**1 Spring Boot version Version**

Spring Boot 2.2.1 ,Spring 2.3.0

**2 where we will keep property files**

Resource package

**3 Profiles**

Test,prod,dev

**4 feature get from Spring Boot**

* Opinionated ‘starter' dependencies to simplify build and application configuration
* Embedded server to avoid complexity in application deployment
* Metrics, Health check, and externalized configuration
* Automatic config for Spring functionality – whenever possible

**5 ACCTUATORS usage**

METRICS,bean details , endpoints exposure for monitoring integration with devOps tools

**6 what is spring boot**

Spring Boot is basically an extension of the Spring framework which eliminated the boilerplate configurations required for setting up a Spring application.

**7 annotaions**

@SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan

**8 internal spring internal works (run as java application)**

-It all starts with adding dependency Starter POM will add jars in build path and create

spring.factoires in META-INF/ spring.factoires

- spring.factoires conatins what should be enabled/disbaled at run time based on @conditional and @Configuration

**Ex: spring-boot-autoconfigure in maven path serach META-INF/spring.factories**

Search for JPA repository go to class you can see in DataSource bean available then create JPA class

**OnBeanCondition,OnClassCondition,OnExpressionCondition,OnJavaCondition,OnJndiCondition**

**@SpringBootApplication =@Configuration+ @EnableAutoConfiguration +@ComponentScan**

@SpringBootApplication annotation will exculde scan of package and class also autoconfig exclude feature

Main method in Spring boot is to

- main method is not required in typical deployment of war file but how ever using in IDE or as execurable jar or war tat can run standalone with boot embeded tomcat by java -jar app.war

SpringApplication.run helps to keep up the application context intialize required bean

From run method main application context kicked off which turns searches for class annotatoed with @Configuration intialize all beans those configured class and stroes beans in JVM knows as IOC after creation of all beans automatically configures dispatcher servlet and register difflut handel mapping ,mesage converter and all basic things

**Internal Run flow**

-Create Application Context

-Create Application Type(web or standalone)

-Register annotated bean class with the context

- create an instance of Embeded Tomcat servlet container and adds the context

**10 swagger and purpose of Swagger**

Swagger provides a set of great tools for designing APIs and improving the work with web services: Swagger Editor – enables to write API documentation, design and describe new APIs, and edit the existing ones.Swagger Inspector allows to easily validate and test APIs with no limits on what you test

**11 Environment specific files**

Application-dev.properties/yml

**12 design pattern used in Spring Boot**

1. Singleton pattern (Singleton Bean ,Autowired Singleton)
2. Factory Method pattern
3. Application Context(BeanFactory -Spring uses this technique at the root of its [Dependency Injection (DI) framework](https://www.baeldung.com/spring-dependency-injection).Fundamentally, Spring treats a bean container as a factory that produces beans.Thus, Spring defines the BeanFactory interface as an abstraction of a bean container: Spring then extends BeanFactory with the ApplicationContext interface, which introduces additional application configuration. Spring uses this configuration to start-up a bean container based on some external configuration, such as an XML file or Java annotations.
4. External Configuration (based on property file profiles)
5. Proxy pattern(Transactional) [proxy pattern](https://www.baeldung.com/java-proxy-pattern) is a technique that allows one object — the proxy — to control access to another object — the subject or service.
6. Template pattern

**13 connecting db in boot**

- add db driver dependnecy and JPA dependenc in pom

Either use extrenalized config like below and directly access dataSource bean

spring.datasource.url=jdbc:h2:mem:test

spring.datasource.driver-class-name=org.h2.Driver

OR

Create bean of DataSource object with db config and process actions

**14 simple to print hello world what dependency we need to have**

Spring-boot-starter package

**15 jersey vs spring boot**

**JAX-RS**

JAX-RS is a [specification](http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-fr-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf) for implementing REST web services in Java, currently defined by the [JSR-370](https://jcp.org/en/jsr/detail?id=370). It is part of the [Java EE technologies](https://stackoverflow.com/a/37083274/1426227), currently defined by the [JSR 366](https://www.jcp.org/en/jsr/detail?id=366).

[Jersey](https://jersey.java.net/) (shipped with GlassFish and Payara) is the JAX-RS reference implementation, however there are other implementations such as [RESTEasy](http://resteasy.jboss.org/) (shipped with JBoss EAP and WildFly) and [Apache CXF](https://cxf.apache.org/) (shipped with TomEE and WebSphere).

**Spring Framework**

The [Spring Framework](http://projects.spring.io/spring-framework/) is a [full framework](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html) that allows you to create Java enterprise applications. The REST capabilities are provided by the [Spring MVC](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html) module (same module that provides model-view-controller capabilities). It is not a JAX-RS implementation and can be seen as a Spring alternative to the JAX-RS standard.

The Spring ecosystem also provides a [wide range of projects](https://spring.io/projects) for creating enterprise applications, covering persistence, security, integration with social networks, batch processing, etc.

worked with both Jersey Rest, spring rest and Jersey Rest with spring. Both of them are very rich frameworks with nice implementations. I would suggest it's better to go with Spring rest if you are using other Spring services such as ORM ,Spring security and DI etc. Both are spring libraries, so I feel a little bit easy for managing code and dependencies

| Spring Annotation | JAX-RS Annotation |
| --- | --- |
| @RequestMapping(path = “/troopers” | @Path(“/troopers”) |
| @PostMapping | @POST |
| @PutMapping | @PUT |
| @GetMapping | @GET |
| @DeleteMapping | @DELETE |
| @ResponseBody | N/A |
| @RequestBody | N/A |
| @PathVariable(“id”) | @PathParam(“id”) |
| @RequestParam(“xyz”) | @QueryParam(‘xyz”) |
| @RequestParam(value=”xyz”) | @FormParam(“xyz”) |
| @RequestMapping(produces = {“application/json”}) | @Produces(“application/json”) |
| @RequestMapping(consumes = {“application/json”}) | @Consumes(“application/json”) |

**JAX-RS pros:**

JSR standard can be run without servlet container (grizzly, simple, ...)

Production-ready implementations (jersey, cxf, resteasy, restlet, ...) designed for REST applications only

**Spring MVC pros:**

Provide "full" stack, not just REST facilities

Dependency injection / AOP / Transactions

Pluggable view templates (JSP, freemarker, velocity, ...)

**16 feature helps in checking production readiness of an application**

Actuator

**17 context path in Spring boot applicaiton**

server.servlet.context-path=/baeldung

**18 how to access value defined in application properties**

Using @Value , @Property values for single value,

@PropertyValue if its group values mapped to entity

Autowire Envirnoment to access values from property files

**19 how to config logging using application properties**

logging.level.root=WARN

logging.level.com.baeldung=TRACE

**20 if parent bean have prototype scope wt is the scope of child bean**

it will be a singleton since it is the default scope.

A child bean definition inherits constructor argument values, property values, and method overrides from the parent, with the option to add new values. Any initialization method, destroy method, and/or static factory method settings that you specify will override the corresponding parent settings.

The remaining settings are always taken from the child definition: depends on, autowire mode, dependency check, singleton, scope, lazy init.

**21 Handle exception in boot**

@ControlAdvice

**22 Achieve transnational management**

@Transactional