**MicroServices**

**Eureka Server Registration**

It is the microservices component, It is used for service registry and service discovery.

**Eureka configuration:**

Create the project and add the spring-starter-web and eureka-server dependencies.

First we have to enable the eureka in the Spring Boot application class.

Next in the properties file we need to write this configuration:

server.port=8086

spring.application.name=eureka

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

eureka.instance.hostname=localhost

**zulu(API GateWay)**

zulu is api gate way when ever we send the request the request will comes to the Api Gateway Only.

It is like a front controller.

In the zulu there are four types of filter are available they are as follows:

1.PreFilter

2.PostFilter

3.ErrorFilter

4.Route Filter

**Zulu configuration:**

First we need to create the project to the following dependencies: spring-cloud-starter-netflix-zuul

Next in the Spring boot application class we need to write one annotation as: @EnableZuluProxy

Then we need to write some configuration in the application.properties file:

zuul:

routes:

order-details:

path: /order-details/\*

service-id: ORDER-DETAILS(this is the name of the application which one you have to send the data to this application ,the name is whatever the application name registered with the eureka server)

strip-prefix: false

**(Hystrix) Fault Tolerance:**

Hystrix is meant for providing the alternate path execution for the particular method . for example we have one application in that there is multiple users are sending the request for the particular resource on that time the server will be down. So we need to provide the some fallback to the user like server busy (or) try after some time like that we need to send to the end user.

**Hystrix Configurations:**

First we have to create the spring starter project and add the dependency : spring-cloud-starter-netflix-hystrix.

Then In the application class we need to enable the @EnableHystrix.

Next we have to write one method and return your fallback data or fallback logical view name.

Find which method you want to give the fall back then we need to annotate with @HystrixCommand(fallbackmethod=”your method name”).

**ELK logging**

ELK Stands for(Elastic Search, Logstash, Kibana)

This is used for developing application along with logging of the application.

Configuration and downloading the ELK:

First we need to download all the three Elastic Search, Kibana, Logstash from the browser.

After that we have to run the bat files from the elastic search and kibana as follows

Elastic search bat file run:

Elastic search/bin> elasticsearch.bat 🡪the default port for the elastic search is 9200.

Kibana bat file run:

Kibana/bin> kibana.bat 🡪the default port for the kibana is 5601.

After running the both elastic search and kibana then that you need the very weather the two bat files are running are not by using default port numbers.

Run the Logstash:

Before running to the logstash we need to write some configuration in the logstash software

**Docker**

**How To convert Your application as Docker Image:**

In Micro Services Docker is a very important as a developers has to give this docker image to the Deployment team, the Deployment team has to run this docker image in the Kubernetes.

How to install the docker in to your system:

First we need to search as DockerToolBox in the browser and download the application from the Browser.

Then we has to use the power shell or command prompt for running our docker file.

Then in the application we need to write on docker file for the application directory File name must and should be DockerFile under that we have to write some configurations.

That is:

FROM openjdk:8-jdk-alpine

EXPOSE 5252

VOLUME /tmp

ADD target/ApplicationName.jar ApplicationName.jar

ENV JAVA\_OPTS=""

ENTRYPOINT [ "java","-jar","ApplicationName.jar" ]

Then finally in the power shell we need to run our docker file as follows :

In the application location we need to write this command:

🡪docker build -t image name(any name) .(Dot)

Run your docker file:

🡪 docekr run -p 8080:8080 -t image name

Stop the server:

🡪 docker stop containerid

Find the containers running in the docker:

🡪docker -ps

**Apache Kafka(IPC)**

IPC(Inter process Communication). If we are able to communicate one microservice to another microservice the we should have to use this Apache kafka the implementations of the kafka is Spring cloud stream, alternate for apache kafka is ActiveMQ and RabitMQ on so on..

How to implement the apache kafka in the application is as follows.

First we need to create the application by adding the spring-kafka, spring-cloud-stream dependencies.

Download the Apache Kafka:

In the browser download apache kafaka

Then run three .bat files from the bin/windows directory

Run the Zookeeper.bat :

🡪in KafkaLoacation in the bin\windows> zookeeper.bat E:\kafka\config\zookeeper.properties

Run the Kafka-server-start.bar:

🡪in KafkaLoacation in the bin\windows> Kafka-server-start.bat E:\kafka\config\server.properties

Create the kafka.topic.bat :

🡪in the kafka location in bin\windows> kafka.topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic usertopic

verify the topic data:

🡪kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Mytopic(Topic Name)

**Writing the Test Cases For our Application**

what is test cases?

test cases is meant for testing the application, that means weather the application is works fine or not. if test cases fails the application will not be packaged ,when ever the test cases runs successfully then only the application will packaged.

What is Junit ?

junit is a most popular testing framework in java.

it is explicitly recommended for unit testing.

junit does not require server for testing web application, which makes the testing faster.

it is also allows quick and easy generation of the test cases and test data. the org.junit package consist of many interfaces and abstract classes for junit testing such as Test, Assert, Before, After, etc.