Full lifecycle of a microservice: from fault-tolerant and reliable architecture to delivery

600  
How to realize, and containerize in a Docker image, a "database per service" REST microservice, using Spring Cloud to simplify the complexity of Eureka service registry and Ribbon client side load balancing.  
Then its lifecycle will be orchestrated till delivery with Paas Cloud environments:  
Github (SCM), Jenkins@Openshift (build pipeline), DockerHub (container registry and automated build) and Pivotal Cloud Foundry (create and bind backing services, service registry and discovery).

Hill

Full lifecycle of a microservice: how to realize a fault-tolerant and reliable architecture and deliver it as a Docker container or in a Cloud environment

400

How to realize, and containerize in a Docker image, a "database per service" REST microservice, using Spring Cloud to simplify the complexity of Eureka service registry and Ribbon client side load balancing. Then its lifecycle will be orchestrated till delivery with Paas Cloud environments: Github, Jenkins@Openshift, DockerHub and Pivotal Cloud Foundry.

1.1 Microservices: implementing the “database per service” pattern

1.2 Implementing a microservice with Spring Boot

1.2.1 Backing Service integration and configuration

1.2.2 Implementing the spring profile pattern

1.2.3 How to create a backing services in Pivotal Cloud Foundry

1.2.4 How to bind a backing services (PAAS CONNECTOR / JAVA CONFIGURATION)

1.2.5 Define an automated build (Jenkins@Openshift) and promote the docker image in container registry (DockerHub)

1.3 Interactions between Microservices: service discovery and service registration

1.3.1 Eureka service registry facilities

1.3.2 How to implement a microservice and register inside Eureka with Spring Cloud

1.3.3 Implementing a client application that consume an Eureka server application with Ribbon client side load balancing features

1.4 Solution delivery

1.4.1 In a Cloud Environment: Pivotal Cloud Foundry

1.4.2 As a docker container

It will be shown, step by step, the full lifecycle development process of a microservice. From architectural (database per service) and technological (Spring Boot) aspects to delivery related scenarios (development, Cloud or dockerized environments), in an ecosystem context where microservices are each other reliable and fault tolerant (Eureka service registry, Ribbon load balancing, Spring Cloud).

[1 Full lifecycle of a microservice: from fault-tolerant and reliable architecture to delivery 4](#_Toc449607447)

[1.1 Microservices: implementing the “database per service” pattern 4](#_Toc449607448)

[1.2 Implementing a microservice with Spring Boot 5](#_Toc449607449)

[1.2.1 Backing Service integration and configuration 5](#_Toc449607450)

[1.2.2 Implementing the spring profile pattern (SPRING PROFILE) 5](#_Toc449607451)

[1.2.3 Implementing spring profile Pattern for a Spring Boot application (SPRING PROFILE) 5](#_Toc449607452)

[1.2.4 How to create a backing services in Pivotal Cloud Foundry 6](#_Toc449607453)

[1.2.5 How to bind a backing services via PAAS CONNECTOR 6](#_Toc449607454)

[1.2.6 How to bind a backing services via JAVA CONFIGURATION 6](#_Toc449607455)

[1.3 Interactions between Microservices: service discovery and registration 6](#_Toc449607456)

[1.3.1 Eureka service registry (EUREKA SERVICE REGISTRY) 7](#_Toc449607457)

[1.3.2 How to implement and register a server application inside Eureka (SERVER APPLICATION REGISTRATION IN EUREKA) 7](#_Toc449607458)

[1.3.3 Implementing a client application that consume an Eureka server application (CLIENT APPLICATION CONSUMING SERVER APPLICATION ON EUREKA) 7](#_Toc449607459)

[1.3.4 AGGIUNGERE UN NUOVO SERVER 7](#_Toc449607460)

[1.4 SOLUTION DELIVERY 7](#_Toc449607461)

[1.4.1 PIVOTAL CLOUD FOUNDRY 7](#_Toc449607462)

[1.4.2 DOCKERIZING 8](#_Toc449607463)

[1.4.3 BUILD PIPELINE WITH JENKINS 8](#_Toc449607464)

# Full lifecycle of a microservice: from fault-tolerant and reliable architecture to delivery

## Microservices: implementing the “database per service” pattern

Affrontiamo l’architettura del database in un microservizio.

Dobbiamo soddisfare:

1. Il servizio deve essere fortemente disaccoppiato e deve essere sviluppato deployato e scalato in modo indipendente
2. Alcune transazioni di business devono aggiornare dati riferiti (di proprietà) di diversi microservizi
3. Alcune query devono eseguire delle join tra dati di riferimento (di proprietà) di diversi micro servizi (esempio elenco dei clienti e dei loro ordini)
4. Micro servizi diversi hanno diversi requisiti inn tema di data storage (database relazionali/nosql MongoDB per persistere dati complessi e non strutturati Neo4J per persistere e accedere a dai grafici)

La soluzione è mantenere la persistenza dei dati del micro servizio privata al micro servizio stesso e accedibile attraverso le sue API.

Il database di servizio è effettivamente parte dell’implementazione del servizio e non può essere acceduto direttamente da altri servizi

Di seguito alcuni modi per mantenere la persistenza dei dati di un servizio privata:

1. Non è necessario fornire un database server per ogni servizio

Nel caso di database relazionali:

1. **Tabelle private per servizio** : ogni servizio controlla un set di tabelle che possono essere acceduta dal servizio stesso
2. **Schema per service**: ogni servizio ha uno schema database privato
3. Database server per service: ogni servizio dispone del proprio database server

1 e 2) forniscono il minore overhead ma servizi da alto livello di throuput hanno bisogno del proprio database server

E’ una buona politica quella di creare barriere che controllino la modularità ad esempio assegnando differenti database userid a ciascun servizio usando i meccanismi di controllo accessi dei database come i grant.

Senza questo controllo all’incapsulamento gli sviluppatori cercheranno sempre di bypassare le api di servizio ed accedere direttamente alle tabelle.

Implementazione

Dati geospaziali.

<http://www.scribd.com/doc/2569355/Geo-Distance-Search-with-MySQL>

## Implementing a microservice with Spring Boot

### Backing Service integration and configuration

INTRO

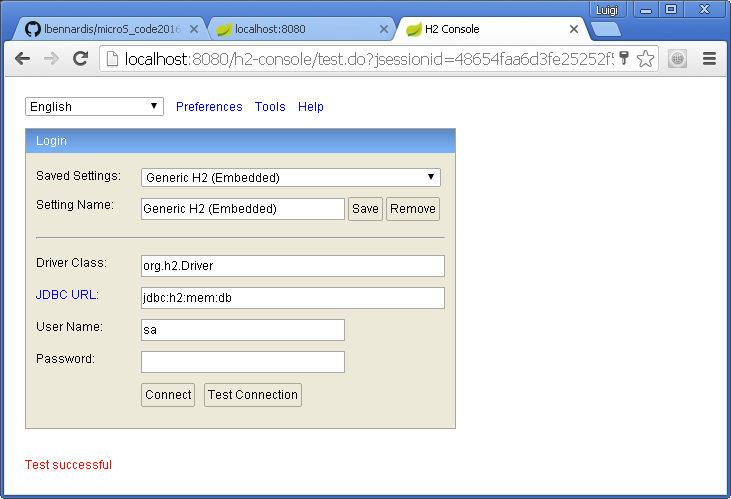
### Implementing the spring profile pattern (SPRING PROFILE)

application.properties

1. localH2
2. mySqlLocal
3. mySqlDocker
4. dockerContainer (specializzare la connessione jdbc secondo lo standard docker run)
5. cloudFoundry

**DEMO**

1. **Esecuzione localH2 (in memory - local Console)**



### Implementing spring profile Pattern for a Spring Boot application (SPRING PROFILE)

1. **Esecuzione mySqlLocal -/Env**
2. **Esecuzione mySqlDocker -/Env**

**PANORAMICA SU DOCKER**

**BOOT2DOCKER**

**DOCKER IMAGES GIA’ SCARICATE**

1. **BUILD maven docker local published -/Env**
2. **Esecuzione dockerContainer**
3. **Promozione sul branch /INTEGRAZIONE del sorgente**

**JENKINS ON OPENSHIFT PANORAMICA**

**ACCESSO A JENKINS CON FILEZILLA SULLA MACCHINA JENKINS**

**BUILD**

**PUBBLICAZIONE E BUILD DEL CONTANER SU DOCKER HUB PER CONDIVIDERE**

### How to create a backing services in Pivotal Cloud Foundry

manifest.yml

cf create-service ClearDb <plan> <name>

env:

SPRING\_PROFILE\_ACTIVE: <profile\_name>

**DEMO**

1. **Esecuzione cloudFoundry /Env**
2. **Accesso al database sul cloud**

### How to bind a backing services via PAAS CONNECTOR

Definizione datasource

@Configuration

@Profile(“cloudFoundry”)

Mettere log print sulla classe che definisce la configuration

**DEMO**

1. **Esecuzione cloudFoundry /Env**

### How to bind a backing services via JAVA CONFIGURATION

Definizione datasource

@Bean

@Profile(“cloudFoundry”)

Definizione di un custom connetion pool

## Interactions between Microservices: service discovery and registration

TUTTE LE APPLICAZIONI DEVONO ESSERE PREDEPLOYATE

I MICROSERVIZI E IL LORO ECOSISTEMA

### Eureka service registry (EUREKA SERVICE REGISTRY)

Cosa è Eureka

Implementazione applicazione

Console

**DEMO**

1. **Esecuzione EUREKA**
2. **Console EUREKA**

### How to implement and register a server application inside Eureka (SERVER APPLICATION REGISTRATION IN EUREKA)

Implementazione applicazione

Maven di deployment

Dettagli degli elementi di registrazione sul file yml

Console di registrazione

**DEMO**

1. **Deployment su Cloud Foundry**
2. **Console EUREKA**

### Implementing a client application that consume an Eureka server application (CLIENT APPLICATION CONSUMING SERVER APPLICATION ON EUREKA)

Implementazione applicazione

Dettagli di bind verso il server sul codice

Dettaglio sul file yml

Console Eureka

### AGGIUNGERE UN NUOVO SERVER

Console Eureka

Fermo di uno dei due server (undeploy)

## SOLUTION DELIVERY

### PIVOTAL CLOUD FOUNDRY

#### -PIVOTAL CLOUD FOUNDRY

### DOCKERIZING

#### –docker

### BUILD PIPELINE WITH JENKINS

#### –build pipeline

OREILLY

# Full lifecycle of a microservice: how to realize a fault-tolerant and reliable architecture and deliver it as a Docker container or in a Cloud environment

How to realize, and containerize in a Docker image, a "database per service" REST microservice, using Spring Cloud to simplify the complexity of Eureka service registry and Ribbon client side load balancing. Then its lifecycle will be orchestrated till delivery with Paas Cloud environments: Github, Jenkins@Openshift, DockerHub and Pivotal Cloud Foundry.

Presentation of myself

Hello, thank you for coming, my name is luigi bennardis and I come from Rome –Italy. First of all I would like to thanks Oreily for having invited me in such an important event.

Just something about me about

MASTER DEGREE in statistcs and economics at the university of rome then twenty year of experience in the field of Information Technology ( seniority ) as a developer than as architect and finally as a system integrator and technical project leader.

Actually for about 6 years Software Configuration Manager - responsible of the design implementation diffusion and management of the ALm platform of an important Italian Company. Platform based both on market and open source tools like GIT Jenkins/ Microsoft TFS and IBM Jazz.

What about this talk: I will try to describe the end to end process of delivery of a microservice from design pattern till delivery

Domain-driven design. Continuous Delivery. On-demand virtualization. Infrastructure automation. ~~Small autonomous teams~~. DevOps. Microservices have emerged from this world: they weren't invented or described before the fact. They emerged as a trend, or a pattern, from real-world use, starting from all that stuff. With Microservices architecture, the use of small developer teams becomes a reality as it is much more feasible to assign smaller independent team of developers compared to building a large monolithic application. Similarly, the responsibility in the IT operations team can be assigned to a smaller group, building a foundation for stronger and deeper collaboration between the developers and IT Ops. Such a close collaboration is at the heart of DevOps and Microservices can go a long way to seamlessly enable such a collaboration. Containers (see docker) offer the right abstraction to encapsulate Microservices. A PaaS offering that takes advantage of containers offers the right standardization to streamline deployment pipelines, maximizing DevOps benefits. When this is combined with the collaboration advantage enjoyed by the small teams building and deploying Microservices, you are in a position to achieve DevOps nirvana. Throughout this talk, we will try to paint a picture of how to design, build, manage, and deploy microservices. And, remember: do one thing and do it well.

focusing which will be the technology choises that will fullfill each phase in terms of development framework and infrastructure

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | pattern | Technology FRAMEWORK | INFRASTRUCTURE |
| Design PATTERN | DATABASE PER SERVICE | JDON | NA |
| Development |  | Spring cloud | DBAS A SERVICE & JENKINS@OPENSHIFT |
| SERVICE REGISTRY | EUREKA | NA |
| LOAD BALANCING | RIBBON | NA |
| Deliver | NA | LOCAL/DOCKER/PIVOTAL | DOCKER HUB |

* 1. **Microservices: implementing the “database per service” pattern**

**Event sourcing and command query segregation pattern**

[**https://github.com/cer/event-sourcing-examples/wiki/DeveloperGuide**](https://github.com/cer/event-sourcing-examples/wiki/DeveloperGuide)

[**https://github.com/cer/event-sourcing-examples/wiki/WhyEventSourcing**](https://github.com/cer/event-sourcing-examples/wiki/WhyEventSourcing)

[**https://webcache.googleusercontent.com/search?q=cache:4O9SFBtXxowJ:https://plainoldobjects.com/2015/09/02/does-each-microservice-really-need-its-own-database-2/+&cd=2&hl=it&ct=clnk&gl=it**](https://webcache.googleusercontent.com/search?q=cache:4O9SFBtXxowJ:https://plainoldobjects.com/2015/09/02/does-each-microservice-really-need-its-own-database-2/+&cd=2&hl=it&ct=clnk&gl=it)

[**http://www.slideshare.net/chris.e.richardson/microservices-in-java-and-scala-sfscala**](http://www.slideshare.net/chris.e.richardson/microservices-in-java-and-scala-sfscala)

**Jdon Framework : a CQRS/DDD/ES Framework**

[**http://en.jdon.com/index.html**](http://en.jdon.com/index.html)

## Description

1. INTRODUCTION  
---------------------------------------------  
JdonFramework is a java framework to build your Domain Driven Design + CQRS + EventSource applications with asynchronous concurrency and higher throughput.  
  
  
  
2. GETTING STARTED  
------------------  
In the "example" directory there are several examples for web application.  
  
You can run runTest.bat in this directory to see how to play JdonFramework in Application.  
  
In the "doc" directory there are all documents about how to use.  
  
Source Project : [https://github.com/banq/jdonframework [https://a.fsdn.com/con/img/icons/external_asset.png](https://github.com/banq/jdonframework)](https://github.com/banq/jdonframework)

**Event sourcing**

[**https://github.com/cer/event-sourcing-examples**](https://github.com/cer/event-sourcing-examples)

[**https://ookami86.github.io/event-sourcing-in-practice**](https://ookami86.github.io/event-sourcing-in-practice)

**1.2 Implementing a microservice with Spring Boot**   
1.2.1 Backing Service integration and configuration   
1.2.2 Implementing the spring profile pattern   
1.2.3 How to create a backing services in Pivotal Cloud Foundry   
1.2.4 How to bind a backing services (PAAS CONNECTOR/JAVA CONFIGURATION)   
1.2.5 Define an automated build (Jenkins@Openshift) and promote the docker image in container registry (DockerHub)

**1.3 Interactions between Microservices: service discovery and service registration**   
1.3.1 Eureka service registry facilities   
1.3.2 How to implement a microservice and register inside Eureka with Spring Cloud   
1.3.3 Implementing a client application that consume an Eureka server application with Ribbon client side load balancing features

**1.4 Solution delivery**1.4.1 In a Cloud Environment: Pivotal Cloud Foundry  
1.4.2 As a docker container (Nota: deprecate boot2docker with docker machine)

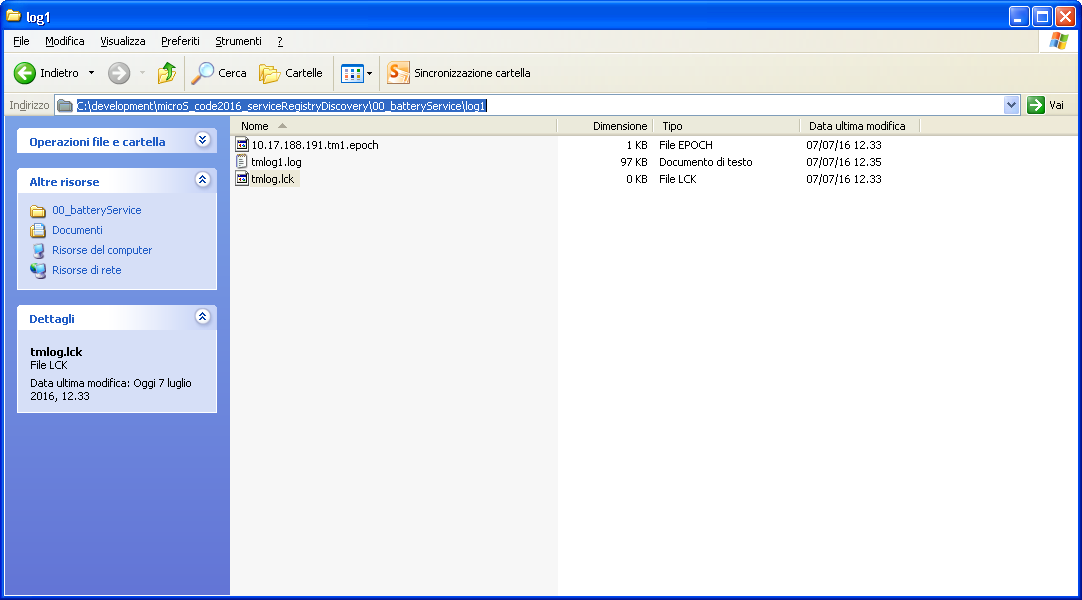
<https://docs.docker.com/engine/installation/windows/>

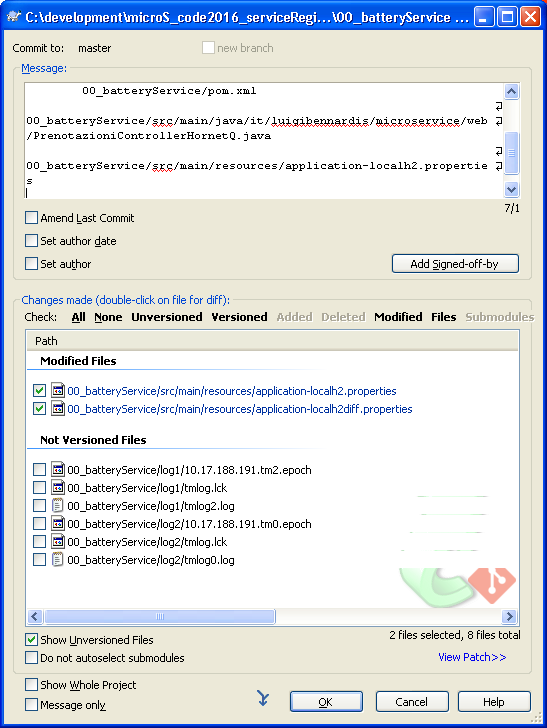
last

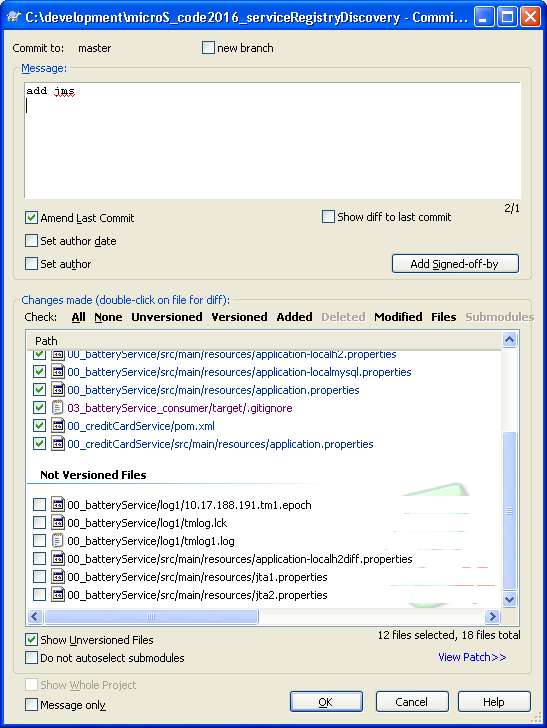
* 1. Microservices: implementing the “database per service” pattern

Atomikos lock file

C:\development\microS\_code2016\_serviceRegistryDiscovery\00\_batteryService\log1







1.2 Implementing a microservice with Spring Boot   
1.2.1 Backing Service integration and configuration   
1.2.2 Implementing the spring profile pattern   
1.2.3 How to create a backing services in Pivotal Cloud Foundry   
1.2.4 How to bind a backing services (PAAS CONNECTOR/JAVA CONFIGURATION)   
1.2.5 Define an automated build (Jenkins@Openshift) and promote the docker image in container registry (DockerHub)

1.3 Interactions between Microservices: service discovery and service registration   
1.3.1 Eureka service registry facilities   
1.3.2 How to implement a microservice and register inside Eureka with Spring Cloud   
1.3.3 Implementing a client application that consume an Eureka server application with Ribbon client side load balancing features

1.4 Solution delivery   
1.4.1 In a Cloud Environment: Pivotal Cloud Foundry  
1.4.2 As a docker container