### Reference links

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

|  |  |
| --- | --- |
| to see all versions of spring boot | <https://docs.spring.io/spring-boot/docs/> |
| single reference | <https://docs.spring.io/spring-boot/docs/3.0.0/reference/htmlsingle/> |
| to see all the dependency versions  ex:- for spring boot 3.2 which hibe jar we hav to use | <https://docs.spring.io/spring-boot/docs/2.7.x/reference/html/dependency-versions.html> |
| to read full documentation of what is released in each version read this doc | <https://github.com/spring-projects/spring-boot/releases/tag/v3.3.5> |
| caching using ehcache | <https://springframework.guru/using-ehcache-3-in-spring-boot/>  <https://github.com/spring-framework-guru/sfg-blog-posts/tree/master/spring-boot-ehcache> |
|  |  |

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

Importatnt points while working with boot

Before adding any jar, check the parent pom “spring-boot-dependencies” and see if they already had this jar, then

1. Add only groupid, artifact id in your child pom
2. If u still want to override/ take the new jar then keep the new version number in this property

Ex:- <assertj.version>3.18.1</assertj.version> //This is one of the property they have defined <properties> tag , so if we want to upgrade/downgrade just keep this prop in new

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

SAM (servers, starters, autoconf, act, microservices, @cond)

1. Auto Configuration –refer spring.factories file in auto configuration jar and their Packages
2. Actuators - /actuators/heapdump, /actuators/env, /actuators/conditions
3. Starter dependency - wide variety of starters –no need to solve version conflicts,

Curated dependencies are present in “spring-boot-dependencies” pom under “<dependencyManagement> tag” in this pom they have defined around 464 dep

Means if we want to use any of them we can straight away include them without mentioning the version, as the version number will be inherited from parent

They would have made ton of testing to check the compatibility with these jar version, so we can happily use those dependencies without compatibility worry

1. Servers- just need to add jars –tomcat server will come automatically
2. With spring boot we can develop microservices
3. @ConditionalAnnotations

**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

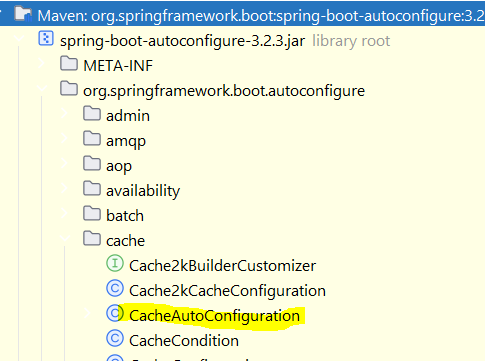
Annotations like @ConditionalOnClass() , @ConditionalOnBean(), @ConditionalOnMissingBean() annotations

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

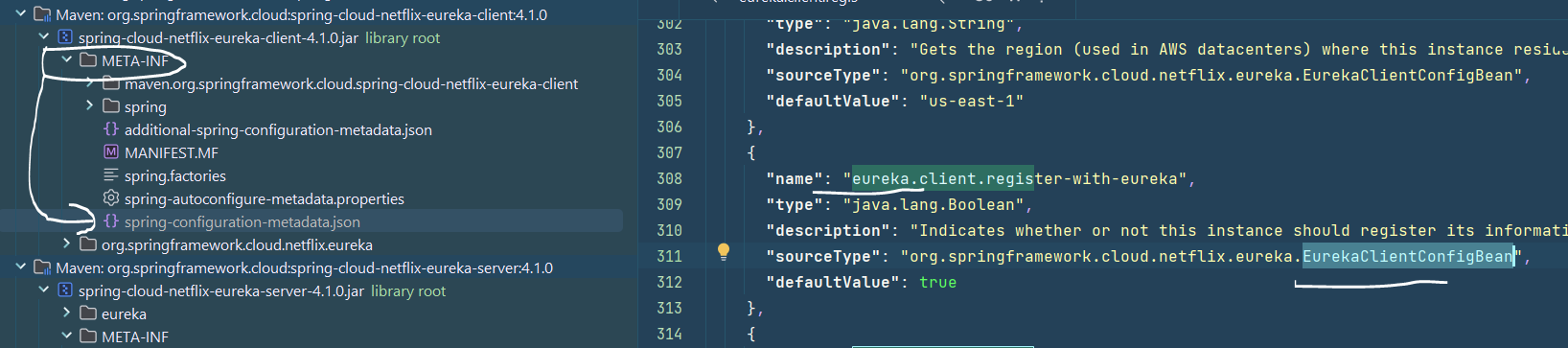
Or u can try this 🡪If you need to find out what auto-configuration is currently being applied, and why, start your application with the --debug



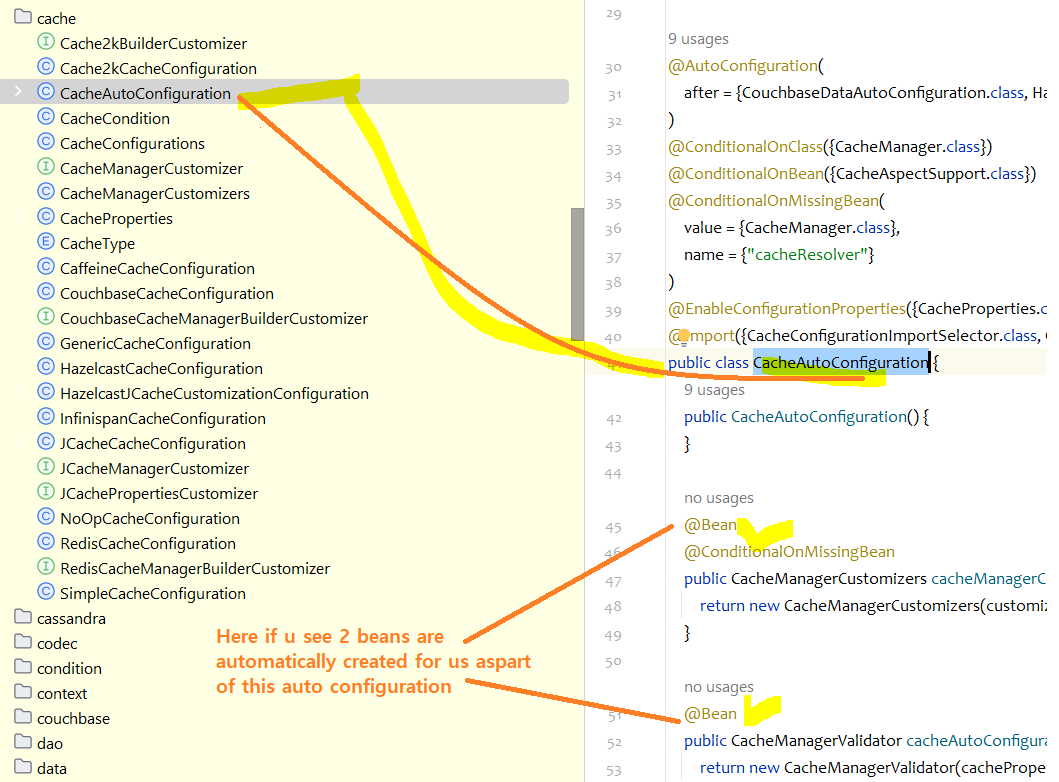
1. All the properties u keep will be binded to some @ConfigurationProperties bean class

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



How to find auto configuration class name

|  |  |
| --- | --- |
|  | for specific libraries, spring will provide a autoconfiguration jar,  in that open this file , there we can see which class is doing the auto configuration  here in above file auto configuration class is clearly mentioned |

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;  } |
|  |  |
|  |  |
|  |  |

How spring boot downloads the jar with version

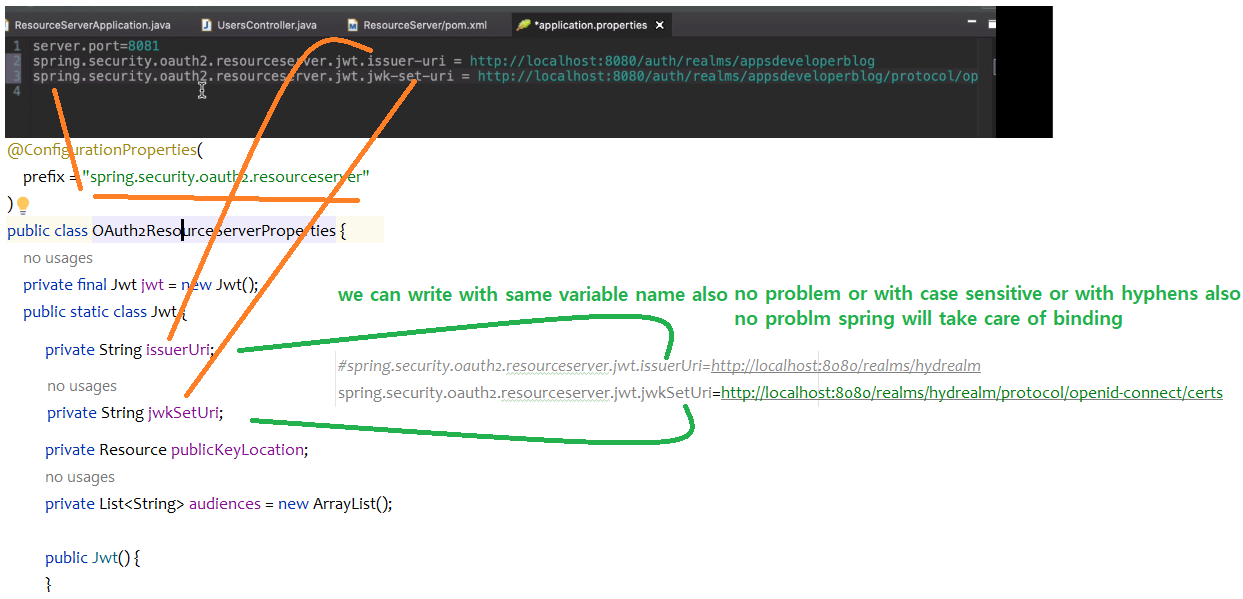
1. Compatible jars are already defined in maven BOM
2. And In spring website
3. Compare the jar released times

|  |  |
| --- | --- |
| dependencies **{** compileOnly 'org.projectlombok:lombok'  }  eventhough we defined jar without version, still maven/gradle downloads the exact version  bec all those versions are pre-defined by spring  if u want to see which jar version is compatible with spring boot parent  they are maintaining all the compatible versions  <https://docs.spring.io/spring-boot/docs/2.7.x/reference/html/dependency-versions.html>  <https://docs.spring.io/spring-boot/3.3/appendix/dependency-versions/coordinates.html> | in build.gradle we will mention only the dependency name, not version then how it will download the exact version, internally it will go and download from spring boot BOM  <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-dependencies/3.3.8> as all the jars are mentioned with version in BOM  this has 100’s of compatible jars |
| **Maven Repository (mvnrepository.com):**   * Search for cxf-spring-boot-starter-jaxws on Maven Repository. * Look for versions that were released around the same time as Spring Boot 3.4.4. Spring Boot 3.4.4 was released on **March 20, 2025**. * While not always explicitly stated, release dates can give you a good indication of compatibility. * The Maven Repository might also have information in the dependency details about which Spring Boot versions a particular CXF version is compatible with, although this is not always guaranteed to be up-to-date. | |

## 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

Conditionally do anything in spring

|  |  |
| --- | --- |
| Conditionally we can load few property files | use case:-  support we have to connect to both mongodb and sqlserver , then we generally keep both those properties in application.yml file – if we hardcode both properties then problem is at any time we should provide both mongodb, sqlserver db details |