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

[1 Microservizi 2](#_Toc444337046)

[1.1 Pattern “database per service” 2](#_Toc444337047)

[1.2 “Backing Service” 2](#_Toc444337048)

[1.2.1 INTRO 2](#_Toc444337049)

[1.2.2 SPRING PROFILE 2](#_Toc444337050)

[1.2.3 BINDING BACKING SERVICES IN CLOUD FOUNDRY 3](#_Toc444337051)

[1.2.4 PAAS CONNECTOR 3](#_Toc444337052)

[1.2.5 JAVA CONFIGURATION 3](#_Toc444337053)

[1.3 “Service discovery and registration” 3](#_Toc444337054)

[1.3.1 EUREKA SERVICE REGISTRY 4](#_Toc444337055)

[1.3.2 SERVER APPLICATION REGISTRATION IN EUREKA 4](#_Toc444337056)

[1.3.3 CLIENT APPLICATION CONSUMING SERVER APPLICATION ON EUREKA 4](#_Toc444337057)

[1.3.4 AGGIUNGERE UN NUOVO SERVER 4](#_Toc444337058)

# Microservizi

## Pattern “database per service”

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>

## “Backing Service”

### INTRO

### 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)**
2. **Esecuzione mySqlLocal -/Env**
3. **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**

### BINDING BACKING SERVICES IN 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**

### PAAS CONNECTOR

Definizione datasource

@Configuration

@Profile(“cloudFoundry”)

Mettere log print sulla classe che definisce la configuration

**DEMO**

1. **Esecuzione cloudFoundry /Env**

### JAVA CONFIGURATION

Definizione datasource

@Bean

@Profile(“cloudFoundry”)

Definizione di un custom connetion pool

## “Service discovery and registration”

TUTTE LE APPLICAZIONI DEVONO ESSERE PREDEPLOYATE

I MICROSERVIZI E IL LORO ECOSISTEMA

### EUREKA SERVICE REGISTRY

Cosa è Eureka

Implementazione applicazione

Console

**DEMO**

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

### 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)