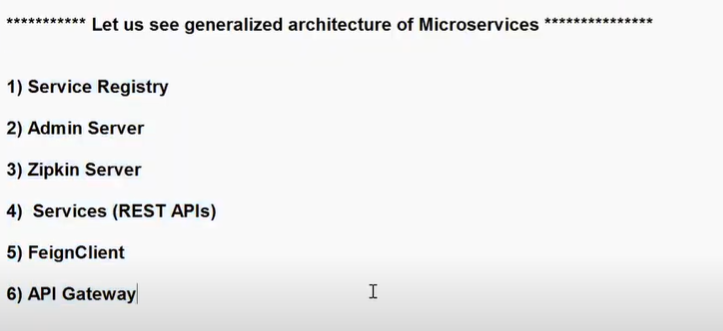
**Microservices Architecture**

Videoplayback(1)





**Service Registry:** Used to register all our backend services**.**Used to maintain list of services, Service names, URLs and status of all services.Eureka can be used as a Service Registry.

Note: **Eureka server provided by Spring Cloud Netflix library**

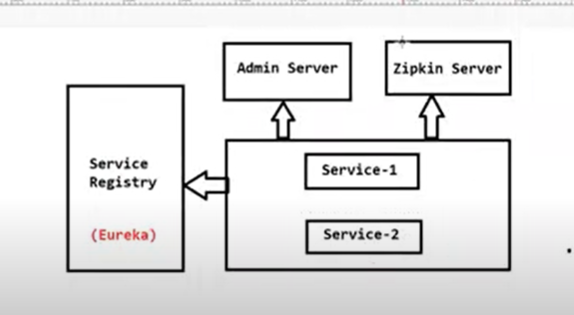
**Admin server:** Used to monitor and manager all services at one place.Currently we are using Actuator.Admin server will provide a User Interface to monitor our application(which has all services)at one place.

Note: **Admin server and Admin client provided by ‘Code Centric’ company(we can integrate with boot).It is a third party service.**

**Zipkin server:** Used for Distributed log tracing**.**We can monitor each request execution details by using Zipkin server.Which request is received, the request is processed by how many services, how much time taken by server to process each request.Provides user interface to monitor application execution details.How many services involved and which service took more time to process the request can be monitored using Zipkin.

Note: **Zipkin is a third party open source server(can be integrated with Spring boot). We need to manually download Zipkin server.**

**Services(REST APIs):** We can have 10 to 100 microservices in an application.These REST APIs are called Microservices.These microservices get registered in Service Registry.These microservices act as client to Admin server, Zipkin server and Service Registry.The actual business logic of the application will be available in services.



**Inter service communication:** There are mutiple REST APIs in a project and one REST API wants to communicate with another REST API and this is called Inter service communication

If one REST API wants to communicate with another REST API , we use a concept called Feign client.

Feign client is used for Inter service communication.If both services belong to same project, then that communication is called Micro services Inter service communication.

Sometimes, our micro services will communicate with external services also.Those external services are third party APIs**.**This is called **Intra-service** communication.

By using **REST TEMPLATE or WEB CLIENT,** you can access third party API.

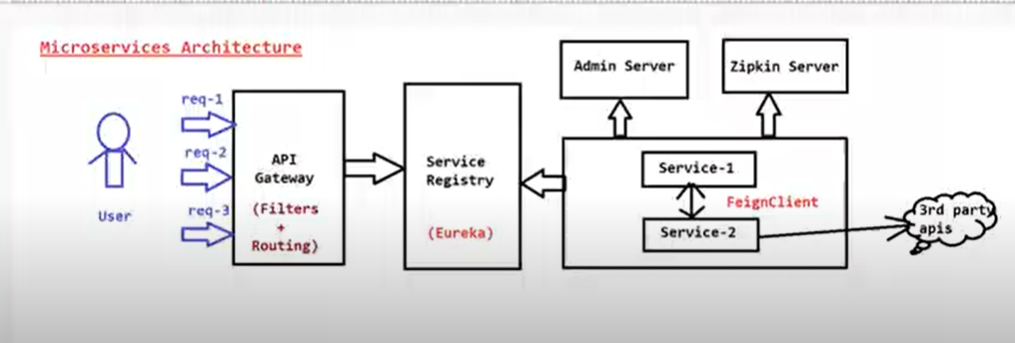
**API Gateway:** It will act as an entry point for micro services based application.It will act as mediator between users and backend APIs.User request will come to API gateway.API gateway can validate the request and will forward the request to Service Registry.Public users will be sending requests to access our application.User request will not go to backend API directly.

**Filters:** Used to decide whether we can process this request or not.If the request is not valid, then we stop the request.

**Routing**:Which request should go to which service in the back end.

In previous versions of Spring, we have **Zuul Proxy** for API gateway but now it got removed from latest version of Spring boot.

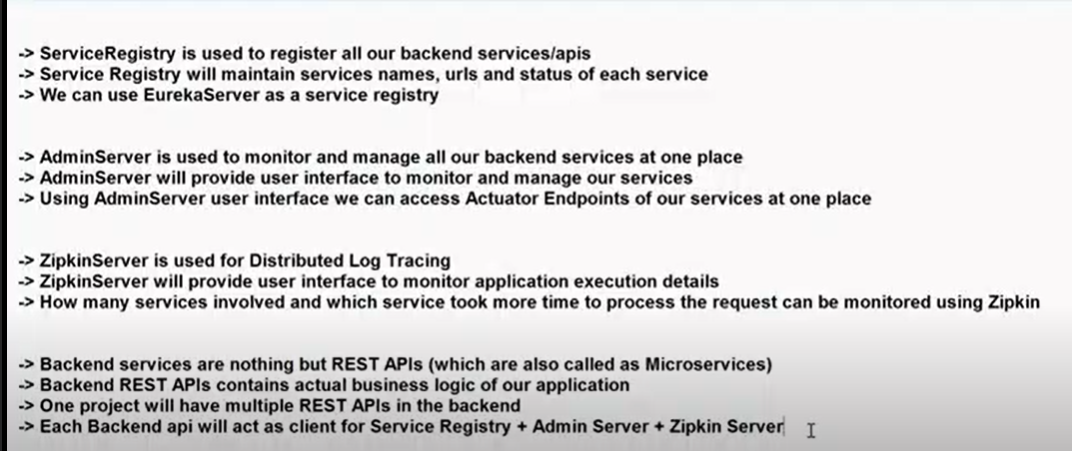
**Spring Cloud gateway** - we can use as API gateway for our application

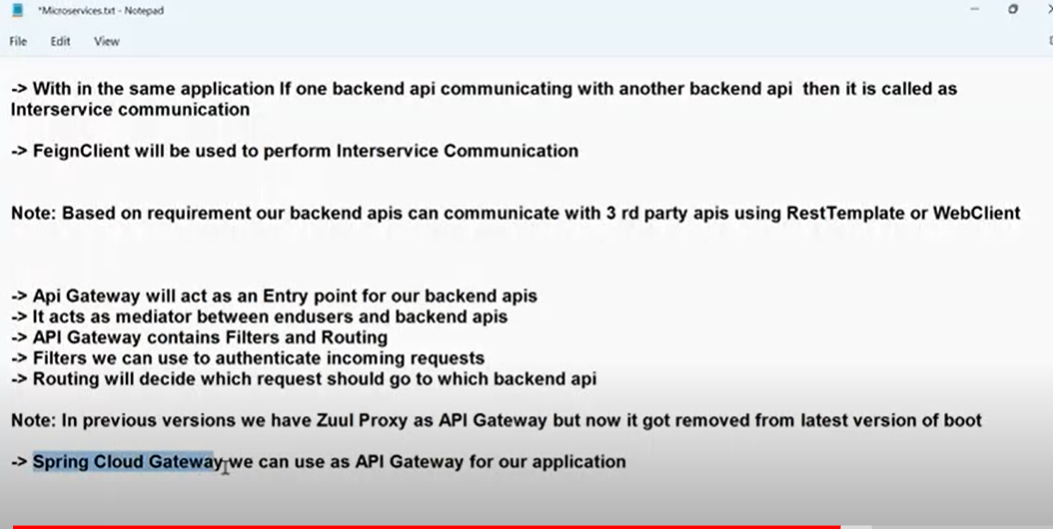


The above picture is Micro service based architecture.But for every micro service project, it is not mandatory to use all the concepts discussed above.It is optional.So for the microservices, we don’t have any fixed architecture.Even if we develop ten REST APIs and we don’t use any Admin server ,Zipkin server,we don’t use any API gateway, even then it is called a Microservice based project.

People are customizing Microservices architecture according to their requirements

Example: You want to know how much Income tax is paid by a person based on his SSN. Here you are communication with SSN application which is a third party application here.Income tax applications is your application.



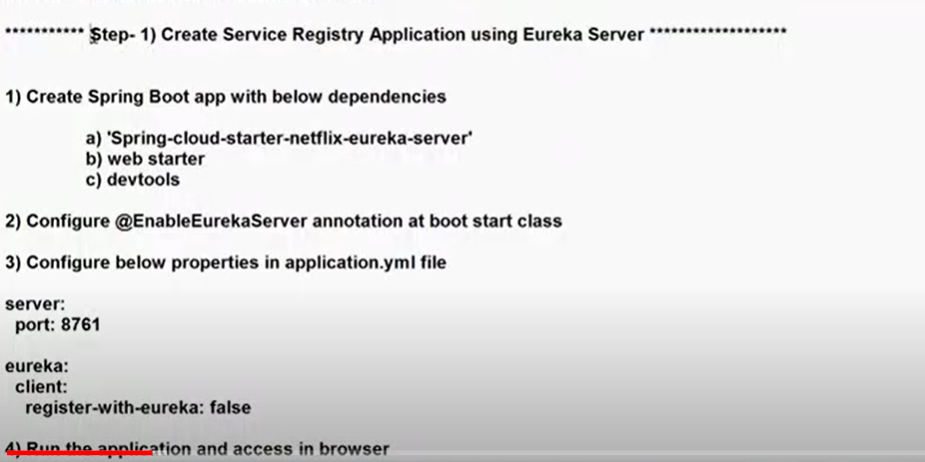


**VIDEOPLAYBACK(2)**

**Micro-services Mini Project Implementation**

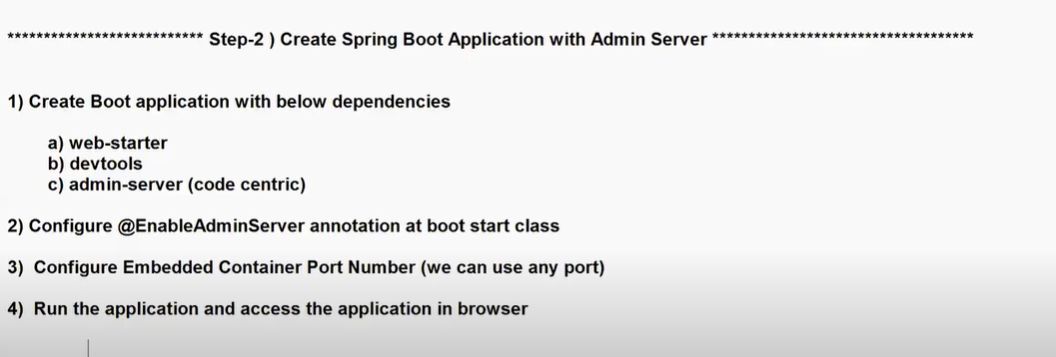
**Server: port-8761 is used as default port for Eureka server.You can also use any port number.If we use 8761 , then we don’t have to explicitly manually register services with Eureka server.Clients can auto-register with Eureka server.**

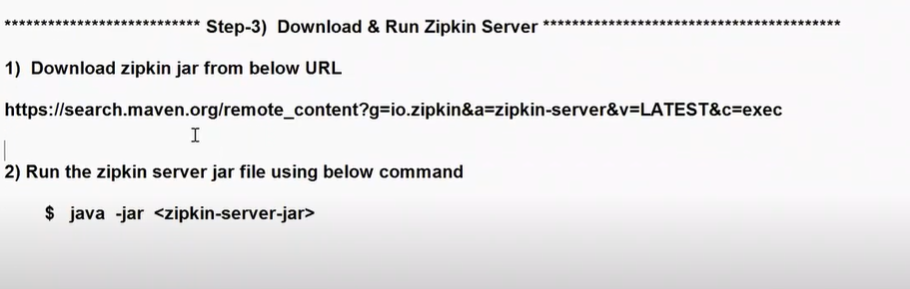
**But if we use any other port number, then we have to manually register services with Eureka server.**



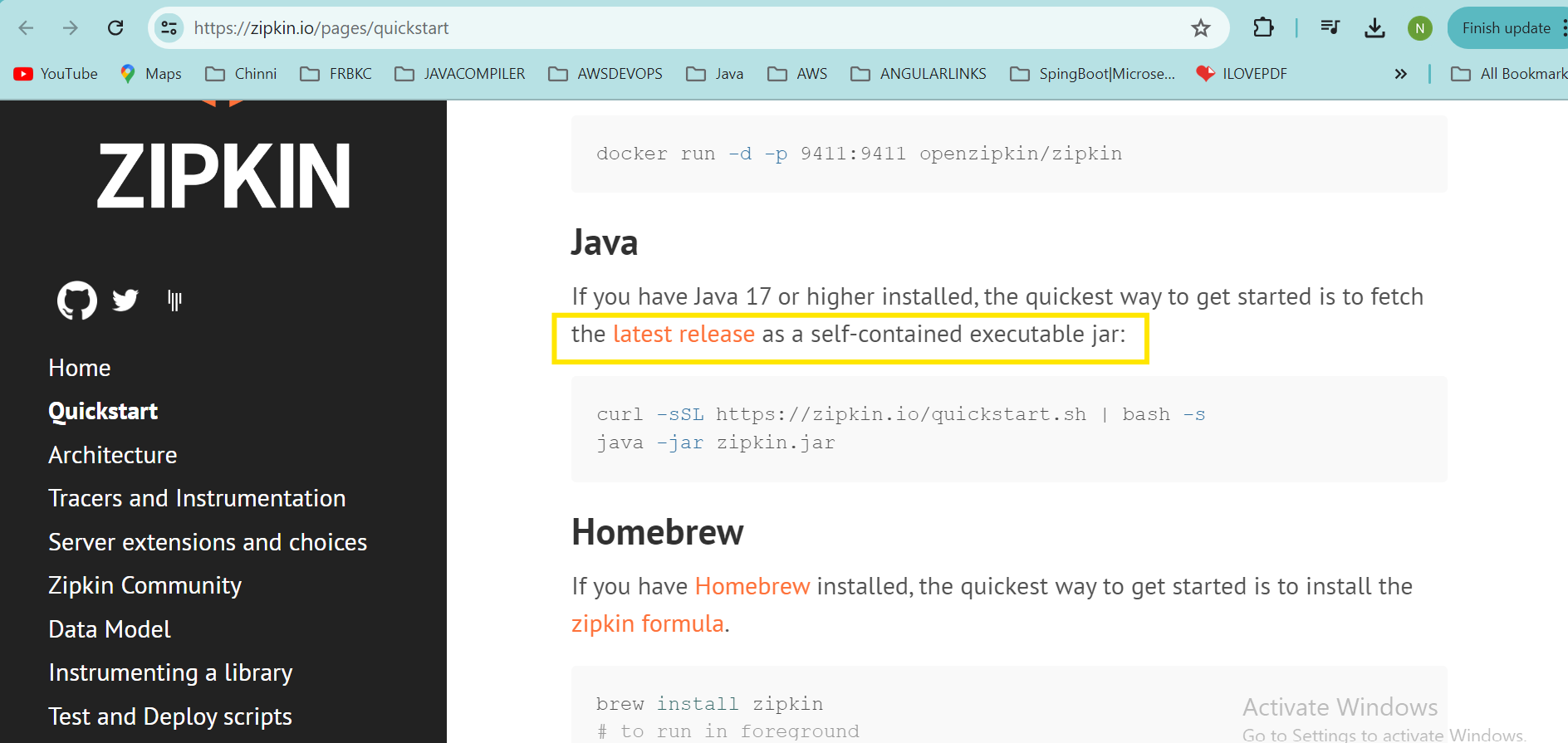
Register-with-eureka:false -- -this property tells that the application itself contains eureka and it need not register with eureka

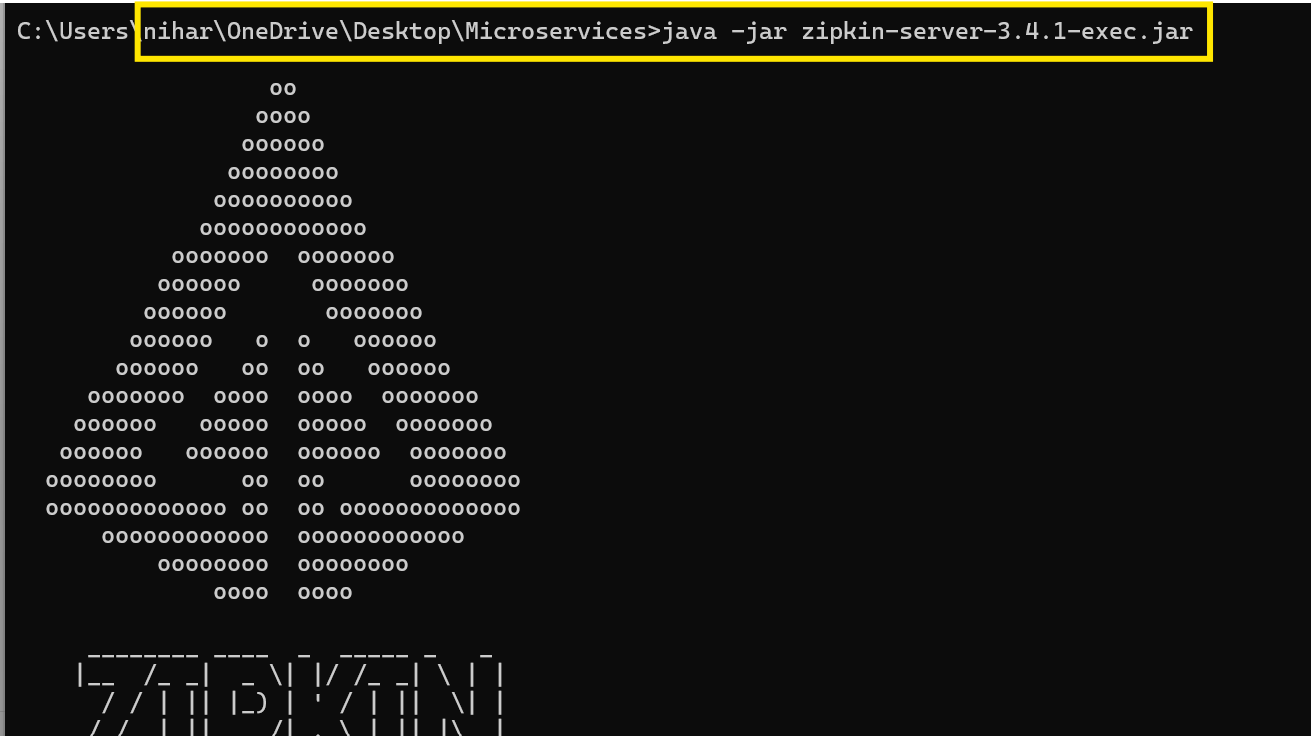
**URL:http://localhost:8761/**

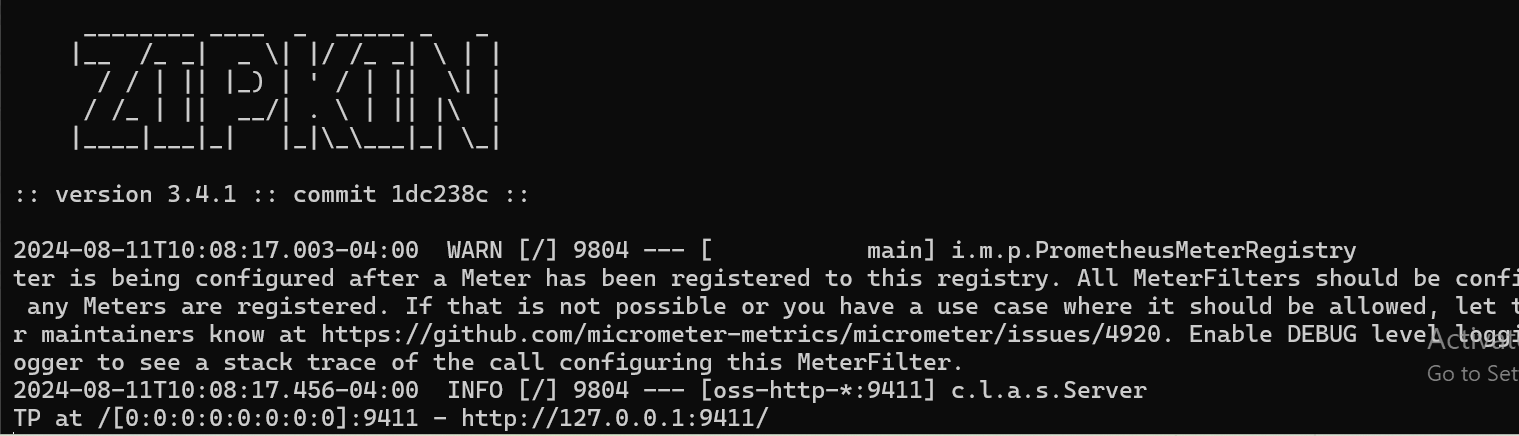




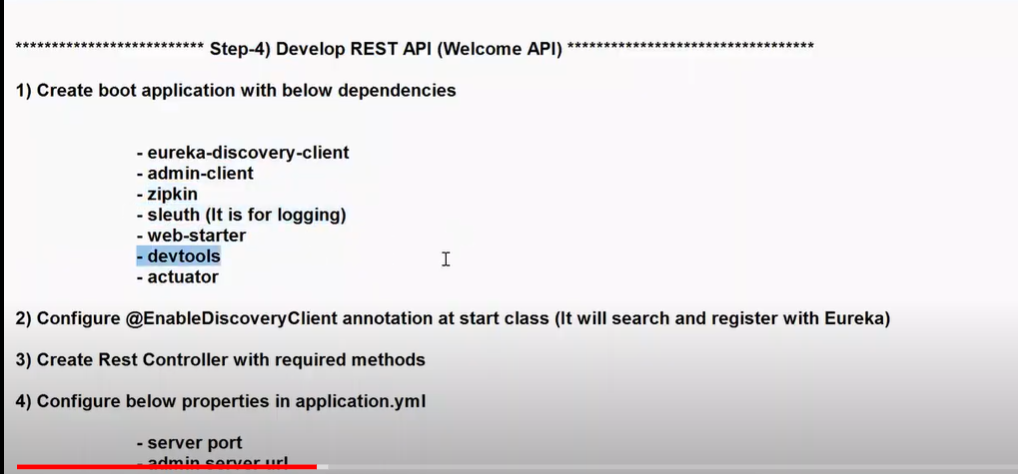
**Zipkin server download**: <https://zipkin.io/pages/quickstart>

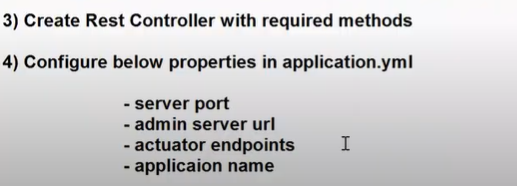


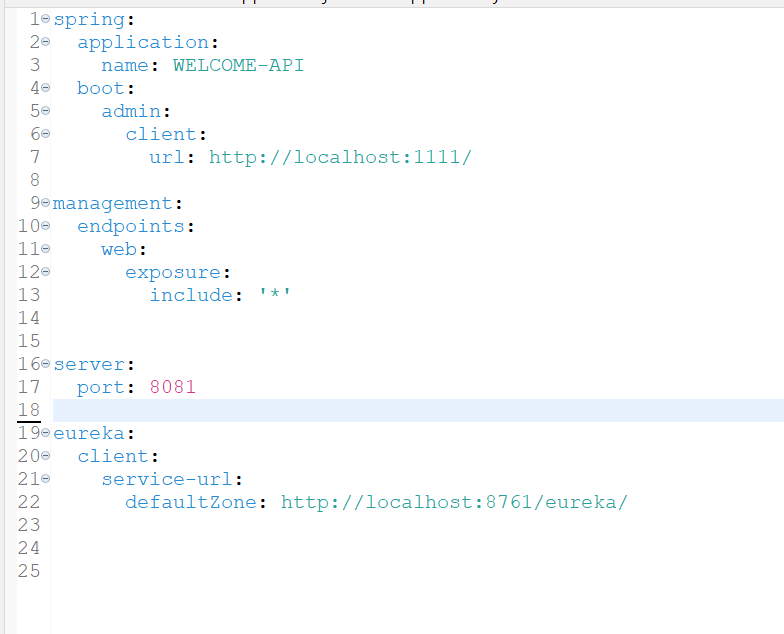




**Sleuth is the dependency which will help Zipkin to trace application details**

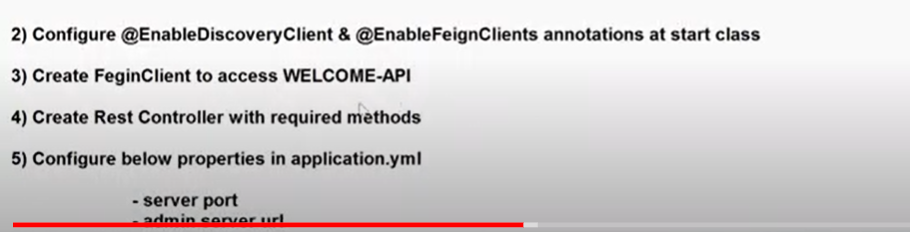








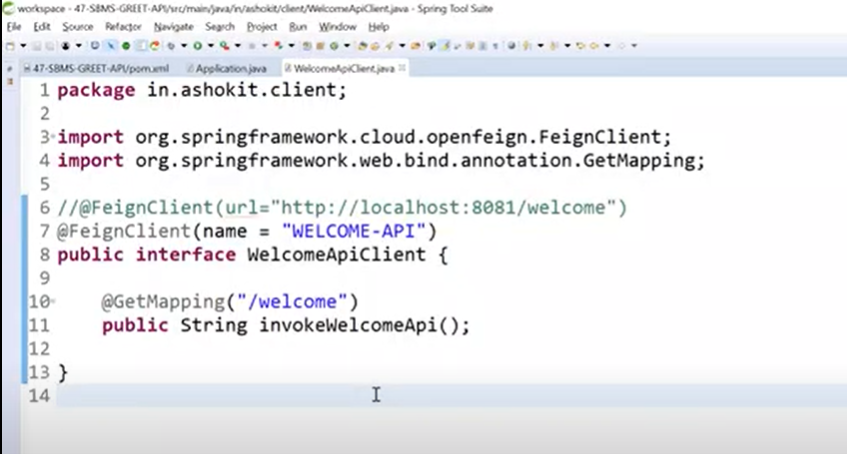




Generally when we access other API through Feign client, it is recommended to use that API by its name , not by URL.If you try to access by its URL, that URL can change in future like if you deploy on AWS.So its recommended to use API by its name than the URL. URLs can change, but service name is fixed.

Here GREET API wants to acess welcome-api. So Feign client is created at GREET API.

If welcome-api wants to access GREET-API, then feign client should be created at welcome-api.



Name = “WELCOME-API” --- by using this, Feign client will talk to service registry and get URL of welcome-api dynamically. For that URL, it is going to send a GET request with “/welcome”

Spring Cloud will provide implementation for this interface at run time.We don’t have to implement this interface or write logic for this interface.Implementation will come in run time.

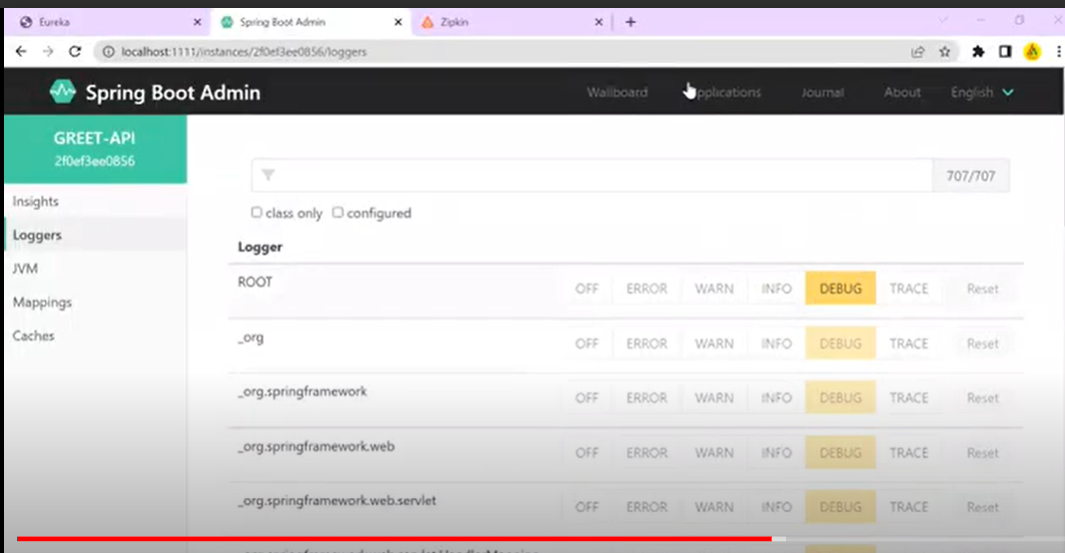
We can access third party APIs also by using Feign client, but we need to give URLs.

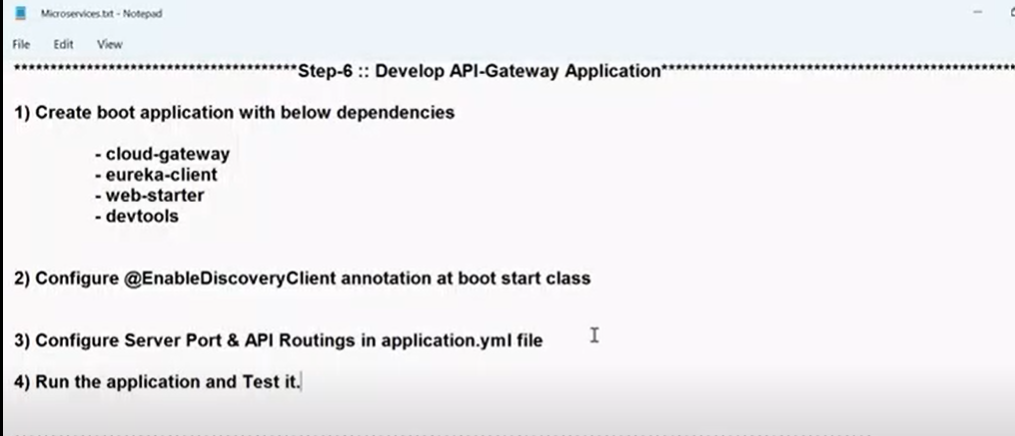
Interview question:

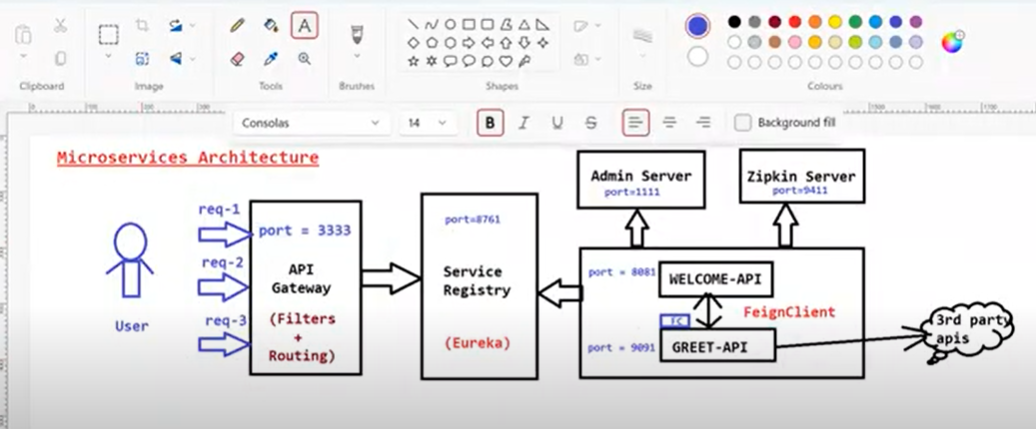
Can we change log level of Spring Boot application without restarting the server?

Yes, By using Admin server, we can change log level of application without restarting the server.

By default, log level of Spring Boot application is INFO, we can change to DEBUG





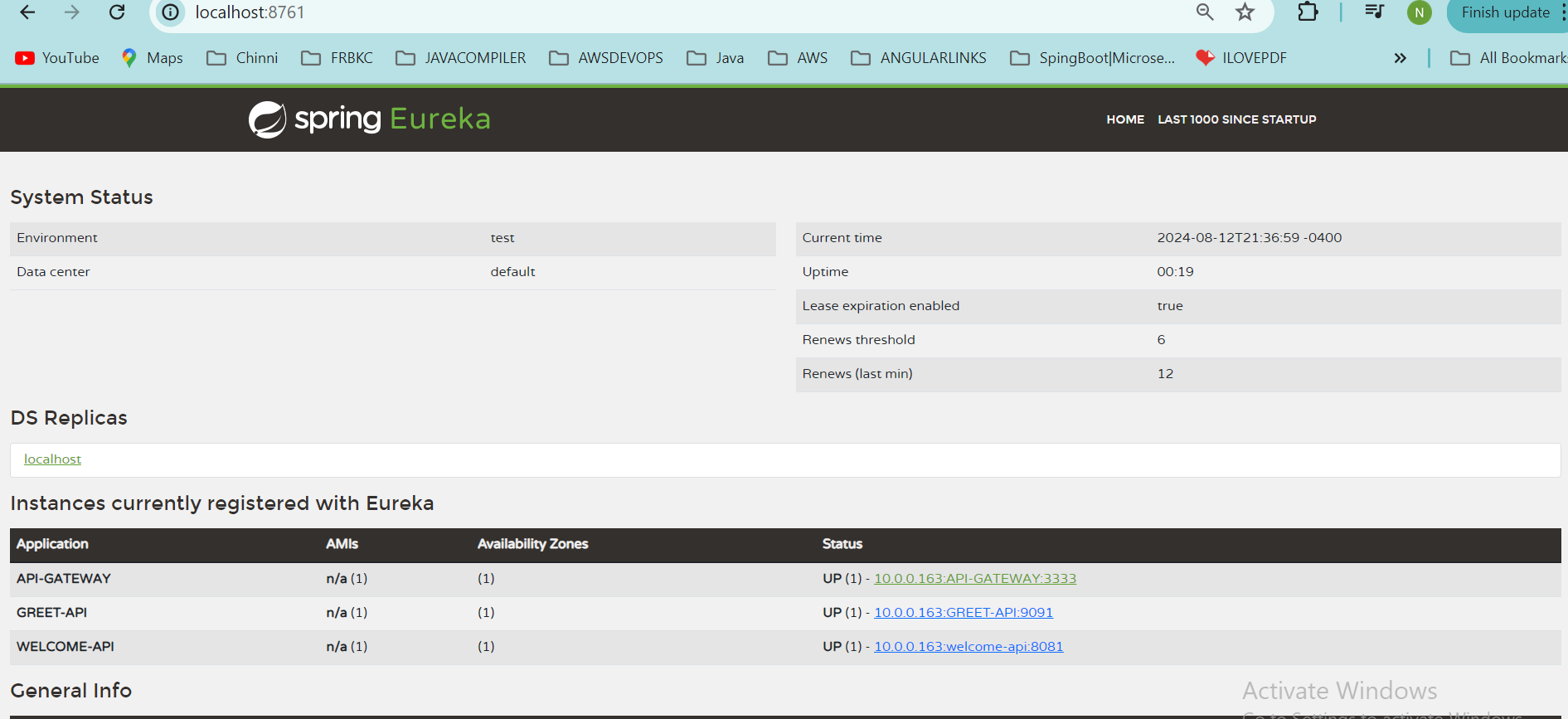


Service registry port number is fixed.

Zipkin server port number is fixed

Admin server, WELCOME-API,GREET-API,API Gateway -- you can give any port numbers

At the end, we can see API-Gateway,GREET-API,WELCOME-API registered with Eureka server



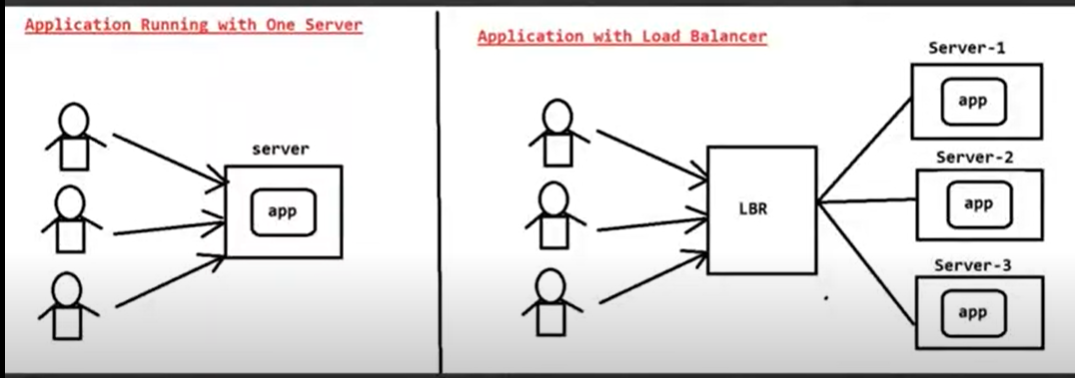
**Videoplayback(3)**

**What is use of @Component annotation?**

**Load Balancing**

Load balancing is the process of distributing load to multiple servers.

Loadbalancer will distribute the load based on Round robin technique.Requests will be distributed in a cyclic process.



**Videoplayback(4)**

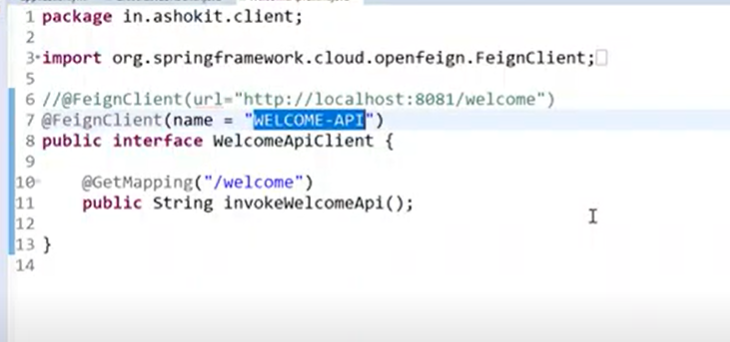
Load balancing can be done in two ways.

**Horizontal scaling**: Increasing the number of servers

**Vertical Scaling**: Increasing the capacity of a server.Vertical scaling can be done only to certain extent depending on the maximum capacity of server.

Therefore Industry people choose Horizontal scaling over vertical scaling.

Welcome-api has three instances, that means it is running in three servers.Load is distributed in Round-Robin fashion.



For the Feign-client, giving the service name is recommended.If you give URL, port number will be fixed and we cannot achieve load balancing.The request always goes to same server.In load balancing, we create instances of a service with different port numbers.

Feign client will communicate with Service registry using Service-name.It will get access to all URLS(instances) available in Service registry.

Feign client internally uses Ribbon concept.That Ribbon will help Feign client to implement Round-Robin algorithm.

API gateway also uses Ribbon concept to implement Round Robin algorithm.

**Zero-Downtime:** 24 hours (365 days)applications will be up and running.It is easy to achieve zero downtime by using Microservices.

But in a Monolithic architecture, when there is a problem, the whole application will be going down.

**The process of increasing number of instances on demand(increase in traffic) is called Auto scaling.**

In this example, the application(GREET-API,WELCOME-API) got scaled manually.

Scale-up -- increase the instances

Scale-down--decrease the instances

We can achieve auto-scaling in AWS and Microsoft GCP.