**SPRING BOOT**

**Spring boot 3.X works with java 17 or later**

**What’s the most important goal of spring boot ?**

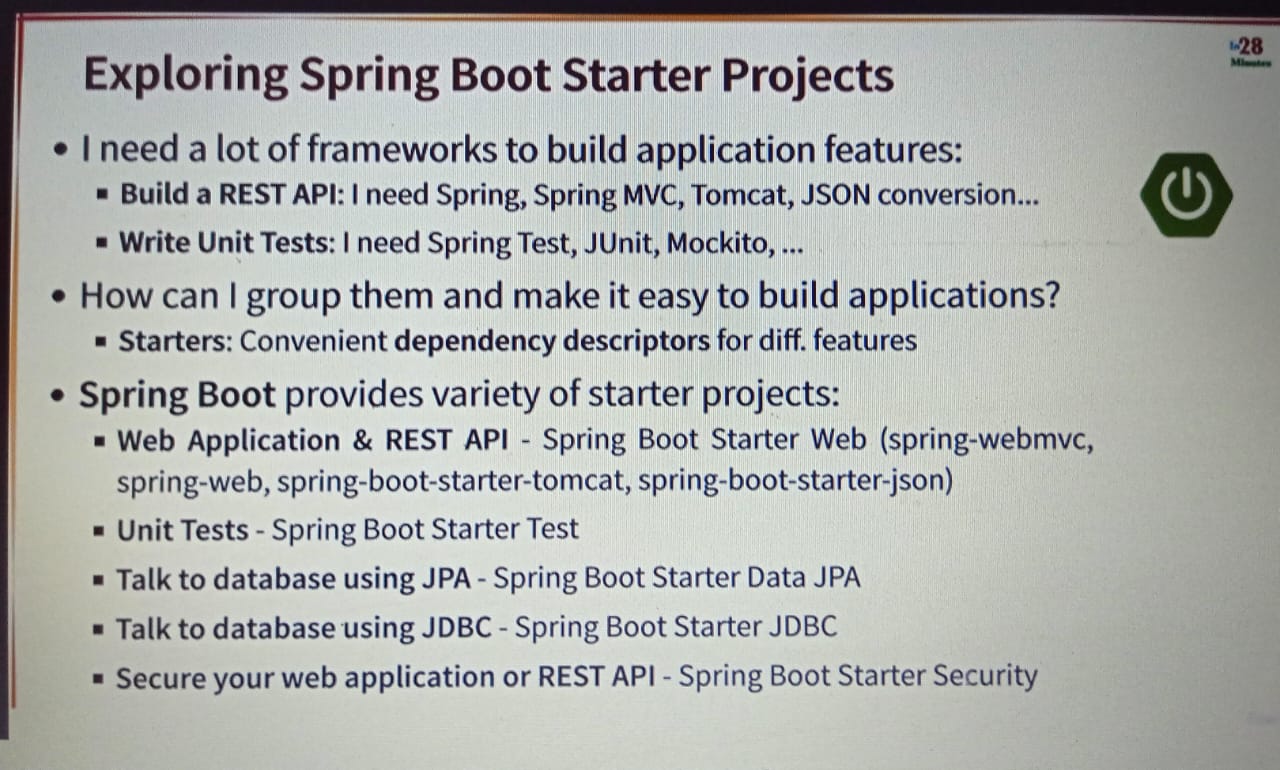
**ANS**: Help you to build production-ready apps quickly.

1. ***Build quickly***  
   spring Initializer

Spring boot starter projects  
spring boot auto configuration

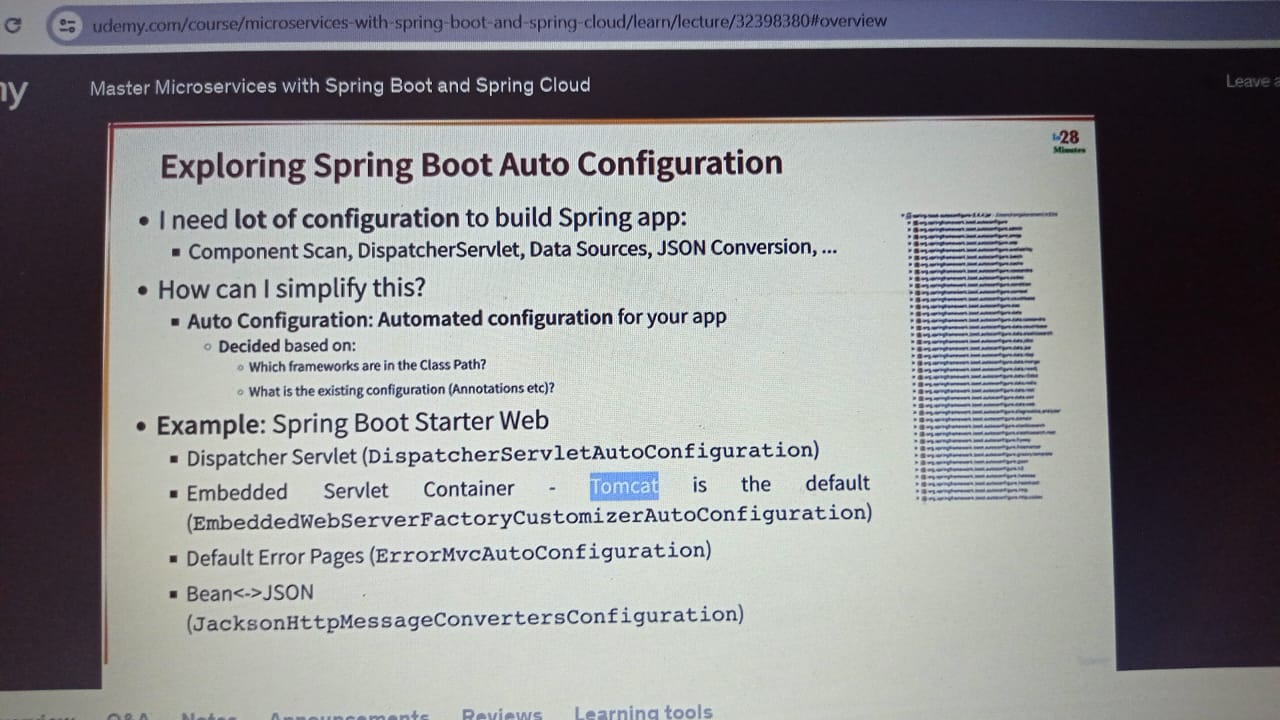
Spring boot Dev tools

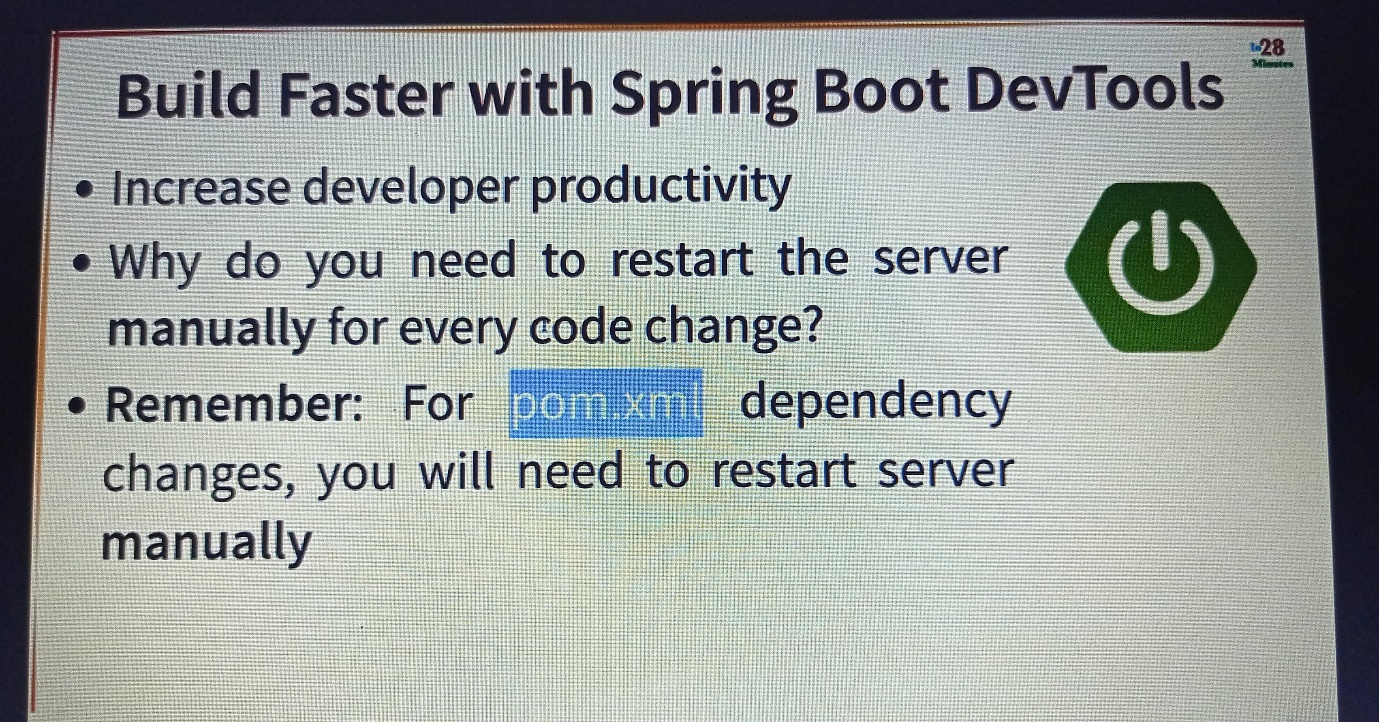
1. ***Be production ready***  
   Logging  
   Different Configuration for different environments -> profiles, Configuration properties,  
   Monitoring



Default logging level is info.

For printing all the dependencies use the below in application.properties file  
**logging.level.org.springframework = debug**

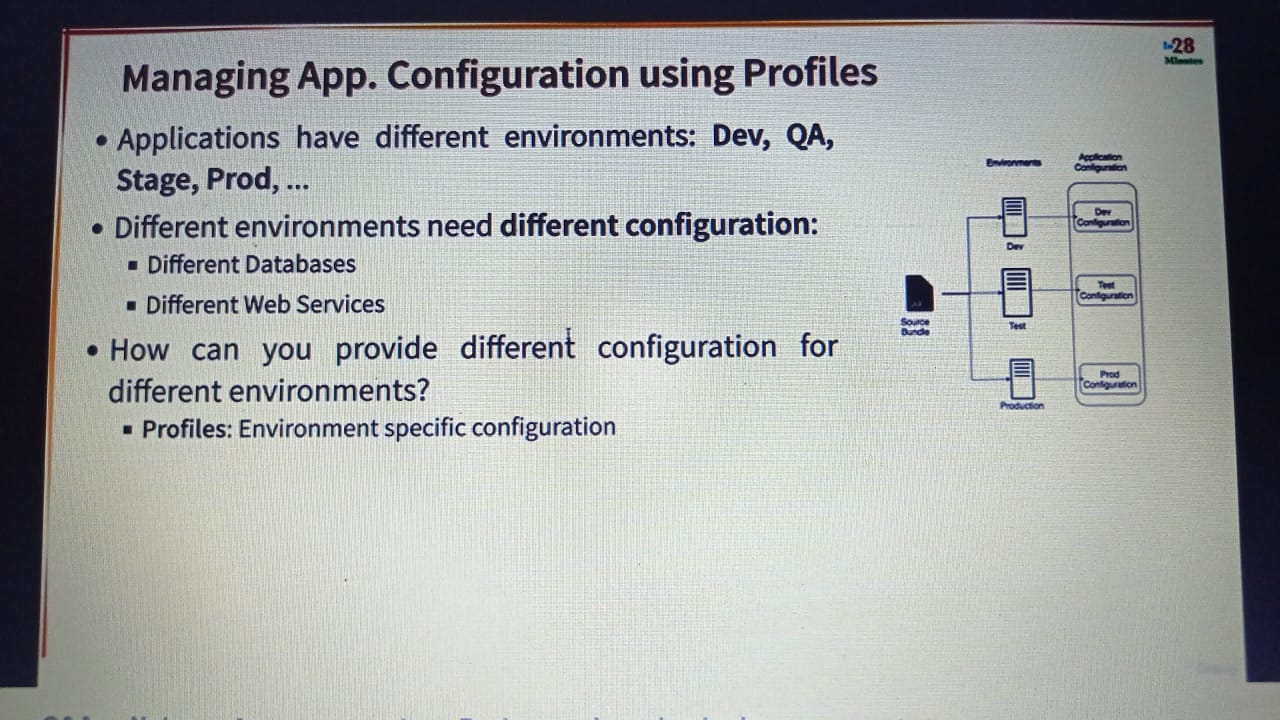




To use other environment in local add below in application.properties file  
**spring.profiles.active = prod**

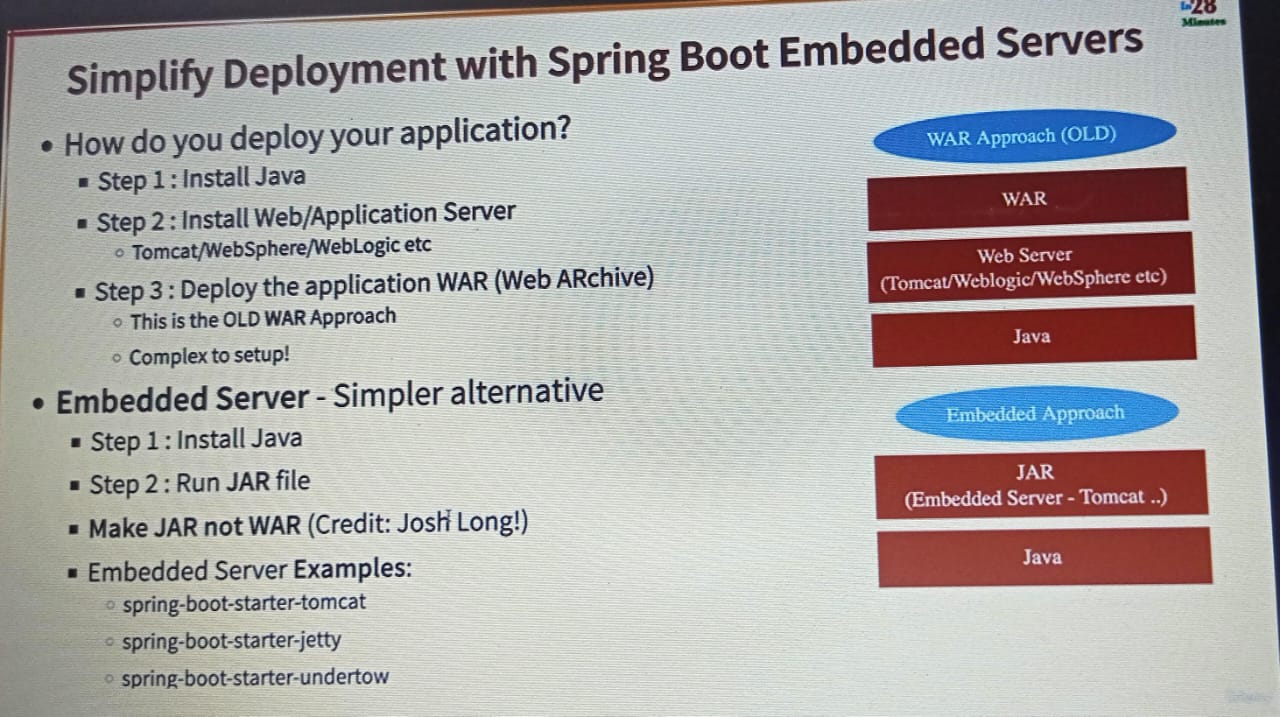
**Order of logging levels:**

1.**trace** -> prints everything there is to log.  
2.**debug** -> lot more information  
3.**info** -> print things logged at info level  
4.**warning** -> a little bit more information  
5.**error** -> only errors and exceptions will be printed  
6.**off** -> to turn off logging



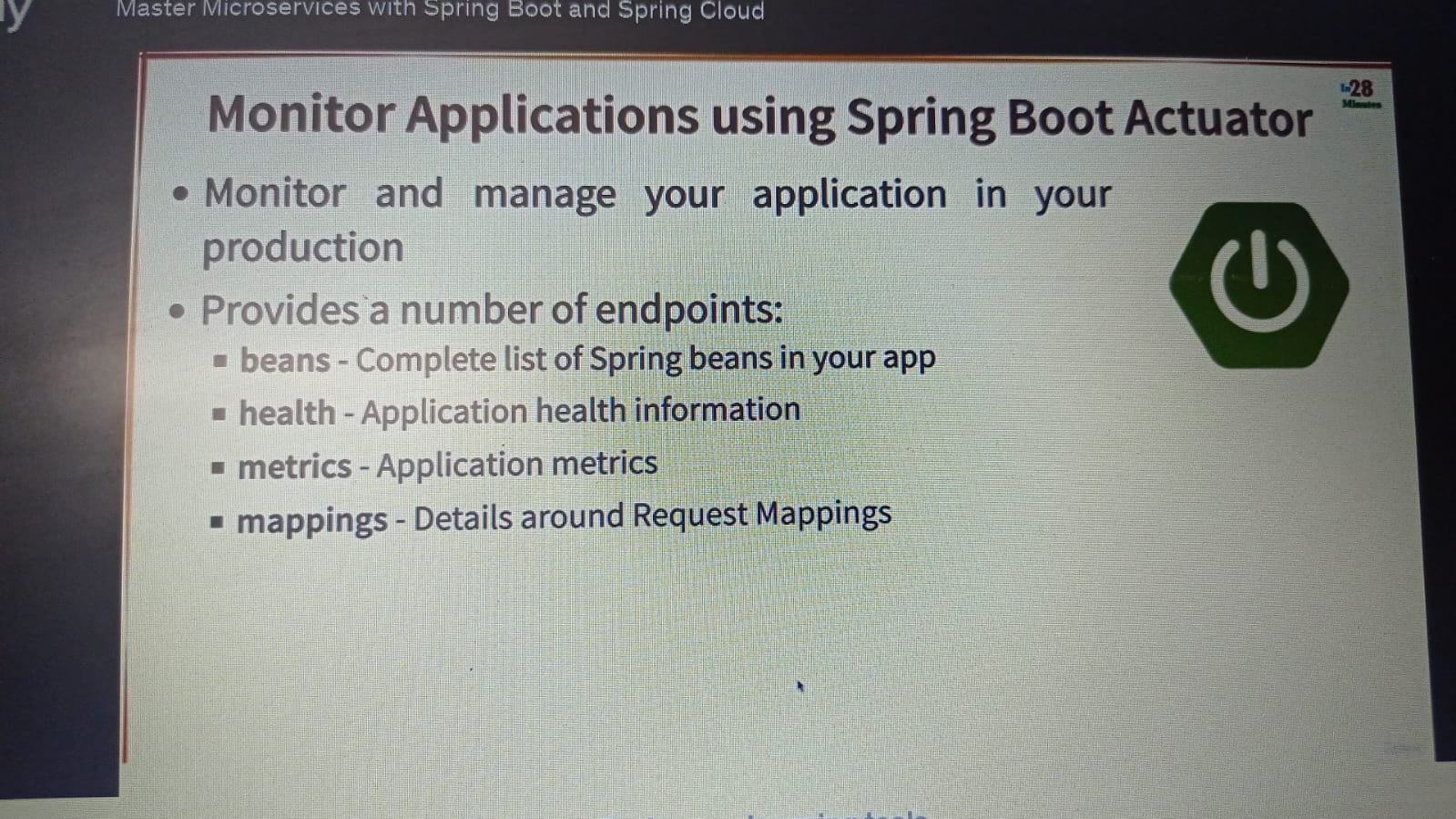
**maven clean install in maven build will give jar file.**

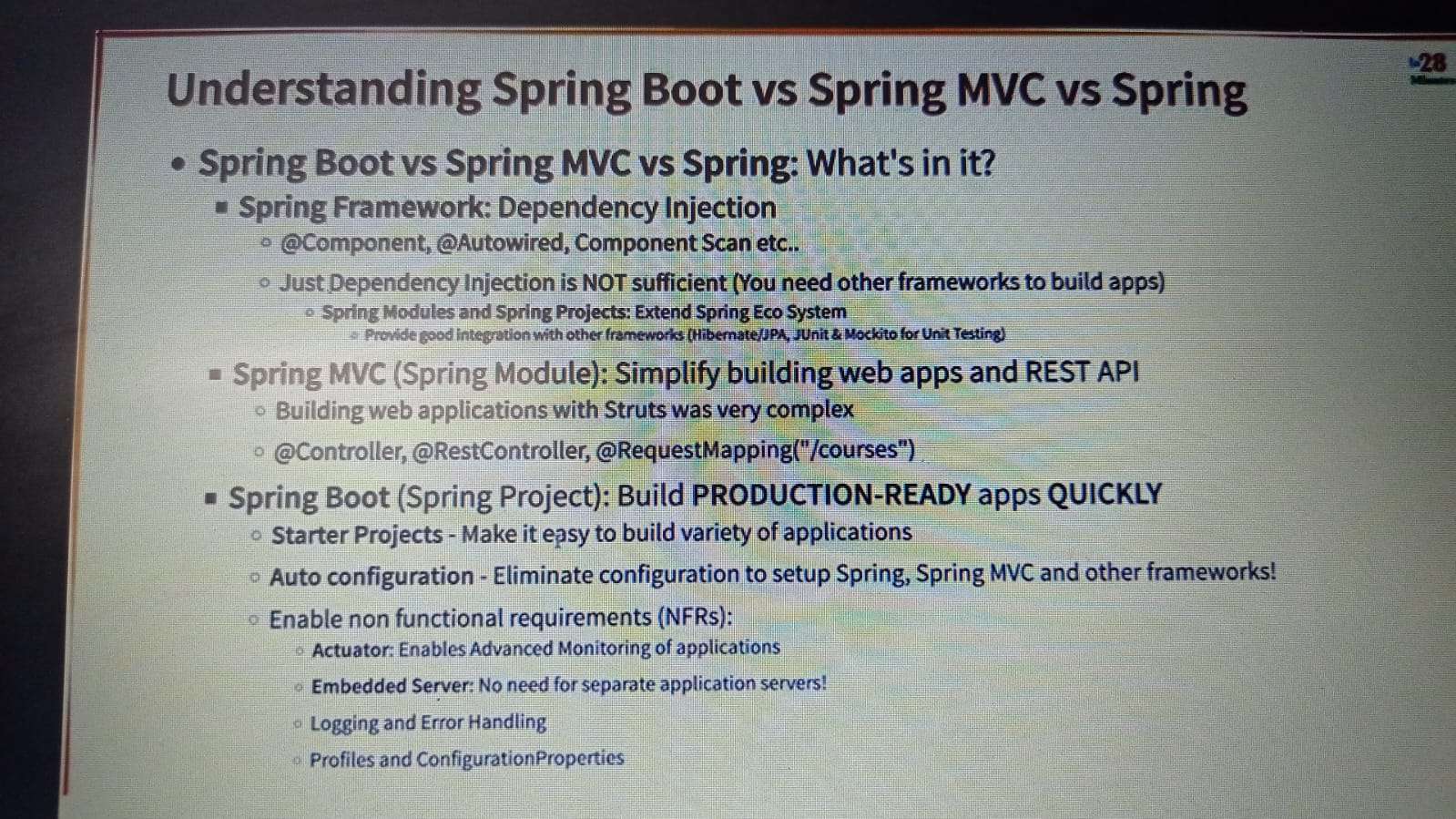
For running jar file use : **java -jar jarfilename**



Localhost:8080/actuator/ -> displays the exposed url  
localhost:8080/actuator/health -> displays status:UP which means UP and running.  
by default it only exposes health end point, if you want more features then include the below in application.properties file.  
**management.endpoints.web.exposure.include=\***

For only specific endpoints you can use   
**management.endpoints.web.exposure.include=health,metrics**

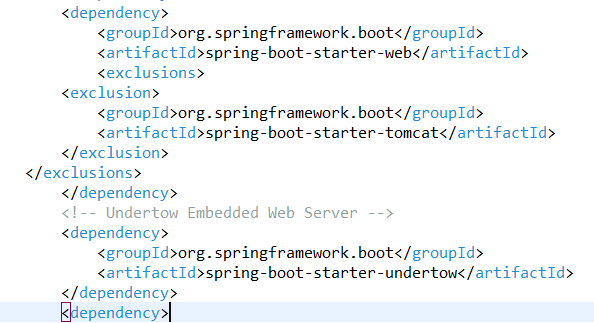


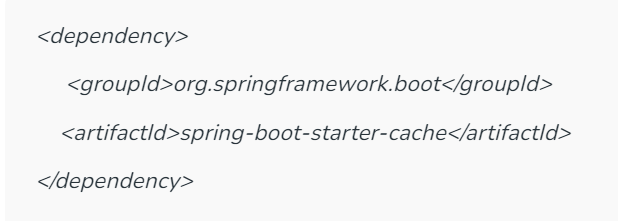


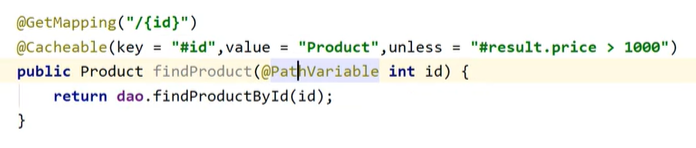
1st revision on April 28.

2nd revision on June 26.

* **How to change default tomcat server in springboot ?**

Other than tomcat you can use jetty or undertow.  
First you need to exclude tomcat server which is present in spring-boot-starter-web and include undertow or jetty as separate dependency.  


* **How to implement caching in spring boot ?**
* **Cache Annotations of Spring Boot**
* 
* Have **@EnableCaching** annotation on the class where you are going to cache.
* The below shown is cache based on condition. Where prices <1000 will be cached.



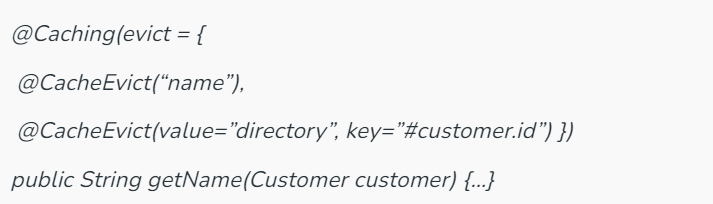
Once I delete the record from database then I need to refresh my cache.  
**@CacheEvict** annotation to remove values so that fresh values can be loaded into the cache again



**@CachePut** annotation can update the content of the cache without interfering with the method execution.

 One of the key differences between **@Cacheable** and**@CachePut** annotation is that the **@Cacheable** skips the method execution while the **@CachePut** runs the method and puts the result into the cache.

**@Caching** is used in the case we want to use multiple annotations of the same type on the same method.



With **@CacheConfig** annotation, we can simplify some of the cache configurations into a single place at the class level, so that we don’t have to declare things multiple times.

