### Reference links

1. https://github.com/spring-projects/spring-boot/releases/tag/v3.3.5

How response Is converted to json :-

@ResponseBody + JacksonHttpMessageConverterAutoConfiguration class or JacksonAutoConfiguration

Class

ErrorMvcAutoConfiguration

Is responsible for getting that white label error page , from this autoconfiguration class only that text is coming from., I mean if mappings are not found for the requested url then those scenarios are handled by this class

# **What is the advantage of spring boot**

1. Auto Configuration –refer spring.factories file in auto configuration jar and their Packages
2. Starter dependency - wide variety of starters –no need to solve version conflicts,
3. Servers- just need to add jars –tomcat server will come automatically
4. With spring boot we can develop microservices

**Auto configuration example :-**

All the logic for Spring Boot Auto Configuration

is present in a jar file

called Spring Boot Auto Configure dot Jar.

Beans like datasource, Dispatcher servlet, Internal view resolver will be configured automatically

And misc beans like json bean converter and many other required beans all will be created automatically by \*Autoconfiguration classes, which will internally work based on

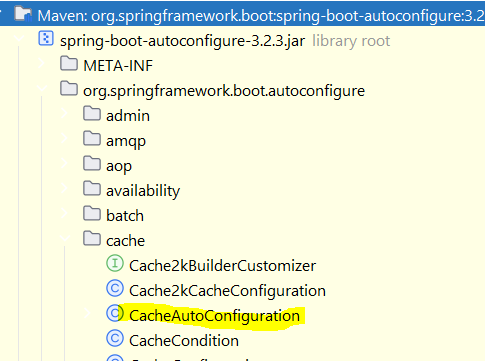
Annotations like @ConditionalOnClass() , @ConditionalOnBean(), @ConditionalOnMissingBean() annotations

Auto configuration

1. If u want to know what beans have been automatically configured for you by spring boot for us then

Open jar named “spring-boot-autoconfigure” and open related package which u are looking for as below

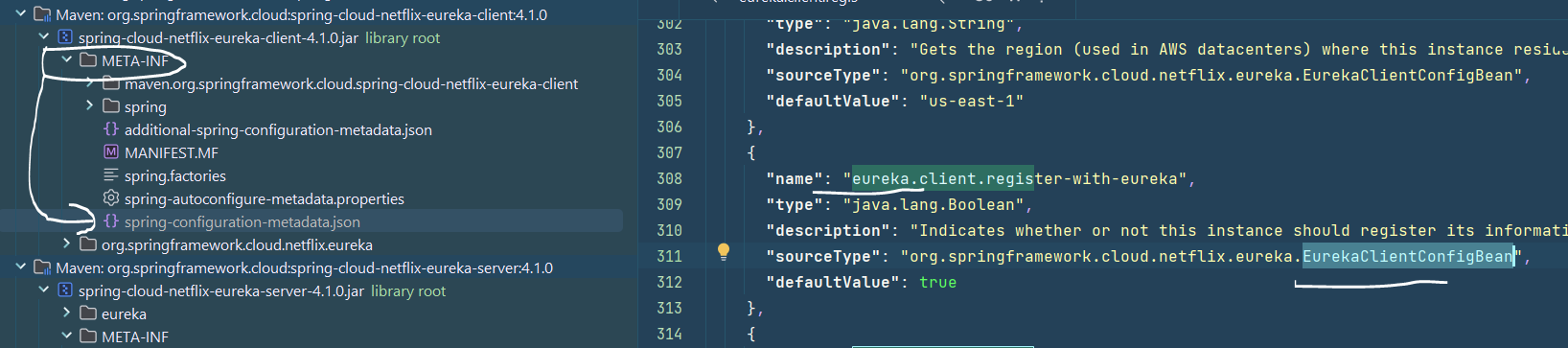
Or u can try this 🡪If you need to find out what auto-configuration is currently being applied, and why, start your application with the --debug



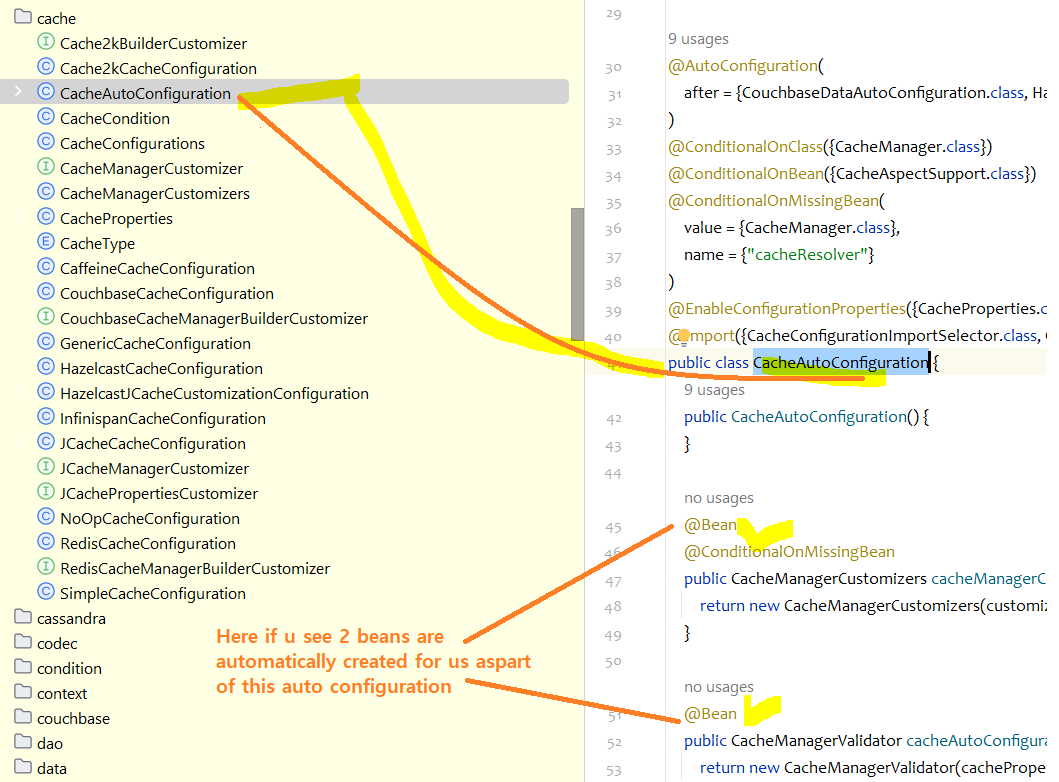
1. All the properties u keep will be binded to some @ConfigurationProperties bean class

If u want to know to which class it is binded then either “Ctrl+shift+F” and search for that property using “Directory” or “scope” button

Or Open any/relevant jar’s META-INF/“spring-configuration-metadata.json” there ur property will be present and it have the class name to which it is going to bind



1. Open related \*Autoconfiguration class and see which beans are automatically configured by that auto configuration class



1. For jdbc auto configuration we should open 🡪

DataSourceAutoConfiguration.class

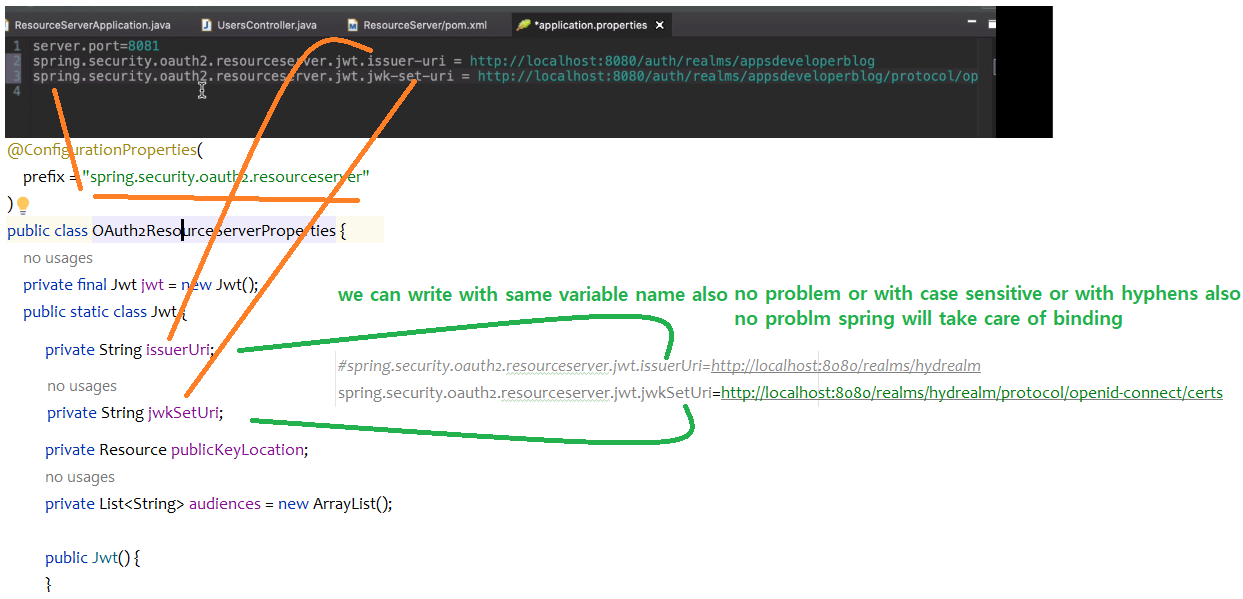
## Annotations

|  |  |
| --- | --- |
| @EnableAutoConfiguration |  |
| @Import(\*Configuration.class) | This annotation is used to import the configuration class  If all configuration classes are present in same package then all be scanned, but if u want to import specific cfgn class present in another package which is not as part of comp scan then this annotation will be used |
| @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class }) | To exclude |
| @AutoConfigurationPackage | If u write any custom auto configuration file, then either add that class under comp scan  Or add this anno to ur main class -- |
| @Autowired | While using constructor injection, generally u don’t need @Autowired annotation, but if u keep 2 constructors  Spring doesn’t know which constructor to call to create bean, so u have to keep @Autowired on the constructor which u want spring to invoke    @Autowired  public MyAccountService(RiskAssessor riskAssessor) {  this.riskAssessor = riskAssessor;  this.out = System.out;  }  public MyAccountService(RiskAssessor riskAssessor, PrintStream out) {  this.riskAssessor = riskAssessor;  this.out = out;  } |
|  |  |
|  |  |
|  |  |

## Auto configuration properties

1. Generally, if u add a property some bean will be created for us as part of @ConditionalOnProperty annotation, if u want to see what are those properties and

To which class these are getting binded then follow below



If u see here these 2 properties are binded to this properties class

### Exclude

@SpringBootApplication(exclude = JdbcTemplateAutoConfiguration.class,  
 excludeName = "org.springframework.boot.autoconfigure.jdbc,JdbcTemplateAutoC.class")

public class MyApplication {

}

1. If the class is not on the classpath, you can use the excludeName attribute of the annotation and specify the fully qualified name instead. If you prefer to use @EnableAutoConfiguration rather than @SpringBootApplication, exclude and excludeName are also available.
2. Finally, you can also control the list of auto-configuration classes to exclude by using the spring.autoconfigure.exclude property.

## Auto-configuration Packages

For example, If you wrote your custom autoconfiguration class which could be in some other package

if u want to include it or to be scanned by spring then u should include that auto configuration class package

In this annotation

@AutoConfigurationPackage(basePackageClasses = JdbcTemplateAutoConfiguration.class,  
 basePackages = "com.nt")

Auto-configuration packages are the packages that various auto-configured features look in by default when scanning for things such as entities and Spring Data repositories. The @EnableAutoConfiguration annotation (either directly or through its presence on @SpringBootApplication) determines the default auto-configuration package. Additional packages can be configured using the @AutoConfigurationPackage annotation