* Service Registry & Discovery - Consul

1. Download Consul
2. consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.99.1

Find out your IP address using ipconfig and use that in bind

Micro-consul. - MicroService 1

Microconsulschoolapp - MicroServices 2 ( This will look up in Consul to invoke MicroServices 1)

Both Service will auto register to Consul

1. <http://localhost:8500>
2. @EnableDiscoveryClient to main class
3. Add following to application.properties

server.port=9098

spring.application.name: student-service

management.security.enabled=false

1. Create School App
2. Clone Client App to clientapp2
3. <http://localhost:8098//getSchoolDetails/abcschool> and see in console - Load Balancing

* Consul from java Client

Download micro-consulapi

Run Main Class in sections & also together to understand the concept.

* Circuit Breaker - Hystrix

Hystrix :

1. Add

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

<dependency>

    <groupId>org.springframework.cloud</groupId>

    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

1. Add @EnableCircuitBreaker annotation
2. Add @EnableHystrixDashboard annotation
3. Add annotation @HystrixCommand(fallbackMethod = "myFallbackMethod")

Add following to hystrixschoolapp properties file

management.endpoints.web.exposure.include=hystrix.stream

This is the endpoint

localhost:8098/actuator/hystrix.stream

Open <http://localhost:8098/hystrix> and add above endpoint

* Service Look up & Registry again - Eureka

Download micro-eureka-server

Create 2 micorservices & register them to Eureka

Dashboard at <http://localhost:8761> and admin at <http://localhost:8761/admin>

* Eureka 2

Spring-eureka-client-student

Spring-edureka-client-school

Spring-eureka-server

application.yml  in server

bootstrap.yml

check server <http://localhost:8761/>

* Externalising the Configuration

Download following projects

Spring-config-client. - MicroService

Spring-config-server - Config server which will host your properties

Config-server-repo. - The repo where your configurations would be there

Test following after getting server up & running:

* http://localhost:8888/config-server-client/development
* <http://localhost:8888/config-server-client/production>

Change in bootstrap.properties of client & check ( from prod to dev or so)

<http://localhost:8080/msg>

1> make changes in application.properties & push to git

2> Ask MicroService (Pull) to refresh itself using post from postman <http://localhost:8080/refresh>

check msg again at <http://localhost:8080/msg>

* Externalising the Configuration ( Polling Method)

Download following projects

Spring-config-client-consul. - MicroService

Now Consul KV Store would act as Config server which will host your properties

Config-server-repo. - The repo where your configurations would be there

Add Property to KV Store config/<Spring.application.name>/property name

check msg at <http://localhost:8080/msg>

Now update value in Kv Store and again visit the same endpoint

Try mapping msg to a property myfolder.subfolder.newmsg ( This would be property at following path in KV Store config/<Spring.application.name>/myfolder/subfolder/newmsg)

* Hoverfly

Hoverfly-actual-service. - MicroService 2 ( Server)

Hoverfly-actual-service-client - MicroService 1 ( Client)

Start Hoverfly using following command ( from Hoverfly-actual-service root folder):

Hoverctl start

Start actual-service

Add vm parameter -Dmode=proxy to service-client and start it

Change mode to Capture from Dashboard or using hoverctl mode capture

Open http://localhost:8888/dashboard and ensure mode is in Capture mode

Now it http://localhost:8080/invoke for multiple times

Ensure it is captured by looking back at Dashboard

hoverctl export simulations.json

( You can use hoverctl import simulations.json in future when you want to import)

Change mode to simulate fro dashboard or

hoverctl mode simulate

http://localhost:8080/invoke and see the simulation count is increasing

Now stop the service

Now hit http://localhost:8080/invoke and see the magic 😊

Now change mode to capture

And now what happens if you refresh invoke endpoint.

* Zipkin

Download projects :

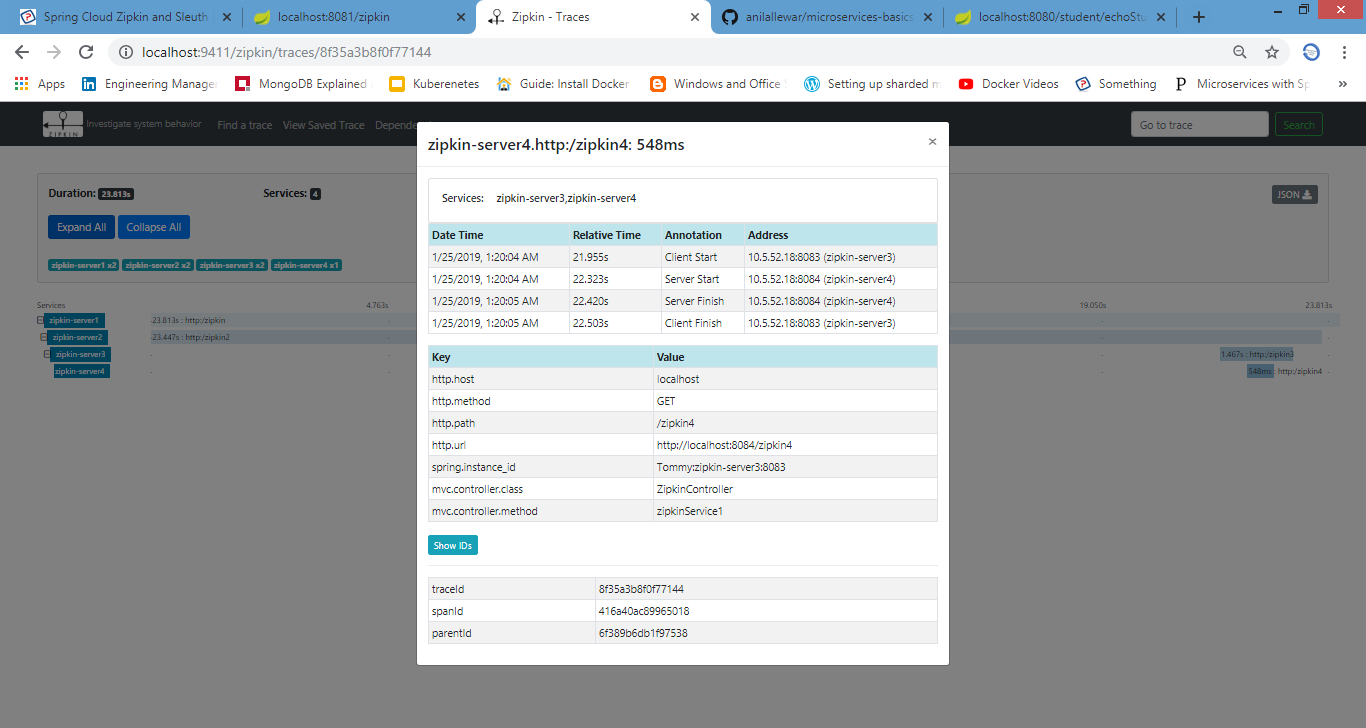
Micro-zipkin which has 4 micro services inside it.

Find pipkin-version.jar at root directory, run it as java -jar .

Go to <http://localhost:9411/zipkin/>

BuildAll and StartAll ( or run projects from STS)

Go back to Dashboard , Refresh Services & find Traces by selecting a Service.



* Externalising using Spring Cloud Bus

springcloudconfigrepo - Configuration

Employee-config - Config Server

Employee-producer-eureka. - MicroService 1

Employee-producer2 - MicroService 2

eureka-server - Eureka Server 1

eureka -server2 - Eureka Server 2

Edit Application.properties in employee-config to point to your git repo

Start servers & visit localhost:8761 where you can see 2 servers ; while null in 8762

Now Edit the application.properties under springcloudconfigrepo to 8762

Post <http://localhost/bus/refresh>

Now all servers should move to 8762

* Load Balancing

**ribbon-client ( This contains ribbon Configuration)**

**ribbon-Server**

To do that we need to use different port for this, to start service in a specific port we need to pass the port in this way.  
java -jar -Dserver.port=XXXX target/YYYYY.jar. We will create 3 instances of this service in ports 9090, 9091 and 9092 ports.

* Zuul Proxy

Projects to be downloaded :

spring-boot-zuulgatwayproxy - Proxy

spring-boot-zuulgatwayproxy-student-service. - MicroService

Start spring-boot-zuulgatwayproxy-student-service

<http://localhost:8090/echoStudentName/Sajal>

<http://localhost:8090/getStudentDetails/Sajal>

start spring-boot-zuulgatwayproxy

and visit

<http://localhost:8080/student/getStudentDetails/Sajal>

<http://localhost:8080/student/echoStudentName/Sajal>

Look at configuration inside spring-boot-zuulgatwayproxy

* Security :

1. Import Spring-MicroServices-OAuth-Server

Start the Project

Go to localhost:9090/oauth/token ( No / ) from POSTMAN ( POST request)

Authorization : Basic Auth -> webapp/websecret

Params -> grant\_type= password, username=user1, password=password1

1. Import OAuth-Resource-Server

See Changes

Go to localhost:9091/resource/endpoint/

Auth Failed

Now get Token for User1/password1 using endpoint in #1

And add a parameter access\_token= <token>

Again Failed

Now get Token for admin/password2 using endpoint in #1

Now it should succeed

* Spring-Actuator

Download actuator project

Understand the Seurity config, custom endpoint & application.properties

* Cloud-Foundry

Spring-helloworld-cf

Download from <https://cli.run.pivotal.io/stable?release=windows64&source=github>

And unzip here

Install exe

Type cf from command line and see

Register yourself at <https://account.run.pivotal.io/z/uaa/sign-up>

Activate your account

Go to webservices

Login using

cf login -a api.run.pivotal.io

Build Project using mvn

F:\sts-workspace\microservices\spring-helloworld-cf>cf push myapptechnopreneur -d domainname.com -p target\sp

ring-helloworld-cf-0.0.1-SNAPSHOT.jar

Optional below :

cf create-domain technopreneur mydomaintechnopreneur.com

cf create-route development mydomaintechnopreneur.com -n www

* Spring Data flow

Sping-data-flow-source

processor

Sink

Run Data flow

<http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-server-local/1.7.3.RELEASE/>

run as java -jar

Download

<http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-shell/1.7.3.RELEASE/spring-cloud-dataflow-shell-1.7.3.RELEASE.jar>

run as java -jar

under this shell:

app register --name source-app --type source --uri maven://com.example:source:jar:0.0.1-SNAPSHOT

app register --name processor-app --type processor --uri maven://com.example:processor:jar:0.0.1-SNAPSHOT

app register --name sink-app --type sink --uri maven://com.example:sink:jar:0.0.1-SNAPSHOT

Go to <http://localhost:9393/dashboard>

stream create --name log-data2 --definition 'source-app | processor-app | sink-app'

stream deploy --name log-data2

#### Reference Projects

#### Reference Project 1:

#### Two Parts :

#### Understanding stream-rabbit

#### Download Project spring-cloud-stream-rabbit project from repository

#### micro-casestudy

#### Run Unit tests and understand the code ( pom.xml, annotations, properties and unit tests)

#### Download event-driven-spring under same repository ( micro-casestudy)

#### You can either run them as Docker OR

#### Simplest way is to run application-process, credit-application, credit-decision, Scoring, customer as Spring Boot Application.

#### Refer readme under event-driven-spring for more details

#### Reference Project 2 :

#### microservices-casestudy ( Simulating Bank Account & Customer - Using Consul + Hysterix Integration)

#### 