->Within Spring applications, you can use these @Enable<Technology> annotations to follow the convention over configuration pattern, making your apps easier to develop and maintain, and without worrying too much about its configuration.

->@EnableAutoConfiguration annotation -> AutoConfigurationImportSelector.class -> getCandidateConfigurations() -> loads spring.factories from META-INF folder.

->spring.factories defines all the auto-configuration classes that are used to set up any configuration that your application needs for running.

->can check in any technology auto configuration class, there will be @conditional annotations, import is applied only when conditions mentioned in @conditional are satisfied.

->auto configuration uses classpath to figure out what is needed for our app. @SpringBootApplication. This annotation is actually the @ComponentScan, @Configuration, and @EnableAutoConfiguration annotations.

**SpringApplication:** This class provides the bootstrap for the Spring Boot application that is executed in the main method. You need to pass the class that is executed.

To disable banner, we can use spring.main.banner-mode = off

When you have SpringApplication.run(SpringBootSimpleApplication.class, args), you have

access to the args in you beans

ApplicationArguments class bean provides access to the arguments that were used to run a SpringApplication.

One of the best features from Spring (and Spring Boot) is access to the properties values by using the @Value annotation.

They can be accessed from the org.springframework.core.env.Environment interface that extends

from the org.springframework.core.env.PropertyResolver interface.

We have the option to inject the properties via a command line.

$ java -jar target/myapp.jar --data.server=remoteserver:3030

can use a specialized environment variable named SPRING\_APPLICATION\_JSON to expose the same properties and its values.

$ SPRING\_APPLICATION\_JSON='{ "data":{"server":"remoteserver:3030"}}' java -jar target/myapp.jar

# Order of overriding application configuration properties

Spring Boot uses an order if you want to override your application configuration properties:

• Command-line arguments

• SPRING\_APPLICATION\_JSON

• JNDI (java:comp/env)

• System.getProperties()

• OS Environment variables

• RandomValuePropertySource (random.\*)

• Profile-specific (application-{profile}.jar) outside of the package jar.

• Profile-specific (application-{profile}.jar) inside of the package jar.  
• Application properties (application.properties) outside of the package jar.  
• Application properties (application.properties) inside of the package jar.  
• @PropertySource  
• SpringApplication.setDefaultProperties

* Spring-boot-configuration-processor dependency is used to process custom properties and code completion.