



Antonio Piovesan

Cybersecurity R&D Committee
member at Datalogic



Who Am I - Antonio Piovesan

<https://github.com/ant1974/ant-repo-public>

Introduzione (`.: the boot .:`)

- Spring Boot rende facile creare applicazioni/web-application di qualità.
- Approccio «opinionated» all'uso del framework Spring e altre librerie evitando codice **boilerplate**
- Ridotta necessità di configurazione

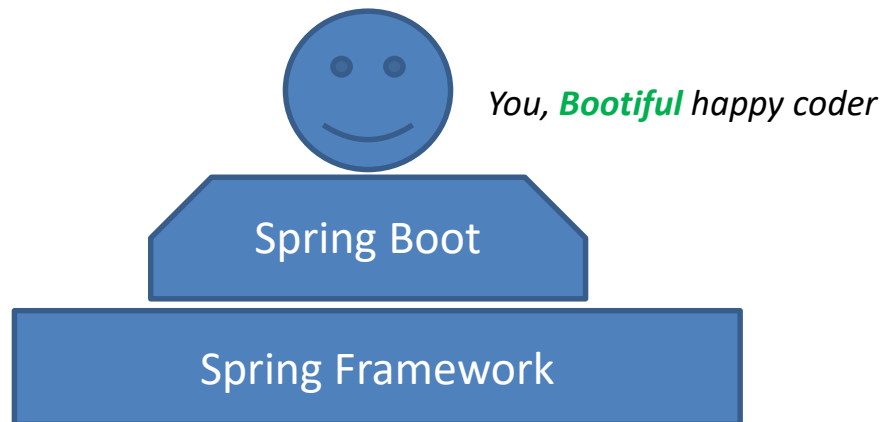
Scopi principali di Spring Boot sono:

- Fornire un avvio molto rapido nello sviluppo usando Spring
- Presumere di fare il meglio per noi, ma lasciandoci la possibilità di scelte personalizzate
- Fornire una vasta gamma di features non funzionali che siano ricorrenti nei progetti SW (web server, sicurezza, Internationalization, databases)
- Mancanza di java code generation e nessuna config. XML richiesta

Introduzione (.. it follows ..)

Alcuni vantaggi di Spring Boot

- Si evitano problemi di conflitti tra versioni e dipendenze: Spring Boot le risolve per noi
- Eccellente integrazione con i più usati IDE (Eclipse – Visual Studio Code – IntelliJ Idea).
- Rapidità di sviluppo e test anche attraverso l'uso di Web Server integrato (Tomcat /Jetty)
- Completa assenza di codice ripetitivo (**boilerplate** code)



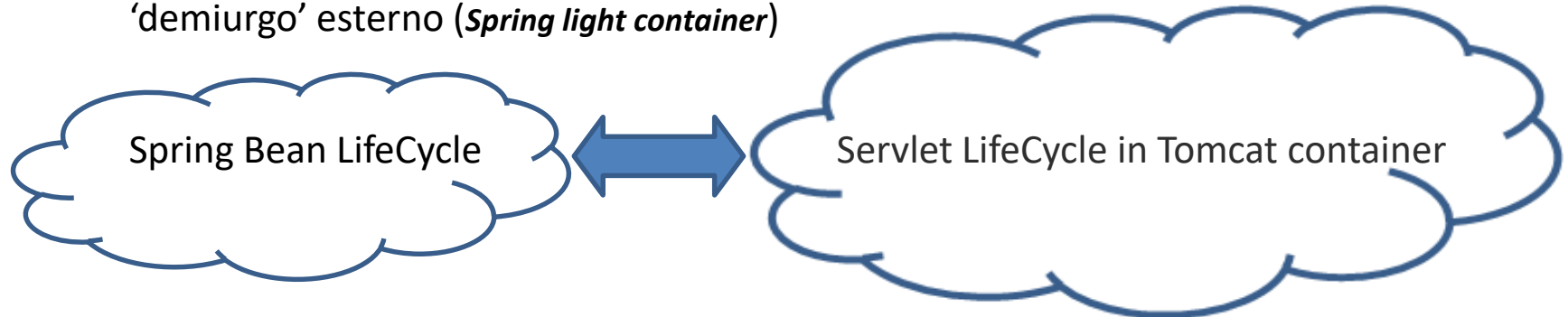
Spring (Say what ?)

- Defacto Standard per sviluppo Java EE
- Light container che offre
 - A. IOC / DI container
 - B. Supporto nativo allo Aspect Oriented Programming
- Ricchezza di «moduli» costruiti con (A) e (B) [<https://spring.io/projects/>]
- Ricchezza di documentazione [<https://spring.io/learn>]

- ## IOC/DI

Hollywood Pattern «Don't call us, we will call you» ...

Si rovescia la dipendenza: un oggetto di tipo A che necessita di un oggetto di tipo B non lo crea ma ne richiede una istanza di classe B (singleton o stereotyp-ed) ad un 'demiurgo' esterno (*Spring light container*)



Spring (.. let's keep it flowin' ..)

- Aspect Oriented Programming (AOP)

Codice cross-cutting/ortogonale

Auditing / Loggin'

JDBC/SQL Transaction

Security / Access Control

Advice: **cosa** fare (commit o rollback ad esempio) .. Cosa invocare **quando** ...

Pointcut: regola che esprime **quando** applicare lo **Advice**

Joinpoint: punti di esecuzione a runtime del codice definiti dal **Pointcut** (specifica del quando)

Aspect = combinazione di **Advice** e **Pointcut**

Before

Around

AfterReturning (exits by normal return)

Afterthrowing (exits by throwing an exception)

After (normal + exception)

```
@SpringBootApplication
@RestController
public class DemoApplication {

    @GetMapping("/helloworld")
    public String hello() {
        return "Hello World!";
    }
}
```

Level up your Java™ code

With [Spring Boot](#) in your app, just a few lines of code is all you need to start building services like a boss.

New to Spring? Try our simple [quickstart guide](#).



Originally [Netflix's Java] libraries and frameworks were built in-house. I'm very proud to say, as of early 2019, we've moved our platform almost entirely over to Spring Boot."




TAYLOR WICKSELL, SENIOR SOFTWARE ENGINEER, NETFLIX
[Watch now](#)

Starter (.. chi era costui ..)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-json</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
  </dependency>
  <dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
  </dependency>
</dependencies>
```



Spring Boot starter: è un template (realizzato come dipendenza MAVEN) che raccoglie le dipendenze delle librerie necessarie per le funzionalità che si dichiara (in pom.xml) di voler usare.

Autoconfiguration (.. it's a kind of magic ..)

La “autoconfiguration” è abilitata attraverso l'uso della java annotation **@EnableAutoConfiguration**

La auto configuration scansiona il java classpath, scova le librerie che sono presenti, imposta la migliore configurazione possibile per queste, istanzia i componenti/beans registrandone le inter - dipendenze (**DI – Dependency Injection**).

@SpringBootApplication


@EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism

@ComponentScan: enable @Component scan on the package where the application is located (see the best practices)

@Configuration: allow to register extra beans in the context or import additional configuration classes

Let's start coding (.. Rock & Roll, baby ..)

<https://start.spring.io/>

 **spring** initializr

Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.3.0 RC1 ☐ 2.3.0 (SNAPSHOT) ☐ 2.2.7 (SNAPSHOT) ☒ 2.2.6
☐ 2.1.14 (SNAPSHOT) ☐ 2.1.13

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

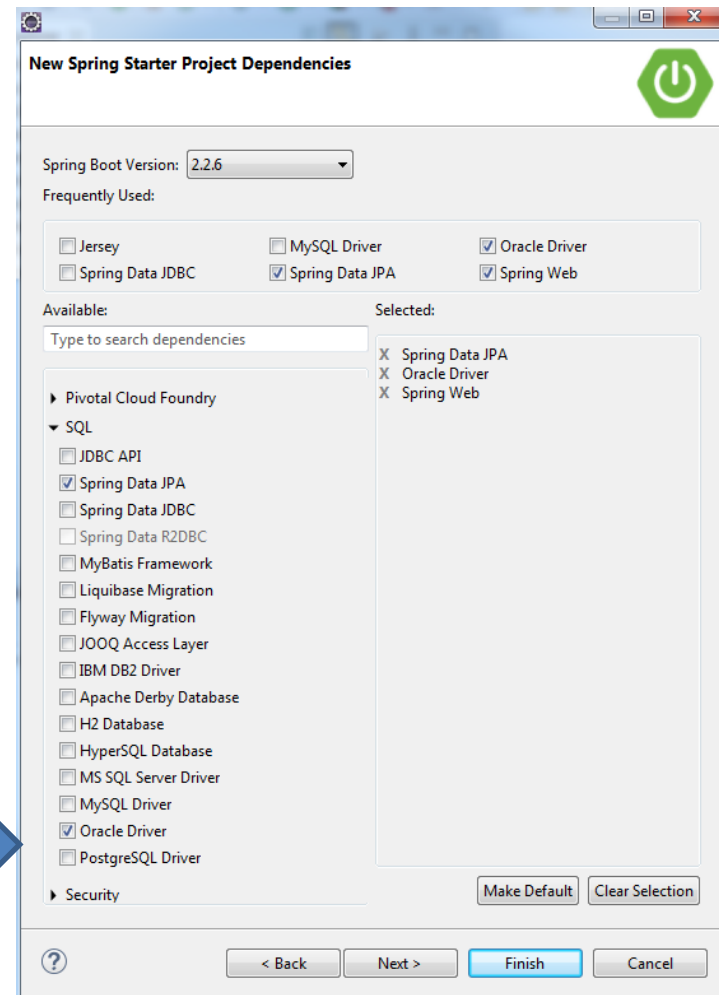
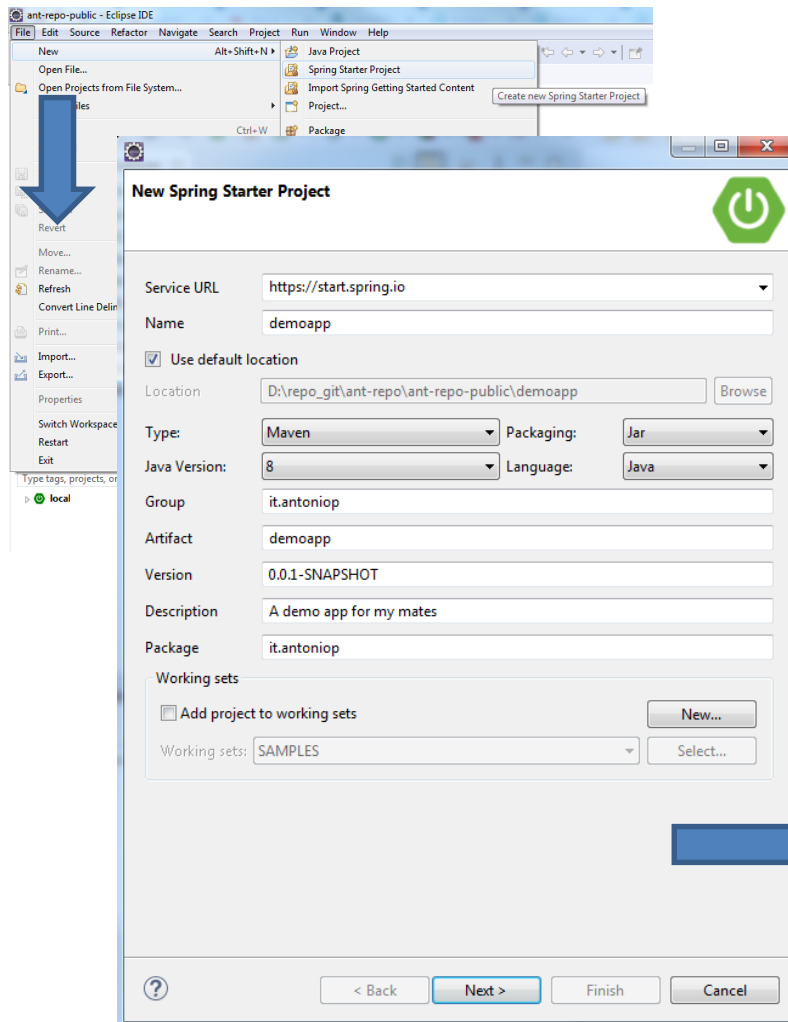
Java ☐ 14 ☐ 11 ☒ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Let's start coding (.. Keep on rockin' in Eclipse world ..)



Spring Boot (.. recap ..)

- Approccio «opinionated» all'uso di Spring et altro (Logging, ORM, REST, JSON, ..) evitando codice ripetitivo
- Auto Configuration
- Maven Starters
 - `spring-boot-starter-web`
 - `spring-boot-starter-data-jpa`
 - `spring-boot-starter-actuator`
- Annotazioni java
 - `@SpringBootApplication`
 - `@Component` | `@Bean` | `@Repository` | `@Service`
 - `@Autowired` | `@Controller`
 - `@RestController`
 - `@RequestMapping`
 - `@RequestBody`
 - `@GetMapping` | `@PostMapping` | `@DeleteMapping`

Continued (.. How many ..)

➤ Maven Starters

- spring-boot-starter-thymeleaf
- spring-boot-starter-mustache
- spring-boot-starter-test
- spring-boot-configuration-processor *{ auto complete su configuration files }*
- spring-boot-starter-aop
- spring-boot-starter-security
 - ❖ thymeleaf-extras-springsecurity5
- spring-boot-starter-actuator
- spring-boot-starter-logging
 - { logging con LogBACK usando Simple Logging Facade for Java (SLF4J) }*
- spring-boot-starter-tomcat *{ Servlet Server Tomcat }*
- spring-boot-starter-log4j2 *{ logging con Log4j }*
- spring-boot-starter-jetty *{ Servlet Server Jetty }*

Continued (.. Whole lotta annotation ..)

➤ Annotazioni java

- @Bean | @Value
- @Repository | @Service
- @Controller | @RestController
- @Configuration | @ConfigurationProperties
- @Profile
- @Aspect
 - @Before: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
 - @AfterReturning: Advice to be executed after a join point completes normally.
 - @AfterThrowing: Advice to be executed if a method exits by throwing an exception.
 - @After: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
 - @Around: Advice that surrounds a join point such as a method invocation.

MVC / DI (.. are there any patterns ?? ..)

The image shows a code editor window with a Java file named `EmployeeRep...`. The code is a Spring MVC controller. Three callouts are present:

- Blue callout:** "I'm an MVC controller" points to the `@Controller` annotation on line 21.
- Red callout:** "Please Spring, gimme the service" points to the `@Autowired` annotation on line 32.
- Yellow callout:** "Let's feed Model... the view is hungry" points to the `model.addAttribute("employees", list);` line on line 42.
- Black callout:** "View! Show me the list" points to the `return "list-employees";` line on line 44.

```
1 package it.antoniop.springdatajpa.web;
2
3 import java.util.List;
4
20
21 @Controller
22 @RequestMapping("/")
23 public class EmployeeMvcController {
24
25     static final Logger LOG = LoggerFactory.getLogger(EmployeeMvcController.class);
26
27
28     @Value("${app.message}")
29     private String welcomeMessage;
30
31
32     @Autowired
33     EmployeeService service;
34
35     @RequestMapping
36     public String getAllEmployees(Model model) {
37         List<EmployeeEntity> list = service.getAllEmployees();
38         if (LOG.isInfoEnabled()) {
39             LOG.info(String.format(" getAllEmployees() is saying [%s]", welcomeMessage));
40         }
41         model.addAttribute("welcomeMessage", welcomeMessage);
42         model.addAttribute("employees", list);
43         // //
44         return "list-employees";
45     }
}
```

REST / DI (.. lotta stuff 1 ..)

```
EmployeeRep... Springdataj... application... EmployeeEnti... EmployeeServ... EmployeeMvcC... application-... add-edit-em...
1 package it.antoniop.springdatajpa.web;
2
3 import java.util.List;
19
20 @RestController
21 @RequestMapping("/employees")
22 public class EmployeeRestController {
23
24     @Autowired
25     EmployeeService service;
26
27     @GetMapping
28     public ResponseEntity<List<EmployeeEntity>> getAllEmployees() {
29         //
30         List<EmployeeEntity> list = service.getAllEmployees();
31         return new ResponseEntity<List<EmployeeEntity>>(list, new HttpHeaders(), HttpStatus.OK);
32     }
33
34     @GetMapping("/{id}")
35     public ResponseEntity<EmployeeEntity> getEmployeeById(@PathVariable("id") Long id) throws RecordNotFoundException {
36         //
37         EmployeeEntity entity = service.getEmployeeById(id);
38         return new ResponseEntity<EmployeeEntity>(entity, new HttpHeaders(), HttpStatus.OK);
39     }
40
41
42     // Use "Key/Value pairs" in form-data ..... as in POST-MAN
43     @PostMapping
44     public ResponseEntity<EmployeeEntity> createOrUpdateEmployee(EmployeeEntity employee)
45         throws RecordNotFoundException {
46         //
47         EmployeeEntity updated = service.createOrUpdateEmployee(employee);
48         return new ResponseEntity<EmployeeEntity>(updated, new HttpHeaders(), HttpStatus.OK);
49     }
50
```

I'm a REST controller

Please Spring, gimme the service



Service Vs Repo (.. lotta stuff 2 ..)

```
EmployeeRep... Springdataj... application... EmployeeEnti... EmployeeServ... EmployeeMvcC...
1 package it.antoniop.springdatajpa
2
3 import java.util.ArrayList
15
16
17
18 @Service
19 @Transactional
20 public class EmployeeService {
21
22     @Autowired
23     EmployeeRepository repository;
24
25     public List<EmployeeEntity> getAllEmployees() {
26         List<EmployeeEntity> employeeList = repository.findAll();
27
28         if (employeeList.size() > 0) {
29             return employeeList;
30         } else {
31             return new ArrayList<EmployeeEntity>();
32         }
33     }
34
35     public EmployeeEntity getEmployeeById(Long id) throws RecordNotFoundException {
36         Optional<EmployeeEntity> employee = repository.findById(id);
37
38         if (employee.isPresent()) {
39             return employee.get();
40         } else {
41             throw new RecordNotFoundException("No employee record exist for given id");
42         }
43     }
44 }
```

I'm a Service component .. I'm Commit/Rollback ready

Please Spring, gimme a JPA ready component



Thanks Spring for the Repo (.. lotta stuff 3 ..)

```
EmployeeRep... Springdatajpa... application... EmployeeEnti... EmployeeServ...
1 package it.antoniop.springdatajpa.repository;
2
3+ import org.springframework
7
8
9
10 @Repository
11 public interface EmployeeRepository extends JpaRepository<EmployeeEntity, Long> {
12     /* */
13 }
14
```

I'm a JPA ready component ...
No code ... It's all free ...
Free as a beer



Spring DOES the
magic

Profiles (.. Test it like PROd ..)

I Profiles sono una funzionalità chiave che ci permette di associare la creazione di componenti a diversi profili/contesti – ad esempio dev, test, prod. ... Possiamo quindi usare profili diversi per situazioni diverse.

Usando la annotazione java `@Profile` associamo un bean ad un particolare profile; l'annotazione si aspetta semplicemente il nome di uno o più profili.

Quando annotiamo un bean/component con un profile “dev” un'istanza del componente `DevDatasourceConfig` sarà creata/attiva solo se allo start-up si chiede di usare il profilo dev

```
@Component
@Profile("dev")
public class DevDatasourceConfig
```

Spring Boot permette di avere/definire profile-specific application.properties files con il formato `applications-{profile}.properties`.

Spring Boot caricherà automaticamente le properties nel file `application.properties` comune a tutti i profiles e quelle definite nei “profile-specific.properties files” riferiti allo start-up come attivi.

I nomi profilo possono essere passati come JVM system parameters; il nome profile usato come segue sarà attivato allo start-up dell'applicativo:

```
java -jar -Dspring.profiles.active=dev ...
```

oppure

```
java -jar app-file-name.jar --spring.profiles.active=dev
```

Security (.. stay safe .. wear masks ..)

Wikipedia – Basic Access AUTH

https://it.wikipedia.org/wiki/Basic_access_authentication

Creare un file in formato

- jks (Java Keystore File) oppure
- p12 [PKCS12 (§)]

<https://www.baeldung.com/spring-boot-https-self-signed-certificate>

<https://mkyong.com/spring-boot/spring-boot-ssl-https-examples/>

https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html#Prepare_the_Certificate_Keystore

(§) PKCS12: Public Key Cryptographic Standards

https://en.wikipedia.org/wiki/PKCS_12

Actuators (.. small step for the human kind but ...)

*"An actuator is a manufacturing term that refers to a mechanical device for moving or controlling something. Actuators can generate a **large amount of motion from a small change.**"*

Spring Boot include delle features aggiuntive per aiutarci a monitorare e a gestire la nostra applicazione quando essa viene usata in produzione.

Possiamo scegliere di gestire/monitorare la nostra applicazione attraverso endpoints HTTP oppure con le API Java JMX.

Esempi:

- `httptrace, health,`
- `metrics, mappings, env,`
- `beans, loggers, logfile,`
- `threaddump, heapdump.`

Starter: `spring-boot-starter-actuator`

Default URL per HTTP endpoints : `/actuator` `/actuator/<actuator-name>`

<https://docs.spring.io/spring-boot/docs/2.2.6.RELEASE/reference/htmlsingle/#production-ready>

Deploying Spring Boot Applications

- **Linux**

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#deployment-service>

- **Windows service**

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#deployment-windows>

<https://github.com/kohsuke/winsw>

<https://github.com/snicoll-scratches/spring-boot-daemon>

- **Deploying in Docker Container**

<https://medium.com/swlh/deploying-spring-boot-applications-15e14db25ff0>

- **Deploying in Microsoft Azure**

<https://spring.io/guides/gs/spring-boot-for-azure/>

<https://docs.microsoft.com/it-it/azure/developer/java/spring-framework/deploy-spring-boot-java-app-with-maven-plugin>

Links

- **Spring «Getting Started» Docs**

<https://spring.io/quickstart>

<https://spring.io/guide>

<https://start.spring.io/>

- **Spring Boot/Spring Docs**

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/index.html>

- **Baeldung**

<https://www.baeldung.com/start-here>

<https://www.baeldung.com/spring-boot>

- **mkyong.com**

<https://mkyong.com/tutorials/spring-boot-tutorials/>

THANK YOU ALL!

Antonio