# @TO Learn

# @TODO

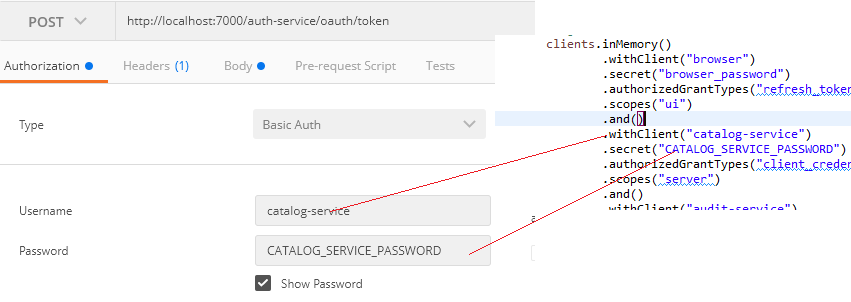
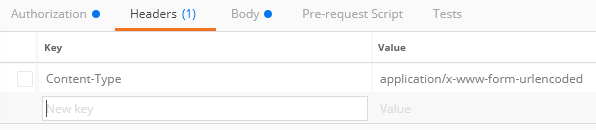
* ~~Design the way to share interface, DTO between Micro service~~
* ~~Design transaction handler with multiple MS in a chain~~
* ~~Integration test with Spring contract~~
* Optimize unit test
* Performance test
* Integration test

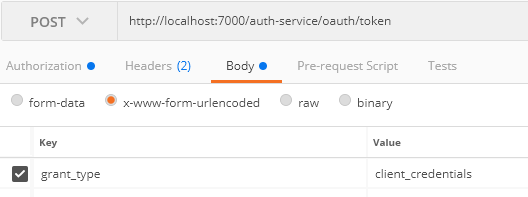
Microservices Integration Testing & Spring Boot  
<https://medium.com/@isadounikau/microservices-integration-testing-spring-boot-404b6f8617d1?source=email-e215461c89f7-1607282236756-digest.reader------0-59------------------79c2768c_15c9_4155_8441_26e0d0f76fe8-1-49d5e1fa_2010_4721_bcb2_99469ba3c411----&sectionName=top>

* **Integrate Elastic log**
* ~~Deploy to Azure Kubernetes cluster~~
* Rabbit MQ scaling for large amount of concurrent user
* Ben-mark Spring web flux

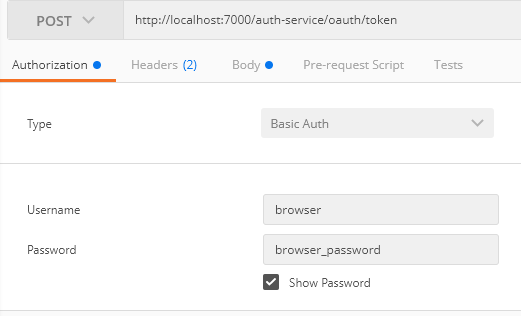
# Postman

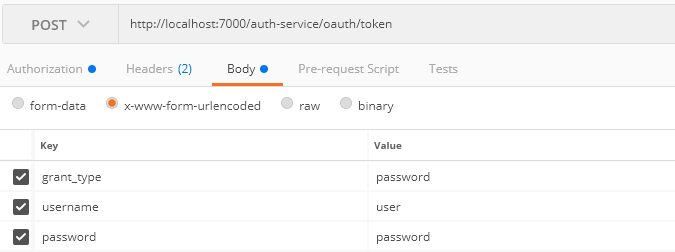
1. Login and get JWT token by **[client\_credentials]** grant type
   1. Authorization tab

* Method must be POST
* Authorization must [Basic Auth]
* User = client ID
* Password = secret  
    
  
  1. Head tab empty  
     
  2. Body tab contains only grant type



1. Login and get JWT token by [password] grant type
   1. Authorization tab

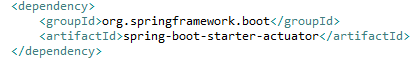
* Type must be basic Auth
* Type must be [Basic Auth]
* Username = client ID
* Password = secret  
    
  
  1. Header tab is empty
  2. Body tab
* Grant\_type = password
* User = user account
* Password = password of the account



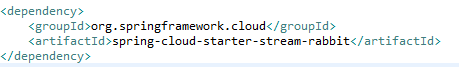
# Authentication

* In order to allow Spring gateway to transfer Header bear token, we need to add some configuration in the configure file of the Spring Gate Way source code  
  

# Refresh – configuration

* In order to refresh value of properties into micro service every time modifying on configuration server, libraries and configuration must be added on each micro service project.
  + Spring actuator must be added in POM file, and the configuration for opening actuator endpoints in application yml file  
      
    
  + Rabbit MQ and spring bus amqb must be added in POM file

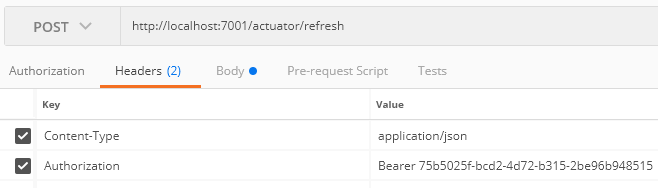
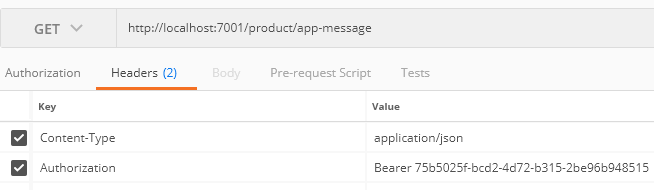




* + Rabbit MQ must be setup in Spring configuration server and in each micro service project

* The value from configuration server in each micro service could be refreshed from configuration server to Micro service application by calling actuator/refresh URL

For example:

Edit the value app.message = new value in config-repo/application.yml file   
Call <http://localhost:7001/actuator/refresh>  
  
Try again to get updated the message from configuration server  
  
In order to allow the configuration value is refreshable, the class must add annotation @RefreshScope



* One Micro Service could notify another service about the update from configuration server by call URL bus-refresh  
  For example: <http://localhost:7001/actuator/bus-refresh>

# How to share DTO between Micro services

## Sharing DTO package

In order to allow one service to work with another service, it needs to know the input DTO parameters and response DTO parameters.

The input and output DTO of each micro service could be different, we could not use the same DTO object sharing for all micro service which lead to redundant code.

For example:

In an ecommerce website using micro service architecture, each and every microservice could represent the user more or less as shown here







To share DTOs of a producer service for other micro services, a project should be divided into 2 sub projects.

For example:

inventory-service includes 2 sub project inventory-client and inventory-server

* Inventory-client contains response DTO classes, request DTO classes, and classes allowing to call API from server which will be provided for another micro services
* Inventory-service contains all API and bussines code
* Once inventory-client is built, it will be deployed to sharing reposity and imported to the consumer micro services. By this way one team could communicate with the another team by version pushed to the repository when they update a service and avoid break the other service.

## Spring contract

* Producer
  + Need to define contract, pushed to repository and provide for consumers
    - Add depending libraries and plugin
    - Define base class, the base class will mock response data for testing
    - Add test stub in /src/test/resources/contracts/ package, the file will be in groovy format
    - When we run the build, the plugin automatically generates a test class named ContractVerifierTest that extends our BaseTestClass and puts it in /target/generated-test-sources/contracts/
* Consumer
  + Need to add spring contract stub runner in pom file & spring contract wire mock
  + Need to implement Junit test
  + The jar client contract file could be declared directly inside pom file or auto detect in Junit file depend on availability of repositoryroot parameters

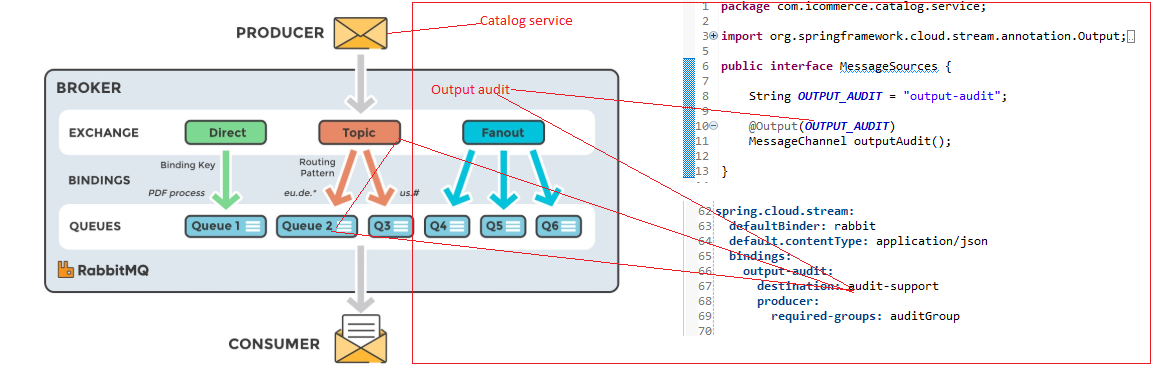
If there repository root, it will look stub jar file in user/.m2/…

If we make any changes on the producer side that directly impact the contract without updating the consumer side, this can result in contract failure.

For example, suppose we're to change the EvenOddController request URI to /validate/change/prime-number on our producer side.

If we fail to inform our consumer of this change, the consumer will still send its request to the /validate/prime-number URI, and the consumer side test cases will throw org.springframework.web.client.HttpClientErrorException: 404 Not Found.

# RabbitMQ



# URLs

## Sleuth zipkin

* + <https://kipalog.com/posts/Distributed-tracing-voi-Spring-Cloud-Sleuth-va-Zipkin>
  + <https://howtodoinjava.com/spring-cloud/spring-cloud-zipkin-sleuth-tutorial/>
  + <http://localhost:9411/zipkin/>

## Rabbit MQ

<http://localhost:15672/>  
guest/guest

<https://dzone.com/articles/rabbitmq-and-spring-boot-integration-with-fault-to>

## Eureka

<http://localhost:8761/>

## Spring contract

<https://www.baeldung.com/spring-cloud-contract>

<https://spring.io/guides/gs/contract-rest/>  
<https://cloud.spring.io/spring-cloud-static/spring-cloud-contract/2.1.1.RELEASE/multi/multi__spring_cloud_contract_stub_runner.html>

## Distributed transaction

https://github.com/solutionmind/orchestration-microservices-transaction

<https://www.vinsguru.com/architectural-pattern-orchestration-saga-pattern-implementation-using-kafka/>

<http://www.vinsguru.com/architectural-design-pattern-saga-pattern-implementation-using-kafka/>

<https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part-2/>  
<https://microservices.io/patterns/data/saga.html>

<http://progressivecoder.com/saga-pattern-implementation-with-axon-and-spring-boot-part-1/>

<https://eventuate.io/exampleapps.html>

<http://progressivecoder.com/saga-pattern-implementation-with-axon-and-spring-boot-part-1/>

<https://medium.com/swlh/handling-transactions-in-the-microservice-world-c77b275813e0>

<https://piotrminkowski.com/2020/06/19/distributed-transactions-in-microservices-with-spring-boot/>

## Best practices Spring JPA

<https://dzone.com/articles/50-best-performance-practices-for-hibernate-5-amp>