Monolith Architecture

==========

-> In our application, we will have several modules and several components also will be available

Components

-> Presentation components : Responsible for User Interface

-> Web Components : Responsible to handle user requests

-> Business components : Responsible to execute business logic

-> Persistence components : Responsible for DB operations

-> Integration components : Responsbile webservices/rest api communications

-> Authorization components : Responsible to authorizing user

-> Notification Components : Responsible for sending email/mobile msg notifications whenver required

-> If we develop all these components as a single project then it is called as Monolith Architecture Based Project.

Benefits Of Monolith Architecture

-> Simple for development: At initial stage it is very easy

-> Easy for testing : End to end testing we can perform

-> Easy for deployment : One war file only we have to deploy

-> Easy for scaling : Mulitple server we can spin easily

Drawbacks of Monolith Architecture

-> Maintence : If application is too large and complex to understand,

making changes for enhancements and CR is very difficult (Lot of Impact Analysis is required)

-> Adopting to new technology is very difficult

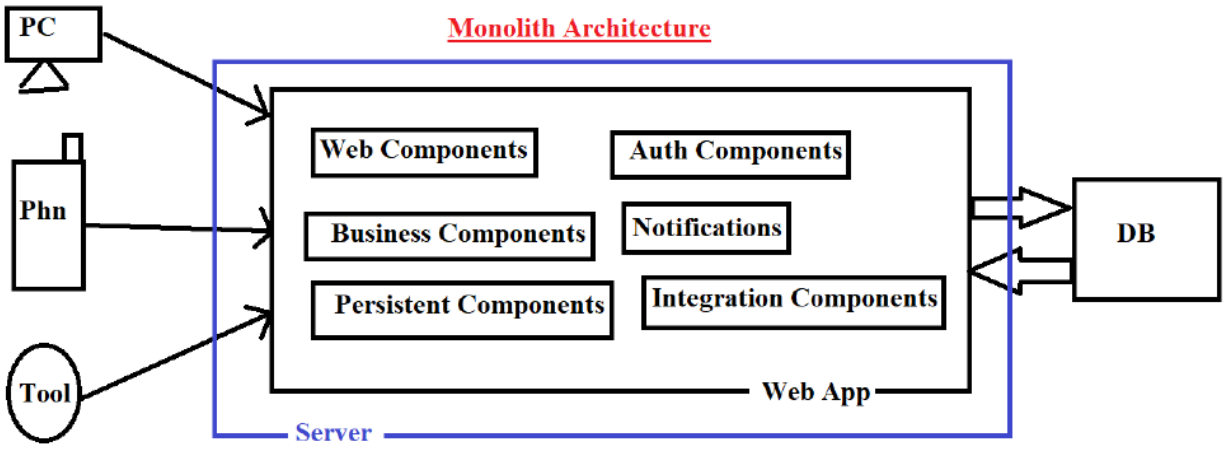
-> The size of the application can increase application startup time

-> Realiability : If there is bug in one module, then it leads to entire application will go down.

-> We must re-deploy entire application when we make some changes to code

-> Quick releases are not possible

-> New team members can't understand the project easily



Today's session : Load Balancer & LB Algos

-> When all components are in same application, all requests comes to same server then burden will increase on server.

-> When burden increased on server, it will process requests slowly sometimes server might get crash also.

-> To reduce burden on the server, people will use Load Balacners for applications.

-> Our application will be deployed to mutlitple servers and all those

servers will be connected Load balancer.

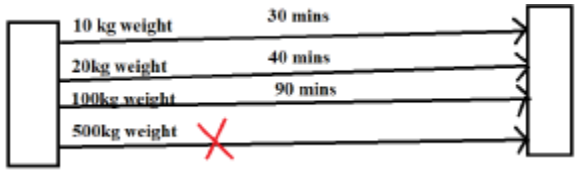
How Load Balancer will distribute the load?

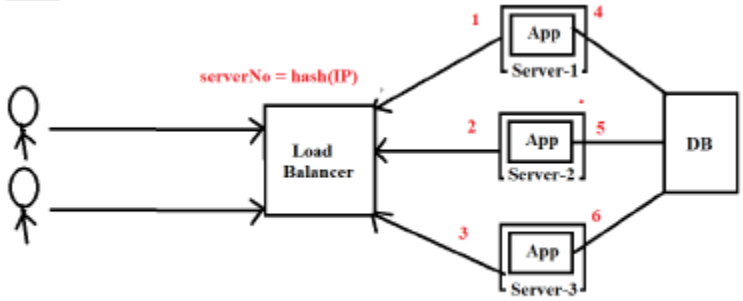
LBR will use alogorithm to distribute incoming to requests to servers.

1) Round Robbin

2) Sticky Session

3) IP Hashing etc....





<u>Today session: Microservices Introduction</u>

-> Microservice is not a technology

-> Microservice is not a framwork

-> Microservie is not an API

-> Microservice is an Architectural Design Pattern

-> Microservices design pattern came into market to avoid the problems of Monolithic Architecture

Challenges with Microservices

1) Bounded Context : It is very difficult to decide boundary for one microsevice. Deciding which functionality should be developed in one microservice is difficult.

2) Lot of configuration : As we are developing one project as multiple services, in every service we have to configuration like DB config, Actuator config, SMTP config, logging config etc....

3) Less visibility

4) Pack of cards problem

Today's session : Challenges & Benefits

-> This design patterns talks about how to design our project architecture and how to develop our project

-> The main aim of Microservices architecture is

'Divide & Conconqure'

Challenges

1) Bounded Context : Deciding functionality for one microservice is difficult

2) Lot Of configuration: In Every Microservice project we have to do configurations like DB config, Kafka Config, Security Config, Log config etc...

3) Visibility : All the team members may not get the chance to work with all microservice hence they will not have complete clarity on project.

4) Pack of cards problem : If any main microservice is failed then the dependent microservices also going to be failed.

Advantages of Microservies

1) Easy Maintenence : As we are dividing our project functionality into multiple microservices it is easy to maintain

2) Faster Releases : As we are developing limited functionality very quickly we can complete development and testing then immediatley we can release our microservices.

3) Parallel Development : Multiple teams can work on multiple services development paralelly.

4) Adopting new technology : There is no rule saying that all microservices should be implemented using same programming language. We can use different technologies to develop our Microservices.

5) Easy Scaling : We can scale our microservices based projects easily.

Microservices Architecture

Note: There is no fixed architecture for this. People are using microservices architecture according to their comfort.

-> Here we will talk about generalized microservices architecture that most of the people will follow in projet development.

Microservices Architecture Components

1) Service Registry (Eureka Server)

2) Microservices (Rest API s)

3) API Gateway (Zuul Proxy)

Microservices Architecture

-> There is no fixed architecture availalble for microservices

-> People are using Microservice Architecture as per their convinience.

Generalized Microservices Architecture

1) Service Registry

2) Services (Rest APIs)

3) API Gateway

What is Service Registry?

-> Service Registry is used to register services available in our project.

-> Service Registry will provide a dashboard with services information

like Status, Health and URL etc...

-> One service nothing but one Rest API

-> We can use Eureka Server as a Servie Registry

Services

-> Rest APIs are called as Services in Microservice Based Project

-> Rest APIs contains business logic to perform Business operations

-> As part of business operation one rest api can communicate with

another rest api.

-> Microservice Based Project Means it is collection of Rest APIs

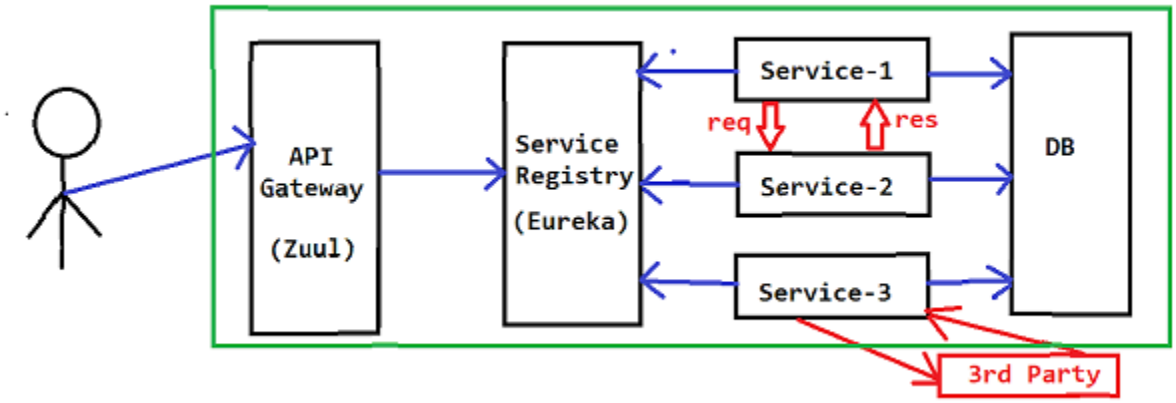
API Gateway

-> An API gateway is an API management tool that sits between a client and collection of backend services.

-> Backend services are nothing but Rest apis.

-> API gateways acts as Single Entrypoint for all clients

-> In API gateway we can write the logic to filter user requests



Today's session : Service Registry & Client applications

-> Service Registry is used to maintain list of services available in our project.

-> Service Registry will provide below details of registered services

1) Service Name

2) Service Status

3) Service EndpointURL

-> We can use Eureka Server as a Service Registry

Procedure to create service registry

-> Create Spring Boot application with below dependencies

1)spring-boot-starter-web

2)netflix-eureka-server (this is from spring cloud)

-> At spring boot start class specify @EnableEurekaServer annotation

-> Configure Embedded Container port number as 8761.

Note: 8761 is default port number for Eureka. If port is 8761, clients can auto detect eureka server and will get registered. If port number is not 8761 then we have to register clients manually.

-> Run our Spring Boot Application and Access Eureka Dashboard using below URL

http://localhost:8761/

Procedure to create Service as Eureka Client

1) Create Spring Boot Project with below dependencies

1) spring-starter-web

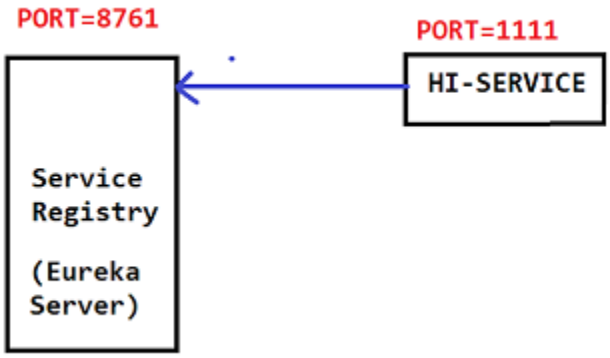
2) spring-cloud-netflix-eureka-client

2) Configure @EnableDiscoveryClient annotation at Spring Boot start class

3) Create Rest Controller with Required methods

4) Change Embedded Container Port No if Required and Run application

5) Verify Eureka Dashboard (Client should be registered)



Eureka Client

-> Service registry is used to register all services available in our project (REST APIs)

-> Service registry will provide a dashboard with details of services

which are registered (name, status & URL)

-> Eureka Server we are using as a Service Registry

-> If Eureka is running on 8761 then then Discovery Clients can auto register with Eureka Server

Developing Second Service as Eureka Client

-> Create Spring Boot app with below dependencies

- spring-boot-starter-web

- spring-cloud-netflix-eureka-client

-> Use @EnableDiscoveryClient annotation at boot start class

-> Create RestController with required methods

-> Configure embedded container port number & application name

-> Run our client application

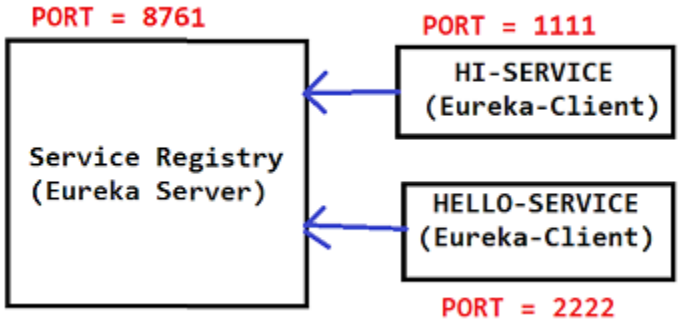
-> Verify Eureka Server Dashboard (Client should be registered)

Note: Before running client application, make sure Eureka Server Project is Running.

----------Eureka Client yml file----------

```
server:
        port: 2222
spring:
        application:
                name: HELLO-SERVICE
eureka:
        client:
                service-url:
                        defaultZone: ${DISCOVERY_URL:http://localhost:9761}/eureka/
```

Note: 9761 is Service Registry Project Port Number



Today's class : FeignClient for Interservice Communication

-> We have created below 3 applications

1) Service-Registry (Port Number : 8761) - Eureka Server

2) HI-SERVICE (Port Number : 1111) - Eureka Client-1

3) HELLO-SERVICE (Port NumBer : 2222) - Eureka Client-2

Inter-Service Communication

In our project, if one micro-services accessing another microservice then it is called as Inter-service communication.

Note: Both Microservices are available in Same Project.

-> We can use Rest Client logic to access one REst API

Working with FeginClient

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

```
package com.ashokit.client;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
@FeignClient(name = "HELLO-SERVICE")
public interface HelloClient {
@GetMapping(value = "/hello/{name}")
public String invokeHelloService(@PathVariable("name") String
name);
}
```

-> When we use FeignClient, we can specify name of the service instead of URL for communication

Steps to Configure FeignClient

1) Add spring-cloud-starter-openfeign in pom.xml

2) Enable Feign Clients using @EnableFiegnClients annotation at boot start class

3) Create a interface for FeginClient and use @FeignClient annotation

Use Case

Develop a project to Calculate stock price

Stock App implementation

StockPriceService : This service is responsible to take company name as an input and return stock price of that company as output . (It is a rest api). This service will maintain companies stock prices in Embedded db.

Steps to develop

1) Create boot application with below dependencies

a) spring-boot-starter-web

b) h2

c) project lombok

d) devtools

e) spring-boot-starter-data-jpa

f) swagger & swagger-ui

2) Create Entity class & Repository interface

3) Create data.sql file with insert queries (src/main/resources)

4) Create Service interface & implementation

5) Create Rest Controller

6) Configure Swagger documentation

Exception Handling using @ControllerAdvice

@ControllerAdvice

```
public class StockPriceServiceExceptionMapper{

@ExceptionHandler(value=CompanyNameNotFoundException.class)

public ResponseEntity<String> handleNotFoundException(){

String msg = "Company Name Not Found";

return new ResponseEntity(msg, HttpStatus.OK);

}

}
```

StockAmountCalculationService

This rest api will take company name and quantity as input. It should get company stock price from 'StockPriceService' api. Then it has to calculate total price for given quantity and it has to return that as an output.

Steps to develop Stock-Amt-Calc-Service

---

1) create spring boot application with below dependencies

a)spring-boot-starter-web

b)spring-cloud-starter-open-feign

c)devtools

2) Create FeignClient to access 'StockPriceService'

3) Create RestController with required method

4) Configure port number, application-name in yml file

Steps to Develop UI application

---

1) Create Spring Boot application with below dependencies

1) spring-boot-starter-webflux

2) tomcat embed jasper

3) devtools

2) Create Service class to access 'Stock-Calculation-Service' using

webclient

3) Create Controller with required methods

4) Create View File

5) configure port number and view resolver

6) Run our application

API Gateway

-> API gateway will act as Entry Point for all microservices available in our project.

-> API Gateway will be called as Edge Microservice.

-> API Gateway is an API management tool which handles all incoming

requests to our backend service

-> API Gateway sits between client requests and backend services

-> Spring Cloud Netflix library provided 'Zuul Proxy' as an API Gateway. It is an open source.

API Gateway Implementation

-> If we don't have API Gateway then we need to implement request filtering logic in every microservice. It increases Maintenense cost of the project.

-> Spring Cloud Netflix Library provided Zuul Proxy as an API Gateway.

-> Zuul Proxy is open source API Gateway

-> Apigee is commercial API Gateway provided by Google

Steps to develop API Gateway

1) Create Spring Boot Application with below dependencies

a) spring-boot-starter-web

b) spring-cloud-netflix-zuul

c) spring-cloud-netflix-eureka-client

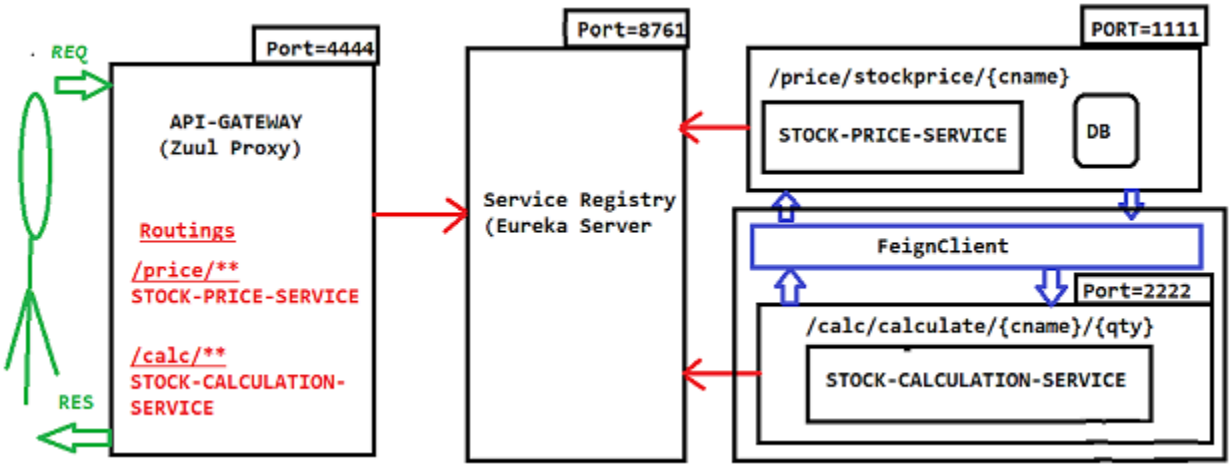
2) Configure @EnableZuulProxy & @EnableDiscoveryClient annotations at

Spring Boot start class

3) Configure API Gateway with Eureka Server Registration in yml

4) Configure Routing to access backend services in yml

5) Configure application name and port number



Mini Project Execution Flow

Stock Application Using Microservices Architecture

SERVICE-REGISTRY : http://localhost:8761/

STOCK-PRICE-SERVICE : http://localhost:1111/price/stockprice/{cname}

STOCK-CALC-SERVICE: http://localhost:2222/calc/calculate/{cname}/{qty}

API-GATEWAY : http://localhost:4444/api/

URLS given to our clients To Get Company Price :

http://localhost:4444/api/price/price/stockprice/{cname}

To get total cost :

http://localhost:4444/api/calc/calc/calculate/{cname}/{qty}

Actuators

-> We are developing applications using Spring Boot and We are deploying our applications in servers

-> Once application is deployed, several users will access our applications

-> When our application is running in production, it is very important to monitor our application

What is Application Monitoring

-> Health Check

-> Beans check

-> configProps check

-> Heap Dump

-> Thread Dump

-> Http Trace etc......

-> To monitor our applications easily, Spring Boot provided Actuators.

-> Actuators are used to provide 'Production Ready Features' for our application.

Working with Actuators

-> We need to add spring-boot-starter-actuator dependency in pom.xml file

-> Actuators provided several pre-defined endpoints to monitor our application. They are

1) health
2) info
3) beans
4) mappings
5) configProps
6) httpTrace
7) heapdump
8) threaddump
9) shutdown (It is special endpoint)

-> To access actuator endpoints we should use /actuator as prefix

Ex : http://localhost:9090/actuator/health

-> In Spring Boot 1.x v '/actuator' should not be there in URL

-> In Spring Boot 2.x v '/actuator' is mandatory in URL to access Actuator endpoints

-> When we add 'actuator' starter, by default it will expose 2 endpoints they are 'health' and 'info' (we can access them directley).

-> To expose all the endpoints we shuld write below property

```
management:
  endpoints:
    web:
      exposure:
        include: *
        exclude: beans
```

-> By default health & info endpoints are exposed.

-> heapdump endpoint is used to down jvm heap details. To analyze heap dump files we can use MAT (Memory analyzer tool).

-> threaddump endpoint provides information about threads available in our application.

-> Shutdown endpoint is a special endpoint. It is used to stop the application.

-> This shutdown endpoint by deafult in disable state.

-> shutdown endpoint is binded to HTTP POST Request.

Spring Boot Admin Server & Admin Client

-> In Microservices Architecture based project, we will have several services (REST APIs)

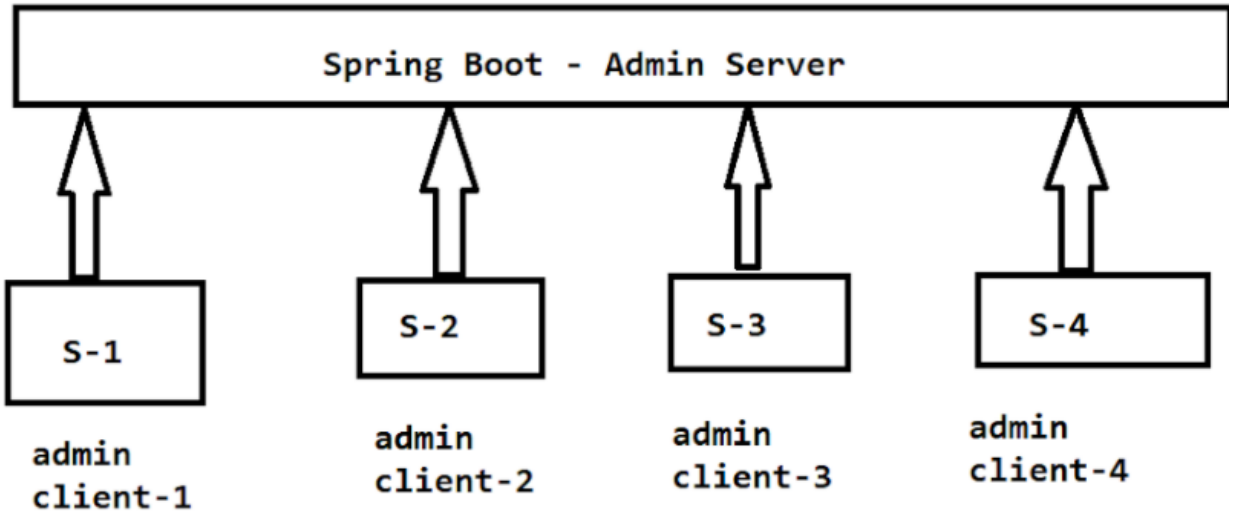
-> To monitor Rest APIs, we will enable and expose actuator endpoints

-> If we have more no.of services, it will would be very difficult to monitor and manage all our services available in project.

*** To overcome this problem, Spring Boot Provided Admin Server & Admin Client Concept **

-> If we use Admin Server, it will provide beautiful user interface to monitor and manage our REST APIs.

Note: Our Rest apis should be registered with Admin server then our rest api is called as Spring Boot Admin Client.



Yesterday's session : Admin Server & Client Introduction

Today's session : Example on Admin Server & Client

-> Spring Boot Admin server is used monitor and manage our client applications (rest apis)

-> By using Admin server at one place we can monitor and manage all our services

Steps to Create Admin Server Application

-> Create Spring Boot Application with below dependencies

1) spring-boot-starter-web
2) spring-boot-starter-admin-server

-> Configure @EnableAdminServer in Spring Boot Start class

-> Configure embedded container port number in application.yml file

-> Run application and access dashboard

Note: By default Admin server will provide UI for monitoring and managing registered clients.

Steps to develop Admin Server Client Application

-> Create Spring Boot Application with below dependencies

1) spring-boot-starter-web
2) spring-boot-starter-admin-client
3) spring-boot-starter-actuator

-> Configure below properties in application.yml file

1) port number
2) application name
3) register with admin server
4) expose actuator endpoints

-> Create Rest Controller with Required Methods

-> Run the application and verify Admin Server Dashboard

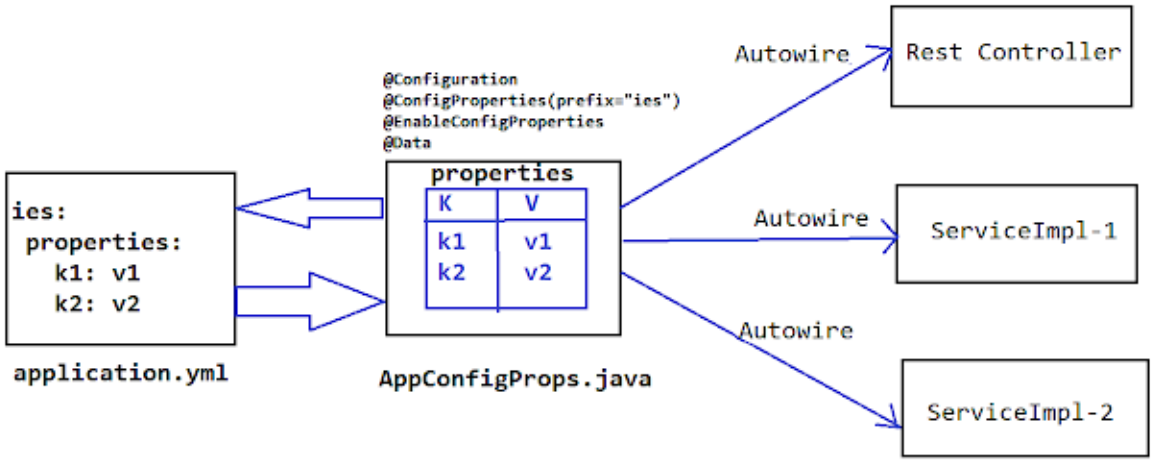
Note: Client application should be displayed in Admin Server Dashboard.

.properties

```
server.port = 9090

spring.application.name=CLIENT-ONE

spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp

demo.welcomeMsg=Hello, How are you?
demo.greetMsg=Good Morning..!!
```

.yml

```
server:
   port: 9090

spring:
  application:
       name: CLIENT-ONE
  mvc:
     view:
        prefix: /views/
        suffix: .jsp

demo:
   welcomeMsg: Hello, How are you>
   greetMsg: Good Morning..!!
```



Last session : Working with Yml or Yaml files

YAML -> Yet Another Markup Language

-> YML files are widely used format for configuration properties

-> Spring Boot is having very good integration with YML files.

-> In Spring Boot we can use .properties and .yml files for configuration properties

-> When we create Spring Boot application, by default it will provide application.properties file

-> We can read YML file data into java bean.

-> Spring Cloud Provied Config Server

-> Config Server is used to Externalize Configuration Properties from our application.

Scenario:> Microservice ----> Config-Server ----> Git Repo

Steps to work with Config Server Project
---
1) Create Spring Boot application with below dependencies

a) spring-boot-starter-web
b) spring-cloud-config-server

2) Configure @EnableConfigServer annotation at Spring Boot Start class

3) Create GIT repository and create config-properties files in repository then configure properties in the form of key and vale format.

EX: https://github.com/TEK-Leads/Config-Properties.git

4) Configure below properties in application.properties or application.yml file

a) server-port
b) spring.cloud.config.server.git.uri=<repo-url>
c) spring.cloud.config.server.git.cloneOnStart=true
d) management.security.enabled=false

Steps to develop Config Server Client
---
1) Create Spring Boot application with below dependencies

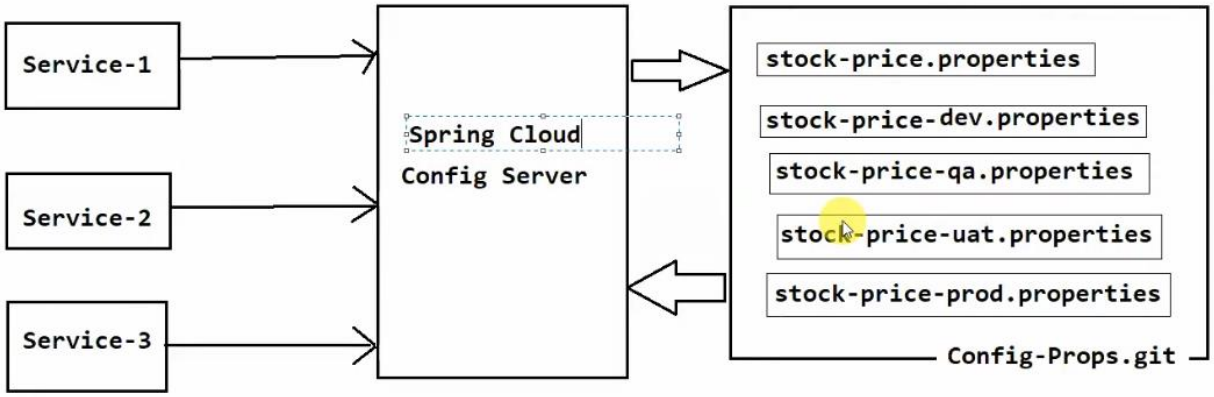
a) spring-boot-starter-web
2) spring-cloud-starter-config

2) Configure below properties in application.properties or application.yml file

a) server-port
b) application-name
c) activate profile
d) spring.cloud.config.uri (It is used to connect with config server)

3) Inject properties values into variables wherever it is required

```
Ex:   @Value("${msg}")
      String msg;
```



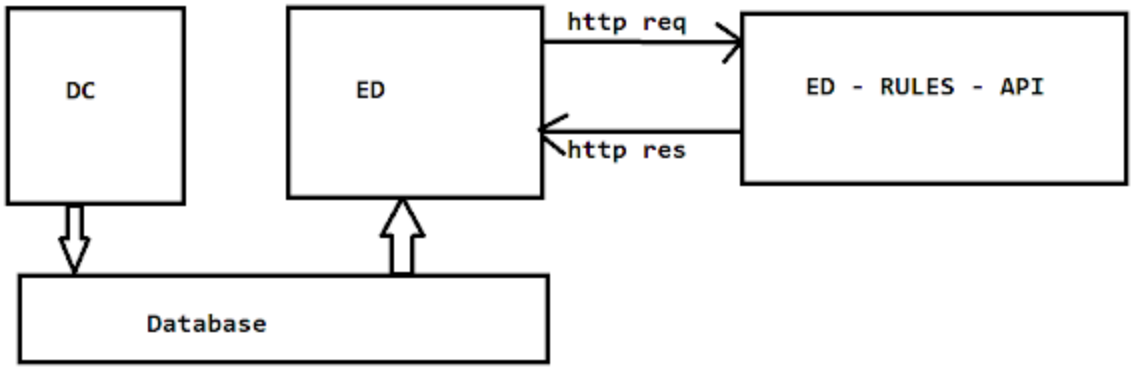
SNAP
---
page-1 ICS
page-2 TXS
page-3 RSS
page-4 PTS

..

Page50-SUS

CCAP
---
page-1
page

Medicaid



esterday's session: Config Server In microservices
---

application.properties or application.yml file
---
bootstrap.properties or bootstrap.yml file

Q) When to use application.yml and when to use bootstrap.yml?

If we have both application.yml and bootstrap.yml in our application, Spring Boot will load bootstrap.yml first.

-> In bootstrap.yml we will configure, application-name, config-server-url etc.

Circuit Breaker

-> Circuit Breaker is used to implement Foult Tolerence.
-> In Microservice architecture, multiple service will available.
-> If one microservice is not able to process the request, then request processing can’t be completed.

To implement Circuit breaker, we will use Spring Cloud Netflix Hystrix Library

-> Circuit Breaker is used to protect us from damages.

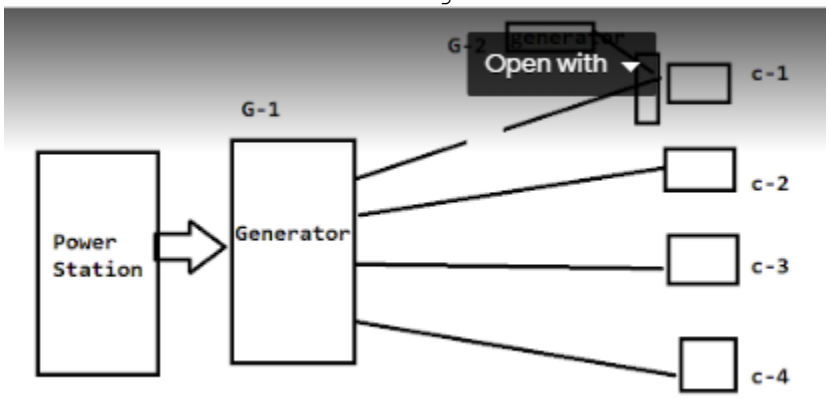
-> A circuit breaker is an automatically operated electrical switch designed to protect an electrical circuit from damage caused by excess current from an overload or short circuit. Its basic function is to interrupt current flow after a fault is detected.

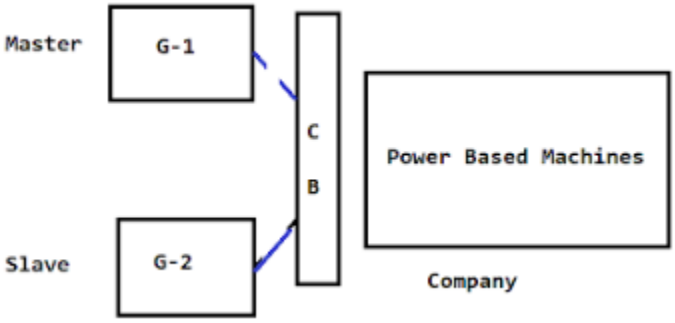
How it is related to Spring Boot ?

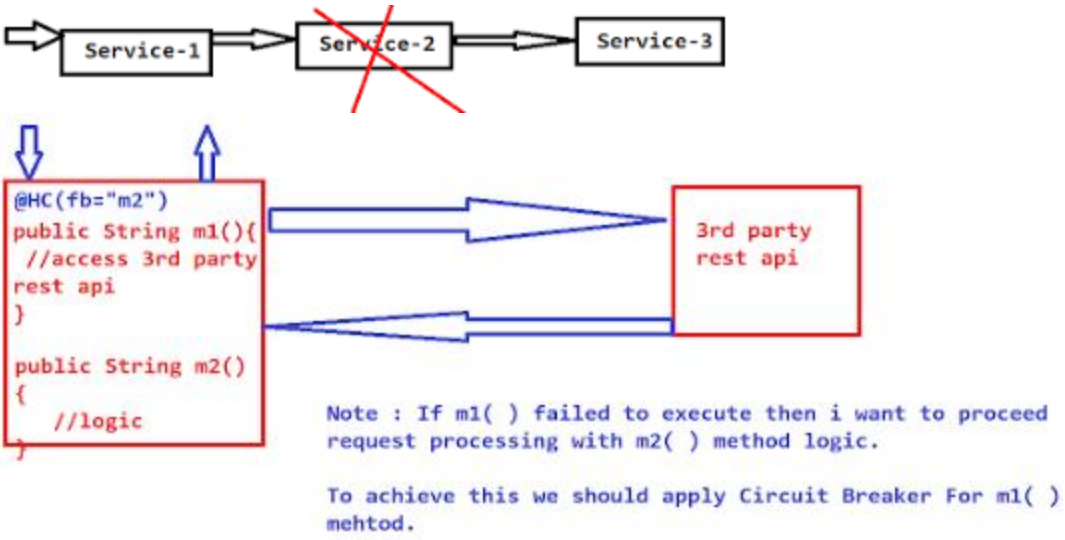
-> Using Spring Boot, we are developing Microservices in Java.

-> When we follow Microservices architecture, we will have several services.

-> Our Services are nothing but Rest APIs.







Steps to work with Circuit Breaker
---

1) Create Spring Boot Application with below dependencies

1) spring-boot-starter-web
2) spring-cloud-starter-netflix-hystrix

2) Configure @EnableCircuitBreaker annotation at Spring Boot Start Class

3) Create Rest Controller with Required Methods (business operation method and fallback method)

4) Configure Circuit Breaker for Business Operation method using below annotation

```
@HystrixCommand(fallBackMethod="...")
```

5) Configure port number and run our application.

In Today's session : Ribbon in Microservices

-> By Using Ribbon we can achieve Client Side Load Balancing.

-> For Running Spring Boot application we will configure Port Number in application.properties or application.yml file

Ex : server.port = 9090

-> We don't configure port number, it will consider 8080 as default port number.

-> Using Environment obj, we can capture on which port number our application is running.

```
@RestController
public class WelcomeRestController {
	@Autowired
	private Environment env;
	@GetMapping("/welcome")
	public String welcomeMsg() {
		String msg = "Welcome to REST API...!!";
		String serverPort = env.getProperty("local.server.port");
		msg = msg.concat(" I am from Server Which is running on Port :: " + serverPort);

		return msg;
	}
}
```

-> We can pass port number to Spring Boot application as VM argument.

Right Click On Project -> Run As -> Run Configuration -> Click on Arguments Tab -> Write below property in VM Arguments -> Click on Apply -> Click Run

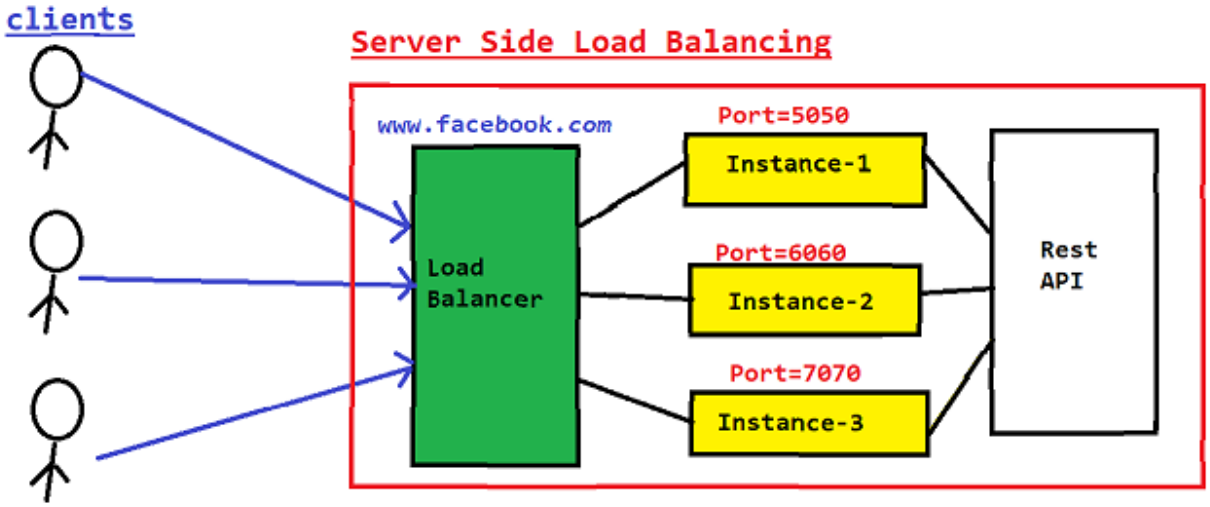
-Dserver.port=9091

==========

First Run with -Dserver.port=5050 ===> http://localhost:5050/welcome

Second Run with -Dserver.port=6060 ===> http://localhost:6060/welcome

Third Run with -Dserver.port=7070 ==> http://localhost:7070/welcome



Working with Ribbon in Stock Market Application

==========

1) Create Service Registry project with Eureak Server

2) Create STOCK-PRICE-SERVICE as Eureka Client

3) Run STOCK-PRICE-SERVICE with 3 instances

4) Create STOCK-CALCULATION-SERVICE USING FeignClient & Ribbon Client