Pre-Requisite
==========
SpringBoot
SpringRest(SpringMVC++)
SpringDataJPA(Working with MySQL,Embeded database(h2))
SpringAOP
Build Project using Monolothic Style

Microservices
==========
What are the challenges which will be faced by the developers if we are following microservices design pattern in our project?
a. Bounded context
b. Lot of configuration
c. Less visibility
d. Pack of Cards Problem

a. Bounded context
It is difficult to decide the boundary for one microservices.
boundary -> which functionality should be devleoped as a seperate project(microservices) is very challenging.

b. Lot of configuration
In microservices architecture we develop mulitple projects or multiple services, so for every service we need to give configuration
like DB config, Actuator Config,SMTP config,logging config, Kafka configuration, etc,...

c. Less visibility
All the team members may not get the chance to work with all microservices, so they will not have complete clarity on the project.
note: Mulitple microservices will be divided into multiple teams(BNG,CHENNAI,HYD,....)

d. Pack of Cards Problem
If any main microservice is failed, then the dependent microservices is going to fail, so it is not possible to process the request.

refer: Challenges of Microservices.png

Advantages of Microservices
==========
1. Easy maintainence(only few requirements are developed as single project becoz we follow divide and conquer strategy)
2. Faster releases => As we are developing limited functionality very quickly we can complete development and testing and we can release microservices.
3. Parallel development => Multiple teams can work with Multiple microservices parallely if there is no dependency which reduces the development time.
4. Adopting to new technology(Cross platform(developing Microservices in different language))
=> There is no rule saying that all Microservices should be implemented in same programming language.
=> We can use different technology to develop different microservices.
5. Easy Scaling => We can scale our microservices based project easily(improving the capacity of the servers)

Microservices Architecture

Note : There is no fixed Microservices Architecture, they would be using as per their comfort and requirement.

Generalized Microservice Architecture

Most of the developers follow this architecture to develop the projects.

MicroServices Architecture Components
- a. Service Registry(Eureka Server)
- b. Micro-Services (RestApi's)
- c. API Gateway(Zuul proxy)

What is Service Registry?

=> Registry(just like school teacher maintaining the register)

=> Service Registry is used to register services(API's) available in our projects.

=> It provides the dash board with services information like
- a. status
- b. health
- c. URL etc,...

=> Service means one ReST api.

eg: Assume there are 70 services in a project, so we can keep track of those services in "Service Registry".

=> We can use Eureka Server as a "Service Registry".

What is Services?

One Service is called as "RestApi".

ReST API's are called as "Services in microservices based project".

ReST API's contains buisness logic to perform buisness operations.

As part of Buisness operation, One ReST Api can communicate with another ReST Api.

Microservice architecture means collection of ReSTApi's.

What is API Gateway?

An API Gateway is an API management tool that sits b/w client and collection of back end services(RestApi's).

API Gateway acts as single entry point for all clients.

In API Gateway we can write the logic to filter the user request.

eg: Zuul proxy.

refer: MicroService Architecture.png

How to register with Eureka Server?

Service Registry is used to register all services available in the project(RestApi's).

What is the advantages of register the services in Eureka Server?

If we register the services, then in the Service Registery DashBoard we would get to know the name of the service,health of the services,

URL of the Services,....

Eurekha Server is been used as Service Registery.

Eurekha Server is runing on 8761 then Discovery clients can auto register with Eureka Server.

Developing the Eurekha Server(default port no : 8761)
==========
Create a Spring boot with the following dependecies
a. spring-boot-starter-web
b. spring-cloud-netflix-eureka-server

use the following annotations
a. @EnableEurekaServer at boot start class to indicate it as Eurkeha Server class

Make the following changes in application.properties file
eureka.client.registerWithEureka = false
eureka.client.fetchRegistry = false
server.port = 8761

run the application, open the browser and type as http://localhost:8761 to see eureka server DashBoard

pom.xml
=======

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
	<groupId>org.springframework.cloud</groupId>
	<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Develop First Service as Eurekha client
==========
Create a Spring boot with the following dependecies
a. spring-boot-starter-web
b. spring-cloud-netflix-eureka-client

use the following annotations
a. @EnableDiscoveryClient at boot start class

Create RestController with the requried methods
Configure embeded container port no and application name
Run our application.
Verify Eurekha Server DashBoard(client should be registered)

Develop Second Service as Eureka Client
==========
Create a Spring boot with the following dependecies
a. spring-boot-starter-web
b. spring-cloud-netflix-eureka-client

use the following annotations
a. @EnableDiscoveryClient at boot start class

Create RestController with the requried methods
Configure embeded container port no and application name
Run our application.
Verify Eurekha Server DashBoard(client should be registered)

Note: Before running client application, make sure Eurekha server project is running

If v want 2 change the port no of eurekha server,we use d following code

a. Service-Register-Eurekha-Server

application.properties
======================
```
eureka.client.registerWithEureka = false
eureka.client.fetchRegistry = false
server.port = 9761
```

b. Eurekha-Client-HI-Service

application.properties
======================
```
server.port=9999
spring.application.name=HI-SERVICE
eureka.client.service-url.defaultZone = ${DISCOVERY_URL:
http://localhost:9761}/eureka/
```

Now run server first by typing the url
http://localhost:9761/

Now run client by typing the url
http://localhost:9999/

We have Created 3 Applications
a. Service-Registry (PortNO: 9761)
b. HI-SERVICE(PortNo: 1111 Eurekha Client-1)
c. HELLO-SERVICE(PortNo: 2222 Eurekha Client-2)

InterService Communication
==========================
In our project, if one microservice wants to communicate with another microservice then it is called as "Inter Service Communication".
Note: Both microservice belongs to same project.

=> We can use RestClient logic to access one Rest Api.
eg: RestTemplate,WebClient,...

=> Feign client will be used if both microservices belongs to same project.
=> Advnatages of Feign Client is we can specify the "service name" not the URL.

Dependencies of Feign Client
============================
pom.xml
=======
```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-openfeign</artifactId>
```

```
</dependency>
```

code
====
#1.
```
package in.ineuron.client;

import org.springframework.cloud.openfeign.FeignClient;

@FeignClient(name = "HELLO-SERVICE") // name is recomended over URL(becoz the URL
will change when we opt for load balancer)
public interface HelloClient {

	@GetMapping("/hello/{name}")
	public String invokeHelloService(@PathVariable String name);
}
```

Note: Through name we are not writing URL, so through service registry the services are loosely coupled.