 Whatever we define in the pom.xml files those are the dependencies that spring auto configures on our behalf. All this configuration is default configuration. Almost 80% of tasks of application can be smoothly done with this default configuration but its the remaining 20% manual intervention is required to make some configuration which spring can not provide .

 even if spring provide the default configuration for the rest 20 percent as well it wont be the wise choice to go with the default one for the application specific settings.



For example:

### 👉 **Default Auto-Configuration for Data Source (Database Connection)**

🚀 By default, if Spring Boot detects

`spring-boot-starter-data-jpa`, it will:



Auto-configure an

**H2 in-memory database** (if no database is specified).



Use default properties like

`username=sa`, `password=`, `driverClassName=org.h2.Driver`.

## 1 When we want to override the dependency defined in the pom.xml

what if you want a MySQL database instead of H2?

We **override the configuration** using `@Configuration` and `@Bean`.

`@Configuration`

```
public class DatabaseConfig {
```

`@Bean`

```
public DataSource dataSource() {
```

```
    HikariDataSource dataSource = new HikariDataSource();
```

```
    dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/mydb");
```

```
    dataSource.setUsername("root");
```

```
    dataSource.setPassword("password123");
```

```
    return dataSource;
```

```
}
```

```
}
```

### ✅ **Why do we need `@Configuration` here?**

Because we want to override the default Data Source Configuration that spring boot provides.

## 2 On the other hand auto configuration doesn't know everything about your Application.

Spring Boot auto Configure things like:

- 🔗 Embedded Tomcat server
- 🔗 Default Data Base Connection
- 🔗 Default Security Settings

But what if you **have a custom business logic component** that Spring Boot **doesn't know about**?

Creating the custom logic :

```
public class MyCustomLogger {  
    public void log(String message) {  
        System.out.println("LOG: " + message);  
    }  
}
```

But here in this case the spring will not understand that this is the custom configuration that we wish to inject where ever it is necessary.

what we do is:

```
@Configuration  
public class AppConfig {  
  
    @Bean  
    public MyCustomLogger customLogger() {  
        return new MyCustomLogger();  
    }  
}
```

✅ Now, this bean can be **@Autowired** anywhere in the project. {we will see what is autowired in further sections}

### 3 To modify auto-configured beans:

Spring also provide us the auto configured beans which we will learn in further sections. For now just considered @RestTemplate we will see what does the rest template does in further sections. For now just understand that the configuration class is also used to change the default behavior of the default bean that is rest template.

```

@Configuration
public class RestTemplateConfig {

    @Bean
    public RestTemplate restTemplate() {
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.setRequestFactory(new SimpleClientHttpRequestFactory() {
            setConnectTimeout(5000); // Set timeout to 5 seconds
        });
        return restTemplate;
    }
}

```

Its okay if you do not understand the code for now. All I have to show is that this is the configuration class that the spring boot will see when we will run the application and it will know that we have manually defined the bean when @SpringBootApplication will get initiated and it has this @Configuration annotation in it which will take care of all the custom definitions.