@Configuration and Auto-Configuration in Spring Boot

What is Auto-Configuration

Auto-configuration in Spring Boot will automatically help you configure your Spring application dependencies based on the jar dependencies that you have added. This helps speed up development time in case you don't want to write any explicit configuration code.

For example, if you have the Spring Boot Redis Starter dependency included: Redis AutoConfigure, and you don't have an explicit Redis configuration configured by your Spring Boot application, then Spring Boot will attempt to auto configure one for you based on the available properties that are prefixed with `spring.data.redis`.

> You can check out more detail about what properties is taken from the application.properties file here.

It is to note that, auto-configuration in Spring Boot is considered to be "non-invasive", meaning, if you start to define your own configuration that replaces part of the auto-configuration, then the auto-configuration will not kick in.

> If you would like to find out which part of auto-configuration is being applied, you can run your application with the --debug flag or put in `debug=true` within the application.properties to generate a report on what auto-configuration is being used.

Additionally, to use auto-configuration you must enable it in your Spring project by using the . If you use Spring Boot then the `@SpringBootApplication` annotation automatically include `@EnableAutoConfiguration`.

How to Write Your Own Auto-Configuration

All auto-configuration classes should be annotated with `@Autoconfiguration`. That annotation by itself is also just annotated with `@Configuration`, but auto-configuration classes should also have additional `@Conditional` annotation such as:

- `@ConditionalOnClass`: To only allow the auto-configuration to kick in if certain classes are available
- `@ConditionalOnMissingBean`: To only allow the auto-configuration to kick in if certain beans are not defined
- `@ConditionalOnProperty`: Let you create auto-configuration based on Spring Environment property

The reason for also adding in those conditional annotations is to make sure that the auto-configuration should only when relevant classes are found or configuration are missing if the user doesn't declare their own `@Configuration`.

Besides that, auto-configuration classes are just essentially `@Configuration` classes that creates beans, you can use `@ConfigurationProperties` to also pull in values from application.properties for when creating your beans

How does Auto-Configuration Get Detected by Spring Boot?

In order for Spring Boot to check whether or not to use the auto-configuration class that you have defined, Spring Boot check for the presence of a `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` in the published jar. The file should include the fully qualified name of your class that does the auto-configuration, with one class per line such as the following example:

```
com.mycorp.libx.autoconfigure.LibXAutoConfiguration
com.mycorp.libx.autoconfigure.LibXWebAutoConfiguration
```

> You can also add in comments within the imports file

Adding Ordering

If you also need your configuration to be applied in a specific order, you can use the `before, beforeName, after` and `afterName` attributes on the `@AutoConfiguration` annotation or the dedicated annotation `@AutoConfigureBefore` and `@AutoconfigureAfter` annotation.

Revision #3
Created 2025-07-16 23:12:11 UTC by Tamarine
Updated 2025-07-17 00:40:10 UTC by Tamarine