1.0 Introduction

This project is to create scaffolding code for spring cloud micro-services as a distributed system.

It will have an evolution from basic stateless micro-services to complex architect with loggers, message queues, elastic search, and redis database which can be used as a learning purpose of spring cloud framework and quickly generated for multiple development purposes, since spring framework is a **Convention Over Configuration** design pattern.

I will use as much producer - consumer model as possible in most cases.

2.0 Spring Cloud Basic Architect

As per graph below,

-API gateway send its node information to eureka server clusters, and receive Iot, Mobile / Browser request and then dispatch to micro services

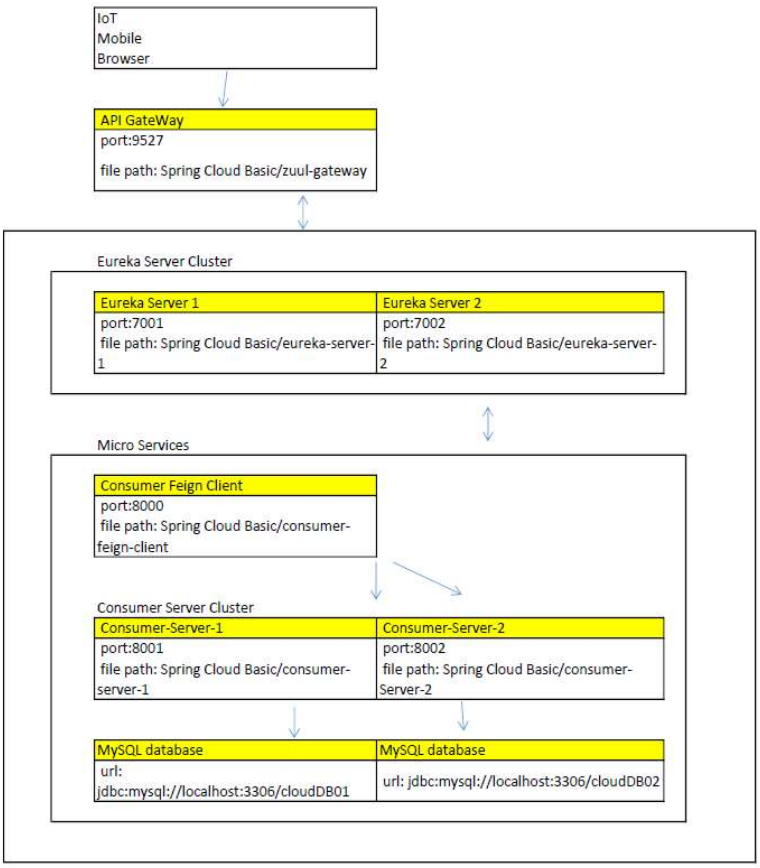
-Eureka server clusters register services from API gate way and micro services.

-Consumer Feign Client is used to achieve load balancing for requests towards consumer server clusters, circuit breaker has been plugged in to degrade services

-Consumer servers are used to process requests and send to MySQL databases by ORM framework (mybatis), hystrix has also been used, these servers are stateless, therefore could be replicated without side effect.

-MySQL database has been used to store and fetch data.

-Source code is in folder Spring Cloud Basic.



2.1 Testing On a Single Machine:

-Set up virtual machine mapping,

In windows hosts file path： C:\Windows\System32\drivers\etc\hosts,

Or in Linux hosts file path: etc\host.conf.

Then add

127.0.0.1 eureka7001.com

127.0.0.1 eureka7002.com

-Start processes from source code file

1. Spring Cloud Basic/zuul-gateway , port:9527

2. Spring Cloud Basic/eureka-server-1 , port:7001

3. Spring Cloud Basic/eureka-server-2 , port:7002

4. Spring Cloud Basic/feign-client , port:8000

5. Spring Cloud Basic/consumer-server-1, port:8001

6. Spring Cloud Basic/consumer-server-2, port:8002

-Create Local databases:

In MySQL:

```
create database cloudDB01;

create database cloudDB02;

CREATE TABLE Consumer
(
    consumer_no INT PRIMARY KEY,
    cname VARCHAR(50),
    db_source VARCHAR(50)
);
```

In database1 cloudDB01, db_source is for name of database , cloudDB01 is 1:

```
INSERT INTO consumer(consumer, cname, db_source) VALUES (1, 'consumer1', '1');
INSERT INTO consumer (consumer, cname, db_source) VALUES (2, 'consumer2', '1');
INSERT INTO consumer (consumer, cname, db_source) VALUES (3, 'consumer3', '1');
```

In database2 cloudDB02, db_source is for name of database , cloudDB01 is 2:

```
INSERT INTO consumer(consumer, cname, db_source) VALUES (1, 'consumer1', '2');
INSERT INTO consumer (consumer, cname, db_source) VALUES (2, 'consumer2', '2');
```

INSERT INTO consumer (consumer, cname, db_source) VALUES (3, 'consumer3', '2');

-Testing request:

Type localhost:9527/api/consumer/get/{id} to request RESTful services.

-Docker / Containerized

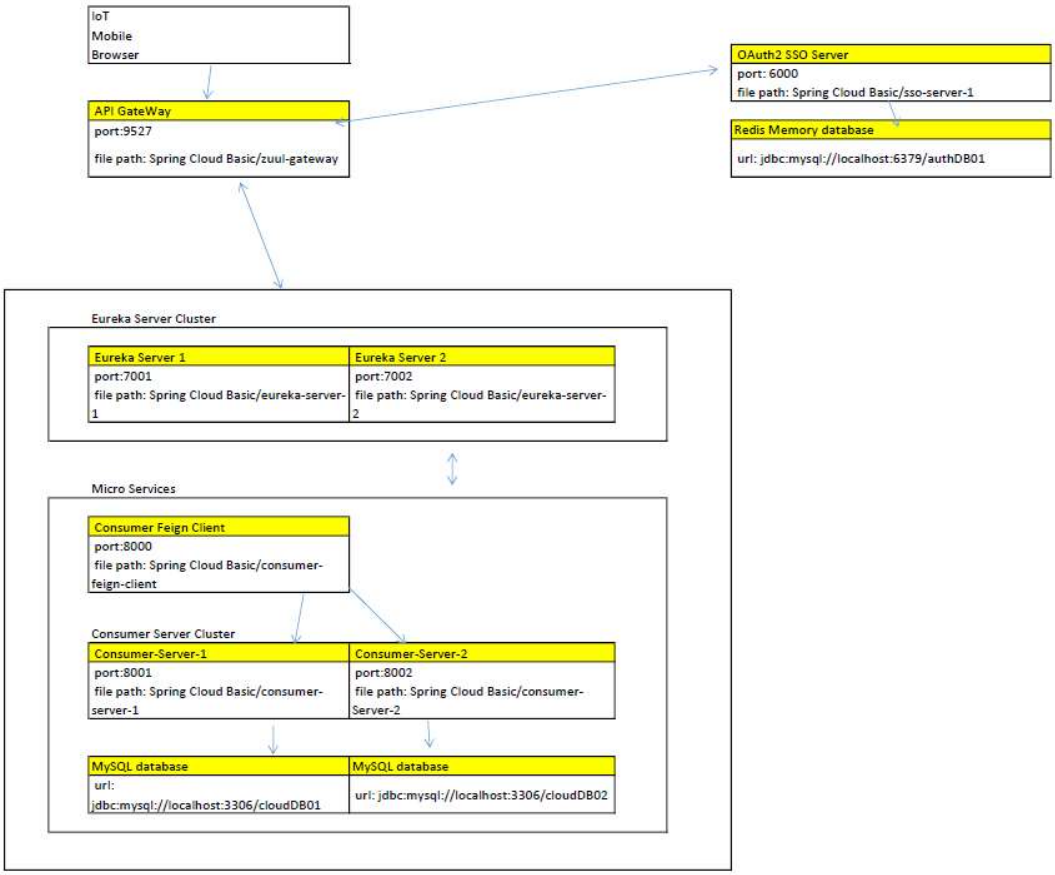
As per above, each node could also be wrap up with docker for testing and deployment.

3.0 Spring Cloud Basic Architect with Security and Single Sign On (SSO)

Based on 2.0, Security and Single Sign On will be added into this server architect.

-Source code is in folder Spring Cloud SSO.

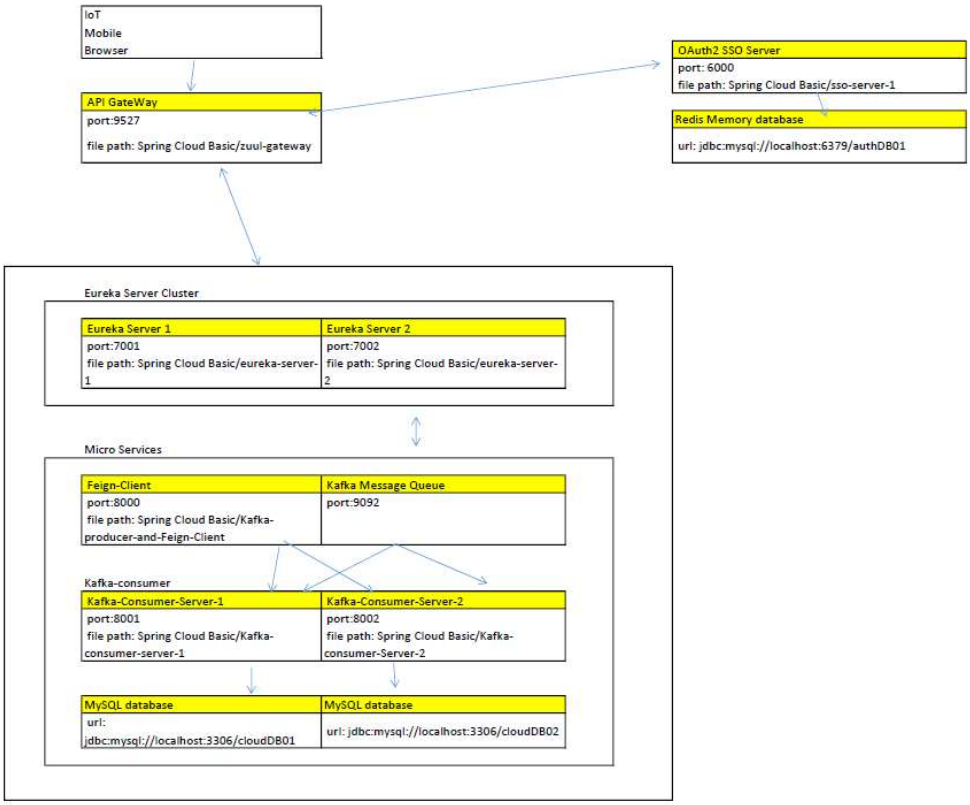
-An Oauth2 SSO Server has been linked to the API Gateway for security purpose to access REST APIs.



4.0 Spring Cloud Basic Architect with Security and Single Sign On (SSO) , Kafka Message Queue

-source code is in folder Spring Cloud Kafka SSO

-As per section 3.0, message queue has been used to deal with high concurrency requests.



5.0 Front End Interface

As a practice purpose, I have put a **POSTMAN Like App** to test RESTful service conveniently without much hardcoding by React.js.