

# Microservizi con Spring Boot

**Andrea Fornaia, Ph.D.**

Department of Mathematics and Computer Science

University of Catania

Viale A.Doria, 6 - 95125 Catania Italy

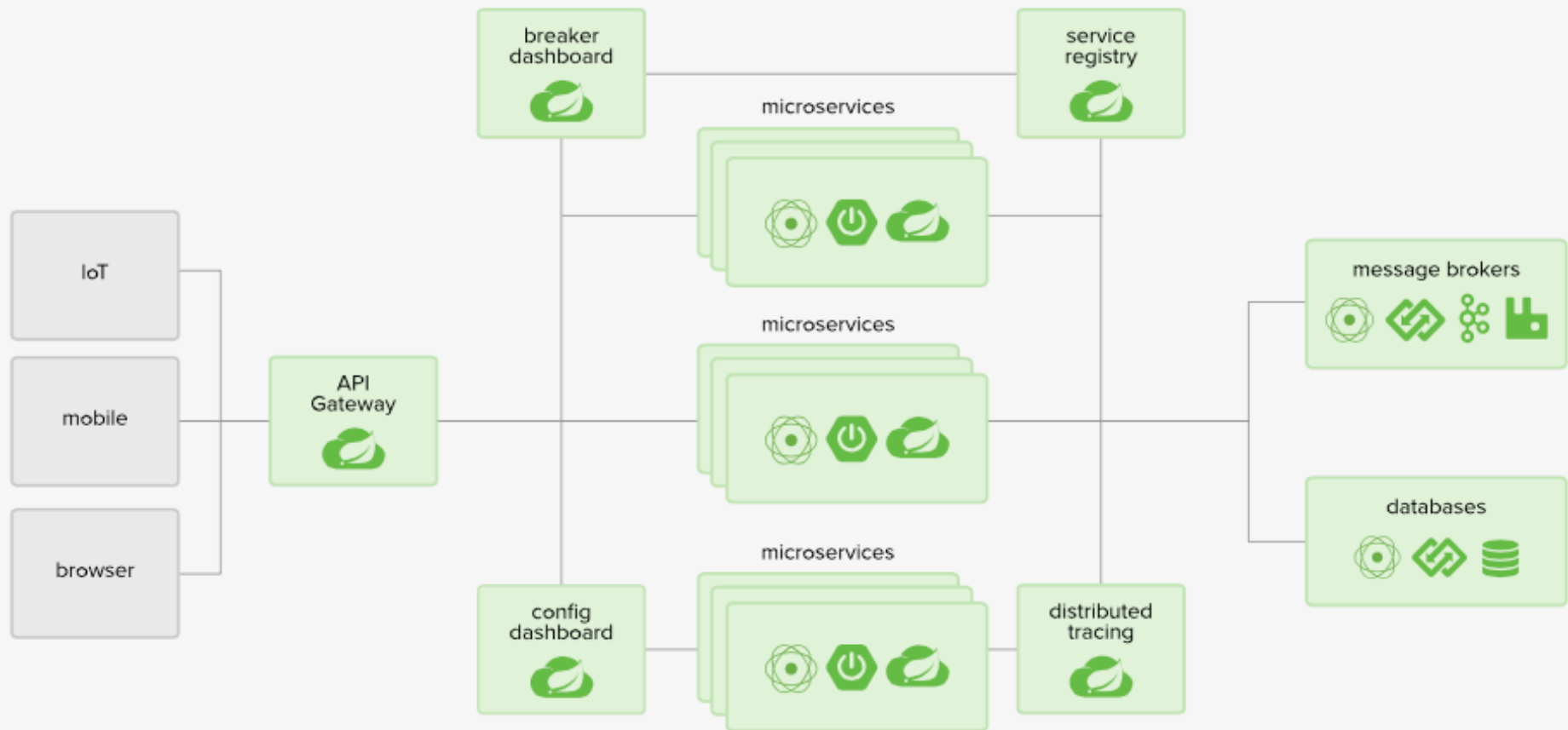
foraia@dmf.unict.it

<http://www.cs.unict.it/~foraia/>

# Spring Framework

- **Framework Java** usato fin dal **2002** per semplificare la creazione di **applicazioni web** (altrimenti basata su Java EE)
  - Molto usato in **ambito enterprise**
  - Permette oggi la creazione di diversi tipi di applicazioni
- **Incorpora molti progetti** (moduli):
  - Spring MVC (lo useremo)
  - Spring Data (lo useremo)
  - Spring Security
  - Spring Integration
  - ...
- **Caratteristiche:**
  - Famoso per il supporto alla **Dependence Injection**
  - **XML-driven** (da sempre)
  - Annotation-driven (negli anni)
- **Problemi:**
  - Difficile da approcciare
  - Difficile da configurare

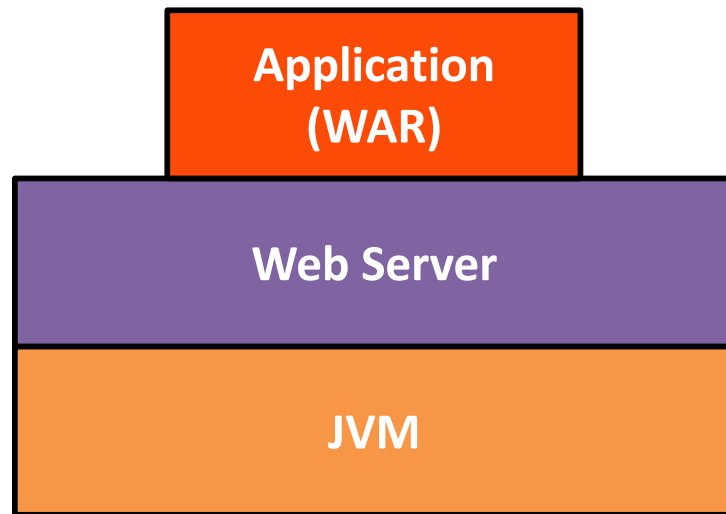
# Spring Cloud



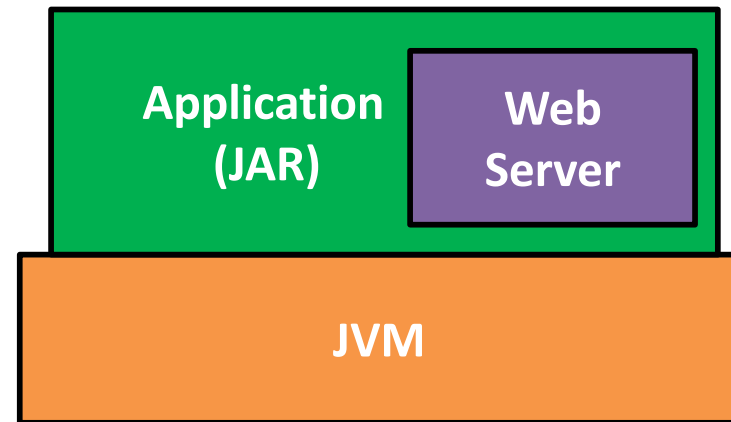
# Spring Boot

- **Tool per semplificare** l'uso di Spring sotto vari aspetti
- Semplifica la creazione di progetti **production-ready**:
  - **Spring Initializr**
  - **Spring starters** (per la gestione semplificata delle **dipendenze**)
- Semplifica la **configurazione**:
  - **Auto-configurazione** in molti casi
  - **Elimina** la necessità di dover definire tutto in **XML** (ma possibile, in casi specifici)
- Ancora **più “Annotation-driven”**
- **Standalone JAR**:
  - Incorpora un web server (**Tomcat**) nel JAR
  - Comunque possibile generare un WAR per il deploy su application/web server

# Spring VS Spring Boot



**Spring**



**Spring Boot**

# Use Case: Space Scanner

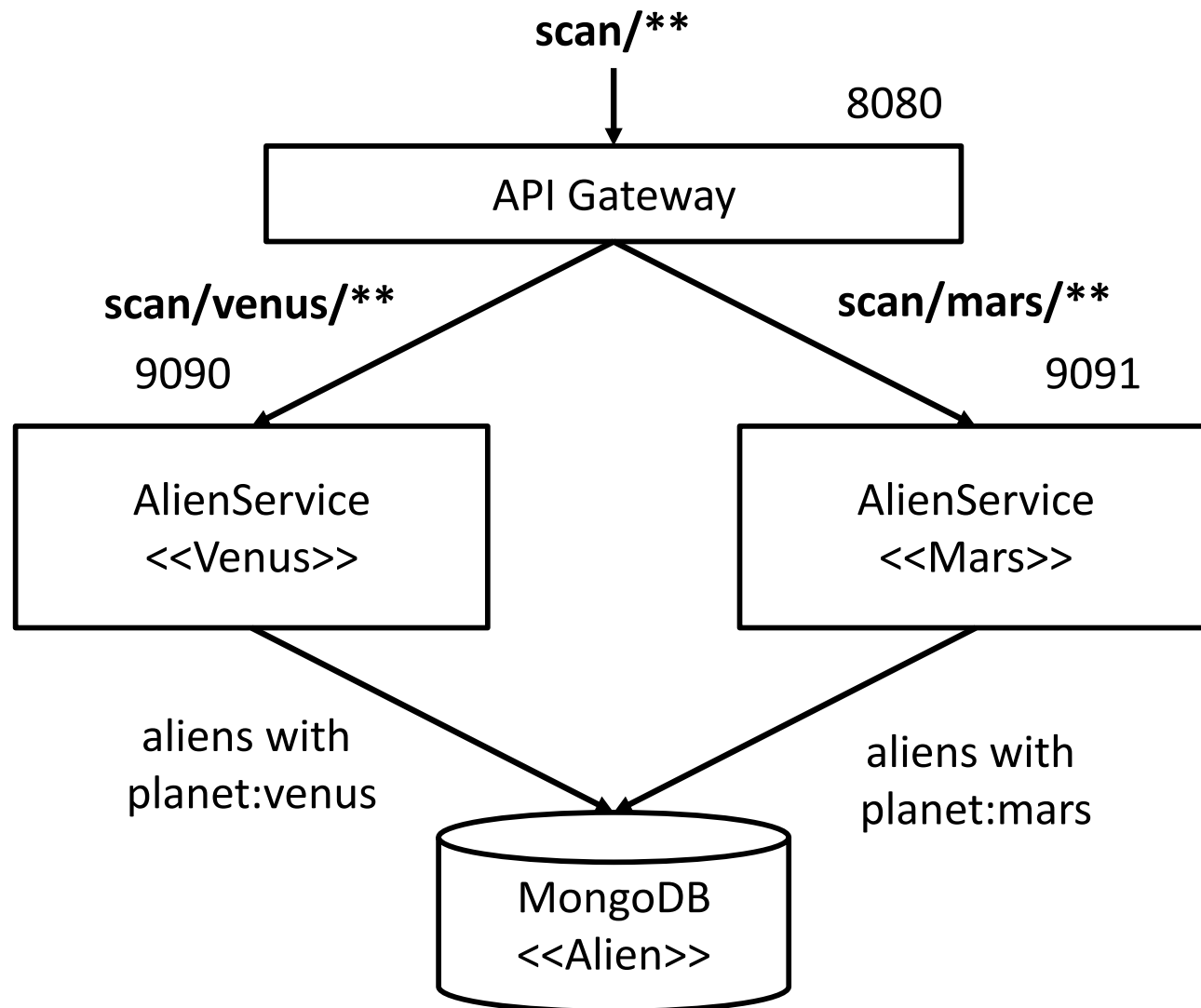
- Vogliamo creare un servizio chiamato Space Scanner che permette di **scoprire nuove forme di vita aliena** sui pianeti della galassia!
- Inizieremo con due pianeti esplorabili: **Marte** e **Venere**
- Ogni **alieno** è descritto da un ***nome***, una ***razza*** e un ***pianeta*** di “residenza”
- Atterrando su un pianeta potremo scoprire solo alieni che vivono **su quel pianeta**
- La razza non deve necessariamente coincidere con la residenza (alieni autostoppisti...)

***Arrivederci e grazie per tutto il pesce! [cit.]***



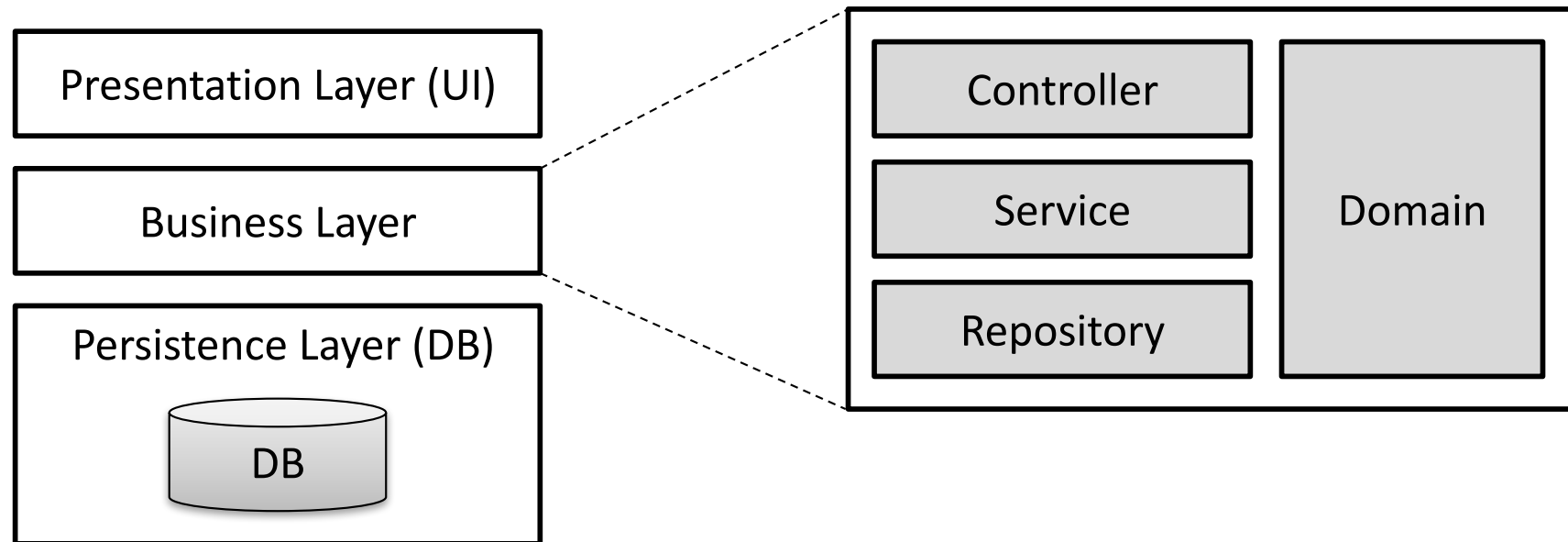
2005

# Microservizi





# Architettura Applicazione Web Spring MVC



**Controller:** interfaccia con i client (UI), espone le API (REST)

**Service:** implementa la logica di business

**Repository:** interfaccia con il DB, espone metodi per l'interrogazione in logica OOP

**Domain:** contiene le entità (oggetti) per la rappresentazione e lo scambio dei dati

# Codice su GitLab

- `git clone https://gitlab.com/fornaia/space-scanner.git`
- `git checkout tags/step1`

- **tags/step1:** After alien.zip unzip
- **tags/step2:** Hello World!
- **tags/step3:** Hello Planet! (Autowire)
- **tags/step4:** Properties
- **tags/step5:** MongoDB con Vagrant & Docker
- **tags/step6:** Some REST APIs
- **tags/step7:** Alien CRUD REST APIs
- **tags/step8:** Zuul API Gateway

# Spring Tool Suite 4 (STS)

The screenshot displays the Spring Tool Suite 4 (STS) IDE interface. The top-left pane shows the Package Explorer with the project structure for 'alien' [boot] [space-scanner master]. The top-right pane shows the Outline view for the 'HelloController' class. The main editor pane displays the source code for 'HelloController.java'.

```
1 package com.space.scanner.alien.controller;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 import com.space.scanner.alien.bean.Planet;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.RequestMapping;
9
10 @RestController
11 public class HelloController {
12     @Autowired
13     Planet planet;
14
15     @RequestMapping("/")
16     public String index() {
17         return "Greetings from " + planet.getGreetings() + "!";
18     }
19 }
20
21
```

The bottom-left pane shows the Boot Dashboard with a search bar and a list of local projects. The bottom-right pane shows the Console view with the output of the application.

alien - AlienApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/jdk1.8.0\_25.jdk/Contents/Home/bin/java (Jan 24, 2019, 1:10:17 AM)

```

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  _/
      |_|_|

:: Spring Boot :: (v2.1.2.RELEASE)

2019-01-24 01:10:18.493 INFO 9752 --- [main] c.space.scanner.alien.AlienApplication : Starting AlienApplication on MB
2019-01-24 01:10:18.497 INFO 9752 --- [main] c.space.scanner.alien.AlienApplication : No active profile set, falling
2019-01-24 01:10:19.405 INFO 9752 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s)
2019-01-24 01:10:19.427 INFO 9752 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-01-24 01:10:19.427 INFO 9752 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache
2019-01-24 01:10:19.433 INFO 9752 --- [main] o.a.catalina.core.AprLifecycleListener : The APR based Apache Tomcat Nat
2019-01-24 01:10:19.508 INFO 9752 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded Web
2019-01-24 01:10:19.508 INFO 9752 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: ini
Landing planet: venus

```

# Spring Boot Initializr

<https://start.spring.io>

**SPRING INITIALIZR** bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.1.2

## Project Metadata

Artifact coordinates

Group

com.space.scanner

Artifact

alien

## Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web

Selected Dependencies

Web x

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)



alien.zip

# Spring Boot Starters

Generate Project % + ↵

## Core

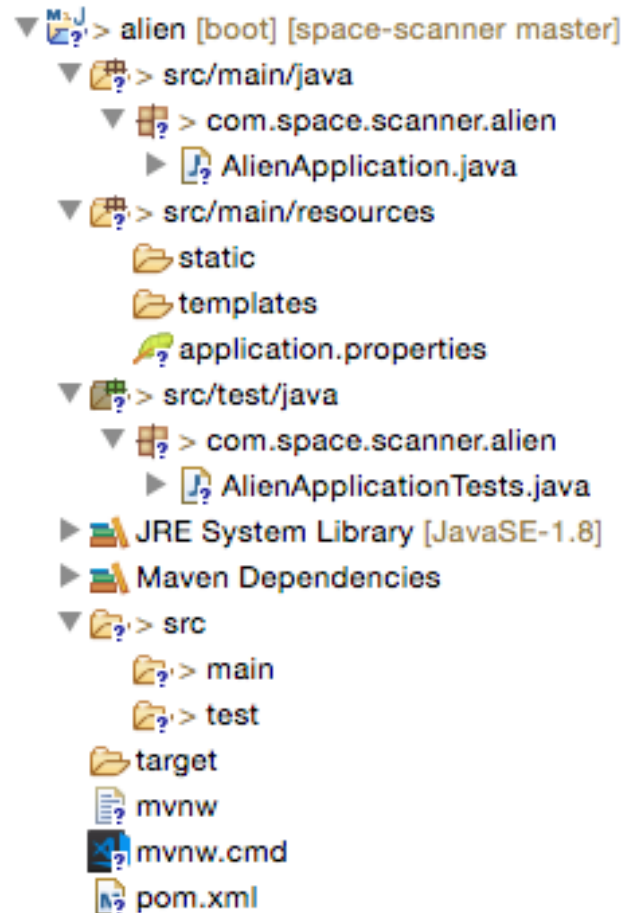
- ☐ DevTools  
Spring Boot Development Tools
- ☐ Security  
Secure your application via spring-security
- ☐ Lombok  
Java annotation library which helps to reduce boilerplate code and code faster
- ☐ Configuration Processor  
Generate metadata for your custom configuration keys
- ☐ Session  
API and implementations for managing a user's session information
- ☐ Cache  
Spring's Cache abstraction
- ☐ Validation  
JSR-303 validation infrastructure (already included with web)
- ☐ Retry  
Provide declarative retry support via spring-retry
- ☐ Aspects  
Create your own Aspects using Spring AOP and AspectJ

## Web

- ☒ Web  
Full-stack web development with Tomcat and Spring MVC
- ☐ Reactive Web  
Reactive web development with Netty and Spring WebFlux
- ☐ Rest Repositories  
Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- ☐ Rest Repositories HAL Browser  
Browsing Spring Data REST repositories in your browser
- ☐ HATEOAS  
HATEOAS-based RESTful services
- ☐ Web Services  
Contract-first SOAP service development with Spring Web Services
- ☐ Jersey (JAX-RS)  
RESTful Web Services framework with support of JAX-RS
- ☐ Websocket  
Websocket development with SockJS and STOMP
- ☐ REST Docs  
Document RESTful services by combining hand-written and auto-generated documentation

**E molti altri (SQL, NoSQL, Integration, Cloud . . .)**

# Generated Project (Maven)



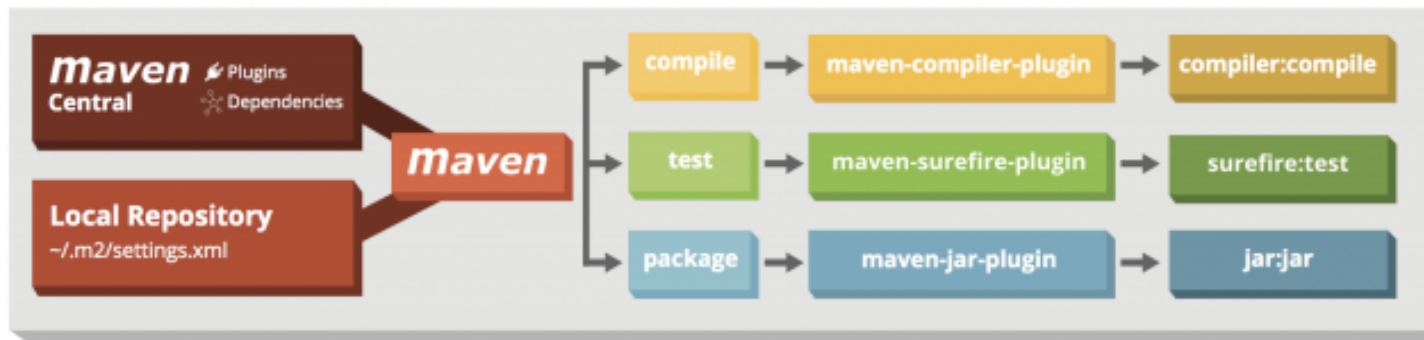
## Crea un progetto Maven:

- **src:** contiene tutti sorgenti e le risorse
  - **main:** per l'applicazione
    - **java:** sorgenti (packages)
    - **resources:** risorse, es. file .properties
  - **test:** per i test
    - **java:** sorgenti
    - **resources:** risorse per il testing
- **target:** creata dopo la compilazione
  - **classes:** risultato compilazione file in main
  - **test-classes:** risultato compilazione file in test
  - **alien-0.0.1-SNAPSHOT.jar** (dopo \$ mvn package)
- **mvnw:** script wrapper per evitare di installare maven
- **mvnw.cmd:** file batch equivalente
- **pom.xml:** descrive dettagli del progetto (es. dipendenze, compilazione, testing)

Importare il progetto su Eclipse o Spring Tool Suite 4:  
import > Maven Project > Browse "alien" extract dir

# Maven

- Software Project Management Tool per Java
  - Automatizzazione:
    - compilazione (con **gestione dipendenze** tramite file di configurazione)
    - testing (es. eseguiti prima del packaging)
    - packaging (es. creazione jar)
    - altro: (installazione in un repository locale, deploy in un repository remoto...)
  - Descrizione basata su file XML (pom.xml)
  - Basato su plugin per estenderne le funzionalità (es. deploy usando docker)
  - Utilizza repository remoto e locale (.m2) per fornire le dipendenze
    - Con i colleghi condivido il file di configurazione (pom.xml) non le librerie (jar)



# mvn

- **clean** — elimina la cartella target (risultato della compilazione precedente)
- **validate** — valida il pom.xml
- **compile** — compila il codice in src, le classi vengono salvate in target/classes
- **test** — esegue i test
- **package** — prende il codice compilato e crea un package di distribuzione (JAR, WAR)
- **verify** — esegue i controlli di verifica sul package, es. controlli di qualità
- **install** — installa (copia) il package nel repository locale (\$HOME/.m2)
- **deploy** — copia il package nel repository remoto, se configurato (artifacts)



# pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.1.2.RELEASE</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.space.scanner</groupId>
12  <artifactId>alien</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>alien</name>
15  <description>Space Scanner Alien Service</description>
16
17  <properties>
18    <java.version>1.8</java.version>
19  </properties>
20
21  <dependencies>
22    <dependency>
23      <groupId>org.springframework.boot</groupId>
24      <artifactId>spring-boot-starter-web</artifactId>
25    </dependency>
26
27    <dependency>
28      <groupId>org.springframework.boot</groupId>
29      <artifactId>spring-boot-starter-test</artifactId>
30      <scope>test</scope>
31    </dependency>
32  </dependencies>
33
34  <build>
35    <plugins>
36      <plugin>
37        <groupId>org.springframework.boot</groupId>
38        <artifactId>spring-boot-maven-plugin</artifactId>
39      </plugin>
40    </plugins>
41  </build>
42
43 </project>
```

← importa le dipendenze di un tipico progetto spring boot

← definisce i dettagli sul progetto alien

← importa le dipendenze per di una web app MVC

← importa le dipendenze per il testing

← plugin maven per il supporto a spring boot

# Dipendenze

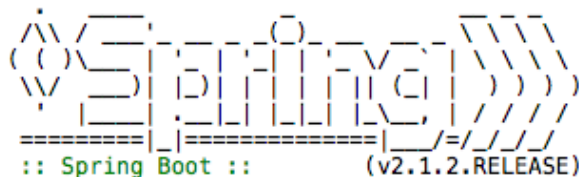
\$ mvn dependency:tree



```
--- maven-dependency-plugin:3.1.1:tree (default-cli) @ alien ---
com.space.scanner:alien:jar:0.0.1-SNAPSHOT
+- org.springframework.boot:spring-boot-starter-web:jar:2.1.2.RELEASE:compile
| +- org.springframework.boot:spring-boot-starter:jar:2.1.2.RELEASE:compile
| | +- org.springframework.boot:spring-boot:jar:2.1.2.RELEASE:compile
| | +- org.springframework.boot:spring-boot-autoconfigure:jar:2.1.2.RELEASE:compile
| | +- org.springframework.boot:spring-boot-starter-logging:jar:2.1.2.RELEASE:compile
| | | +- ch.qos.logback:logback-classic:jar:1.2.3:compile
| | | | \- ch.qos.logback:logback-core:jar:1.2.3:compile
| | | +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.11.1:compile
| | | | \- org.apache.logging.log4j:log4j-api:jar:2.11.1:compile
| | | \- org.slf4j:jul-to-slf4j:jar:1.7.25:compile
| | +- javax.annotation:javax.annotation-api:jar:1.3.2:compile
| | \- org.yaml:snakeyaml:jar:1.23:runtime
| +- org.springframework.boot:spring-boot-starter-json:jar:2.1.2.RELEASE:compile
| | +- com.fasterxml.jackson.core:jackson-databind:jar:2.9.8:compile
| | | +- com.fasterxml.jackson.core:jackson-annotations:jar:2.9.0:compile
| | | \- com.fasterxml.jackson.core:jackson-core:jar:2.9.8:compile
| | +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.9.8:compile
| | +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.9.8:compile
| | \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.9.8:compile
| +- org.springframework.boot:spring-boot-starter-tomcat:jar:2.1.2.RELEASE:compile
| | +- org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.14:compile
| | | +- org.apache.tomcat.embed:tomcat-embed-el:jar:9.0.14:compile
| | | \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.14:compile
| | +- org.hibernate.validator:hibernate-validator:jar:6.0.14.Final:compile
| | | +- javax.validation:validation-api:jar:2.0.1.Final:compile
| | | +- org.jboss.logging:jboss-logging:jar:3.3.2.Final:compile
| | | \- com.fasterxml:classmate:jar:1.4.0:compile
| +- org.springframework:spring-web:jar:5.1.4.RELEASE:compile
| | \- org.springframework:spring-beans:jar:5.1.4.RELEASE:compile
| \- org.springframework:spring-webmvc:jar:5.1.4.RELEASE:compile
| +- org.springframework:spring-aop:jar:5.1.4.RELEASE:compile
| +- org.springframework:spring-context:jar:5.1.4.RELEASE:compile
| \- org.springframework:spring-expression:jar:5.1.4.RELEASE:compile
\-- org.springframework.boot:spring-boot-starter-test:jar:2.1.2.RELEASE:test
| +- org.springframework.boot:spring-boot-test:jar:2.1.2.RELEASE:test
| +- org.springframework.boot:spring-boot-test-autoconfigure:jar:2.1.2.RELEASE:test
| +- com.jayway.jsonpath:json-path:jar:2.4.0:test
| | +- net.minidev:json-smart:jar:2.3:test
| | | \- net.minidev:accessors-smart:jar:1.2:test
| | | \- org.ow2.asm:asm:jar:5.0.4:test
| | \- org.slf4j:slf4j-api:jar:1.7.25:compile
| +- junit:junit:jar:4.12:test
| +- org.assertj:assertj-core:jar:3.11.1:test
| +- org.mockito:mockito-core:jar:2.23.4:test
| | +- net.bytebuddy:byte-buddy:jar:1.9.7:test
| | | +- net.bytebuddy:byte-buddy-agent:jar:1.9.7:test
| | | \- org.objenesis:objenesis:jar:2.6:test
| +- org.hamcrest:hamcrest-core:jar:1.3:test
| +- org.hamcrest:hamcrest-library:jar:1.3:test
| +- org.skyscreamer:jsonassert:jar:1.5.0:test
| | \- com.vaadin.external.google:android-json:jar:0.0.20131108.vaadin1:test
| +- org.springframework:spring-core:jar:5.1.4.RELEASE:compile
| | \- org.springframework:spring-jcl:jar:5.1.4.RELEASE:compile
| +- org.springframework:spring-test:jar:5.1.4.RELEASE:test
| \- org.xmlunit:xmlunit-core:jar:2.6.2:test
```

# Run

- Su STS: Run as Spring Boot Application (oppure Java Application)
- Da riga di comando usando maven (uno dei due) se installato:
  - \$ mvn spring-boot:run
  - \$ mvn package && java -jar target/alien-0.0.1-SNAPSHOT.jar
- Usando il wrapper di maven: sostituire **mvn** con **./mvnw**



```
2019-01-23 17:59:13.224 INFO 7428 --- [main] c.space.scanner.alien.AlienApplication : Starting AlienApplication on MacBook-Pro-di-And
2019-01-23 17:59:13.227 INFO 7428 --- [main] c.space.scanner.alien.AlienApplication : No active profile set, falling back to default
2019-01-23 17:59:14.128 INFO 7428 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2019-01-23 17:59:14.155 INFO 7428 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-01-23 17:59:14.155 INFO 7428 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.14]
2019-01-23 17:59:14.161 INFO 7428 --- [main] o.a.catalina.core.AprLifecycleListener : The APR based Apache Tomcat Native library whic
2019-01-23 17:59:14.232 INFO 7428 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationCont
2019-01-23 17:59:14.232 INFO 7428 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization comp
2019-01-23 17:59:14.446 INFO 7428 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskEx
2019-01-23 17:59:14.645 INFO 7428 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with con
2019-01-23 17:59:14.648 INFO 7428 --- [main] c.space.scanner.alien.AlienApplication : Started AlienApplication in 1.739 seconds (JVM
```

## Whitelabel Error Page

<http://localhost:8080>



This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jan 23 18:00:34 CET 2019

There was an unexpected error (type=Not Found, status=404).

No message available

# AlienApplication

```
package com.space.scanner.alien;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AlienApplication {

    public static void main(String[] args) {
        SpringApplication.run(AlienApplication.class, args);
    }
}
```

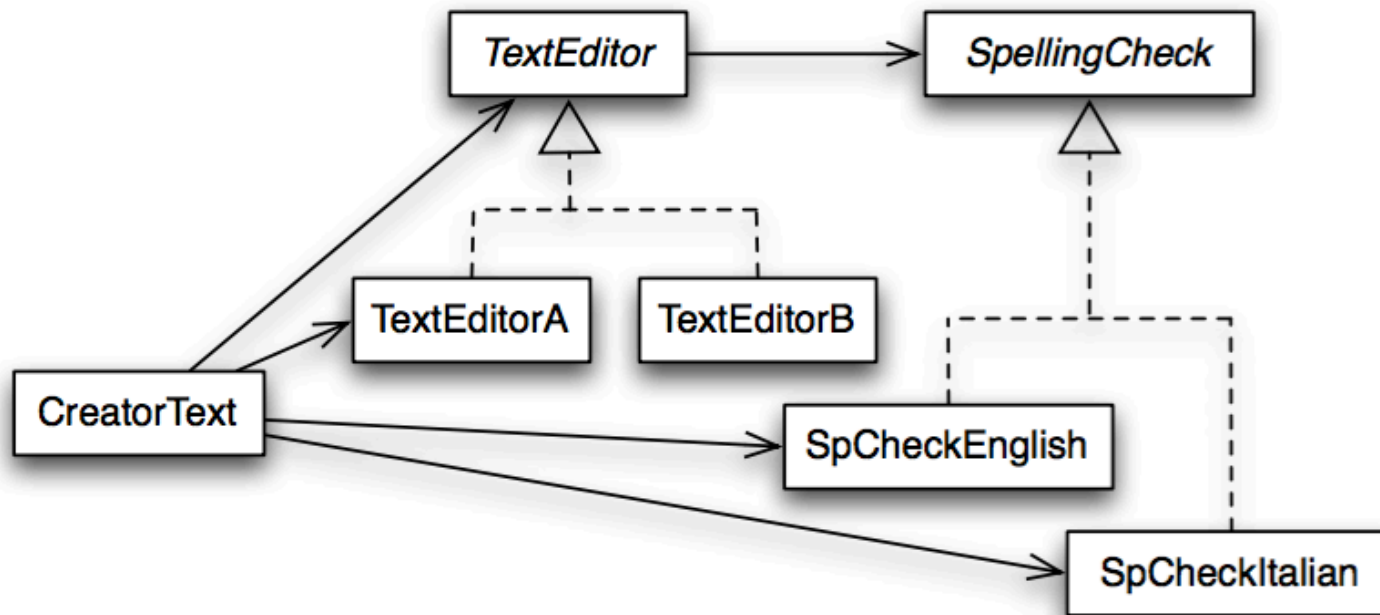
- Contiene il **main**, che si limita a passare il controllo al framework tramite **SpringApplication.run()**. Di solito si inserisce nel **root package**
- **@SpringBootApplication** è un'annotazione di convenienza che aggiunge in realtà:
  - **@Configuration**: etichetta una classe che contiene delle definizioni di **bean** (oggetti da inserire nello SpringContainer e da fornire tramite **dependency injection**)
  - **@EnableAutoConfiguration**: per dire a Spring di inferire alcuni parametri di configurazione in base, ad esempio, alle dipendenze (es. @EnableWebMvc viene inferito dal framework perché spring-webmvc si trova nel classpath)
  - **@ComponentScan**: per dire a Spring di controllare (nel package e sotto-package) la presenza di componenti da inserire nello Spring Container e da fornire tramite DI

# Dependency Injection (ripasso)

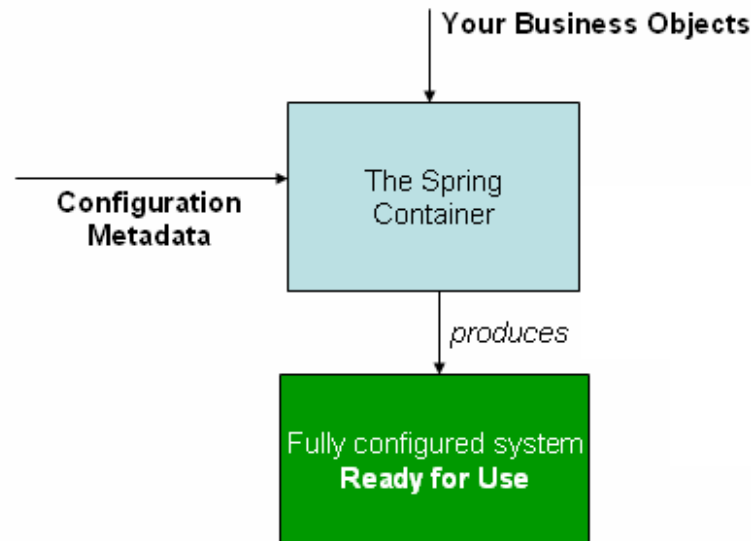
- Tecnica usata per inserire le dipendenze necessarie ad altri oggetti (es. usando Factory Method)
  - Una classe C usa un servizio S (ovvero C dipende da S)
  - Esistono tante implementazioni di S (ovvero S1, S2), e la classe C non deve dipendere dalle implementazioni S1, S2
  - **Al momento di creare l'istanza di C, indico con quale implementazione di S deve operare, passandogli un'istanza di S1 o di S2 (es. nel costruttore)**
- Esempio di dependency
  - Una classe TextEditor usa un servizio SpellingCheck
  - Ci sono tante classi che implementano il servizio SpellingCheck, in base alla lingua usata: SpCheckEnglish, SpCheckItalian, etc.
  - TextEditor deve poter essere collegato ad una delle classi che implementano SpellingCheck



# Dependency Injection (ripasso)

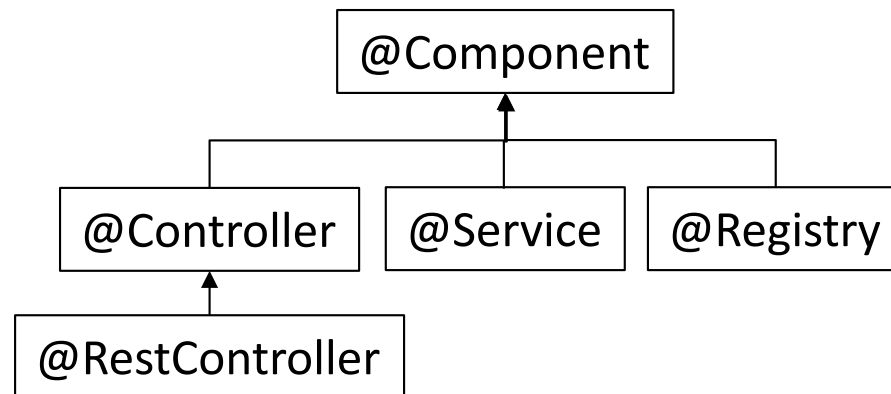
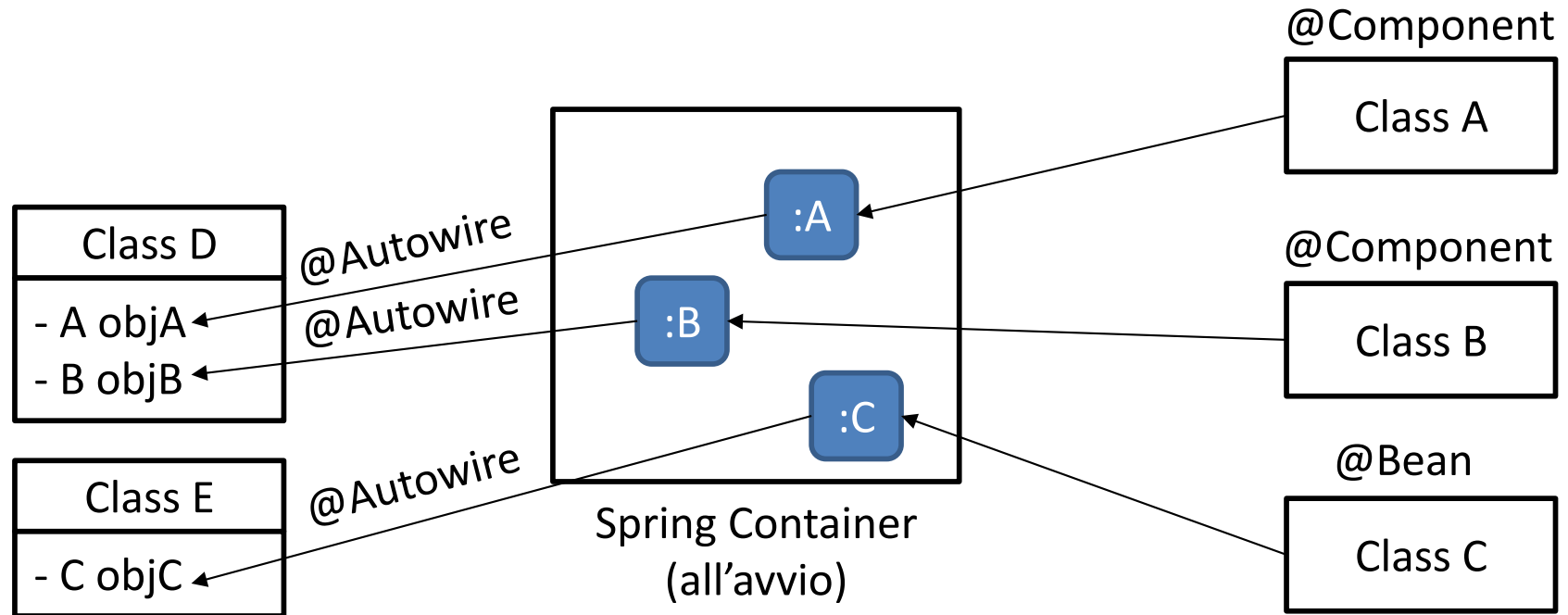


# Spring Container



- Lo **Spring Container** viene usato per gestire la dependency injection (DI)
  - Utilizza i metadati di configurazione per istanziare, configurare e assemblare gli oggetti (bean)
- I metadati di configurazione esprimono le relazioni tra gli oggetti (**object graph**):
  - i bean oggetto di dependency injection
  - le dipendenze tra questi bean
- I metadati (in **Spring Boot**) possono essere rappresentati:
  - Annotazioni (@Autowire, @Component, @Controller, @Service, @Registry)
  - Java (tramite @Bean nelle classi annotate con @Configuration)
  - in XML (usato originariamente da Spring, prolisso ma maggior controllo)

# Configuration





# Hello World

```
package com.space.scanner.alien.controller;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from World!";
    }
}
```

- **@RestController**: informa Spring Framework che si tratta di una *componente* (@Component) di tipo *controller REST*.
- All'avvio, i componenti vengono inseriti dal framework nello **Spring Container**
- **@RequestMapping("/")**: aggiunge una **regola di routing**: tutte le **richieste http** su questo percorso (localhost:8080) verranno indirizzate al metodo **index()**
- Da browser, visitando **http://localhost:8080** si ottiene ***Greetings from World!***

# Hello Planet (@Autowired)

- Vogliamo fare in modo che il messaggio di benvenuto sia personalizzato in base al “pianeta” di atterraggio (servizio Alien<<Mars>> o Alien<<Venus>>)
- Definiamo un’interfaccia **Planet** e due classi **Mars** e **Venus** che la implementano
- Quale delle due istanza usare verrà decisa a **runtime** tramite **Dependency Injection**
- Con **@Autowired** viene chiesto al framework di inserire la dipendenza corretta per Planet con il **bean** opportuno (a runtime)

```
package com.space.scanner.alien.controller;

[...]
import com.space.scanner.alien.bean.Planet;

@RestController
public class HelloController {
    @Autowired
    Planet planet;

    @RequestMapping("/")
    public String index() {
        return "Greetings from " + planet.getGreetings() + "!";
    }
}
```

# @Bean

- Il metodo **@Bean** di una classe **@Configuration** viene chiamato dal framework all'avvio
- L'istanza restituita viene inserita nello **Spring Container**

```
package com.space.scanner.alien;

[...]  
import com.space.scanner.alien.bean.Mars;  
import com.space.scanner.alien.bean.Planet;  
import com.space.scanner.alien.bean.Venus;  
  
@SpringBootApplication  
public class AlienApplication {  
  
    private String planet = "venus";  
  
    public static void main(String[] args) {  
        SpringApplication.run(AlienApplication.class, args);  
    }  
    /*@Bean vanno definiti in una classe @Configuration*/  
    @Bean  
    public Planet planet() {  
        System.out.println("Landing planet: " + planet);  
  
        switch (planet) {  
            case "mars": return new Mars();  
            case "venus": return new Venus();  
            default: return new Mars();  
        }  
    }  
}
```

```
package com.space.scanner.alien.bean;  
  
public interface Planet {  
    public String getGreetings();  
    public String getName();  
}
```

```
package com.space.scanner.alien.bean;  
  
public class Mars implements Planet {  
    @Override  
    public String getGreetings() {  
        return "Red Planet";  
    }  
    @Override  
    public String getName() {  
        return "mars";  
    }  
}
```

```
package com.space.scanner.alien.bean;  
  
public class Venus implements Planet {  
    @Override  
    public String getGreetings() {  
        return "Yellow Planet!";  
    }  
    @Override  
    public String getName() {  
        return "venus";  
    }  
}
```

# @Value e le Properties

- Attualmente la scelta di Planet è hardwired nel codice
  - `http://localhost:8080` restituisce sempre ***Greetings from Yellow Planet!!***
- Possiamo usare le **properties** per configurare il servizio
- **application.properties** contiene le proprietà dell'applicazione:
  - Semplice formato **chiave=valore** (es. `server.port=9090`)
  - Sono supportati anche altri formati, es. YML
- Il valore di una proprietà può essere inserito in una variabile con **@Value("\${prop.name}")**
- Le proprietà possono essere ridefinite (in ordine di precedenza)
  - Come variabile di ambiente (**export SERVER\_PORT=9090**)
  - Da riga di comando (`java -jar -Dserver.port=9090 service.jar`)

```
## application.properties
# App configuration
app.planet = venus

# Tomcat configuration
server.port = 9090
```

**Vengono usate anche da terze parti,  
es. Per settare la porta di ascolto**

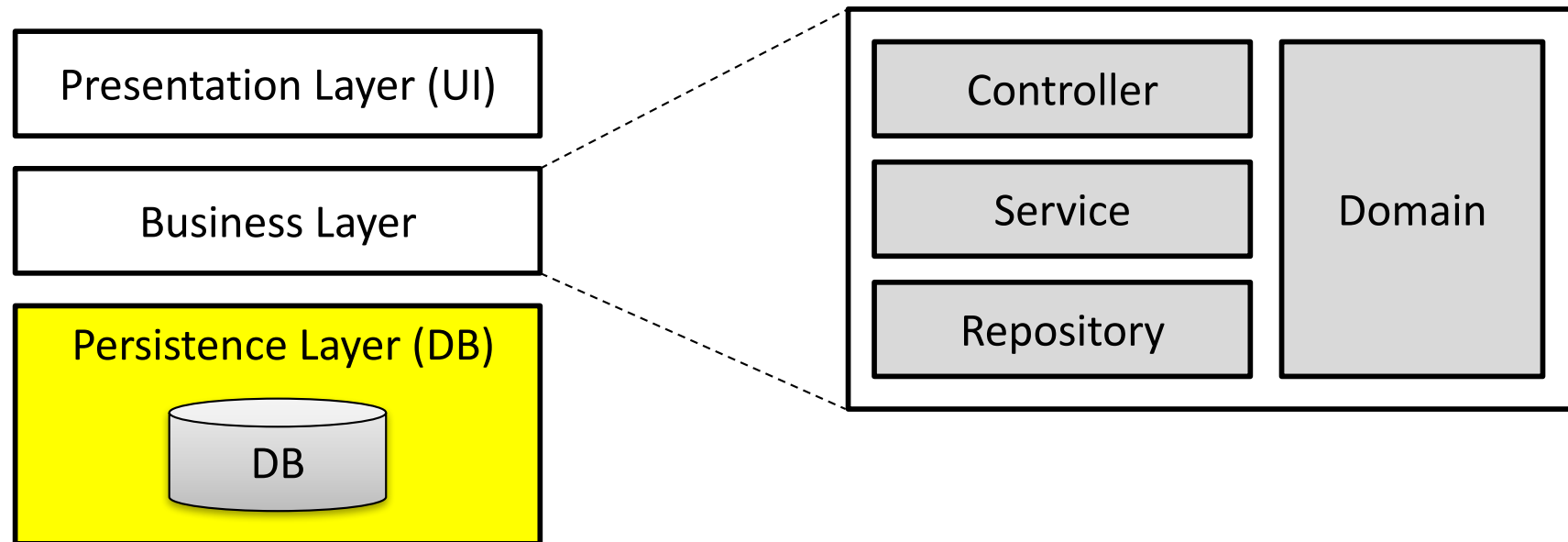
```
@SpringBootApplication
public class AlienApplication {

    @Value("${app.planet}")
    private String planet;

    public static void main(String[] args) {
        SpringApplication.run(AlienApplication.class, args);
    }

    @Bean
    public Planet planet() {
        System.out.println("Landing planet: " + planet);
        switch (planet) {
            case "mars": return new Mars();
            case "venus": return new Venus();
            default: return new Mars();
        }
    }
}
```

# Architettura Applicazione Web Spring



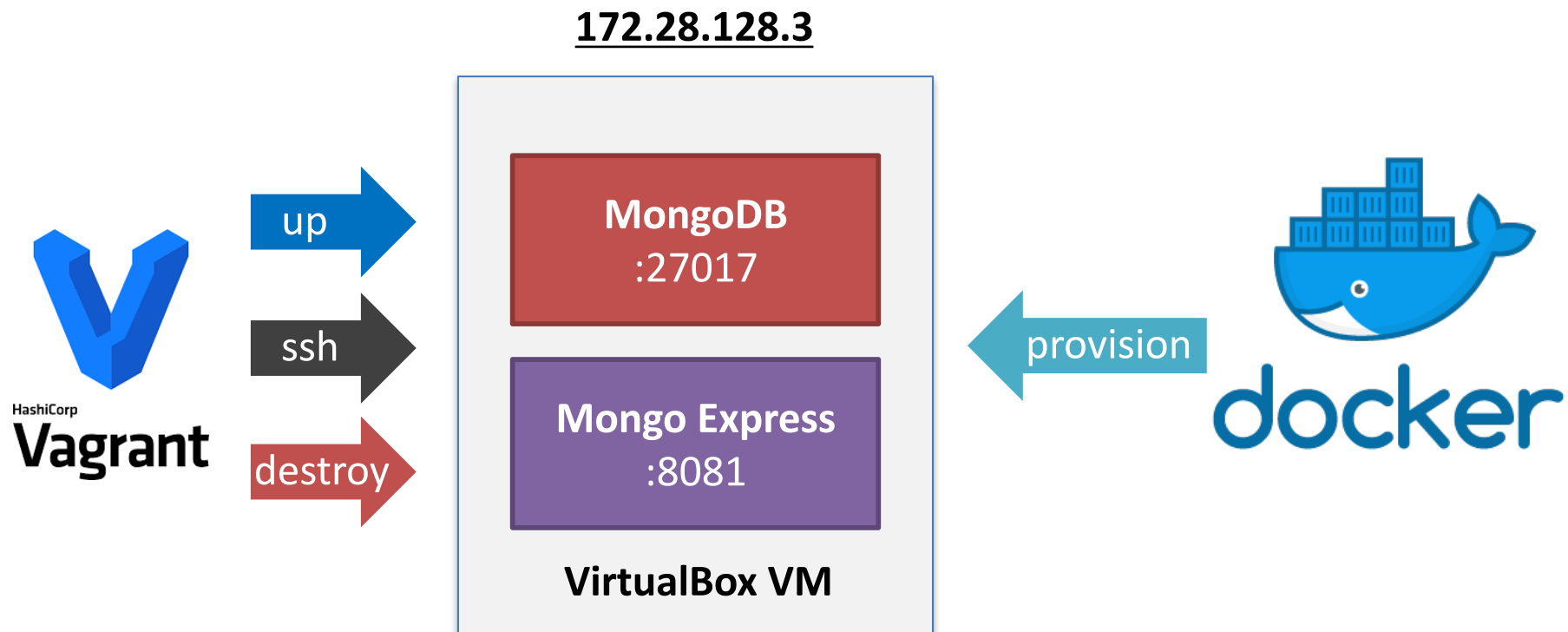
**Controller:** interfaccia con l'esterno, espone le API (REST) e valida l'input

**Service:** implementa la logica di business

**Repository:** interfaccia con il DB, espone metodi per l'interrogazione

**Domain:** contiene le entità (oggetti) per rappresentare e scambiare i dati tra i 3 layer

# Persistence Layer con MongoDB



# Vagrant

```
$ mkdir project (project root)
$ cd project
$ vagrant init (opzionale, crea la cartella .vagrant/ e un Vagrantfile di esempio)

$ edit Vagrantfile (descrive come creare e configurare la VM)
$ edit bootstrap.sh (provisioning shell script)
$ echo "Some file content" > input_file.txt (la project root è condivisa con la VM)

$ vagrant up (la VM viene creata seguendo il Vagrantfile)
$ vagrant ssh (ssh sulla VM basato su chiave)
    $ ls /vagrant → bootstrap.sh Vagrantfile input_file.txt (root -> /vagrant)
    $ echo "output content" > output_file.txt
    $ logout
$ vagrant destroy (elimina la VM)
$ ls → ... output_file.txt (i file condivisi sopravvivono alla VM)
```

# Vagrantfile





```
# Esempio Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, guest: 80, host: 8080, host_ip: "127.0.0.1"
end
```

## Provisioning: bootstrap.sh

```
# Di default il provisioning viene fatto solo al vagrant up (configurabile)
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
if [ ! -L /var/www ]; then
  rm -rf /var/www
  ln -fs /vagrant /var/www
fi
```

**\$ vagrant up**  
**localhost:8080**


### Index of /

	<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
	<a href="#">Vagrantfile</a>	15-Jan-2019 11:55	211	
	<a href="#">bootstrap.sh</a>	15-Jan-2019 11:51	136	
	<a href="#">input_file.txt</a>	15-Jan-2019 14:30	0	
	<a href="#">outout_file.txt</a>	15-Jan-2019 12:31	15	

*Apache/2.2.22 (Ubuntu) Server at localhost Port 8080*

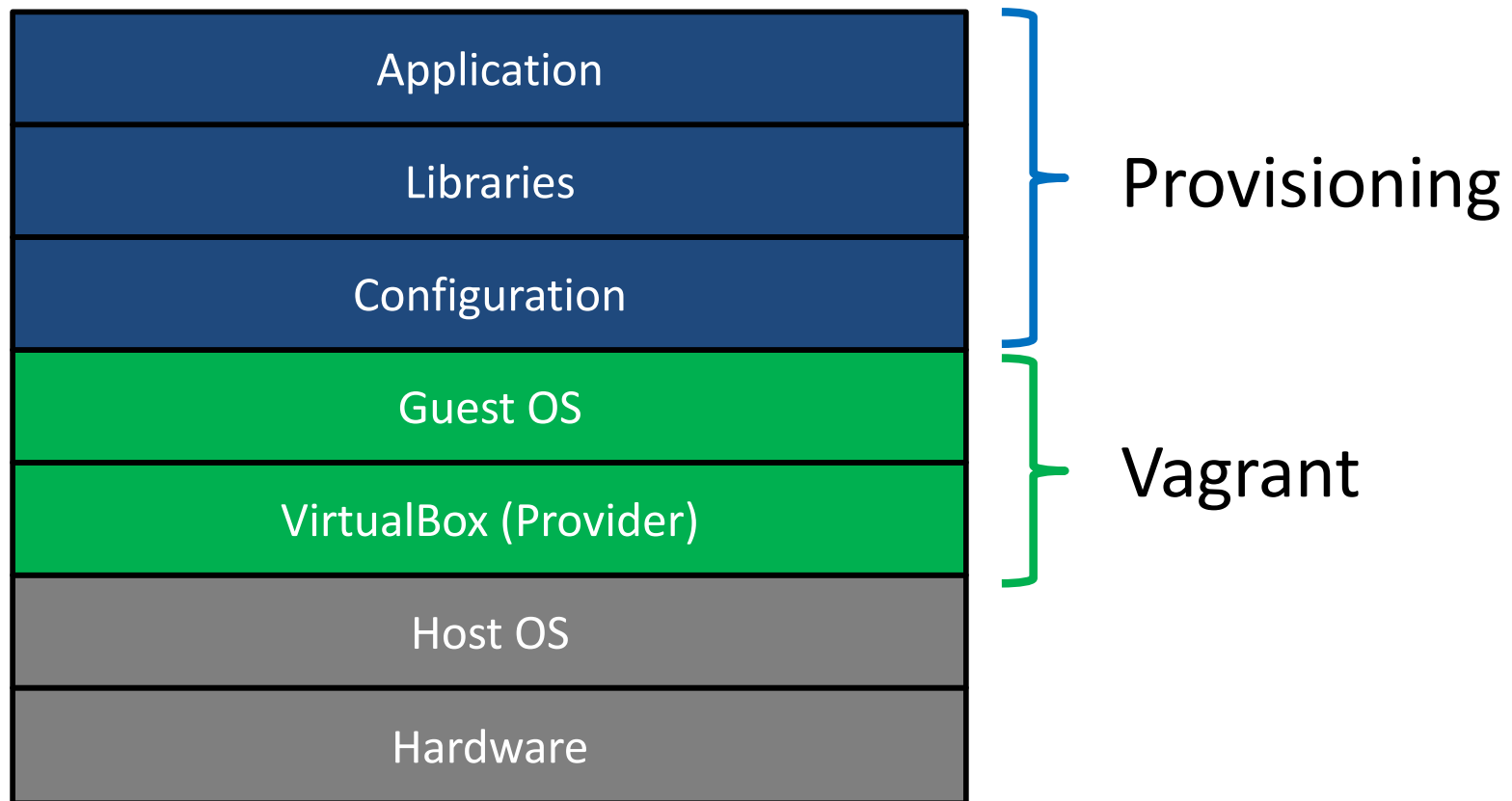


# VM Provider: VirtualBox

General	Preview
Name: Docker_default_1547564211927_80925 Operating System: Ubuntu (64-bit)	
<b>System</b> Base Memory: 1024 MB Processors: 2 Boot Order: Floppy, Optical, Hard Disk Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization	
<b>Display</b> Video Memory: 16 MB Remote Desktop Server: Disabled Video Capture: Disabled	
<b>Storage</b> Controller: IDE Controller: SCSI SCSI Port 0: ubuntu-xenial-16.04-cloudimg.vmdk (Normal, 10,00 GB) SCSI Port 1: ubuntu-xenial-16.04-cloudimg-configdrive.vmdk (Normal, 10,00 MB)	
<b>Audio</b> Host Driver: CoreAudio Controller: ICH AC97	
<b>Network</b> Adapter 1: Intel PRO/1000 MT Desktop (NAT)	
<b>USB</b> Disabled	
<b>Shared folders</b> Shared Folders: 1	

# Provisioning

Indica come personalizzare la VM dopo l'avvio (installazione pacchetti, configurazione, deploy)  
Supportati: Shell Script, Ansible, Chef, Puppet, Docker ...)



[<https://www.vagrantup.com/docs/provisioning/>]

# Docker Provisioner

```
# Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.provision "docker" do |d|
    d.run "hello-world"
  end
end
```

Modo veloce per installare docker: sfruttare il **docker provisioner**

In realtà viene usato al posto di shell per eseguire comandi docker all'avvio (provision)

- scaricare immagini
- creare container
- ...

```
$ vagrant up
$ vagrant ssh
$ docker --version
Docker version 18.09.1, build 4c52b90
$ logout
$ vagrant suspend
$ vagrant resume
```

# Creazione Ambiente

- Vogliamo un server con Mongo DB e un'interfaccia web per la gestione del DB
- Possiamo installarli entrambi in una macchina virtuale di sviluppo usando Vagrant e Docker Compose
- Vagrant userà docker-compose per il provisioning della VM

```
# Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.provision "docker" do |d|
    d.run "hello-world"
  end
  config.vm.provision "shell", path: "vagrant-install-docker-compose.sh"
  config.vm.provision "shell", inline: "docker-compose -f /vagrant/docker-compose.yml up"
  config.vm.network "private_network", ip: "172.28.128.3"
end
```

# Docker Compose

## **vagrant-install-docker-compose.sh**

```
# reference: https://docs.docker.com/compose/install/  
sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
sudo chmod +x /usr/local/bin/docker-compose
```

## **docker-compose.yml**

```
version: '3.1'  
services:  
  mongo:  
    image: mongo  
    restart: always  
    ports:  
      - 2701:2701  
    environment:  
      MONGO_INITDB_ROOT_USERNAME: root  
      MONGO_INITDB_ROOT_PASSWORD: example  
  mongo-express:  
    image: mongo-express  
    restart: always  
    ports:  
      - 8081:8081  
    environment:  
      ME_CONFIG_MONGODB_ADMINUSERNAME: root  
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
```

# Run del server

```
$ cd space-scanner
$ ls
README.md          alien              vagrant-install-docker-compose.sh
Vagrantfile        docker-compose.yml

$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/xenial64'..
==> default: Setting the name of the VM: space-scanner_default_1548291686683_56948
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
==> default: Mounting shared folders...
==> default: Running provisioner: docker...
==> default: Starting Docker containers...
==> default: -- Container: hello-world
==> default: Running provisioner: shell...
default: Creating network "vagrant_default" with the default driver
default: Pulling mongo (mongo:)...
default: latest: Pulling from library/mongo
default: Pulling mongo-express (mongo-express:)...
default: latest: Pulling from library/mongo-express
default: Creating vagrant_mongo-express_1 ...
default: Creating vagrant_mongo_1 ...
Creating vagrant_mongo-express_1 ... done
Creating vagrant_mongo_1 ... done
```

# Mongo Express







**http://172.28.128.3:8081**

 Mongo Express Database ▾

## Mongo Express

### Databases

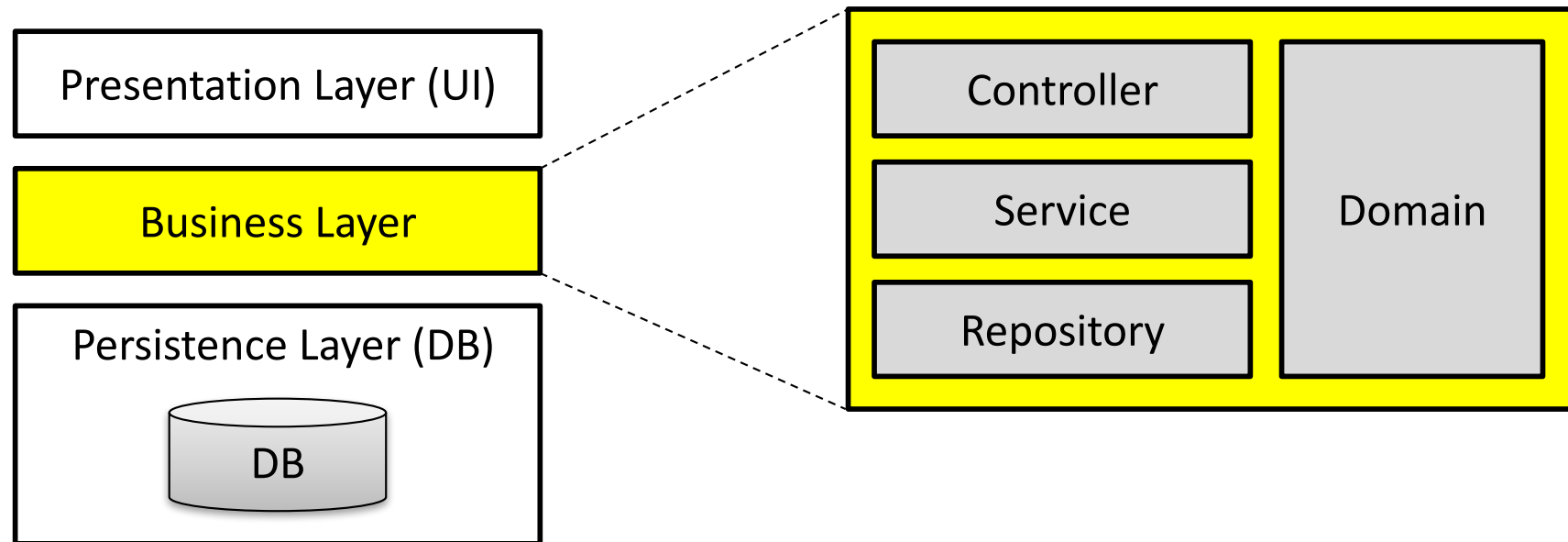
[+ Create Database](#)

 View	admin	 Del
 View	config	 Del
 View	local	 Del

### Server Status

Hostname	05738e98d976	MongoDB Version	4.0.5
Uptime	1185 seconds	Server Time	Thu, 24 Jan 2019 01:24:07 GMT

# Architettura Applicazione Web Spring



**Controller:** interfaccia con l'esterno, espone le API (REST) e valida l'input

**Service:** implementa la logica di business

**Repository:** interfaccia con il DB, espone metodi per l'interrogazione

**Domain:** contiene le entità (oggetti) per rappresentare e scambiare i dati tra i 3 layer



# Package Structure

```
com.space.scanner.alien
+- AlienApplication.java
+- bean
|   +- Planet.java
|   +- Mars.java
|   +- Vanus.java
+- domain
|   +- Alien.java
+- controller
|   +- HelloController.java
|   +- AlienController.java
+- service
|   +- AlientService.java
+- repository
    +- AlienRepository.java
```

**VS**

```
com.space.scanner.alien
+- AlienApplication.java
+- bean
|   +- Planet.java
|   +- Mars.java
|   +- Vanus.java
+- hello
|   +- HelloController.java
+- alien
    +- Alien
    +- AlienController.java
    +- AlientService.java
    +- AlienRepository.java
```

# Domain: Alien

```
package com.space.scanner.alien.domain;

import [...]
@Document
public class Alien {
    @Id
    private String id;
    @NotNull
    @Size(min=3, message="Name should have at least 2 characters")
    private String name;
    @NotNull
    @Size(max=10, message="Race should have no more than 10 characters")
    private String race;
    // Set by Service with setter
    @Null
    private String planet;

    public Alien() { }

    /* Getters and Setters */
}
```

- **@Document** mappa la classe ad un'entità (documento, in Mongo) nel DB
- **@NotNull**, **@Size**, **@Null** verranno usati per la validazione automatica (tramite **@Valid**)
- Le **entità** del dominio vengono create (e populate dal framework tramite setter) per lo **scambio dati** in entrambe le direzioni tra il DB, i layer e i client REST

# Controller: AlienController

```
package com.space.scanner.alien.controller;
import [...]

@RestController
@RequestMapping("/aliens")
public class AlienController {
    @Autowired
    AlienService alienService;

    @GetMapping("/add")
    public @ResponseBody String addAlien (@RequestParam String name, @RequestParam String race) {
        alienService.addAlien(name, race);
        return "Saved";
    }

    @GetMapping("")
    public @ResponseBody List<Alien> getAliens (@RequestParam Optional<String> race) {
        if (race.isPresent())
            return alienService.getAliens(race.get());
        else
            return alienService.getAliens();
    }
}
```

**@ResponseBody:** per codificare l'oggetto (entità o lista di entità) in oggetto JSON

**@RequestParam:** per associare i parametri della richiesta (?race=marziano) ad un parametro del metodo (Optional<String> race)

# Service: AlienService

```
package com.space.scanner.alien.service;  
import [...]
```

```
@Service
```

```
public class AlienService {
```

```
    @Autowired
```

```
    private AlienRepository alienRepository;
```

```
    @Autowired
```

```
    Planet planet;
```

```
    public void addAlien(String name, String race) {
```

```
        Alien a = new Alien();
```

```
        a.setName(name);
```

```
        a.setRace(race);
```

```
        a.setPlanet(planet.getName());
```

```
        alienRepository.save(a);
```

```
    }
```

```
    public List<Alien> getAliens() {
```

```
        return alienRepository.findByPlanet(planet.getName());
```

```
    }
```

```
    public List<Alien> getAliens(String race) {
```

```
        return alienRepository.findByPlanetAndRace(planet.getName(), race);
```

```
    }
```

```
}
```

## NOTA

Il filtro su planet (“usa solo dati del pianeta gestito”) è un concetto di business gestito dal service layer  
**(separation of concern)**

# Repository: AlienRepository

Dipendenza (starter) da aggiungere al pom.xml

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

```
package com.space.scanner.alien.repository;

import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.space.scanner.alien.domain.Alien;

public interface AlienRepository extends MongoRepository<Alien, String> {
    public List<Alien> findByPlanet(String planet);
    public List<Alien> findByPlanetAndRace(String planet, String race);
}
```

- L'interfaccia padre contiene già molti metodi per l'interazione con il db, come **save(alien)** e query comuni, come **findById(id)**
- Altre **query** possono essere definite nell'interfaccia usando un formato intuitivo
- I metodi dell'interfaccia verranno **implementati dal framework**

# Connessione con MongoDB

# App configuration

`app.planet = venus`

**application.properties**

# Tomcat configuration

`server.port = 9090`

# Mongo connection

`spring.data.mongodb.authentication-database=admin`

`spring.data.mongodb.username=root`

`spring.data.mongodb.password=example`

`spring.data.mongodb.port=27017`

`spring.data.mongodb.host=172.28.128.3`

`spring.data.mongodb.database=space_scanner`

# Test dell'interfaccia REST

- **service:localhost:9090**
  - **GET /**
    - Greetings from Yellow Planet!!
  - **GET /aliens/add?name=Urza&race=martian**
    - Saved
  - **GET /aliens/add?name=Gin&race=vulcan**
    - Saved
  - **GET /aliens**
    - [{"id":"5c492986a21e442781894b96","name":"Urza","race":"martian","**planet**":"**venus**"}, {"id":"5c492a09a21e442781894b97","name":"Gin","race":"vulcan","**planet**":"**venus**"}]
  - **GET /aliens/?race=vulcan**
    - [{"id":"5c492a09a21e442781894b97","name":"Gin","race":"vulcan","**planet**":"**venus**"}]

# RESTful APIs

- **REST** sta per **RE**presentational **State Transfer**
- L'astrazione principale in REST è la **risorsa** (es. alien, population)
- Evitare quindi **azioni** (es. **add**)
  - usare nomi, non verbi per definire le API
- Ogni risorsa ha un **URI** (**U**niform **R**esource **I**dentifier)
  - aliens
  - aliens/5c492986a21e442781894b96 (id)
- Ogni risorsa ha una sua rappresentazione (XML, HTML, JSON, TEXT) che ne identifica lo **stato attuale**
- REST utilizza **verbi di HTTP** per codificare le **operazioni** sulle risorse
  - **POST:**      **Crea** una risorsa                      (**C**REATE)
  - **GET:**        **Leggi** una risorsa                      (**R**EAD)
  - **PUT:**        **Aggiorna** una risorsa                      (**U**PDATE)
  - **DELETE:** **Elimina** una risorsa                      (**D**ELETE)



# Alien CRUD: Controller

## AlienController

```
// CREATE
@PostMapping("")
Alien createAlien(@Valid @RequestBody Alien newAlien) {
    return alienService.createAlien(newAlien);
}

// READ
@GetMapping("/{id}")
public @ResponseBody Alien getAlien (@PathVariable String id) {
    return alienService.getAlien(id);
}

// UPDATE
@PutMapping("/{id}")
public @ResponseBody Alien updateAlien (@PathVariable String id,
    @Valid @RequestBody Alien newAlien) {
    return alienService.updateAlien(id, newAlien);
}

// DELETE
@DeleteMapping("/{id}")
void deleteAlien(@PathVariable String id) {
    alienService.deleteAlien(id);
}
```

- **@PostMapping, @GetMapping, @PutMapping, @DeleteMapping** sostituiscono e specializzano **@RequestMapping**
- **@Valid @RequestBody**: l'oggetto JSON nel payload della richiesta viene convertito in un'entità del dominio (Alien) applicando le regole di Validazione nell'entità

# Alien CRUD: Service

## AlienService

```
public Alien createAlien(@Valid Alien newAlien) {
    newAlien.setPlanet(planet.getName());
    return alienRepository.save(newAlien);
}

public Alien getAlien(String id) {
    return alienRepository.findByPlanetAndIdQuery(planet.getName(), id);
}

public Alien updateAlien(String id, @Valid Alien newAlien) {
    return alienRepository.findByPlanetAndId(planet.getName(), id)
        .map(alien -> {
            alien.setName(newAlien.getName());
            alien.setRace(newAlien.getRace());
            return alienRepository.save(alien);
        })
        .orElseGet(() -> {
            System.out.println("put new with specified ID");
            newAlien.setId(id);
            newAlien.setPlanet(planet.getName());
            return alienRepository.save(newAlien);
        });
}

public void deleteAlien(String id) {
    alienRepository.deleteById(id);
}
```

# Alien CRUD: Repository

```
AlienRepository
@Query("{planet: ?0, id: ?1}")
public Alien findByPlanetAndIdQuery(String planet, String id);
public Optional<Alien> findByPlanetAndId(String planet, String id);

// Other queries
public Long countByPlanet(String planet);
public Long countByPlanetAndRace(String planet, String race);
```

- Oltre ad usare il formato del nome dei metodi, è possibile usare **@Query("quey")** e creare query nel linguaggio specifico del Database (es. SQL per MySQL)
  - in questo caso opportuni oggetti JSON per interrogare MongoDB
- Oltre a **findBy\***, anche **countBy\*** può essere usato per creare query di conteggio

# Test dell'interfaccia REST

- **Usare curl da riga di comando**
  - **curl -X POST localhost:9090/aliens**
    - H 'Content-type:application/json'
    - d '{"name":"Ubaldo","race":"terran"}'
  - **Sostituire PUT con il comando HTTP appropriato (GET, PUT, DELETE)**
  - **Content-type (-H) va specificato solo se viene inviato un oggetto (-d)**
- **service:localhost:9090**
  - **POST /aliens + '{"name":"Ubaldo","race":"terran"}'**
    - {"id":"5c493a49a21e4427cd414e85","name":"Ubaldo","race":"terran","planet":"venus"}
  - **GET /aliens/5c493a49a21e4427cd414e85**
    - {"id":"5c493a49a21e4427cd414e85","name":"Ubaldo","race":"terran","planet":"venus"}
  - **PUT /aliens/5c493a49a21e4427cd414e85 + '{"name":"Ubaldo","race":"cylon"}'**
    - {"id":"5c493a49a21e4427cd414e85","name":"Ubaldo","race":"cylon","planet":"venus"}
  - **DELETE /aliens/5c493a49a21e4427cd414e85**
    - <vuoto>
  - **GET /aliens/?race=vulcan**
    - [{"id":"5c492a09a21e442781894b97","name":"Gin","race":"vulcan","planet":"venus"}]

# Test dei servizi: Venus e Mars

```
$ java -jar  
-Dserver.port=9090  
-Dapp.planet=venus  
target/alien-0.0.1-SNAPSHOT.jar &  
$ curl -X GET localhost:9090
```

9090

AlienService  
<<Venus>>

aliens with  
planet:venus

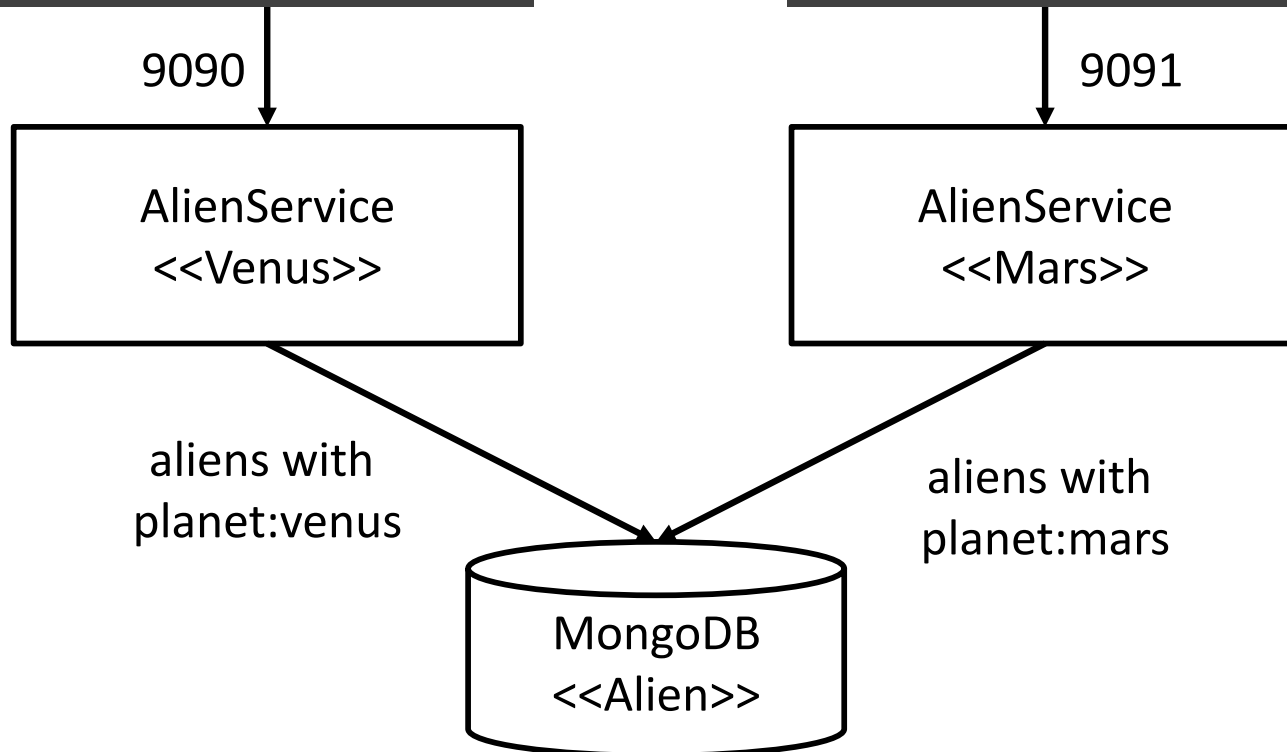
```
$ java -jar  
-Dserver.port=9091  
-Dapp.planet=mars  
target/alien-0.0.1-SNAPSHOT.jar &  
$ curl -X GET localhost:9091
```

9091

AlienService  
<<Mars>>

aliens with  
planet:mars

MongoDB  
<<Alien>>



# API Gateway con Zuul

- Dobbiamo creare un **nuovo progetto (servizio gateway)**:
  - usiamo sempre Initializr o STS
  - com.space.scanner.gateway
  - starter: Zuul
    - groupId: org.springframework.cloud
    - artifactId: **spring-cloud-starter-netflix-zuul**

```
package com.space.scanner.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@EnableZuulProxy
@SpringBootApplication
public class GatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }
}
```

# application.yml

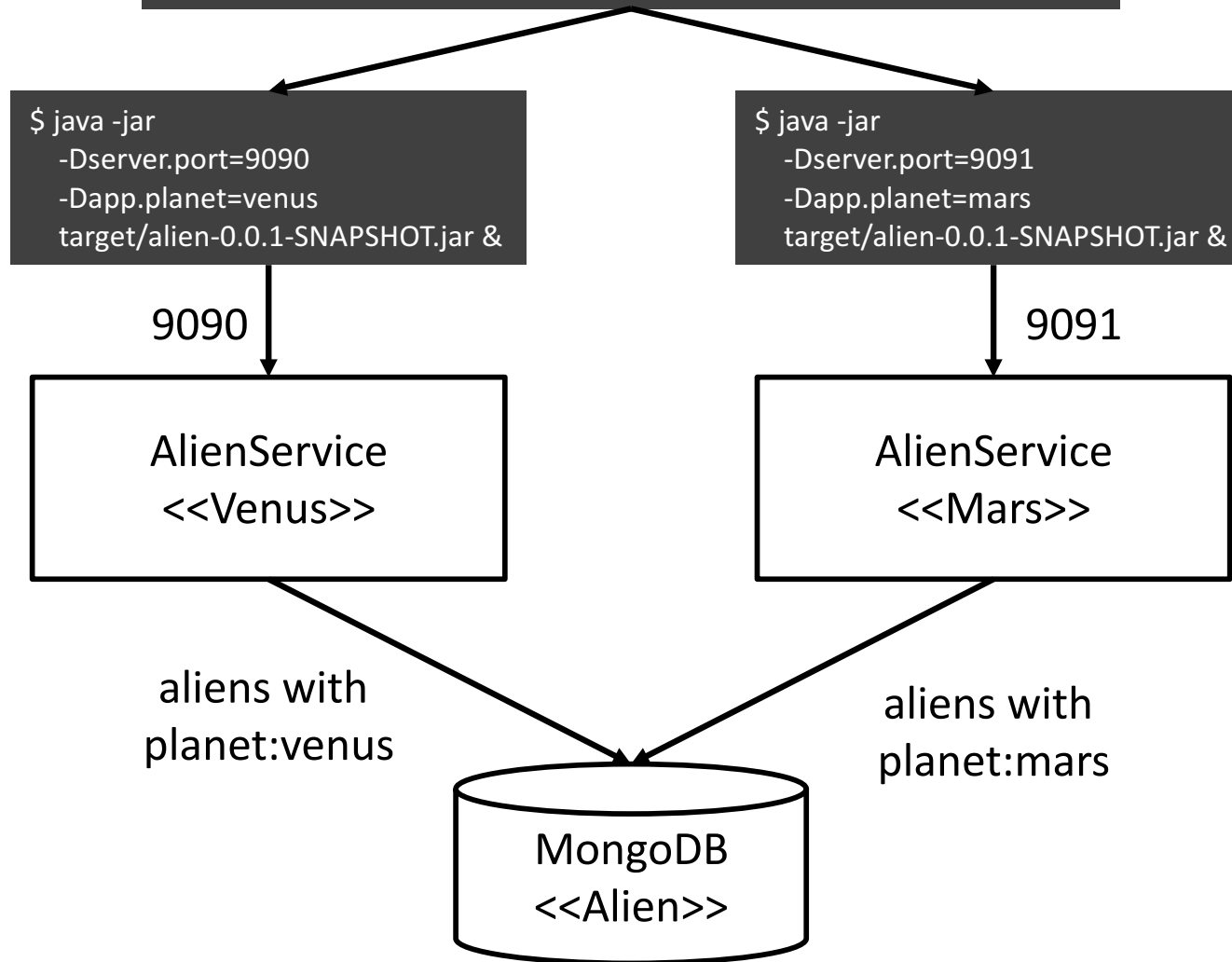
```
server:
  port: 8080

zuul:
  prefix: /scanner
  routes:
    venus:
      path: /venus/**
      url: http://localhost:9090
    mars:
      path: /mars/**
      url: http://localhost:9091
```

- Rinominare il file **application.properties** in **application.yml**
  - Il framework si occuperà di convertire le entry YML in java properties
  - Es. server.port=8080, **zuul.routes.mars.path=/venus/\*\***
- Definisce le **regole di routing** per l'API gateway
- Le **richieste sulla 8080** verranno **reindirizzate** al servizio appropriato

# Test del gateway

```
$ java -jar target/gateway-0.0.1-SNAPSHOT.jar &  
$ curl -X GET localhost:8080/scanner/venus  
$ curl -X GET localhost:8080/scanner/mars
```





# Riferimenti

- M. Macero: “Learn Microservices with Spring Boot”. Apress, 2017
- Building an Application with Spring Boot  
<https://spring.io/guides/gs/spring-boot/>
- Spring Boot DOC  
<https://docs.spring.io/spring-boot/docs/2.1.2.RELEASE/reference/pdf/spring-boot-reference.pdf>
- Spring Framework Annotations  
<https://springframework.guru/spring-framework-annotations/>
- Spring Component Scan  
<https://springframework.guru/spring-component-scan/>
- Structuring Your Code  
<https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-structuring-your-code.html>
- The IoC container  
<https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html>
- Dependency injection  
<https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-collaborators>
- Understanding Spring Web Application Architecture: The Classic Way:  
<https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/>