O'REILLY®
Software Architecture
ENGINEERING THE FUTURE OF SOFTWARE

softwarearchitecturecon.com
#OReillySACon

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

Luigi Bennardis

l.bennardis@email.it
https://it.linkedin.com/in/luigi-bennardis-0a56a72

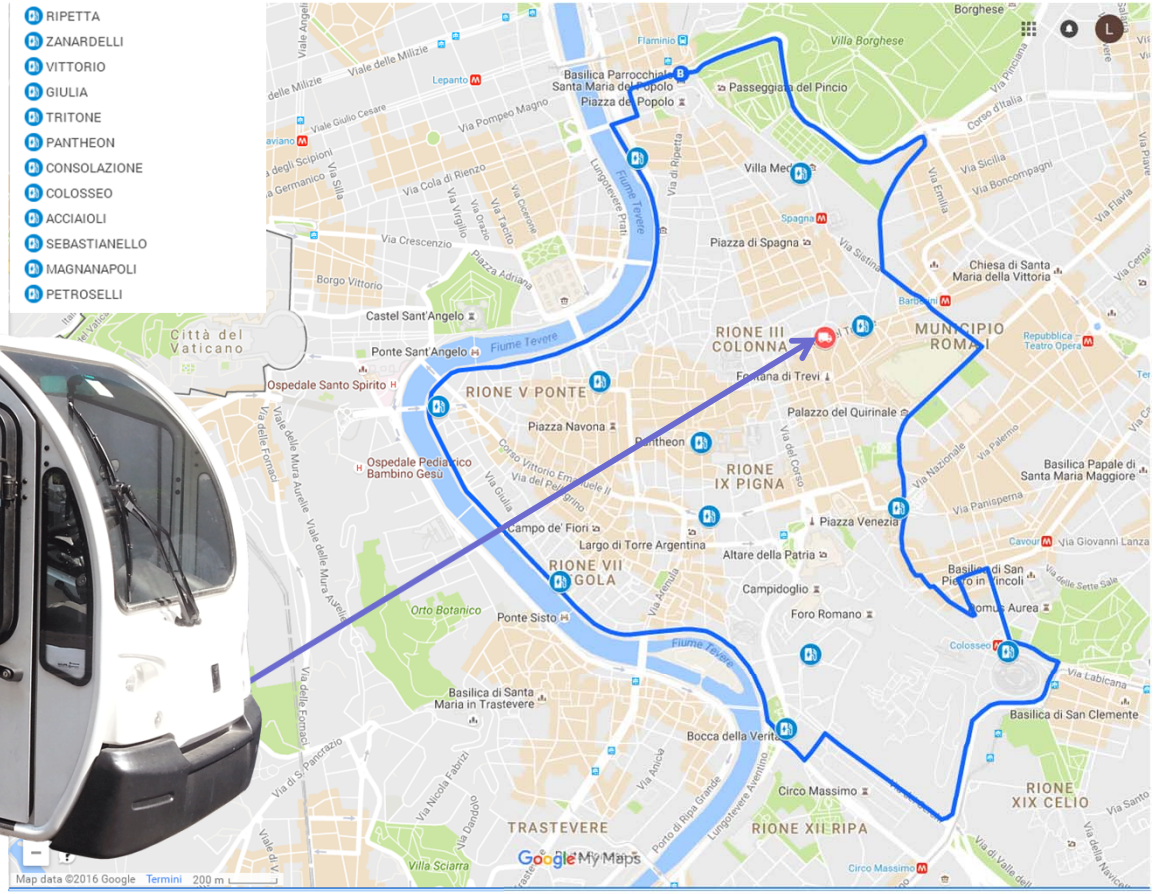
The goals of this project:

- The development of a microservice-based, fault-tolerant, reliable architecture
- Its lifecycle
- Its delivery in a cloud environment or as a Docker container

To this end we will create:

- A simple microservices Spring Boot-based architecture and its development lifecycle
- An **Event-Driven Architecture** implemented by a **Database-per-service poliglot persistence** pattern
- The implementation of two key patterns of the microservice design: **service discovery** and **client side load balancing**
- A **smooth deployment scenario** which starts from a local environment and subsequently moves to a Docker container before ending up in the Cloud

A microservice-based digital platform for a sustainable mobility



Functional requirements

- Each driver, according to his daily delivery schedule, will decide the expected pit stops for battery replacement in one of the stations within the historical center by means of a mobile application.
- The smart cockpit of each vehicle will be able to locate the nearest pit stop station for an emergency battery switch and go.
- The digital platform, due to its mission critical ends, should be fairly resilient.

Technical Requirements

- The services of this computing system should be **loosely coupled**. Each of these services is independent of the others during the development and deployment phases, as well as during the process of scaling out.
- For each service **the adoption of new technology** or design pattern should be quick and seamless with respect to the overall system
- **Deployment should be flexible** regardless of the environments
- Data consistency shoud be achieved by **asynchronous non-blocking operations**
- Data persistence will be accomplished by a **«poliglot approach»** with different and specialized data storage for each service
- The **infrastructure scale-out** should be as simple as possible (zero service downtime)
- **Service discovery** should be achieved without the need for additional coding
- **Client side load balancing** features should be obtained without additional infrastructure needs, simply adding new service instances

Project Management Requirements

- The software delivery should be more streamlined, eliminating the need for coding and redeploying at every step while maintaining a cohesive overall structure
- Thus, we avoid the management of great codebases
- The project staff will be organized into small, self-contained feature teams, geographically distributed according to each specific need (for example, we do not need engineers/developers working in test or functional teams)

Microservices-based architecture: «database per service» pattern

- The services will be **loosely coupled**, so that they may be developed, deployed and scaled-out independently
- Each service could be developed using a different technology
- The need for **service improvement** could be addressed using a different technological stack without any impact on the rest of the system
- The design of each service could be carried out using the type of database that is best suited to its need. For example **neo4j** for social graph, **elasticsearch** for text searches
- It will be possible to apply scaling only for those services that need it, resulting in greater cost efficiency
- Deployment could be more efficient compared to a monolitic architecture: the change to a single service could be immediately deployed, isolated, and quickly rolled back
- The persistence data of each microservice is kept private and exclusive to that service and must be accessible only via its API, so that changes to one service's database do not impact any other services.
- This behaviour could be enforced through the following approaches:
  - **Private table per service** (the weakest approach **but** with the lowest overhead)
  - **Schema per service** (making clear the ownership structure of database tables)
  - **Database server per service** (the strongest approach, to be applied only for high throughput services)

Requirements fullfilment: technology stack

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Boot

- Spring Boot simplifies Spring application configuration by applying **Convention over Configuration** (CoC)
- Spring Boot **auto-configuration** feature provides a set of default behaviors that are driven by what libraries are on the classpath
- Spring Boot simplifies deployment, **packaging application as an executable jar** containing all the dependency libraries and an embedded web container
- To run a Spring Boot microservice need only to have Java installed
- Makes easy to get a new micro-service up and running only having java installed with little or no configuration while preserving the ability to customize your application.

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Cloud Stream

- Spring Cloud Stream, based on integration between Spring Integration and Spring Boot, is a framework for building event-driven microservices.
- Spring Cloud Stream simplify the programming model, focusing on application business logic: the messaging middleware access comes out-of-the-box, for free

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Data REST

- Simplify the development of hypermedia-based RESTfull web services on top of Spring Data repositories.

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Data JPA

- Spring Data Java Persistance Api improves the implementation of data access layers by reducing the coding effort, providing the automatic implementation of repository interfaces and custom finder methods

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Mongo DB

- The Spring Data MongoDB project provides integration with the MongoDB document database.

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Spring Cloud

- Spring cloud, built on top of spring boot to support development of microservices ,
- Provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing,
- Services and applications Spring Cloud based will work in any distributed environment, including the developer's own laptop and managed platforms such as Cloud Foundry.

Netflix Eureka

- Is a REST (Representational State Transfer) based service for locating services with the purpose of basic round-robin load balancing and failover of middle-tier servers.

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Netflix Ribbon

- Ribbon provides software-based load balancers with rotation features

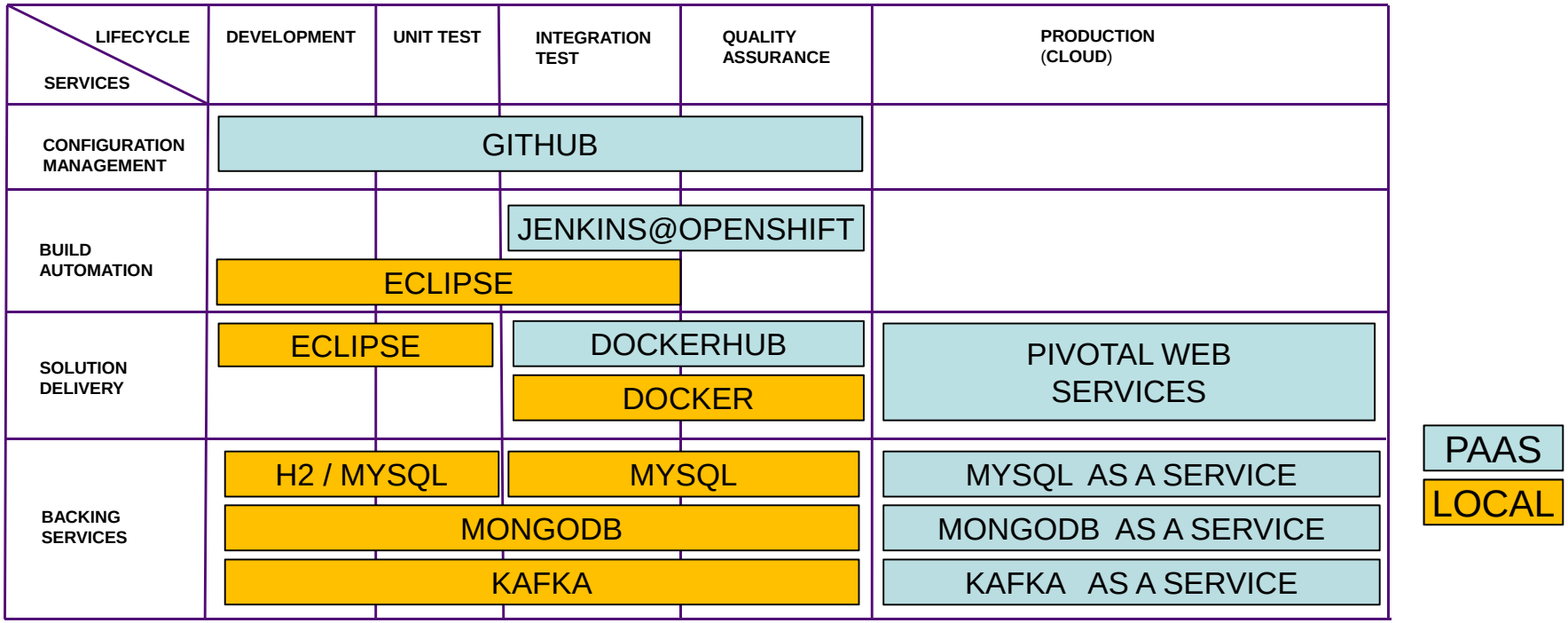
DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

Apache Kafka

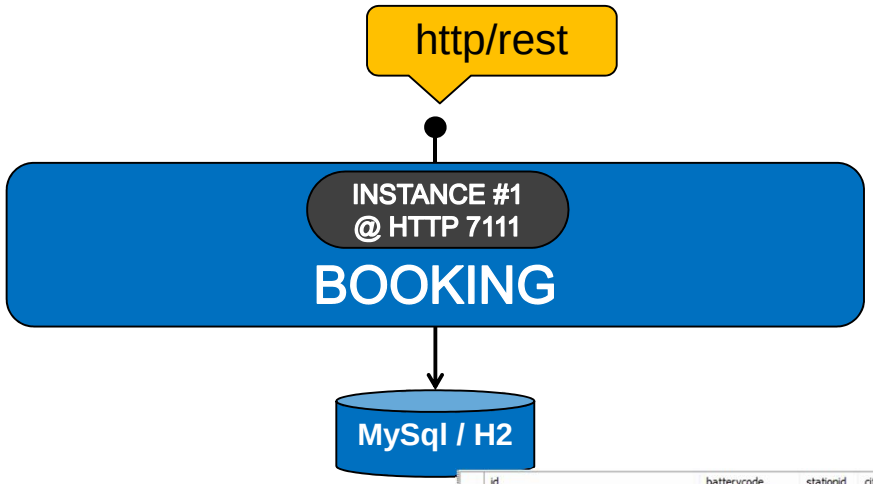
- Apache Kafka is a platform for handling real-time data feeds designed to be highly available;
- Apache Kafka uses Apache Zookeeper to coordinate cluster information in which all nodes are interchangeable.
- Data is replicated from one node to another to ensure that it will still be available in the event of a failure

DESIGN PATTERN		TECHNOLOGY STACK
CORE MICROSERVICE		SPRING BOOT
DATABASE PER SERVICE	EVENT DRIVEN ARCHITECTURE	SPRING CLOUD STREAM
	MATERIALIZED VIEW	
	REST API	SPRING DATA REST
DATA ACCESS	RELATIONAL DATABASE (MYSQL-H2)	SPRING DATA JPA
	NO SQL DATABASE (MONGO DB)	SPRING MONGO DB
CLOUD BASED ARCHITECTURE		SPRING CLOUD
SERVICE DISCOVERY		NETFLIX EUREKA
LOAD BALANCING		NETFLIX RIBBON
MESSAGE BROKER		APACHE KAFKA

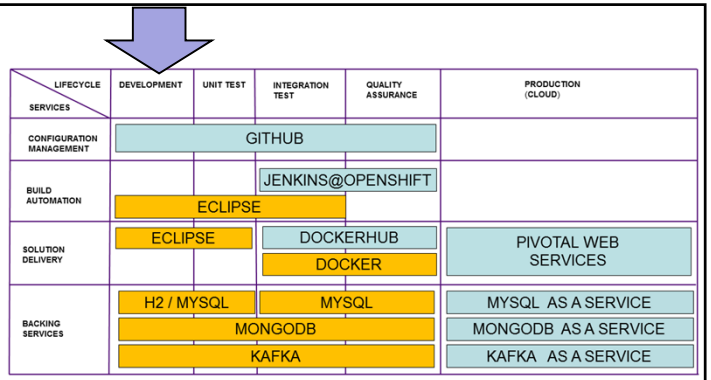
Tools of lifecycle process



DEVELOPMENT / UNIT TEST



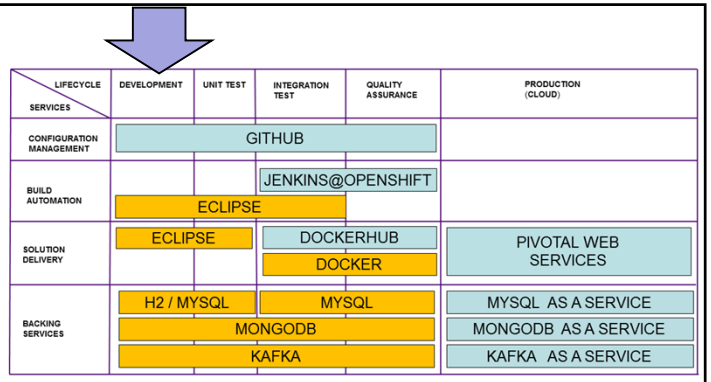
id	batterycode	stationid	city	latitude	longitude
04abfea3-2241-481a-8ec2-30d8c6523864	STAZ001BATT001	STAZ001	ROMA	42.89898000	16.98989800
5563b981-f400-427b-9a4a-e46a9b03b53c	STAZ001BATT011	STAZ001	ROMA	42.98454545	16.44444000
6b1bea06-afe6-41e9-b9fa-21d57d3b54d2	STAZ001BATT001	STAZ001	ROMA	42.78987000	16.89098000
813c22f8-65d7-423a-bd4c-591ebd0ba61d	STAZ001BATT001	STAZ001	ROMA	42.00000000	16.00000000
b8d0008f-4dc4-4d77-82b0-2d148e4c4da8	STAZ001BATT001	STAZ001	ROMA	42.00000000	16.00000000
df4a3110-09ac-43a3-a4be-0331b9069679	STAZ001BATT002	STAZ001	ROMA	42.34489800	17.98989800
f264184d-7d53-493c-933b-70725c5dbc28	STAZ001BATT011	STAZ001	ROMA	42.98454545	16.74444000
ffc01b43-5694-44ad-9ead-6ca513ff4ed4	STAZ001BATT011	STAZ001	ROMA	42.94445455	16.44444000



- **Microservices technical layout:**
  - HTTP Rest API
  - Incapsulation of core functionality and database access
  - Invariant database access implementation
    - embedded H2 in memory
    - MySql as a server instance

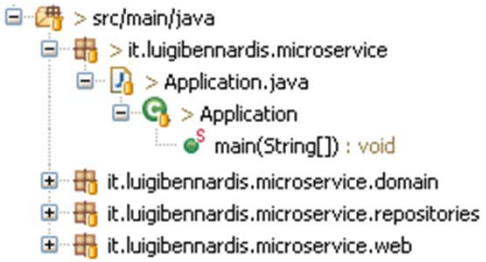
DEVELOPMENT / UNIT TEST

- flyway-core : 3.2.1 (managed:3.2.1)
- mysql-connector-java (managed:5.1.37)
- spring-boot-starter-web (managed:1.3.0.RELEASE)
- spring-boot-starter-data-rest (managed:1.3.0.RELEASE)
- spring-boot-starter-data-jpa (managed:1.3.0.RELEASE)
- h2 (managed:1.4.190)

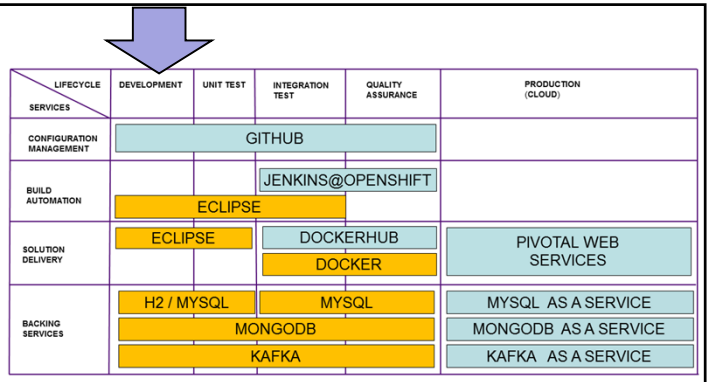


- **Development dependencies**
  - Flyway: the database migration tool
  - Java connector for MySQL
  - Starter web: the Spring Boot capabilities for web application
  - Data rest: the Spring Boot capabilities for rest web services on top of data repositories
  - JPA for the data management in a relational database
  - H2 database

DEVELOPMENT / UNIT TEST



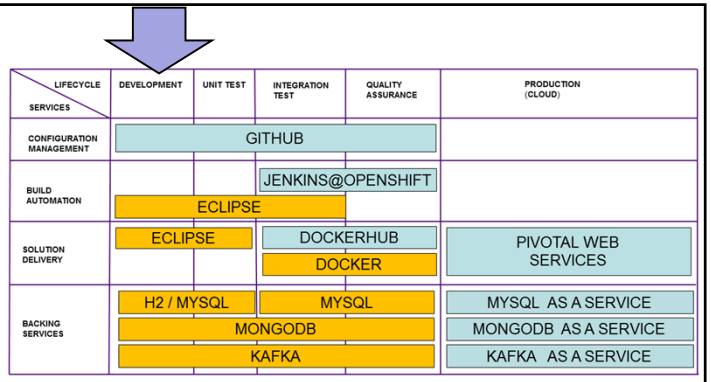

```
@SpringBootApplication
public class Application {
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
```



- **Development details**
  - Microservice's packaging hierarchy
  - *@SpringBootApplication:* the annotation needed by a Spring Boot main class
  - Domain classes
  - JPA Data Access reposisories
  - Rest API

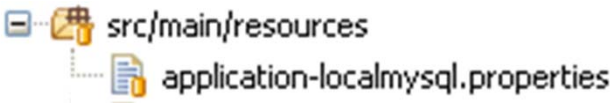
DEVELOPMENT / UNIT TEST


```
@RestController
@RequestMapping(value = "/bookABattery")
public class BookABatteryController {

@Autowired
private final IBookingInfoRepository prenotazioniRepository;

@Autowired
BookABatteryController(IBookingInfoRepository prenotazioniRepository) {
        this.prenotazioniRepository = prenotazioniRepository;}

@RequestMapping(value= "/addBooking/{stazione}/{batteria}/{citta}/{latitudine}/{longitudine}")
public Booking addBook(@PathVariable String stazione, @PathVariable String batteria,
                           @PathVariable String citta , @PathVariable String latitudine,
                           @PathVariable String longitudine) {

      Booking prenotaBatteria = new Booking(stazione +
      batteria,stazione,citta,Double.valueOf(latitudine),Double.valueOf(longitudine));

      prenotazioniRepository.saveAndFlush(prenotaBatteria);

      return prenotaBatteria;}
```

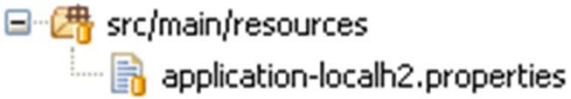


- **Development details**
  - REST Controller implementation
  - Uses the data access methods provided by the JPA repositories (i.e: save&flush)

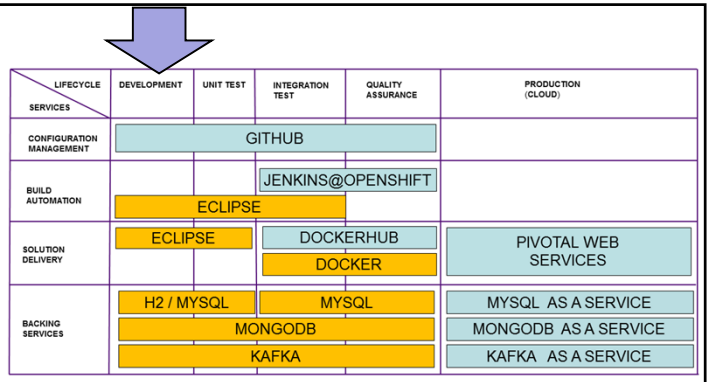
DEVELOPMENT / UNIT TEST




```
spring.datasource.url=jdbc:mysql://localhost/bookabattery_db_pws
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.username=bab_USER
spring.datasource.password=bab_USER
```



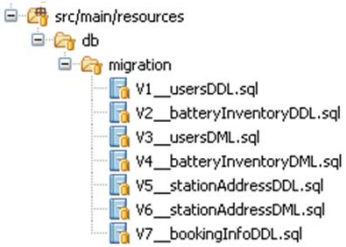

```
spring.datasource.url=
jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;
MODE=MySQL;INIT=CREATE SCHEMA IF NOT EXISTS "public"
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

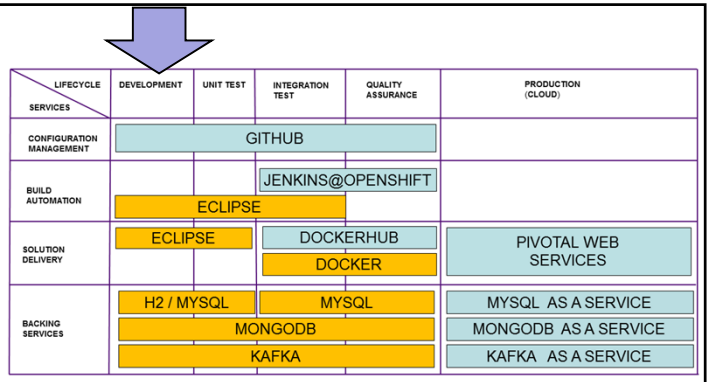


- **Configuration details**
  - This is an example of database connection declarative approach
  - This approach is provided by specific properties files called during application start up
  - The invariant access implementation derives from the existing compatibility between H2 and MySql

DEMO

DEVELOPMENT / UNIT TEST



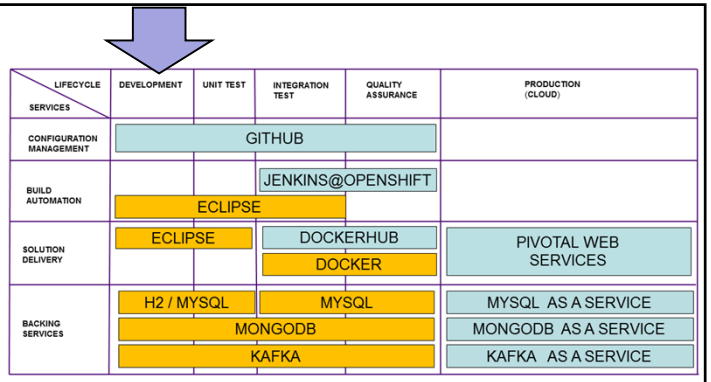



```
TomcatEmbeddedServletContainer: Tomcat initialized with port(s): 7111 (http)
StandardService: Starting service Tomcat
StandardEngine: Starting Servlet Engine: Apache Tomcat/8.0.28

VersionPrinter: Flyway 3.2.1 by Boxfuse
DbSupportFactory: Database: jdbc:mysql://localhost/bookabattery_db_pws (MySQL 5.6)
DbValidate: Validated 7 migrations (execution time 00:00.016s)
MetaDataTableImpl: Creating Metadata table: `bookabattery_db_pws`.`schema_version`
DbMigrate: Current version of schema `bookabattery_db_pws`: << Empty Schema >>
DbMigrate: Migrating schema `bookabattery_db_pws` to version 1 - usersDDL
DbMigrate: Migrating schema `bookabattery_db_pws` to version 2 - batteryInventoryDDL
DbMigrate: Migrating schema `bookabattery_db_pws` to version 3 - usersDML
DbMigrate: Migrating schema `bookabattery_db_pws` to version 4 - batteryInventoryDML
DbMigrate: Migrating schema `bookabattery_db_pws` to version 5 - stationAddressDDL
DbMigrate: Migrating schema `bookabattery_db_pws` to version 6 - stationAddressDML
DbMigrate: Migrating schema `bookabattery_db_pws` to version 7 - bookingInfoDDL
DbMigrate: Successfully applied 7 migrations to schema `bookabattery_db_pws`
           (execution time 00:03.278s).
```

- **Spring Boot start up evidences**
  - The web container starting up
  - Flyway: a database migration tool

DEVELOPMENT / UNIT TEST

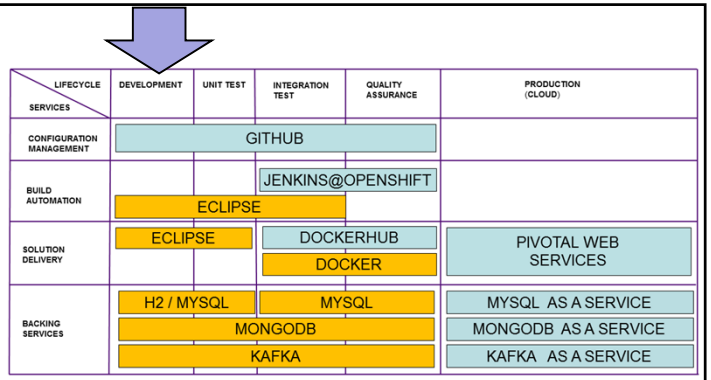
```
RequestMappingHandlerMapping: Mapped
      "{[/bookABattery/list]}"

RequestMappingHandlerMapping: Mapped
      "{[/stationAddresses],methods=[GET]}"

RequestMappingHandlerMapping: Mapped
      "{[/findNearestStation/{latitudine}/{longitudine}/{distanza}],methods=[GET]}"

RequestMappingHandlerMapping: Mapped
      "{[/bookABattery/addBooking/{stazione}/{batteria}/
      {citta}/{latitudine}/{longitudine}],methods=[POST]}"
```



- **Spring Boot start up evidences**
  - REST methods that achieve the API of this service

DEVELOPMENT / UNIT TEST

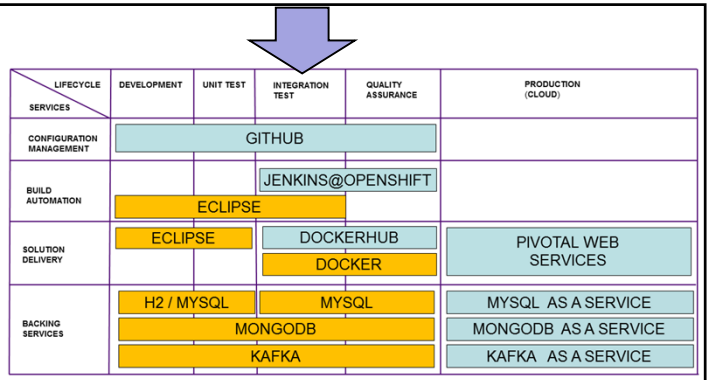
```
EndpointHandlerMapping: Mapped "{[/trace || /trace.json
EndpointHandlerMapping: Mapped "{[/flyway || /flyway.json
EndpointHandlerMapping: Mapped "{[/mappings || /mappings.json
EndpointHandlerMapping: Mapped "{[/metrics || /metrics.json
EndpointHandlerMapping: Mapped "{[/beans || /beans.json
EndpointHandlerMapping: Mapped "{[/health || /health.json
EndpointHandlerMapping: Mapped "{[/env || /env.json]
EndpointHandlerMapping: Mapped "{[/autoconfig || /autoconfig.json
EndpointHandlerMapping: Mapped "{[/info || /info.json

TomcatEmbeddedServletContainer: Tomcat started on port(s): 7111 (http)
     cloud.services.mySqlBackingServices.connection.jdbcurl JDBC URL= NOT IN A CLOUD ENV
     DATASOURCE URL=jdbc:mysql://localhost/bookabattery_db_pws.
Application: Started Application in 12.083 seconds (JVM running for 16.066)
```



- **Spring Boot start up evidences**
  - Environment end-point
  - Web container
  - Datasource resolution
  - Application

DEMO

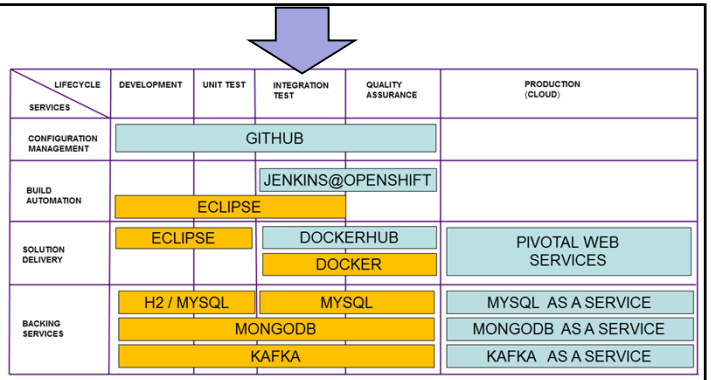
INTEGRATION TEST - Docker



- **Docker** is a way to containerize applications, allowing us to package a microservice in a standardized and portable format, independently of the technology used for its implementation
- At runtime it provides a high degree of isolation between different services
- Docker containers are extremely lightweight and as a result can be built and started extremely quickly
- A container can typically be built in just a few seconds and starting a container simply consists of starting the service's processes
- Provide ah high level of portability: infact many clouds have added extra support for Docker

Docker image

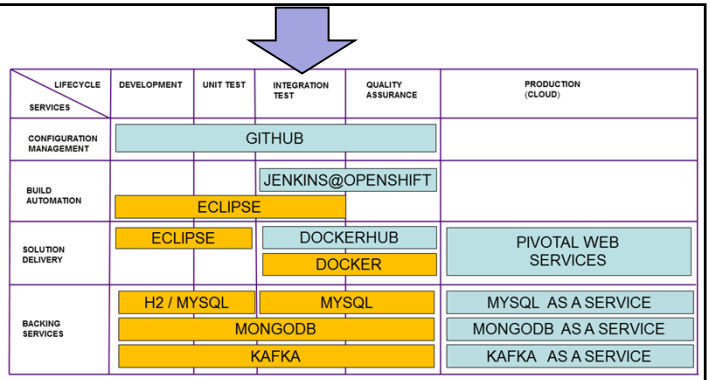
- A **Docker image** is read-only file system image of an operating system and an application.
- An image is self-contained and will run on any Docker installation.
- To create an image containing a Spring Boot-based application, we could start from an Ubuntu image, install the JDK, and then install the executable JAR.
- A Docker image has a layered structure, a feature that reduces the amount of time needed to build and deploy a Docker image.



- An image consists of a sequence as well as parent referenced layers, each corresponding to the command that changes the file system.
- The layered structure makes building an image extremely fast and efficient.
  1. Docker enables **the sharing of layers between images**, and it does not need to move an entire image over the network, since it copies only the layers that do not already exist on the destination machine.
  2. Docker **caches layers when building an image**. When re-executing a command, Docker automatically skips the execution of the command, and instead reuses the output layer previously built.

Docker container

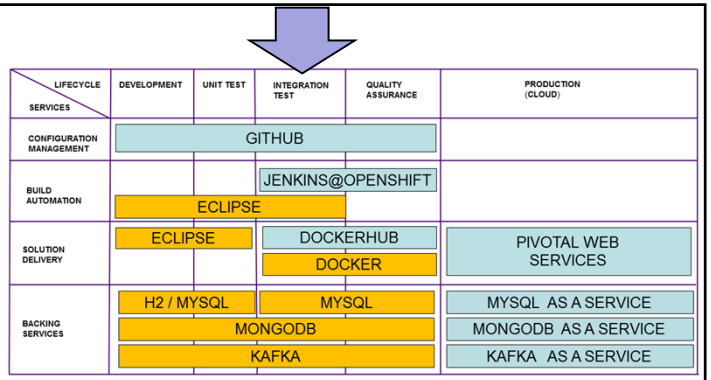
A **Docker container** is a running image consisting of one or more sandboxed processes.

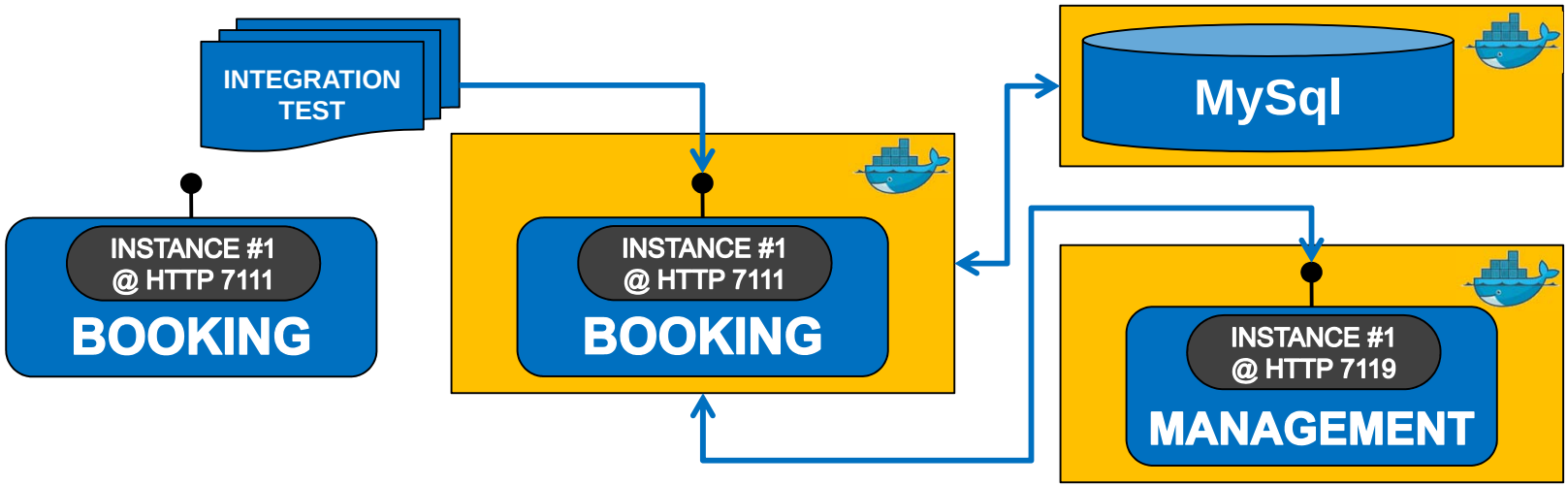


- Docker isolates a container's process and its networking portion
- Docker enables a service to be accessed from outside its container by setting up a port mapping feature through which we associate a host port to a container port.
- Eventhough an image contains an entire OS, a **Docker container only consists of the application's processes**
- For instance, a Docker **container that runs a Spring Boot application only requires running Java, making it extremely efficient**
- As a result, a Docker container has a minimal runtime overhead and its startup time is the startup time of your application.

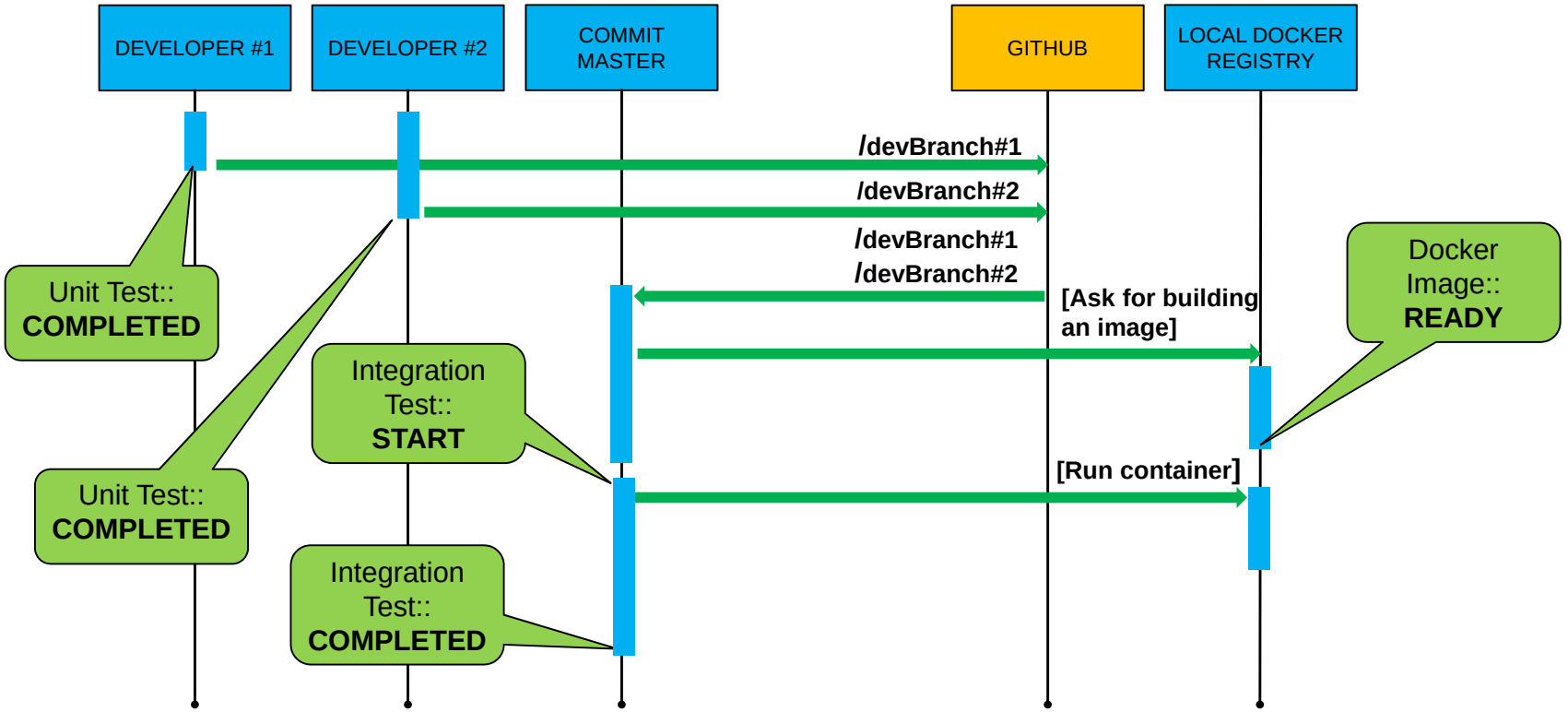
INTEGRATION TEST

- Spring Boot packaging features
- Running this **microservice inside a Docker container** needs to build a Docker image containing the service library and Java
- The **integration test** then will be done using the docker containers of the microservices that establish the digital platform





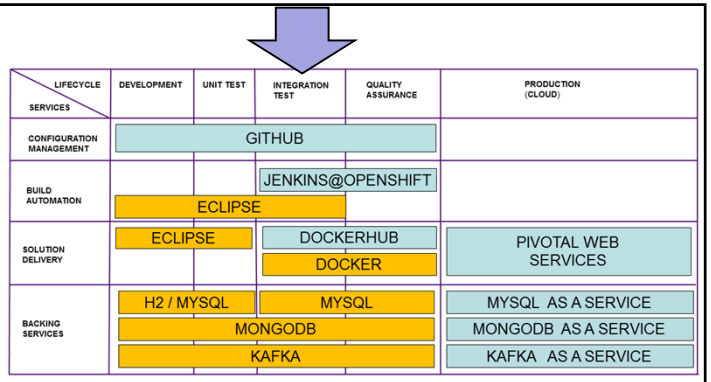
INTEGRATION TEST



INTEGRATION TEST


```
FROM java:8
MAINTAINER l.bennardis@email.it
VOLUME /tmp
ADD /00dloc-bookabattery-service-release.jar app.jar
RUN bash -c 'touch /app.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

```
<plugin>
      <groupId>com.spotify</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <version>0.2.3</version>

      <configuration>
            <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
            <dockerDirectory>src/main/docker</dockerDirectory>
            <resources>
                  <resource>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.jar</include>
                  </resource>
            </resources>
      </configuration>
</plugin>
```



- **Docker integration**
  - Dockerfile with the definition of the Docker image
  - Docker Maven Plug-in specifications

INTEGRATION TEST

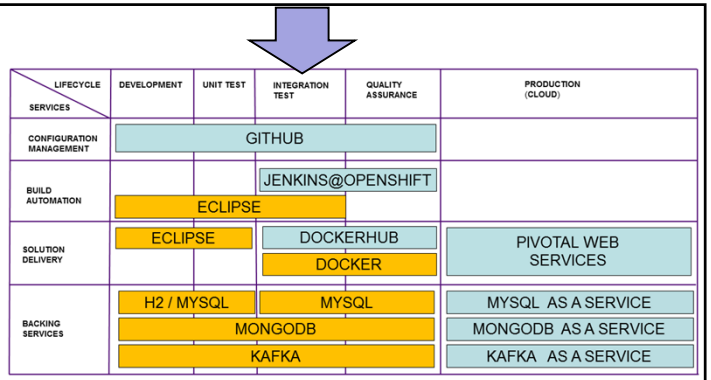
Goals: package docker:build

[INFO] Building image **luigibennardis/00dloc-bookabattery-service**

Step 0 : **FROM** java:8
---> 9a7221d5adb5

Step 1 : **VOLUME** /tmp
---> Using cache
---> 52e9b30dca40

Step 2 : **ADD** /00dloc-bookabattery-service-release.jar app.jar
---> ac4489f8ba5b
Removing intermediate container ec4d480719c5
Step 3 : **RUN** bash -c 'touch /app.jar'
---> Running in 61262e1379a2
---> 78c5b5d831ce
Removing intermediate container 61262e1379a2
Step 4 : **ENTRYPOINT** java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
---> Running in 50e9016f1701
---> 4f2e75b6d815
Removing intermediate container 50e9016f1701
Successfully built 4f2e75b6d815
[INFO] **Built luigibennardis/00dloc-bookabattery-service**
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 01:13 min



- **Maven docker: build evidences**
  - «package docker:build» is the Maven goal for building a Docker image
  - «Layering» features corrisponding to each command

INTEGRATION TEST

```
C:\Users\3volv-04>docker images
REPOSITORY                                  TAG      IMAGE ID       CREATED          VIRTUAL SIZE
luigibennardis/00dloc-bookabattery-service  latest   4f2e75b6d815   13 minutes ago   758.4 MB
mysql                                       5.6      37c900662eb2   10 days ago      328.9 MB
java                                        8        9a7221d5adb5   5 months ago     642.4 MB
```

RUNNING THE DATABASE CONTAINER:

docker run --**name** mysqldb
-e MYSQL_USER=bab_USER
-e MYSQL_PASSWORD=bab_USER
-e MYSQL_DATABASE=batteryService
-e MYSQL_ROOT_PASSWORD=root
-d
mysql:5.6

RUNNING THE MICROSERVICE CONTAINER:

docker run --**name** appdemo
--**link** mysqldb:mysql
-**p** 7111:7111
-t
luigibennardis/00dloc-bookabattery-service

OBTAINING THE LIST OF IMAGES:

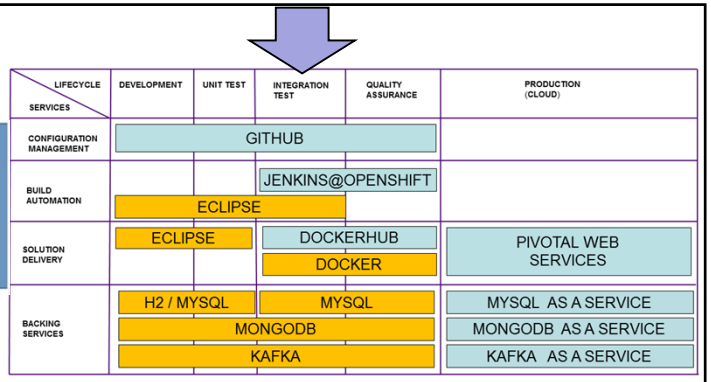
docker images

OBTAINING THE LIST OF CONTAINERS:

docker ps –a

STOPPING AND REMOVING CONTAINERS:

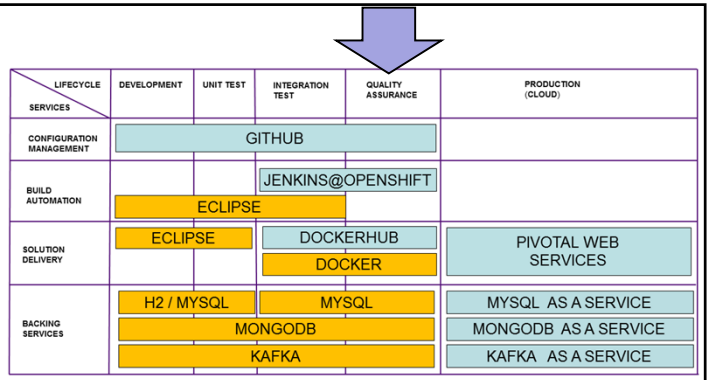
Docker stop/rm <container-id>



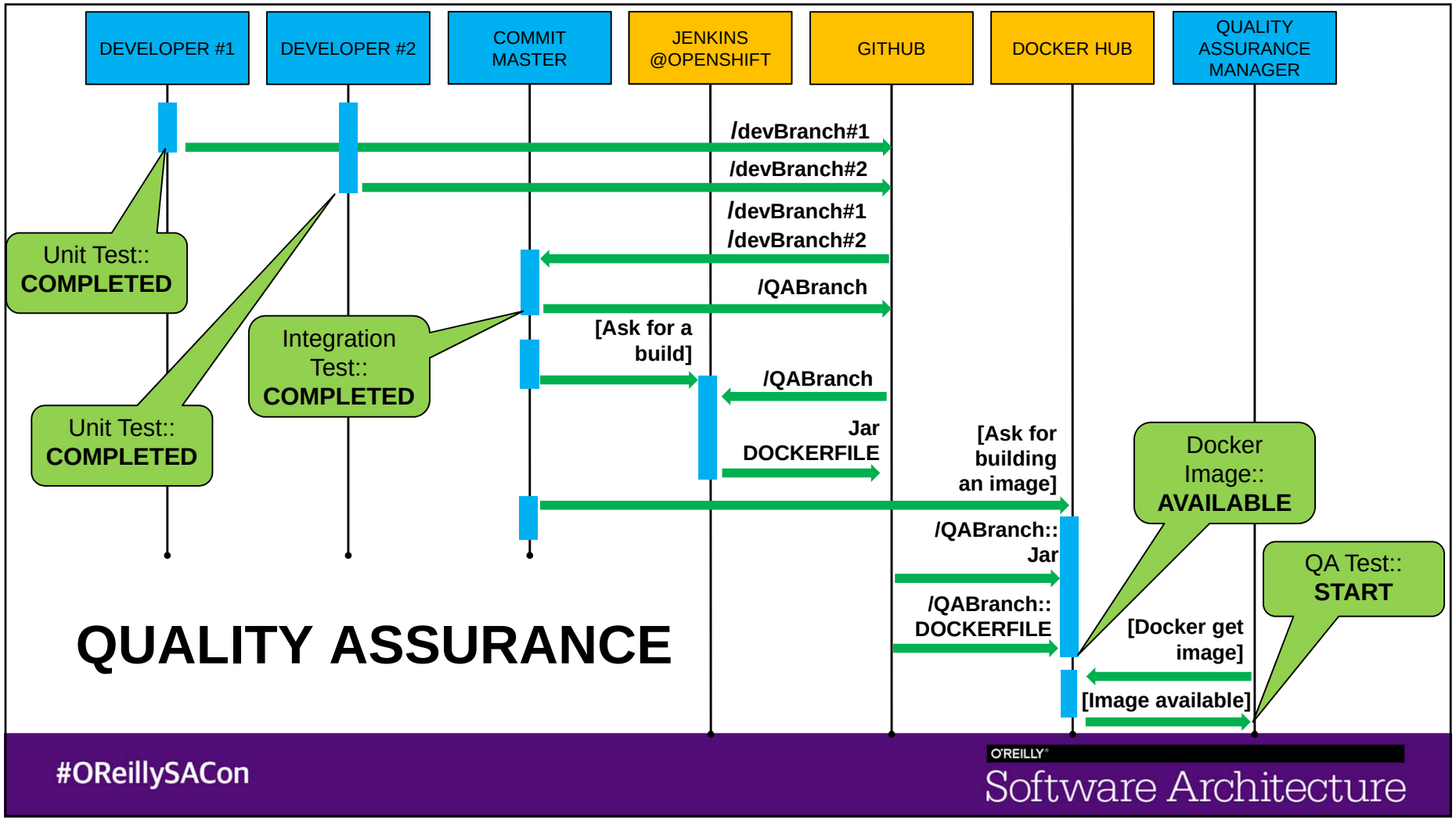
- **Docker instance details**
  - List of available images available within the docker registry
  - Docker run commands details
  - MySql instance data will be load from scratch by means of the sql statements provided in the application's package
  - For specific test purposes it will be possible to build and run database images with test-distinguishing data

DEMO

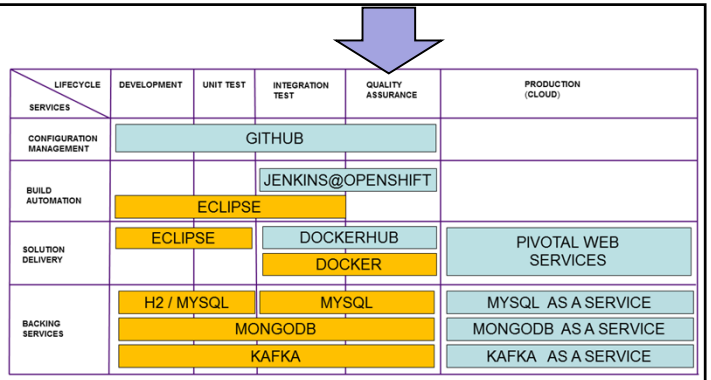
QUALITY ASSURANCE



- **Docker Hub** is the cloud-based registry service of Docker images. It provides:
  - Official and private image repositories: finding, managing, pushing and pulling images from the Docker command line
  - Automated builds: creating new images, either manually triggered or as a result of changes detected in the source code repository.
  - Webhooks features: triggering actions after a successful push to a repository.
  - Team collaboration and workflow automation.
- **OpenShift** is Red Hat's Platform-as-a-Service cloud environment, where for this project a **Jenkins** instance will provide the features of Continuous Integration
- **Docker Hub and Jenkins** are both integrated with the **GitHub** repository


DEVELOPER #1
DEVELOPER #2
COMMIT MASTER
JENKINS @OPENSHIFT
GITHUB
DOCKER HUB
QUALITY ASSURANCE MANAGER
/devBranch#1
/devBranch#2
/devBranch#1
/devBranch#2
/QABranch
[Ask for a build]
/QABranch
Jar DOCKERFILE
[Ask for building an image]
/QABranch:: Jar
/QABranch:: DOCKERFILE
[Docker get image]
[Image available]
Unit Test:: COMPLETED
Integration Test:: COMPLETED
Unit Test:: COMPLETED
Docker Image:: AVAILABLE
QA Test:: START
QUALITY ASSURANCE
#OReillySACon
O'REILLY
Software Architecture

QUALITY ASSURANCE

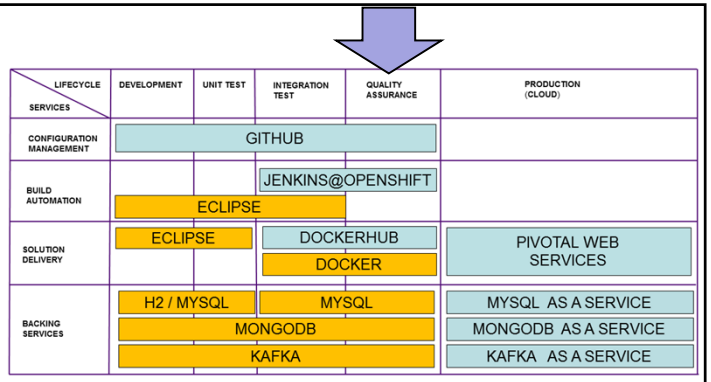



```
FROM java:8
MAINTAINER l.bennardis@email.it
VOLUME /tmp
RUN mkdir /temp

RUN git clone -b qualityassurance https://github.com/lbennardis/bookabatteryservice.git /temp

RUN bash -c 'touch /temp/it/luigibennardis/00D-bookABattery_SERVICE/@version@/@jar_name@-@version@.jar'

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/temp/it/luigibennardis/00D-bookABattery_SERVICE/@version@/@jar_name@-@version@.jar"]
```

- **Dockerfile template**
  - Directive of a git clone command
  - Entrypoint of the Docker image

QUALITY ASSURANCE

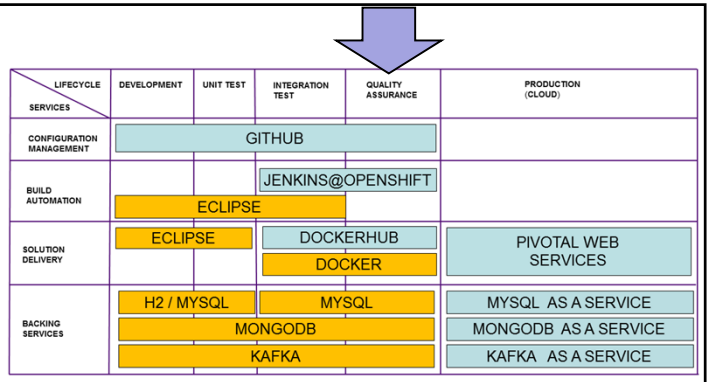
```
<groupId>com.google.code.maven-replacer-plugin</groupId>
<artifactId>replacer</artifactId>
<version>1.5.3</version>

<!-- DELETED FRAGMENTS FOR DEMO PURPOSE  -->

<configuration>
<file>src/main/docker/dockerfileTemplate</file>

     <!--PATH_TO_REPO
     /var/lib/openshift/566ae57d0c1e6629760000cd/
     app-root/data/buildjenkins/qualityassurance -->

     <outputFile>${PATH_TO_REPO}/Dockerfile</outputFile>
          <replacements>
               <replacement>
                    <token>@jar_name@</token>
                         <value>00D-bookABattery_SERVICE</value>
               </replacement>

               <replacement>
                    <token>@version@</token>
                         <value>${project.version}</value>
               </replacement>
<!-- DELETED FRAGMENTS FOR DEMO PURPOSE  -->
```



- **Maven Replacer Plug**
  - Updates the @...@ tags in the dockerfile template file

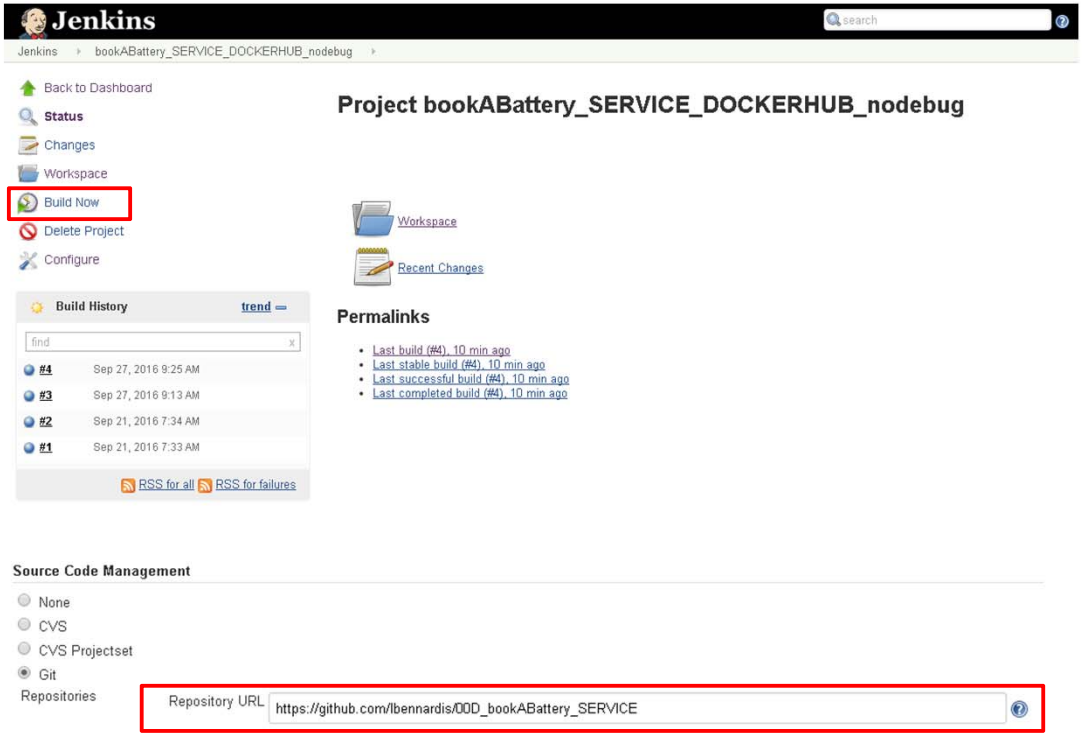
QUALITY ASSURANCE

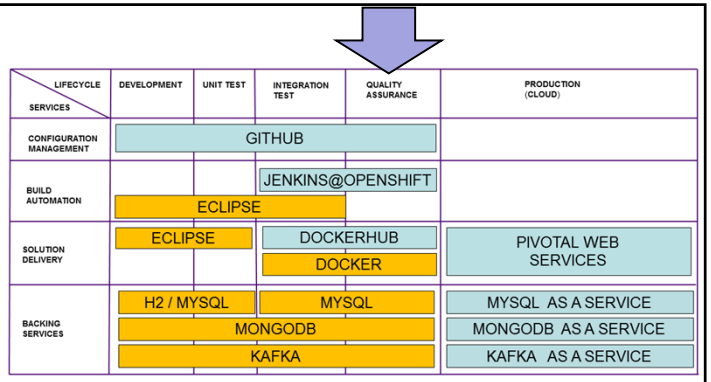
```
<groupId>com.github.github</groupId>
<artifactId>site-maven-plugin</artifactId>
<version>0.12</version>

<configuration>
     <message>Maven artifacts for ${project.version}</message>
     <noJekyll>true</noJekyll>
     <outputDirectory>${PATH_TO_REPO}</outputDirectory>

     <!-- REMOTE BRANCH NAME-->
     <branch>refs/heads/qualityassurance</branch>

     <includes><include>**/*</include></includes>
     <merge>true</merge>

     <!-- GITHUB REPOSITORY NAME-->
     <repositoryName>bookabatteryservice</repositoryName>

      <!-- USERNAME-->
     <repositoryOwner>lbennardis</repositoryOwner>

</configuration>
```



- **Maven GitHub Plug-in**
  - Promotes built artifacts within GITHUB, according to the Maven repository model.

QUALITY ASSURANCE

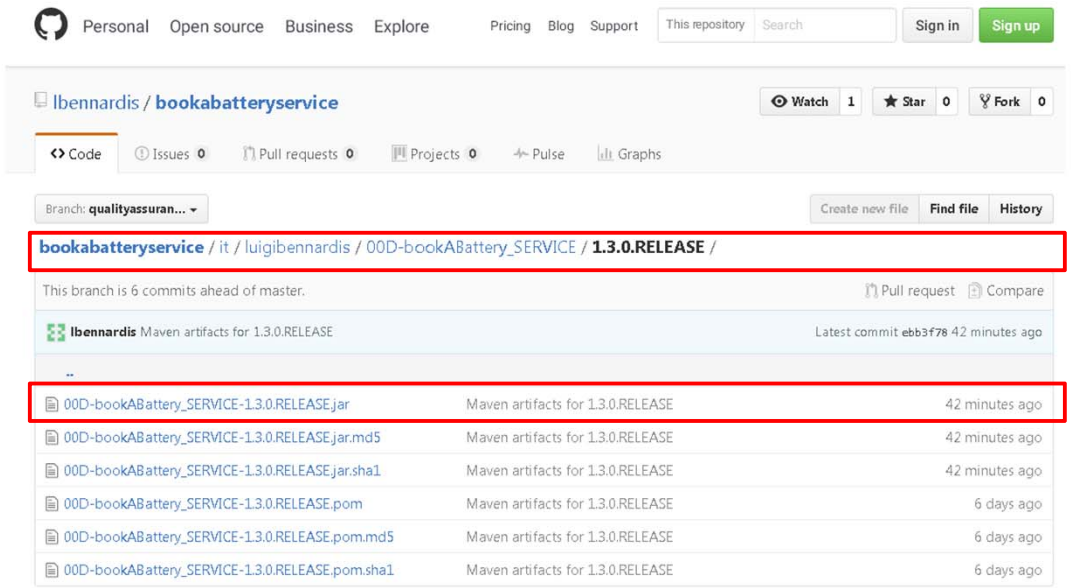


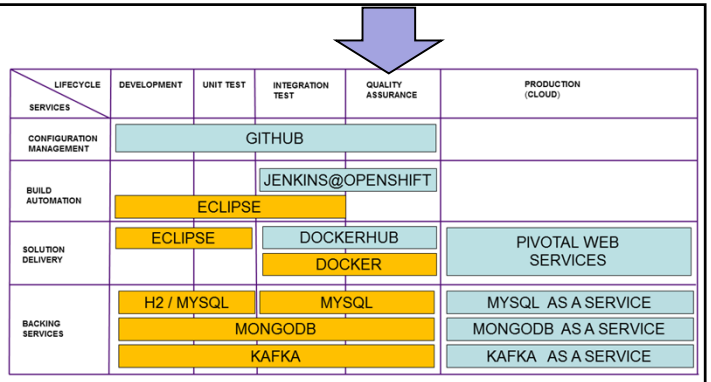


- **Jenkins' build project**
  - Project dashboard
    - Features manual build capabilities
    - Build history
  - GITHUB configuration details

DEMO

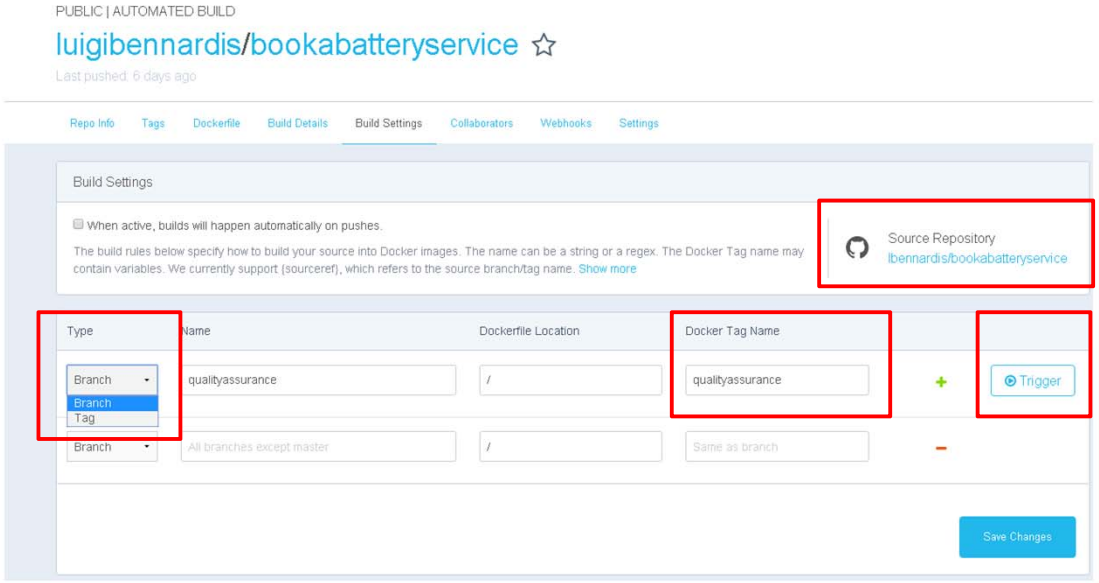
QUALITY ASSURANCE

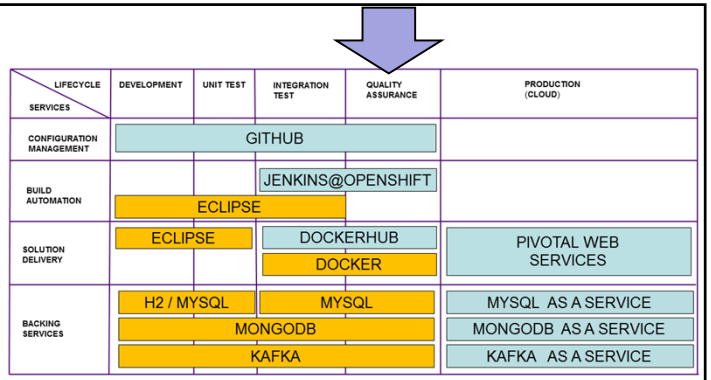




- **GitHub Maven repository**
  - Executable artifacts stored inside GITHUB according to the Maven repository specifications

QUALITY ASSURANCE





- **DockerHub configuration details**
  - Source GITHUB repository
  - Branch or Tag references
  - Tagname of the Docker image

QUALITY ASSURANCE

PUBLIC | AUTOMATED BUILD

luigibennardis/bookabatteryservice ☆

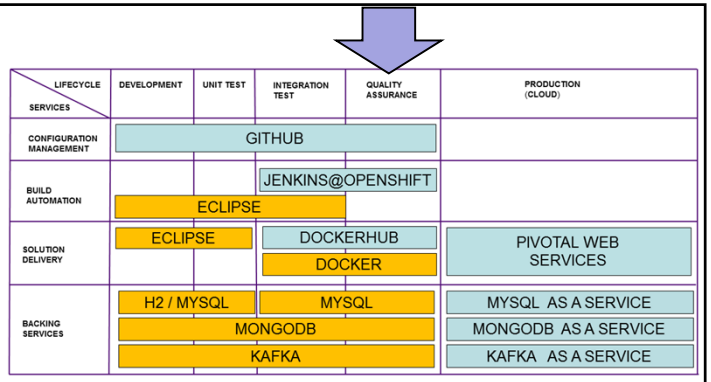
Last pushed: 15 minutes ago

Repo Info | Tags | Dockerfile | Build Details | Build Settings | Collaborators | Webhooks | Settings

Status	Tag	Created	Last Updated
Queued	qualityassurance	a few seconds ago	a few seconds ago

Status	Tag	Created	Last Updated
Building	qualityassurance	a few seconds ago	a few seconds ago

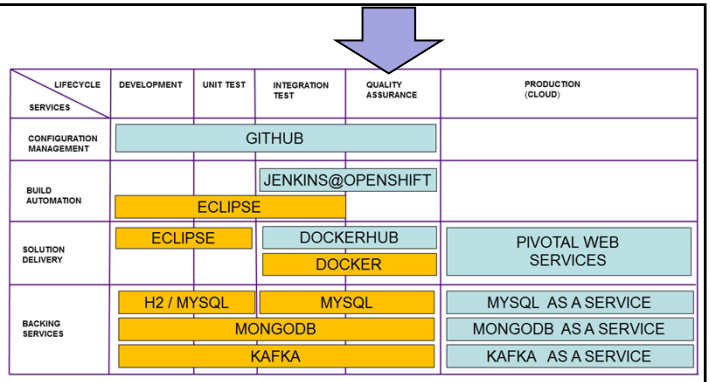
Status	Tag	Created	Last Updated
Success	qualityassurance	3 minutes ago	2 minutes ago



- **DockerHub build status**
  - Moves from «queue» status to «completed» status

QUALITY ASSURANCE

- **DockerHub build evidences**
  - Dockerfile

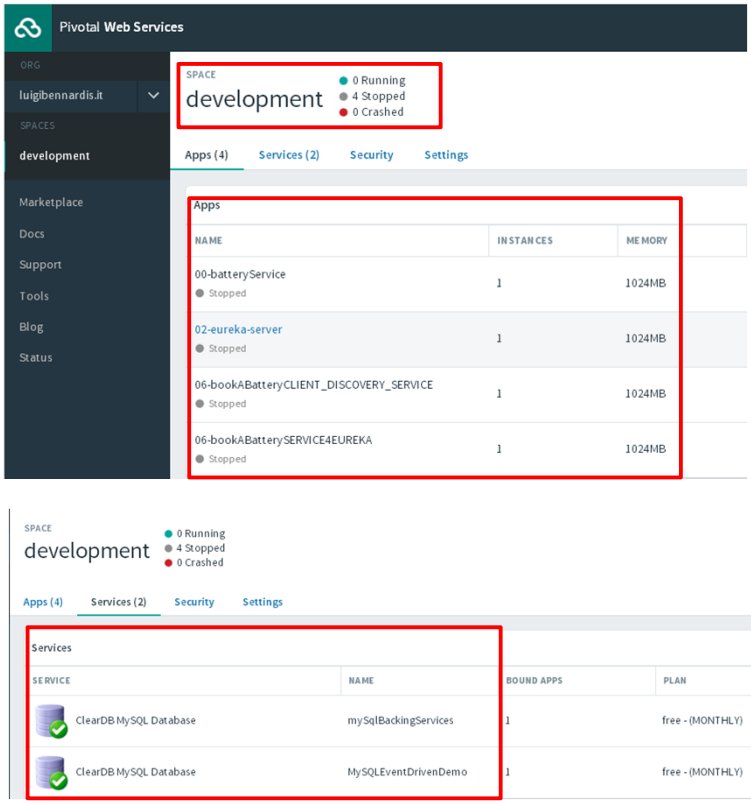


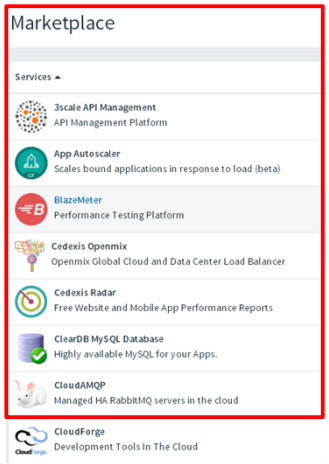
Dockerfile

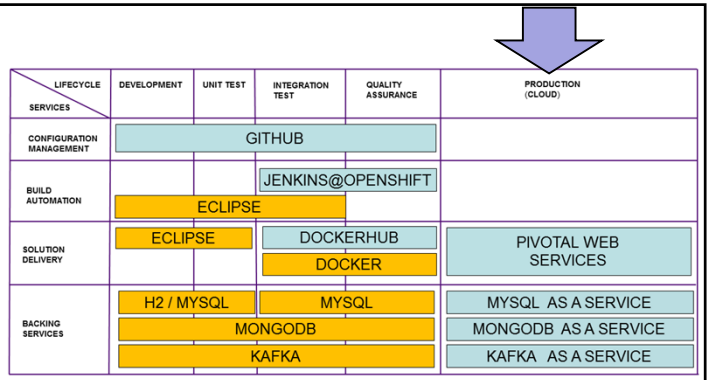
```
FROM java:8
MAINTAINER l.bennardis@email.it
VOLUME /tmp
RUN mkdir /temp
RUN git clone -b qualityassurance https://github.com/lbennardis/bookabatteryservice.git /temp
RUN bash -c 'touch /temp/it/luigibennardis/00D-bookABattery_SERVICE/1.3.0.RELEASE/00D-bookABattery_SERVICE-1.3.0.RELEASE.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/temp/it/luigibennardis/00D-bookABattery_SERVICE/1.3.0.RELEASE/00D-bc
```

DEMO

PRODUCTION





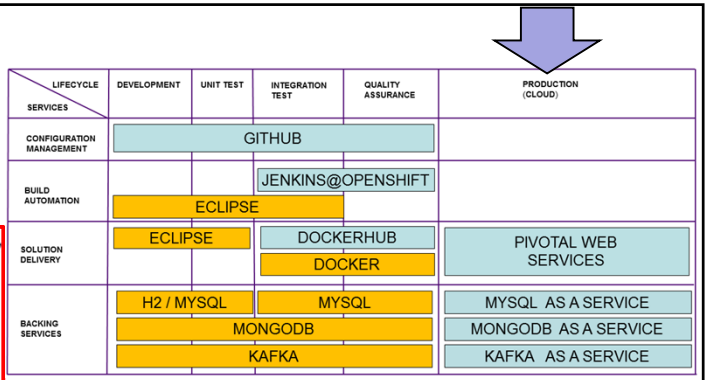


- **Pivotal web service console**
  - Developmnet space
  - Deployed application
  - Available services
  - Marketplace

DEMO

PRODUCTION


```
"VCAP_SERVICES": {
  "cleardb": [
    {
      "credentials": {
        "jdbcUrl": "jdbc:mysql://us-cdbr-iron-east-03.cleardb.net/ad_9eccf35d79407b7?user=b6feefbfb1e277&password=bc81c4b6",
        "uri": "mysql://b6feefbfb1e277:bc81c4b6@us-cdbr-iron-east-03.cleardb.net:3306/ad_9eccf35d79407b7?reconnect=true",
        "name": "ad_9eccf35d79407b7",
        "hostname": "us-cdbr-iron-east-03.cleardb.net",
        "port": "3306",
        "username": "learefbfb1e277",
        "password": "bclearb6"
      },
      "syslog_drain_url": null,
      "volume_mounts": [],
      "label": "cleardb",
      "provider": null,
      "plan": "spark",
      "name": "mySqlBackingServices",
      "tags": [
```

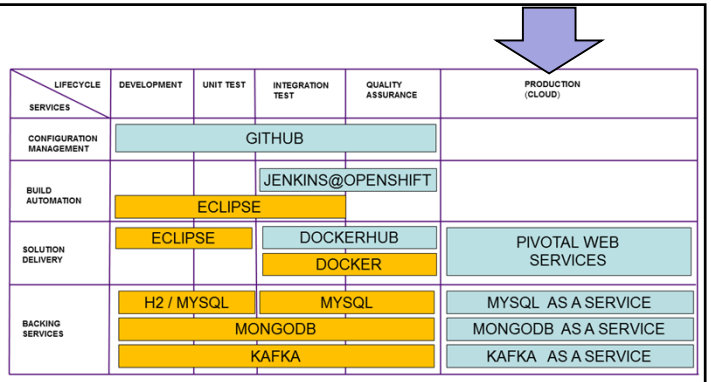
- **PWS environment variable**
  - ClearDB backing service details

PRODUCTION

```
@Bean
@Profile("cloudfoundry")

DataSource dataSource
 (@Value("${cloud.services.mySqlBackingServices.connection.jdbcurl}")
      String    jdbcUrl) {

try {    return new SimpleDriverDataSource(
         com.mysql.jdbc.Driver.class.newInstance() , jdbcUrl);}
```

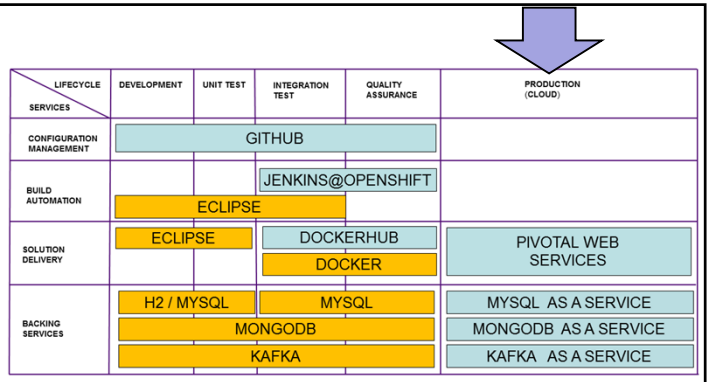
```
 catch (Exception e) {
         throw new RuntimeException(e) ; }}
```

```
 @Bean
CommandLineRunner checkDatasource(
          DataSourceProperties dataSourceProps,

 @Value("${cloud.services.mySqlBackingServices.connection.jdbcurl:}")
          String jdbcUrl)
{
    return args -> System.out.println(
      "\n\n cloud.services.mySqlBackingServices.connection.jdbcurl JDBC URL="
      + jdbcUrl +    " \n\n the DATASOURCE URL=" + dataSourceProps.getUrl() +
      ".\n\n");    }}
```



- **PWS Spring profile**
  - Define the coded way for datasource binding
  - The «cloudfoundry» profile will be provided at deploy time
  - Useful bean for database connection log details

PRODUCTION



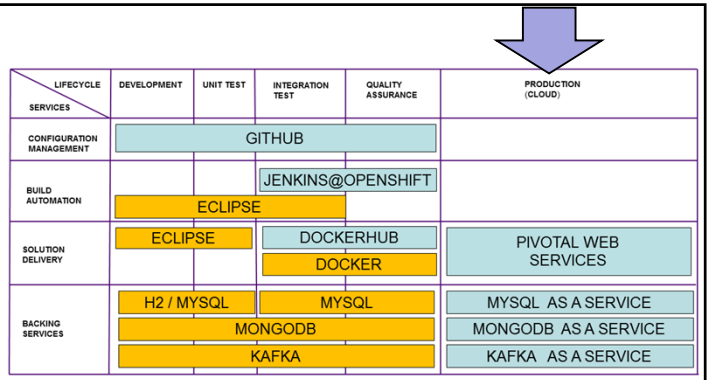

```
[APP/0] [OUT] the DS URL=jdbc:mysql://localhost/bookabattery_db_pws

[APP/0] [OUT] cloud.services.mySqlBackingServices.connection.jdbcurl

[APP/0] [OUT]
JDBC URL=jdbc:mysql://us-cdbr-iron-east-03.cleardb.net/ad_9eccf35d79407b7?user=b6feefbfb1e277&password=bcSED4b6
```

```
HOST NAME= us-cdbr-iron-east-03.cleardb.net
DATABASE NAME= as_9eccf35d79407b7
USER NAME= bserefbfb1e277
PASSWORD= AS1dD4b6
```

- **Application start-up evidence**
  - JDBC URL
  - ClearDb instace parameters

PRODUCTION

```
--- manifest.yml
applications:
  - name: 00-batteryService

    memory: 1024M
    instances: 1

    buildpack: java_buildpack
    host: 00-batteryService

    domain: cfapps.io
    path: target/00-bookABattery_SERVICE-1.0.0.BUILD-SNAPSHOT.jar

    services:
     - mySqlBackingServices

     datasource:
    jpa:
    hibernate.ddl-auto: none
    show_sql: false
env:
    SPRING_PROFILES_ACTIVE: cloudfoundry
    DEBUG: "true"
    debug: "true"
```

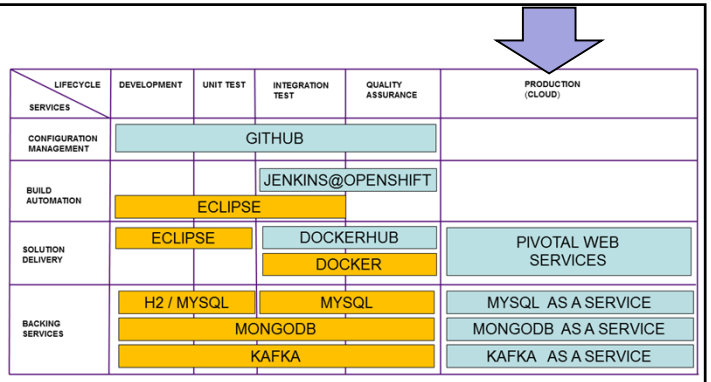


- **PWS deployment descriptor**
  - Instance parameters
  - Artifact location
  - Backing services
  - Environment's variable

PRODUCTION deploy pws

```
<plugin>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cf-maven-plugin</artifactId>
    <version>1.1.2</version>

        <configuration>
            <server>cloudfoundry-pws-instance</server>

            <target>http://api.run.pivotal.io</target>
            <org>mycloudfoundry-org</org>
            <space>development</space>
            <appname>00-batteryService</appname>
            <url>00-batteryService.cfapps.io</url>

            <memory>1024</memory>
            <instances>1</instances>

             <env>
                 <ENV-VAR-NAME>SPRING_PROFILES_ACTIVE</ENV-VAR-NAME>
             </env>
             <services>
                 <service>
                      <name>mySqlBackingServices</name>
                 </service>
             </services>
        </configuration>
    </plugin>
```

$ mvn clean package cf:push



- **Cloud Foundry Maven plug-in**
  - Instance parameters
  - Artifact location
  - Backing services
  - Environment's variable
  - Maven command line

DEMO

Microservices: data consistency across services

- The «database per service architecture» introduces a distributed data transaction challenge.
- In this architecture the need of business transactions to span over different services could not be satisfied by distributing transaction.
- Infact each service kepts his database private and in a poliglot persistance approach some database do not support distributing transaction.
- Besides we have also stated in the requirements that the persistance model should be made by non-blocking operations

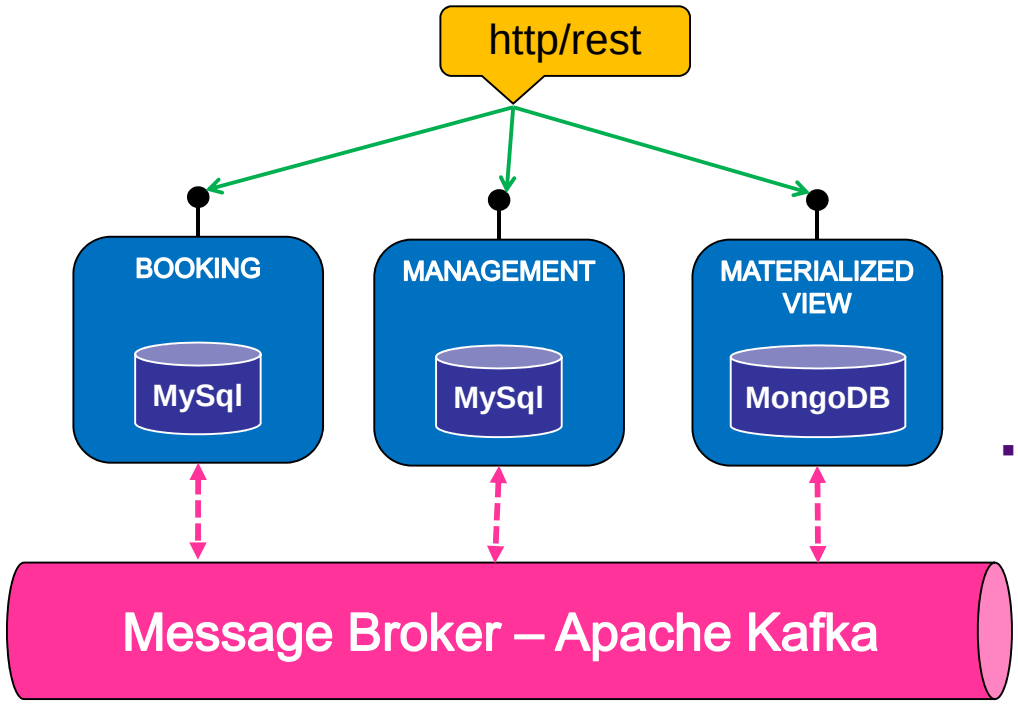
Microservices: event driven architecture

- *The **"Event-driven architecture"*** is the mechanism that will ensure data consistency across services without using distributed transactions
- This approach is based on a dedicated **message broker** responsible for event distribution
- With this mechanism, each service publishes an event whenever it updates its data. Other services subscribe to these events and whenever an event is received, the subscriber service automatically updates its data

- The solution has the following drawbacks:
  - It is a **more complex programming** model
  - It has an **infrastructure management overhead**
  - It needs the implementation of **compensation transactions in order to recover from** failures
  - It needs also the **development of queries** that retrieve data from multiple services

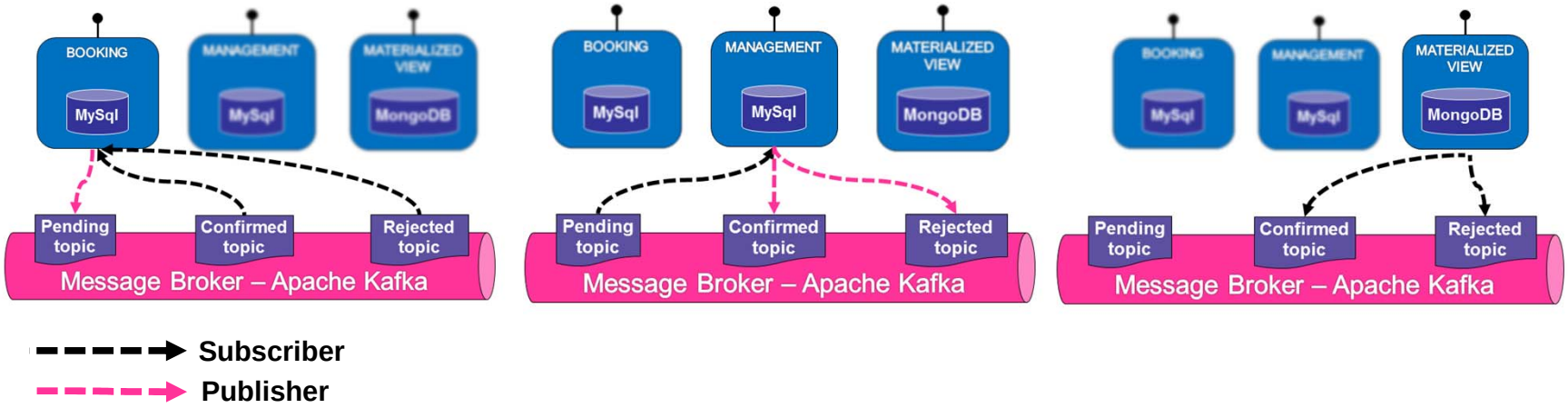
- The failure issues could be mitigated by:
  - choosing a **high-reliability message broker** infrastructure
  - the self-contained scaling-out feature allows us to obtain a more reliable message broker

Event driven architecture: «base model»

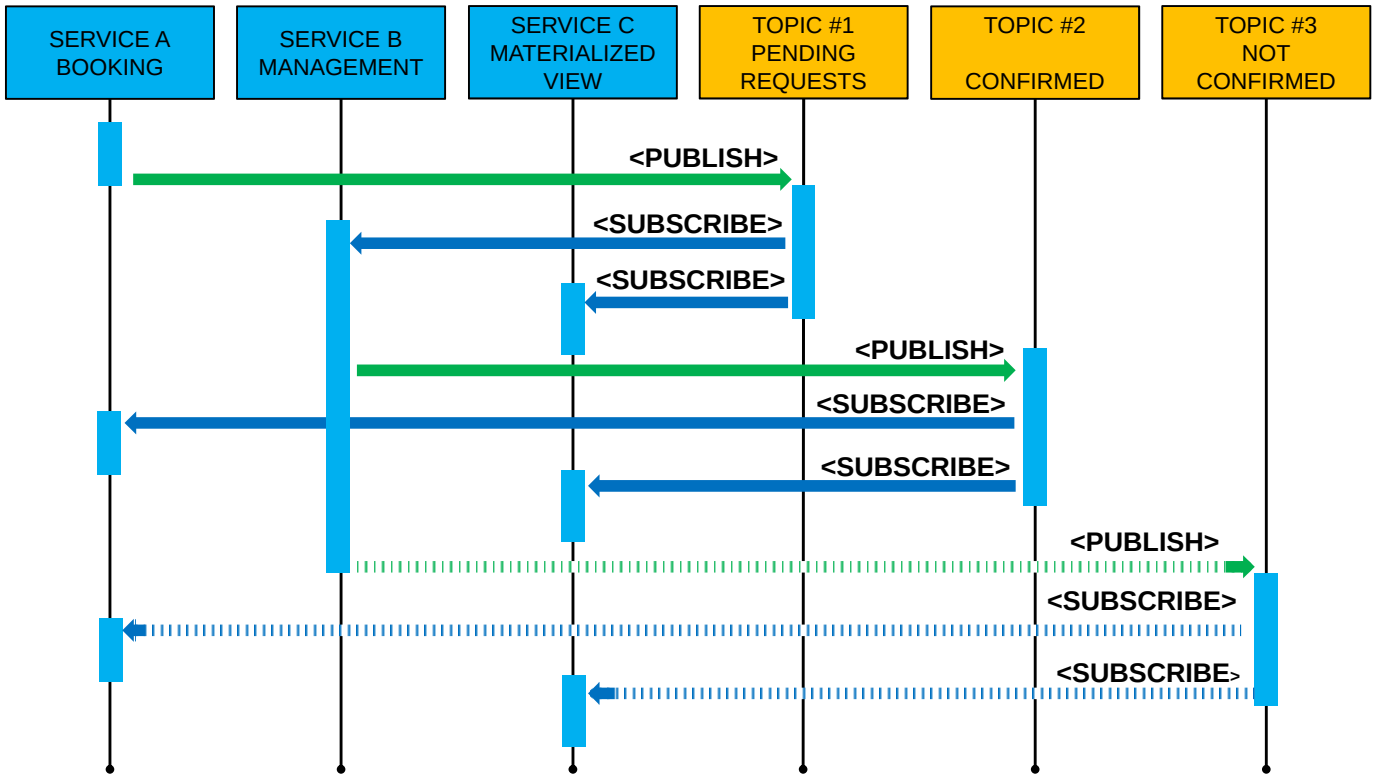


- The **«Base model»** of an event-driven architecture
  - Each business transation corresponds to a series of steps
  - For each step, one microservice updates a business entity and publishes an event, thereby triggering the next step
  - Materialized view uses these events to join together and store different sets of data owned by different services
- Example of **«poliglot persistence»**
  - ***Booking*** and ***Management*** services with MySql data storage
  - ***Materialized view*** service with MongoDB data storage

Base model: topics subscription and publishing



Base model: sequence diagram

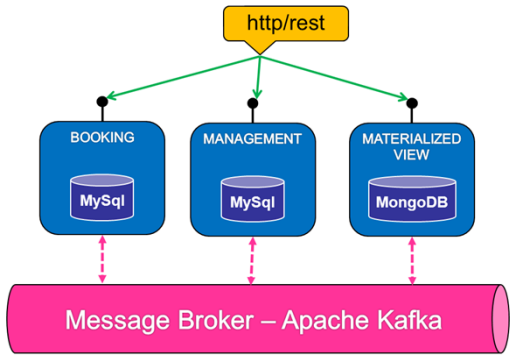


Topics subscription: declaration and interface

spring-cloud-stream (managed:1.0.2.RELEASE)

```
#output_pending_topic
spring.cloud.stream.bindings.output_pending_topic.
        content-type=application/x-java-object;
        type=it.luigibennardis.microservice.domain.Booking

#input_confirm_topic
spring.cloud.stream.bindings.input_confirm_topic.
        content-type=application/x-java-object;
        type=it.luigibennardis.microservice.model.TransactionDetails
```

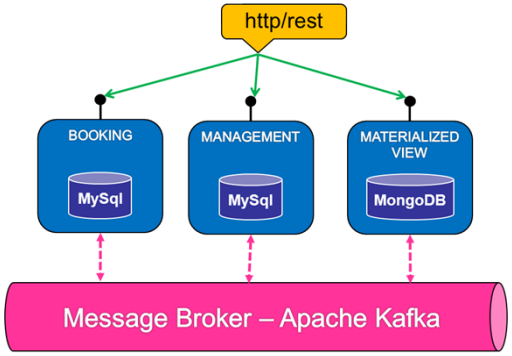
```
public interface ISinkConfirmTopic {

      String INPUT_CONFIRM_TOPIC = "confirmBookingTopic";

      @Input(ISinkConfirmTopic.INPUT_CONFIRM_TOPIC)
      SubscribableChannel confirmBookingTopic();

      String INPUT_NOT_CONFIRM_TOPIC = "notConfirmBookingTopic";

      @Input(ISinkConfirmTopic.INPUT_NOT_CONFIRM_TOPIC)
      SubscribableChannel notConfirmBookingTopic();
```



- **Topics subscription: implementation details**
  - Spring Cloud Stream dependency
  - Declarative topics binding definition with java object typed content
  - Interface declaration of subscribing topics

Topics subscription: implementation of the interface

```
@Component
@EnableBinding(ISinkConfirmTopic.class)

public class ReadReturnTopic {

      @Autowired
      private ApplicationContext context;

      @ServiceActivator(inputChannel = ISinkConfirmTopic.INPUT_CONFIRM_TOPIC)
      public void updateConfirmPending(GenericMessage<TransactionDetails> message) {

      BookingService service = context.getBean(BookingService.class);

      service.updatePendingBooking(message.getPayload().getIdReservation()); }



      @ServiceActivator(inputChannel = ISinkConfirmTopic.INPUT_NOT_CONFIRM_TOPIC)
      public void updateNotConfirmPending(GenericMessage<TransactionDetails> message) {

      BookingService service = context.getBean(BookingService.class);

      service.updateNotConfirmedBooking(message.getPayload().getIdReservation()); }}
```

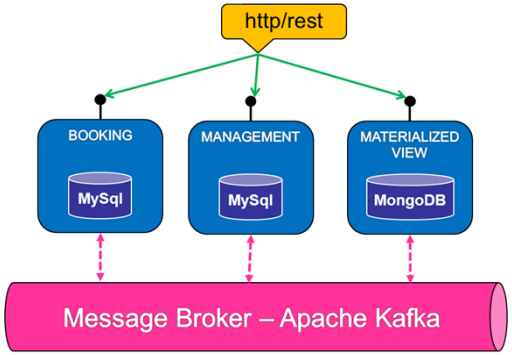


- **Topics subscription: implementation details**
  - Implementation of the interface
  - Methods triggered by the subscribed topics
  - Typed message payload

Topics publishing: interface declaration and implementation

```
public interface ISinkOutputTopic {

     String OUPUT_PENDING_TOPIC = "pendingBookingTopic";

     @Output(ISinkOutputTopic.OUPUT_PENDING_TOPIC)
     SubscribableChannel outputPendingTopic(); }
```

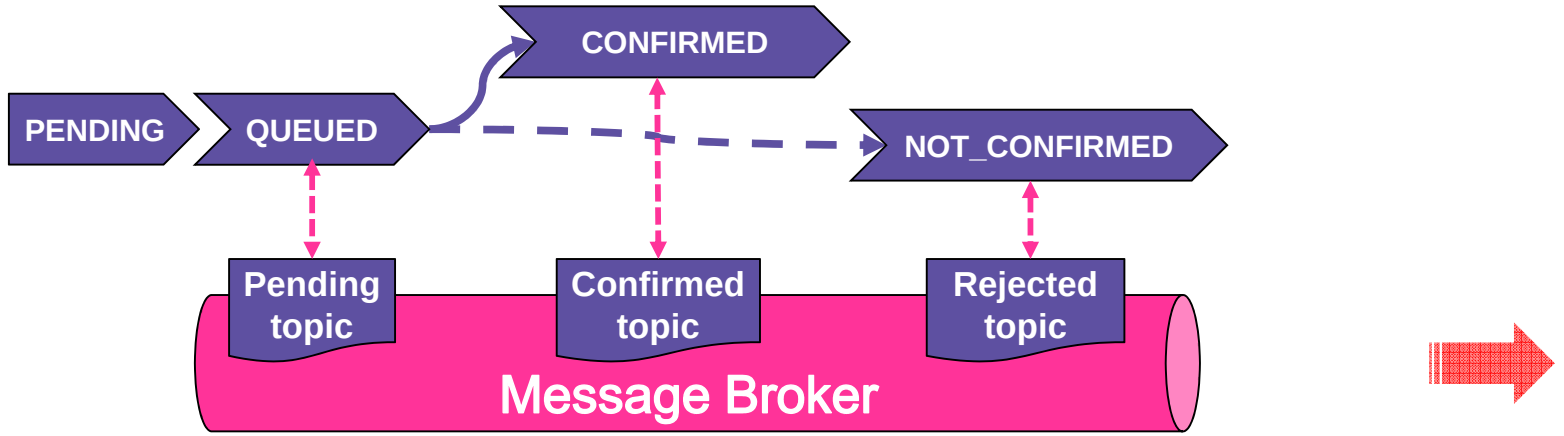
```
@Component
@EnableBinding(ISinkOutputTopic.class)
public class WritePendingTopic {

@Autowired
private ISinkOutputTopic kafkaChannel;

@Autowired
public WritePendingTopic() { }

     @Autowired
     public WritePendingTopic(ISinkOutputTopic kafkaChannel) {
          this.kafkaChannel = kafkaChannel;
     }

     public void writePendingTopic(List<Booking> dtInfo) {
          if (!dtInfo.isEmpty()){
          kafkaChannel.outputPendingTopic().send(MessageBuilder.withPayload(dtInfo).build()); }}
```



- **Topics publishing: implementation details**
  - Interface definition
  - Class which implements the interface
  - Method which implements the publishing logic

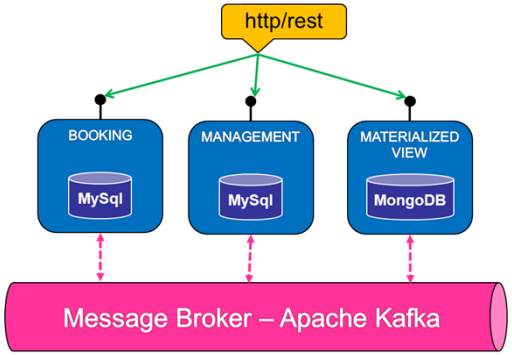
Base model: record workflow

id	batterycode	stationid	city	latitude	longitude	bookingstate	tscreation	tsupdate
e9784d49-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.98450000	16.67560000	PENDING	2016-09-25 17:57:44	2016-09-25 17:57:44
ea1caa41-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.98460000	16.56640000	PENDING	2016-09-25 17:57:45	2016-09-25 17:57:45
eac86911-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.95450000	16.34430000	QUEUED	2016-09-25 17:57:46	2016-09-25 17:57:46
eb76ce8d-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.96450000	16.21230000	QUEUED	2016-09-25 17:57:47	2016-09-25 17:57:47
ec1dd28a-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.89560000	16.56650000	CONFIRMED	2016-09-25 17:57:48	2016-09-25 17:57:48
ecc4a006-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.76560000	16.76540000	CONFIRMED	2016-09-25 17:57:49	2016-09-25 17:57:49
ed679d44-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.54340000	16.23340000	NOT_CONFIRMED	2016-09-25 17:57:50	2016-09-25 17:57:50
ee0dd780-8338-11e6-af3c-a02bb8273f35	STAZ001BATT011	STAZ001	ROMA	42.23430000	16.55560000	NOT_CONFIRMED	2016-09-25 17:57:52	2016-09-25 17:57:52



Base model: implementation of the scheduler

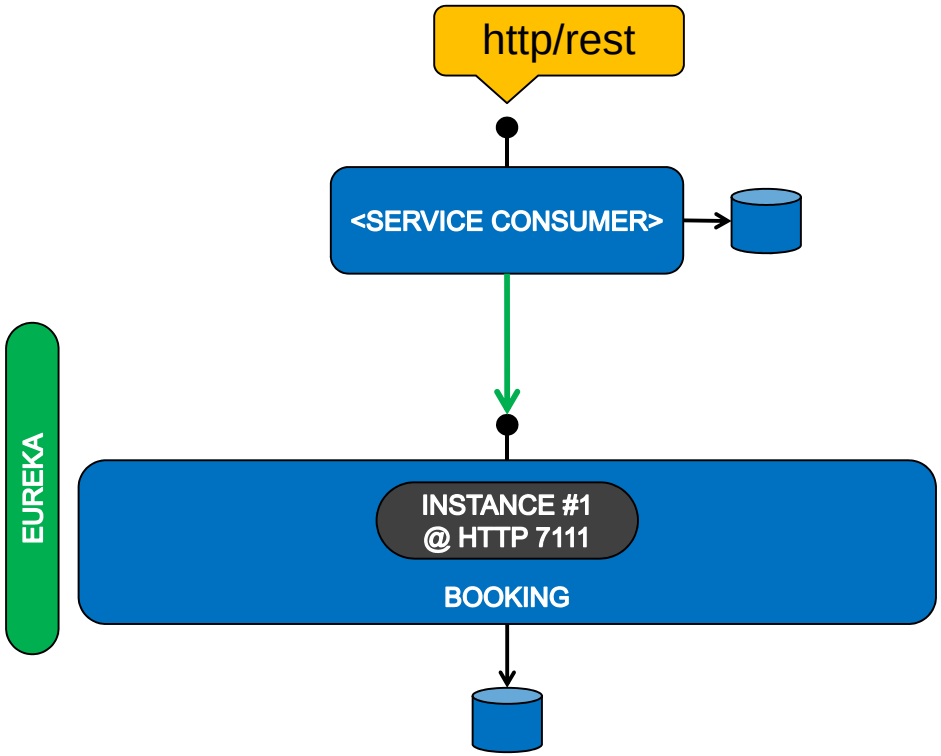
```
@SpringBootApplication
@EnableScheduling
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);}}


@Component
public class DbPollingPending {

@Autowired
private ApplicationContext context;

    @Scheduled(fixedRate = 60000)
    public void pollingPending() {


      BookingService service = context.getBean(BookingService.class);

      WritePendingTopic serviceTopic = context.getBean(WritePendingTopic.class);

      List <Booking> listaItem = service.getPendingBooking();

      serviceTopic.writePendingTopic(listaItem);

      service.updateQueued(listaItem);  }}
```



- **Scheduler: implementation details**
  - *@EnableScheduling* directive
  - *@Scheduled* directive
  - Data access and topic publishing business logic

DEMO

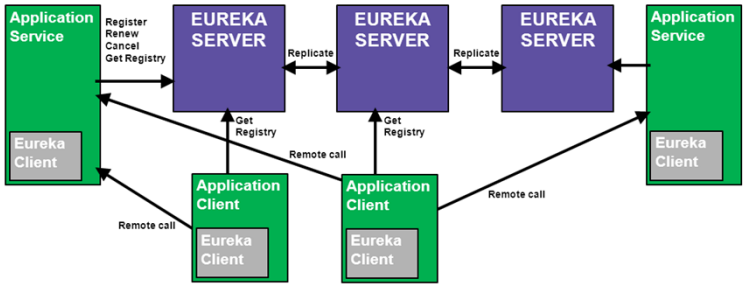
Wiring Microservice: Service Discovery



- In a microservices-based architecture, services are not capable of acting on their own
- In fact, services need to cooperate with each other to obtain the required system functionalities
- **Service Discovery,** with its registry function, is one of the key tenets of a microservices-based architecture
- This pattern allows microservices to find each other dynamically without *point to point* wiring

Netflix Eureka: characteristics

- Eureka is the **Netflix Discovery Service** provider. It consists of two modules:
  - "**Eureka Server**": a REST-based engine for locating services
  - "**Eureka Client"**: a client component that interacts with the Eureka server. It provides a built-in load balancer that achieves basic round-robin load balancing.



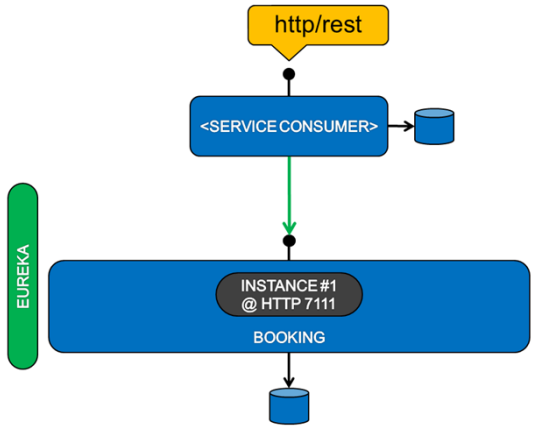
- Eureka can be configured and deployed in a manner that ensures a high degree of availability by replicating the current operating status of each registered service.
- Once registered, services send heartbeats to renew their leases.
- If the **heartbeat** fails to occur over a configurable timetable, the instance will be removed.
- Eureka does not use a back-end storage system, which means that all instance services in the registry are required to send a heartbeat in order to keep their registration up-to-date
- To maintain their synchronization, Client and Server use the provided memory cache
- Eureka works as a registry, keeping and providing information about the registered services, regardless of whether the instances are available or not, or the method used to access them
- The combination of two caches and the hearthbeat function provide a very resilient standalone Eureka server configuration

Discovery Service: Eureka Server

spring-cloud-starter-eureka-server (managed:1.1.5.RELEASE)

```
@EnableEurekaServer
@SpringBootApplication
public class Application {

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

```
server:
  port: 8761 #LISTENING PORT

eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: false #STANDALONE MODE
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
  server:
    enableSelfPreservation: true
```



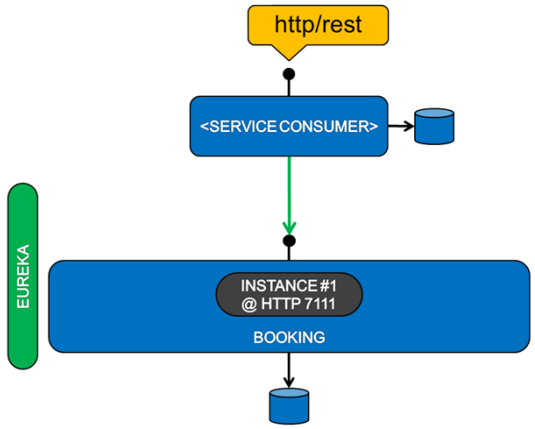
- **Eureka Server implementation details**
  - Spring Cloud Eureka Server Maven dependency
  - @EnableEurekaServer directive
  - Eureka listening port
  - Eureka Standalone mode

Discovery Service: service provider

spring-cloud-starter-eureka

```
@SpringBootApplication
@EnableEurekaClient
@EnableDiscoveryClient
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);}}
```

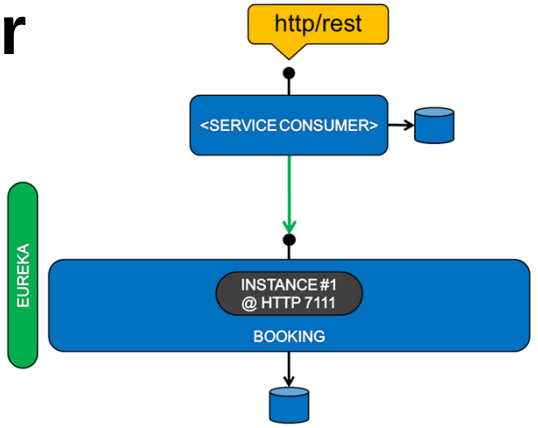
```
eureka:
  password: password
  client:
    serviceUrl:
      defaultZone: http://user:${eureka.password}@localhost:8761/eureka/
    healthcheck:
      enabled: true
    lease:
      duration: 5
  instance:
    leaseRenewalIntervalInSeconds: 1
    leaseExpirationDurationInSeconds: 2
    metadataMap:
      instanceId:
        ${vcap.application.instance_id:${spring.application.name}:${spring.application.instance_id:${server.port}}}
```



- **Eureka service implementation details**
  - *@EnableEurekaClient* directive
  - *@EnableDiscoveryClient* directive
  - Eureka service configuration keys

Discovery Service: service consumer

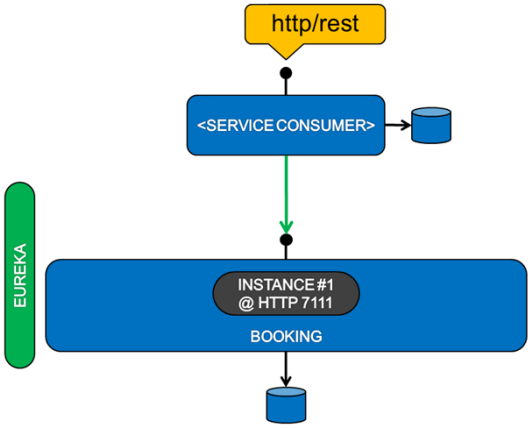
```
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;

@Autowired
DiscoveryClient discoveryClient;

@RequestMapping("/listDiscovery")
public String listDiscovery() {
     List<ServiceInstance>
     instances = this.discoveryClient.getInstances("BOOKABATTERYSERVICE4EUREKA");

     if(instances != null && !instances.isEmpty()) {
        for(int i=0; i<instances.size();i++ ){

           ServiceInstance serviceInstance = instances.get(i);

          System.out.println("hostname-> " + serviceInstance.getHost());
          System.out.println("port-> " + serviceInstance.getPort());
         }
     }
  }
```



- **Service Consumer implementation details**
  - Service instances resolution by Eureka Service Discovery capabilities
  - Details of the available service instances

Discovery Service: Feign client

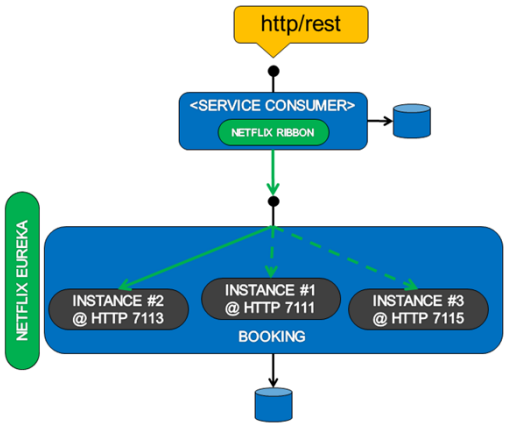
```
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.cloud.netflix.feign.FeignClient;

@SpringBootApplication
@EnableDiscoveryClient
@RestController
@EnableFeignClients
public class EurekaFeignClientApplication {

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

     @FeignClient("BOOKABATTERYSERVICE4EUREKA")
     interface IServiceBookAbattery {
          @RequestMapping(method = RequestMethod.GET, value = "/bookABattery/list")
          List<Booking> getBookingList();}

     @Autowired
     IServiceBookAbattery client;

     @RequestMapping("/")
     public List<Booking> getBookingList() {
          return client.getBookingList(); }}
```



- **Netflix Feign Client implementation details**
  - *@EnableFeignClient directive*
  - *@FeignClient* service resolution
  - Rest service method invocation

Dynamic routing and load balancing

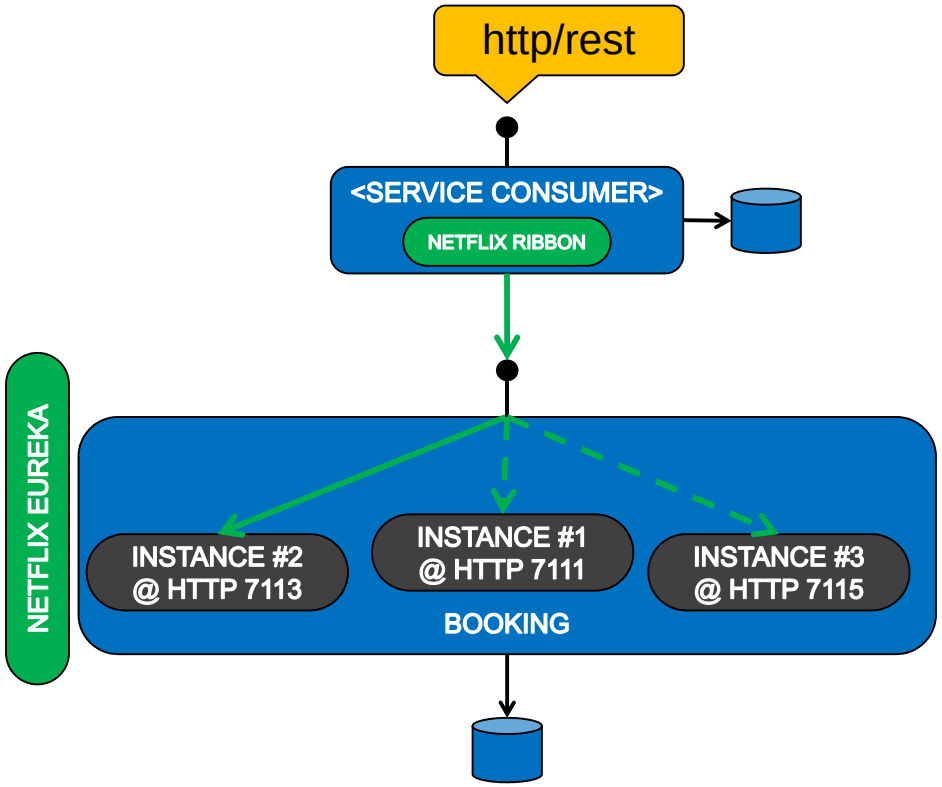
```
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;

@Autowired
LoadBalancerClient loadBalancer;

@Autowired
RestTemplate restTemplate;

@Bean
RestTemplate restTemplate() {return new RestTemplate(); }

@RequestMapping("/loadBalancerBooking")
public Booking[] getBooking(){

     ServiceInstance instance =
          this.loadBalancer.choose("BOOKABATTERYSERVICE4EUREKA");

     URI uri = UriComponentsBuilder.fromUriString(instance.getUri().toString())
          .path("/bookABattery/list").build().toUri();

     Booking[] listBooking = restTemplate.getForObject(uri , Booking[].class);

     return listBooking;
}
```



- **Load balancing implementation details**
  - Service instance resolution provided by Spring Cloud load balancer
  - The invocation of the service Rest method is done by means of a RestTemplate class

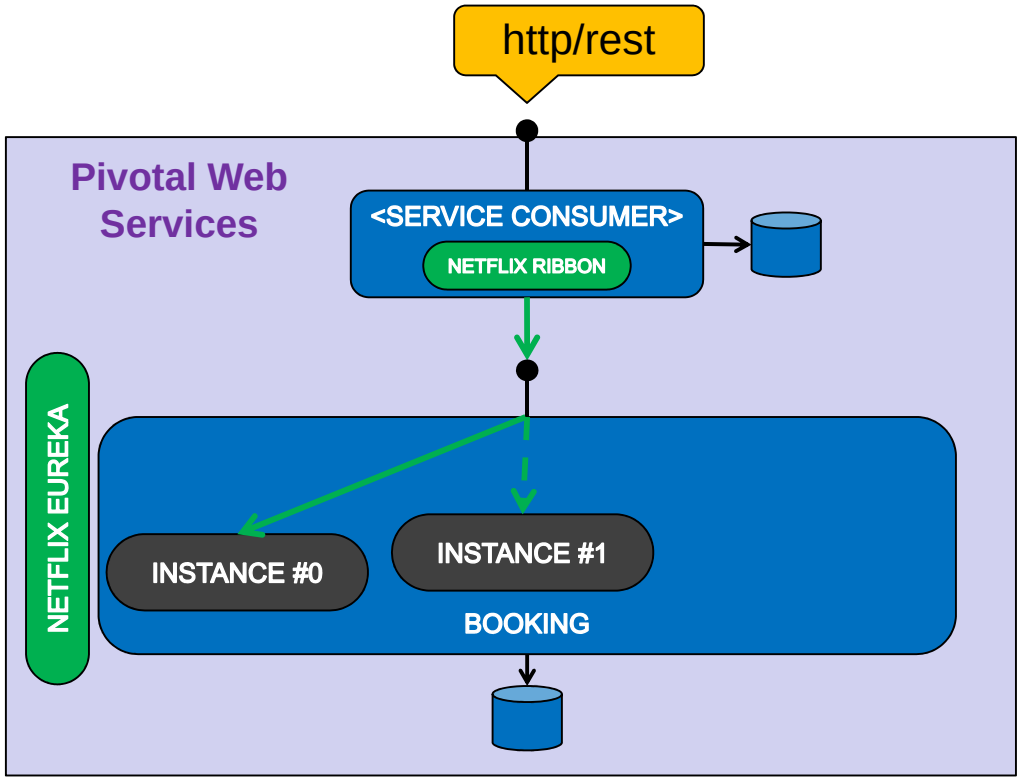
DEMO

Dynamic routing and load balancing: Ribbon



- In order to avoid service outages, it is common practice to make a system landscape with more than one service instance of the same type running.
- To this end, a load balancer will spread all the incoming calls over every available instance.
- As stated in the requirements, load balancing shoud be obtained simply by adding references to the new services, without the need to configure any load balancer
- **Ribbon** is the Netflix solution to these **load balancing requirements**
  - Ribbon uses the information available in Eureka to locate adequate service instances
  - If more than one instance is found, Ribbon will apply load balancing to spread the request over the available instances
  - Ribbon does not run as a separate service, but rather as an embedded component within each service consumer

Eureka and PWS: service instance scale up and load balancing

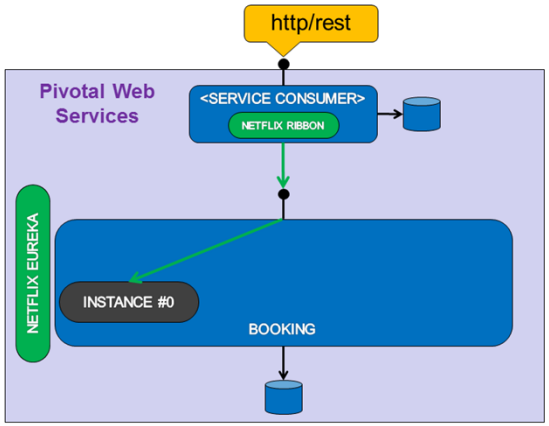


- To develop its service scaling capabilities, Eureka uses the PWS features of dynamic allocation ports
- It is sufficient to scale out a registered service instance and PWS will automatically allocate a new port and thus notify the service discovery server of the new instance
- The application receives an entry in a dynamic routing tier of Eureka Server, which performs a load balancing function for traffic across the available instances.

DEMO

PWS: service instance scale up and load balancing

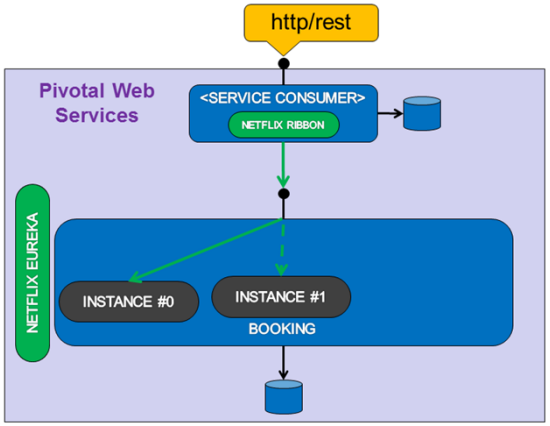
```
[APP/0] [OUT] 2016-09-19 20:27:43.431 INFO 17 ---
[ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)

[APP/0] [OUT] 2016-09-19 20:27:43.433 INFO 17 ---
[ main] c.n.e.EurekaDiscoveryClientConfiguration : Updating port to 8080

[APP/0] [OUT] 2016-09-19 20:27:43.438 INFO 17 ---
[ main] i.l.microservice.Application : Started Application in 18.953 seconds
          (JVM running for 20.013)

[RTR/2] [OUT] 06-bookABatterySERVICE4EUREKA.cfapps.io - [19/09/2016:20:28:45.667 +0000]
     "GET /bookABattery/list HTTP/1.1" 200 0 1485 "-" "Java/1.8.0_101" 10.10.66.180:64424
     x_forwarded_for:"54.86.233.33"
     x_forwarded_proto:"http"
     vcap_request_id:b0cffe94-a597-42fe-500d-4e13585f8677
     response_time:0.299727231
     app_id:e33d3d68-2efb-41db-8ac5-d61d43c4af3f
     index:0


[APP/0] [OUT] 2016.10.19.20.28.50
[APP/0] [OUT] ************************************************************************
[APP/0] [OUT] *********** CALLED BOOKING LIST ***********
[APP/0] [OUT] ************************************************************************
```



- **PWS balancing evidence**
  - Tomcat container correcly started
  - Client invocation details
  - APP/0 is the responding service instance

PWS: service instance scale up and load balancing

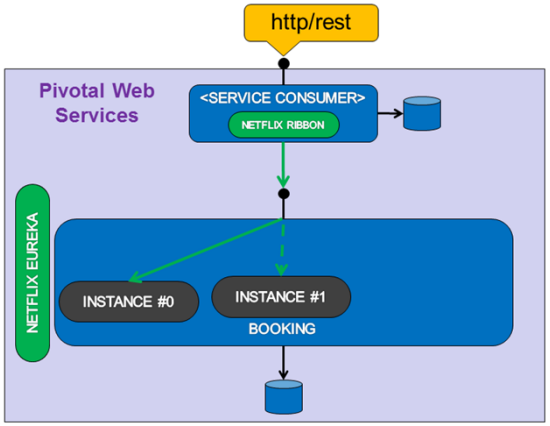
```
[APP/1] [OUT] 2016-10-19 21:27:43.431 INFO 17 ---
[ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)

[APP/1] [OUT] 2016-10-19 21:27:43.433 INFO 17 ---
[ main] c.n.e.EurekaDiscoveryClientConfiguration : Updating port to 8080

[APP/1] [OUT] 2016-10-19 21:27:43.438 INFO 17 ---
[ main] i.l.microservice.Application : Started Application in 18.953 seconds (JVM running
for 20.013)

[RTR/2] [OUT] 06-bookABatterySERVICE4EUREKA.cfapps.io - [19/10/2016:20:28:45.667 +0000]
      "GET /bookABattery/list HTTP/1.1" 200 0 1485 "-" "Java/1.8.0_101" 10.10.66.180:64424
      x_forwarded_for:"54.86.233.33"
      x_forwarded_proto:"http"
      vcap_request_id:b0cffe94-a597-42fe-500d-4e13585f8677
      response_time:0.299727231
      app_id:e33d3d68-2efb-41db-8ac5-d61d43c4af3f
      index:0

[APP/0] [OUT] 2016.10.19.20.28.50
[APP/0] [OUT] ************************************************************************
[APP/0] [OUT] *********** CALLED BOOKING LIST ***********
[APP/0] [OUT] ************************************************************************
```



- **PWS balancing evidence**
  - Start-up of the second instance (APP/1)
  - Evidence of the Eureka Discovery configuration

PWS: service instance scale up and load balancing

```
[RTR/2] [OUT] 06-bookABatterySERVICE4EUREKA.cfapps.io - [19/10/2016:20:28:45.667 +0000]
      "GET /bookABattery/list HTTP/1.1" 200 0 1485 "-" "Java/1.8.0_101" 10.10.66.180:64424
      ……..
      app_id:e33d3d68-2efb-41db-8ac5-d61d43c4af3f
      index:1

[APP/1] [OUT] 2016.10.19.20.28.50
[APP/1] [OUT] ********************************************************************
[APP/1] [OUT] ********** CALLED BOOKING LIST **********
[APP/1] [OUT] ********************************************************************

[RTR/3] [OUT] 06-bookABatterySERVICE4EUREKA.cfapps.io - [19/10/2016:20:28:47.969 +0000]
      "GET /bookABattery/list HTTP/1.1" 200 0 1485 "-" "Java/1.8.0_101" 10.10.66.180:2509
       ……..
      app_id:e33d3d68-2efb-41db-8ac5-d61d43c4af3f
      index:0

[APP/0] [OUT] 2016.10.19.20.28.55
[APP/0] [OUT] ********************************************************************
[APP/0] [OUT] ********** CALLED BOOKING LIST **********
[APP/0] [OUT] ********************************************************************
```



- **PWS balancing evidence**
  - Requests are then balanced towards the available instances

    (APP/1 and APP/0)

DEMO

…in the last hour we saw:

- That the benefits of the **Event-Driven Architecture** generate additional overhead in systems management and a more complex coding process
- The increased coding complexity of the Event-Driven architecture could be mitigated by the adoption of **Spring Stream**
- That the extra cost in infrastructure handling will result in a **High Availability** central data storage system
- It is unusual for an architectural approach of this kind to be so **brutally efficient**, offering an **immediate and significant reduction in operating costs**

What's next?

- Other complementary technologies include:
  - **Spring Cloud Config:** the Spring framework providing Git-managed versioning for configuration data. It enables the dynamic refresh of configuration data without the need for re-start services (zero downtime service)
  - **Circuit breaker** with **Histrix:** a fault tolerance library designed to isolate points of access from remote systems. It enables us to halt episodes of cascading failure and ensure resilience in complex distributed systems
  - **Routing and filtering** with **Zuul:** an edge service application built to enable dynamic routing, monitoring, resilience and security
  - **Event sourcing, Database triggers** and **Transaction log-tailing patterns within the Event Driven Architecture** as a way to atomically update the status and publish events

References

- https://12factor.net/
- https://spring.io/guides/gs/spring-boot-docker/
- https://github.com/Netflix/eureka
- https://www.infoq.com/news/2015/06/cqrs-eventsourcing-demoapp
- https://plainoldobjects.com/presentations/building-and-deploying-microservices-with-event-sourcing-cqrs-and-docker/
- https://blog.codecentric.de/en/2016/04/event-driven-microservices-spring-cloud-stream/
- http://samchu.logdown.com/posts/313414-microservice-with-springcloud
- https://spring.io/blog/2015/07/14/microservices-with-spring
- https://blog.pivotal.io/pivotal-cloud-foundry/products/spring-cloud-stream-the-new-event-driven-microservice-framework
- https://www.3pillarglobal.com/insights/building-a-microservice-architecture-with-spring-boot-and-docker-part-i
- https://spring.io/blog/2015/04/15/using-apache-kafka-for-integration-and-data-processing-pipelines-with-spring

Source Code

- https://github.com/lbennardis/microS_code2016_serviceRegistryDiscovery

Questions?

l.bennardis@email.it
https://it.linkedin.com/in/luigi-bennardis-0a56a72