Rome is the city where I live since I wos born. It counts about 3.5 official inhabitants to witch we have to add about 1 million of non resident people (students, tourists, workers). To these figures we have also to add about 700 vehicles each 1000 people, a rate that makes mobility a challenge.

Everyone could easily realize that with there figures besides mobility there are also pollution issues.

A partial solution of these issues could be addressed reducing the vehicles that daily deliver goods.

For a more sustainable for the environment and efficient mobility in such a context the daily delivery of goods could be done by means of (through) “smart” electrical vehicles.

To overcome the limited life of the batteries these vehicles has been enginnered with special battery pack that could be easily and rapidly changed in dedicated service station.

In this way these vehicle will not need to stop for charging their batteries and so could guarantee much more short breaks

Could be much more efficient in comparison to the other electric vehicle (that do non support this kind of batteries – not engineered in such a way)

I do not know if this is a realizable scenario but if so it is necessary a software solution by witch manage this scenario.

Each driver of this ecological delivery company according to his delivery plan each morning program the expected battery changes that will reasonably occur during a day. By means of a mobile application (web application) he will book one or more fresh batteries supplied by the stations distributed in the city.

In case of emergency it will be the vehicle itself that will notify to the driver the necessity to come to the nearest station with available fresh batteries, find by a software running in the vehicle.

Today's Rome is a modern [metropolis](https://en.wikipedia.org/wiki/Metropolis), yet it reflects the stratification of the epochs of its long history. The historical centre, identified as those parts within the limits of the ancient Imperial walls, contains archaeological remains from Ancient Rome. These are continuously being excavated and opened to the public, such as the [Colosseum](https://en.wikipedia.org/wiki/Colosseum" \o "Colosseum); the [Roman Forum](https://en.wikipedia.org/wiki/Roman_Forum), and the [Catacombs](https://en.wikipedia.org/wiki/Catacombs_of_Rome). There are areas with remains from [Medieval](https://en.wikipedia.org/wiki/Medieval) times. There are palaces and artistic treasures from the [Renaissance](https://en.wikipedia.org/wiki/Renaissance); fountains, churches and palaces from [Baroque](https://en.wikipedia.org/wiki/Baroque) times. There is art and architecture from the [Art Nouveau](https://en.wikipedia.org/wiki/Art_Nouveau), [Neoclassic](https://en.wikipedia.org/wiki/Neoclassical_architecture), [Modernist](https://en.wikipedia.org/wiki/Modernist) and [Rationalist](https://en.wikipedia.org/wiki/Rationalist) periods. There are museums, such as the[Musei Capitolini](https://en.wikipedia.org/wiki/Capitoline_Museums), the [Vatican Museums](https://en.wikipedia.org/wiki/Vatican_Museums), [Galleria Borghese](https://en.wikipedia.org/wiki/Galleria_Borghese).

New map

<https://www.google.com/maps/d/u/0/viewer?mid=1KXbNWEmT194EMIVopTnmVTihcyA&hl=it>

I do  not know if this is realizable because i am not an engineer but is so the management should be addressed by a system that will satisfy the following requirements

**Requirements**

Funtional

Non Functional

Project Management

So I will try to show a possible process of development of a microservices ecosystem from the very first phase of design to delivery

**Patterns vs technology stacks**

In this chart i have tried to summarize the technology stacks that realize the design patterns defined for choose for the development.

In this chart I have tried to summarize the technology stacks that realize the design patterns choosed for the architecture of this system

**Lifecycle vs service vs environment specifity**

In this chart I have arranged that will accomplish the lifecycle steps imagined for this project and the corresponding realizing services both in a local and in a cloud environment.

In this chart i have arraged the    that accomplish the lifecycle steps imagined for this project and the corrispondending realizing services   
both in a local and in a cloud environment.

**Database per service will fulfill the non functional requirements**

Description of database per service pattern for developing Micorservices

**Demo**

**Demo Pws**Database as service pattern will enforce database per service pattern.  Dedicated instance , dedicated schema and horizontal scale up

## Demo

## Integration test - Docker Local

Docker definition

-I have provided an empty database image that will be load by flyway at start up

Show Dockerfile

Show dockerfile template

Show maven implementation and plug in

Create a local docker image

Run the container + database

Show bindings

## Quality assurance - Docker Hub - Jenkins@openshift

## Transactional

1. DESCRIPTION
2. Start ZOOKEEPER
3. Start KAFKA
4. Start MongoDB
5. START LISTENER TO TOPIC
   1. confirmBookingTopic
   2. notConfirmBookingTopic
   3. pendingBookingTopic
6. project **01-batteryBookingInformation\_SERVICE**
   1. DETAILS: TOPIC IMPLEMENTATION
   2. DETAILS: PROCESS DB INQUIRY
   3. DETAILS: POM
      1. spring-cloud-stream
      2. spring-cloud-starter-stream-kafka
      3. spring-boot-starter-data-mongodb
   4. RUN FROM .BAT
   5. LIST BOOKING INFOS

<http://localhost:7112//bookingInfoMaterializedView/list> NULL

1. project **01-batteryManagement\_SERVICE**
   1. DETAILS: TOPIC IMPLEMENTATION
   2. DETAILS:
   3. RUN 01\_**batteryManagement** \_SERVICE
      1. Show log output
      2. Show database load
   4. Show topic subscriber **EMPTY**
   5. RUN FROM .BAT
2. project 01\_bookAbattery\_SERVICE
   1. DETAILS: TOPIC IMPLEMENTATION
   2. DETAILS: PROCESS DB INQUIRY
   3. DETAILS: POM
      1. spring-cloud-stream
      2. spring-cloud-starter-stream-kafka
   4. RUN 01\_bookAbattery\_SERVICE –profile localmysql FROM .BAT
      1. Show log output
      2. Show database load



* + 1. Show log file with executing jobs

DbPollingPendingCleaner Job -> 09/06/2016 16:29:13

DbPollingPending Job -> 09/06/2016 16:29:13

DbPollingPending Job -> 09/06/2016 16:29:23

DbPollingPending Job -> 09/06/2016 16:29:33

DbPollingPendingCleaner Job -> 09/06/2016 16:29:38

DbPollingPending Job -> 09/06/2016 16:29:43

DbPollingPending Job -> 09/06/2016 16:29:53

DbPollingPending Job -> 09/06/2016 16:30:03

DbPollingPendingCleaner Job -> 09/06/2016 16:30:03

DbPollingPending Job -> 09/06/2016 16:30:13

DbPollingPending Job -> 09/06/2016 16:30:23

DbPollingPendingCleaner Job -> 09/06/2016 16:30:28

* + 1. Show Topic subscriber



* + 1. Show database update



* + 1. Blah













1. Stop core services and check information services still running

## Service Discovery

### 02-eureka-server

Overview of the project

File yml etc

Configurations of Eureka

Run project from command line .bat

Show console



#### Detail the message “THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS”

2016-09-08 14:52:35.431 WARN 8232 --- [a-EvictionTimer] c.n.eureka.PeerAwareInstanceRegistry : The self preservation mode is disabled!. Hence allowing the

instances to expire.

<https://github.com/ExampleDriven/spring-cloud-eureka-example/blob/master/eureka-server/src/main/resources/application.yml>

http://stackoverflow.com/questions/33921557/understanding-spring-cloud-eureka-server-self-preservation-and-renew-threshold

### 06\_bookABatterySERVICE4EUREKA

Overview of the project

Eureka directives on application .class

File yml etc

Run project from command line .bat

Check registration



#### 06\_bookABatterySERVICE4EUREKA 7113-7115

Overview of the project

Eureka directives on application .class

File yml etc

Run project from command line .bat

Check registration



### Discovery

### 06\_bookABatteryCLIENT\_DISCOVERY\_SERVICE

Code description

Run from .bat

Call:

## Load balancing with ribbon

### 06\_bookABatteryCLIENT\_FEIGN\_SERVICE

Show implementation

<http://sdpsvrsa094:7112/>

show requested on the tree instance



Stop two and how requested on the one running





Suppress warn and info on services display

## Load Balancing

## Load Balancing

To cover in such a short time all these aspect each technology and process aspects are been described at a very high level,and need to be detailed in deep

07 June 2016 on [netflix](https://luizkowalski.net/tag/netflix/), [oss](https://luizkowalski.net/tag/oss/), [eureka](https://luizkowalski.net/tag/eureka/)

In this series of posts, I'll try to get you inside the Netflix stack: understand how it works and get your feet wet in the microservices world.

So, to begin, we need to know the first component of the Netflix stack: Eureka

The Eureka is described by Netflix as

a REST based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers. We call this service, the Eureka Server. Eureka also comes with a Java-based client component,the Eureka Client, which makes interactions with the service much easier. The client also has a built-in load balancer that does basic round-robin load balancing.

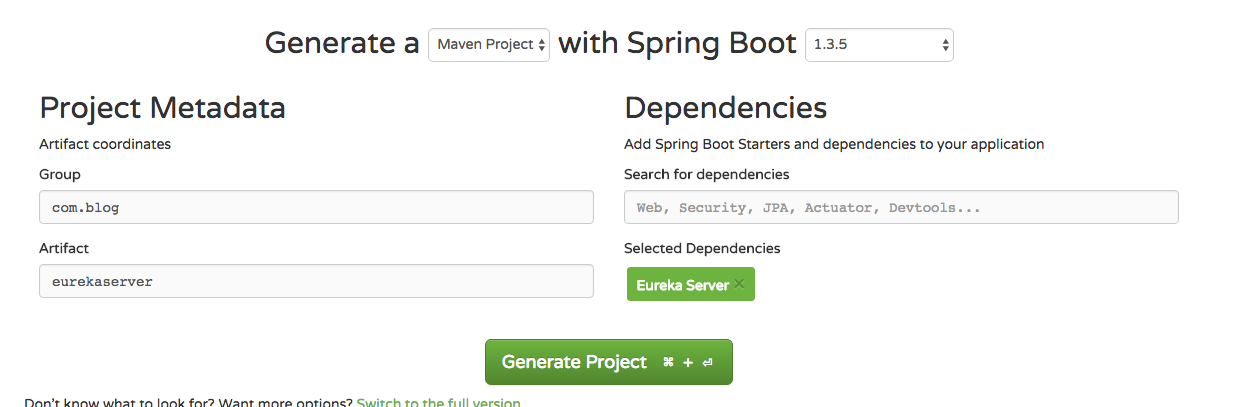
So basically, Eureka is a register, that will know where which one of our services lives, how many instances of they are up (or down) and how to access them.

Thanks to Spring, all the complexity to get a Eureka server up and running was wrapped inside useful libraries that we'll be using in this series.

So, enough with chit chat, let's get to work.

#### Creating an Eureka Server using Spring

Get a Eureka Server to run is unbelievable easy. First, visit [http://start.spring.io](http://start.spring.io/) and fill the form as follow:



Pay attention to "Eureka Server" dependency listed.   
Now, download the project clicking "Generate Project" and import it to your IDE (if you are using Eclipse, you can "mavenize" it running mvn eclipse:eclipse on root).

Now, if you open it, you'll see an empty Spring Boot project, then, go to your application.yml (or application.properties if you prefer) and change as follow:

server:

port: 8761

eureka:

numberRegistrySyncRetries: 1

instance:

hostname: localhost

client:

registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

server:

enable-self-preservation: true

Explaining the most important lines:

server:

port: 8761

Here, we are configuring the Eureka Server to run on port 8761. This is the default port and you can change, but you need to give this port to the clients later on.

eureka:

numberRegistrySyncRetries: 1

If you are running locally, there is a 2 to 3 minutes wait until fulling boot up. This happens because Eureka will be looking for peers. To disable this, set to 0 (although you should never do this in production)

client:

registerWithEureka: false

As this is the Eureka Server, we do not want it to register itself. Will always be set to false on server and true on the clients.

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

the defaultZone is the fallback URL for every client that doesn't specify a preference for a server.

Now, the only thing we should do is enable the Eureka Server.   
To do so, go to the main class (at this point, you should only have one class in the project, though) and annotate it with @EnableEurekaServer, as follow

@EnableAutoConfiguration

@EnableEurekaServer

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

and start the server with mvn spring-boot:run

the latest log lines would be something like this

2016-06-07 21:34:38.624 INFO 723 --- [ main] o.s.c.support.DefaultLifecycleProcessor : Starting beans in phase 0

2016-06-07 21:34:38.627 INFO 723 --- [ main] .p.EurekaConfigBasedInstanceInfoProvider : Setting initial instance status as: STARTING

2016-06-07 21:34:39.120 INFO 723 --- [ main] com.netflix.discovery.DiscoveryClient : Not registering with Eureka server per configuration

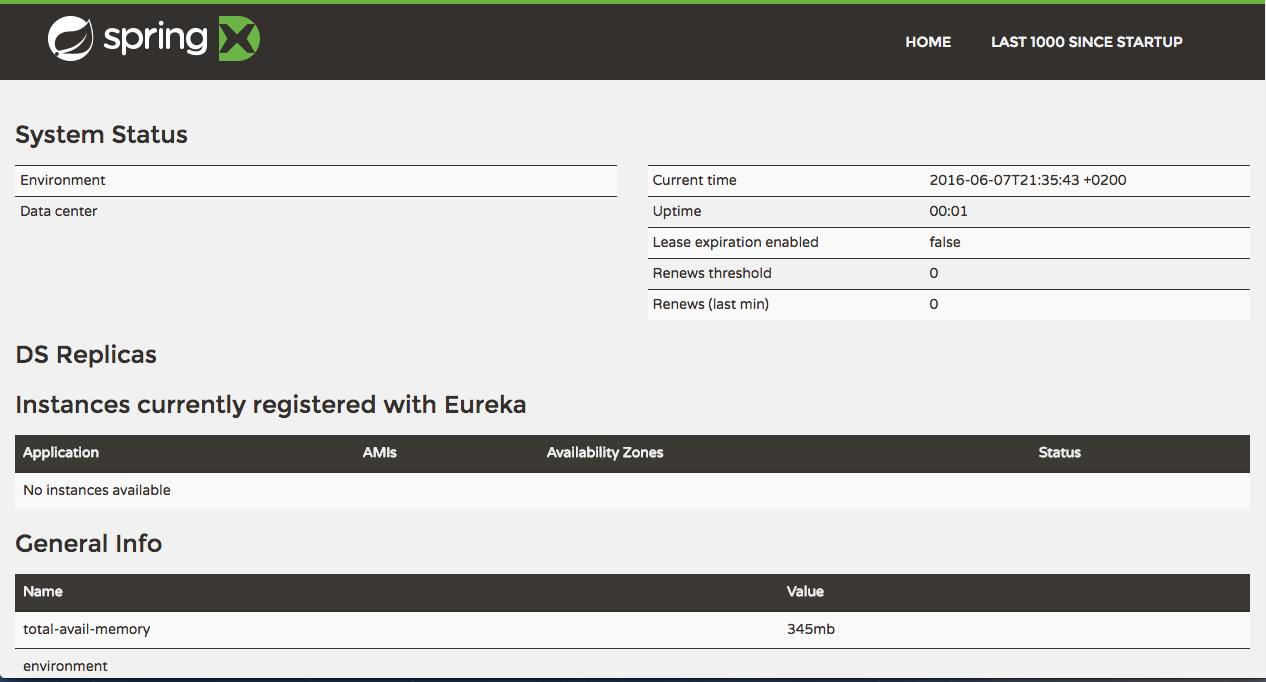
2016-06-07 21:34:39.127 INFO 723 --- [ main] c.n.e.EurekaDiscoveryClientConfiguration : Registering application bootstrap with eureka with status UP

2016-06-07 21:34:39.202 WARN 723 --- [ Thread-2] c.n.eureka.PeerAwareInstanceRegistry : The replica size seems to be empty. Check the route 53 DNS Registry

2016-06-07 21:34:39.362 INFO 723 --- [ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8761 (http)

2016-06-07 21:34:39.366 INFO 723 --- [ main] com.inkdrop.EurekaServerApplication : Started EurekaServerApplication in 12.2 seconds (JVM running for 18.242)

Access [http://127.0.0.1:8761](http://127.0.0.1:8761/) and you'll see this:



and that's it! Your Eureka Server is up and running.   
In the next post, we'll create a microservice that will register itself on Eureka.

If you want to see the code already, go to the final [github project](https://github.com/luizkowalski/microservices-netflix-sample)

🍻 🍻

[Luiz Eduardo's Picture](https://luizkowalski.net/author/kowalski/)

#### [Luiz Eduardo](https://luizkowalski.net/author/kowalski/)

Read [more posts](https://luizkowalski.net/author/kowalski/) by this author.

Berlin

#### Share this post

[Twitter](https://twitter.com/intent/tweet?text=Netflix%20OSS%3A%20A%20beginner%27s%20guide%20%5Bpt1%5D&url=https://luizkowalski.net/netflix-oss-a-beginners-guide-pt1/)[Facebook](https://www.facebook.com/sharer/sharer.php?u=https://luizkowalski.net/netflix-oss-a-beginners-guide-pt1/)[Google+](https://plus.google.com/share?url=https://luizkowalski.net/netflix-oss-a-beginners-guide-pt1/)