***REST Webservices:***

* + 3 key characteristics of web service,
    1. Macine-to-machine (app-to-app) communication
    2. Platform independent
    3. Should allow communication over network

Spring boot in background,

* DispatcherServlet - Front controller pattern
* Java beans are converted into json by using, @ResponseBody + JacksonHttpMessageConverters
* Errors are autoconfigured - ErrorMVCAutoConfiguration

We will be creating a social media applications.

It has 2 components, users and posts

User entity have id, name, birth date

Post entity have id, description

Users REST API,

* Get all users,

GET /users

* Add user

POST /users

* Get single user

GET /users/{id}

* Delete user

DELETE /users/{id}

Posts REST API,

* Get all posts related to user,

GET /users/{id}/posts

* Create posts for a user

POST /users/{id}/posts

* Get a specific post for a user

GET /users/{id}/posts/{post\_id}

We can use ResponseEntityExceptionHandler class from spring to create our custom exception handler and then use @ControllerAdvice at class level and @ExceptionHandler(ExcecptionClassName.class) to map any method to perticular exception so that logic in this method will execute whenever any controller gets that perticular exception

Advanced REST API features,

* Documentation
* Content negotiation
* Internationalization - i18n
* Versioning
* HATEOAS
* Static filtering
* Dynamic filtering
* Monitoring

Content-negotiation in REST API happens with Accept or Accept-language header.

E.g. accept header value as application/xml returns response in xml format. However, accept header value as application/json returns json formatted response.

Accept-Language header value as fr returns response in French language.

Versioning REST API - Options

* URI Versioning - have separate URI for different versions.

E.g. /v1/person - return {"name" : "ajay murtekar"}

/v2/person - return {"name" : {"firstname": "ajay", "lastname":"murtekar"}}

* Request Param Versioning - differentiate based on request params

E.g. /person?version=1 - returns {"name" : "ajay murtekar"}

/person?version=2 - returns {"name" : {"firstname": "ajay", "lastname":"murtekar"}}

* Header versioning - differentiate based on different header value

E.g. /person/header with header as x-api-version=1 returns returns {"name" : "ajay murtekar"}

/person/header with header as x-api-version=2 returns {"name" : {"firstname": "ajay", "lastname":"murtekar"}}

* Media type versioning or content-negotiation versioning

E.g. using accept header value as application/vnd.company.app-v1+json vs application/vnd.company.app-v2+json to return different outputs

Factors to consider while REST API versioning,

* URI Pollution - URI and Request param versioning have URI pollution
* Misuse of HTTP Headers - originally headers are never meant to be used for versioning
* Caching - caching of rest api happens based on uri and hence we cannot cache rest api with header or media type versioning
* Can we execute request in browser
* API documentation

HATEOAS - Hypermedia as the engine of application state

HATEOAS uses two main components - EntityModel and WebMvcLinkBuilder

Static filtering can be used to remove any class member from response json.

@JsonIgnore can be used on class member.

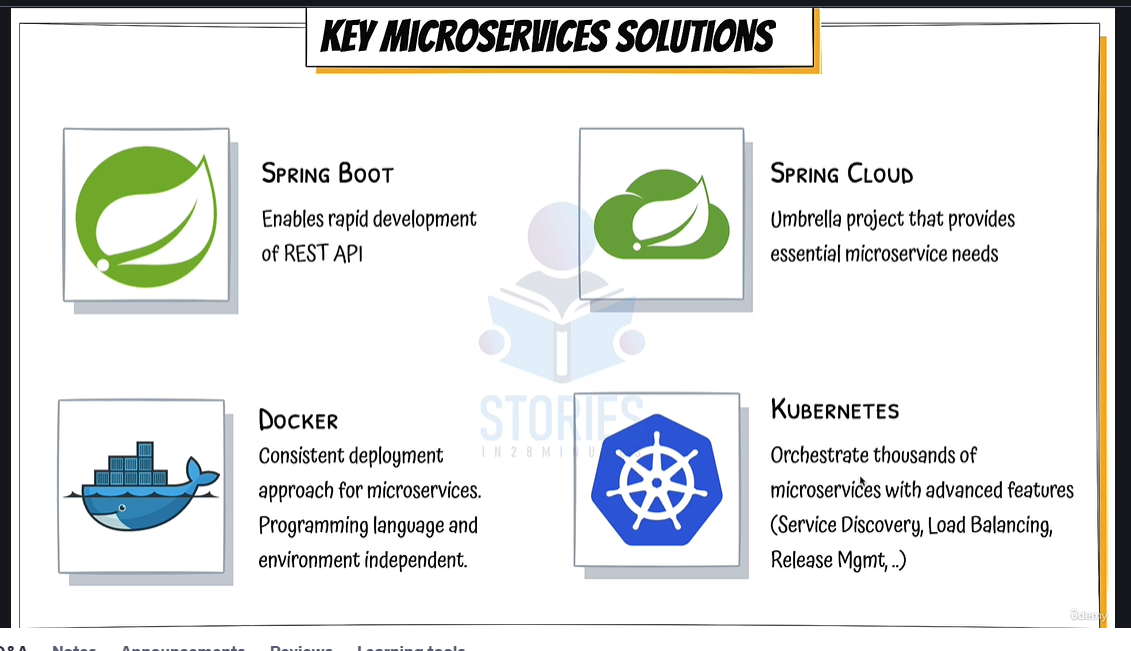
@JsonIgnoreProperties can be used at class level by adding class variable names

Dynamics filtering means filtering different class members based on different rest api calls.

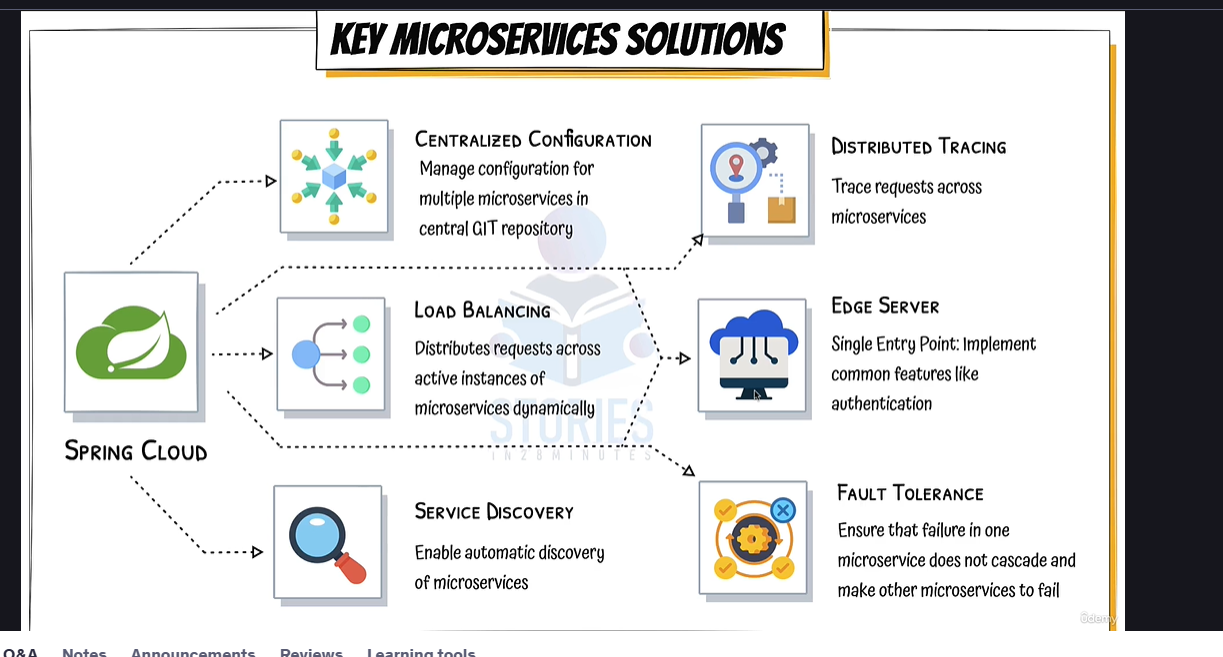
MappingJacksonValue class is used to implement dynamic filtering at controller level.

**MICROSERVICES:**

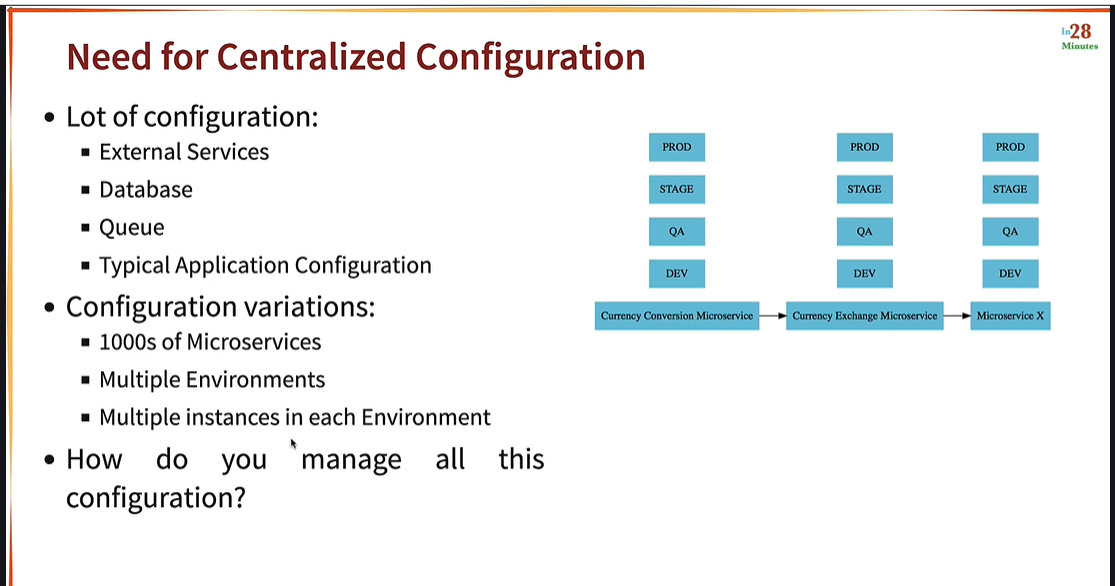
Key microservices solutions:



Spring cloud is an umbrella of solutions used for microservices:



Centralized configurations:



Ports configuration for examples:

**1. Limits Microservice**  
Ports: **8080, 8081**, etc.

**2. Spring Cloud Config Server**  
Port: **8888**

**3. Currency Exchange Microservice**  
Ports: **8000, 8001, 8002**, etc.

**4. Currency Conversion Microservice**  
Ports: **8100, 8101, 8102**, etc.

**5. Netflix Eureka Naming Server**  
Port: **8761**

**6. API Gateway**  
Port: **8765**

**7. Zipkin Distributed Tracing Server**  
Port: **9411**

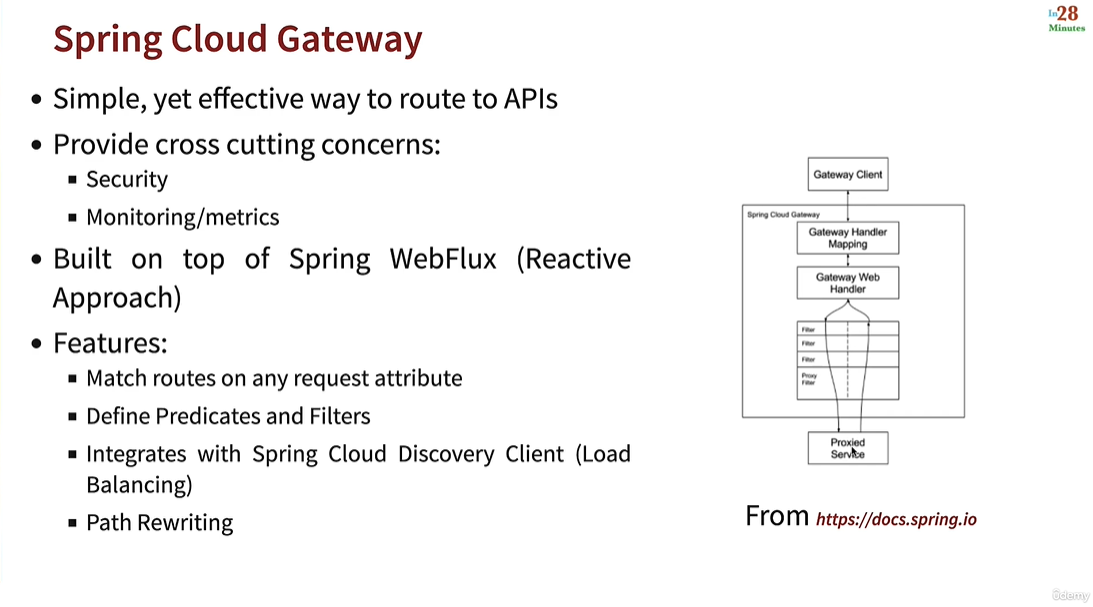
Currency exchange service:

<http://localhost:8000/currency-exchange/from/USD/to/INR>

Currency conversion service:

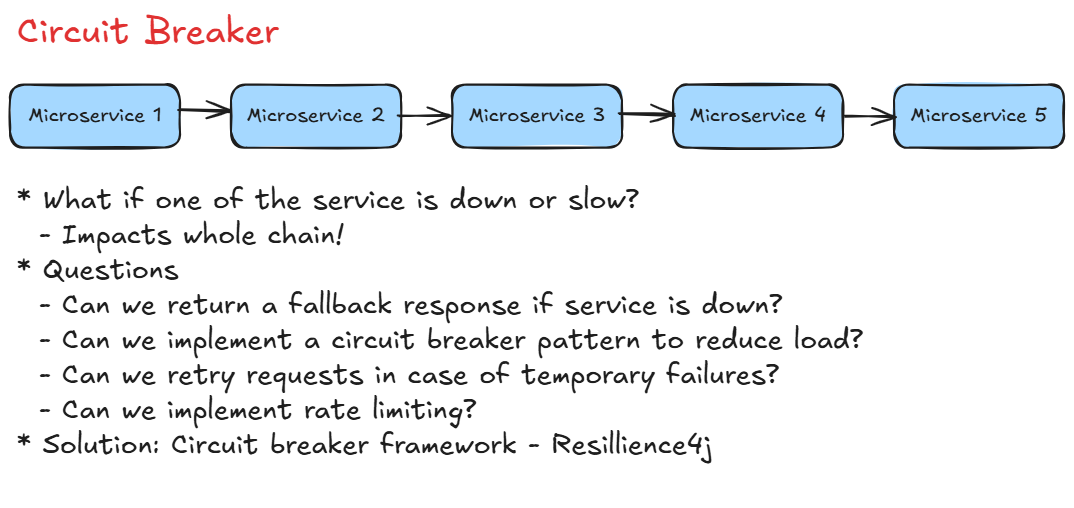
<http://localhost:8100/currency-conversion/from/USD/to/INR/quantity/10>

Spring cloud Gateway. Functionalities offered,



RouteLocator class is used to create a route configuration in API gateway in spring cloud.

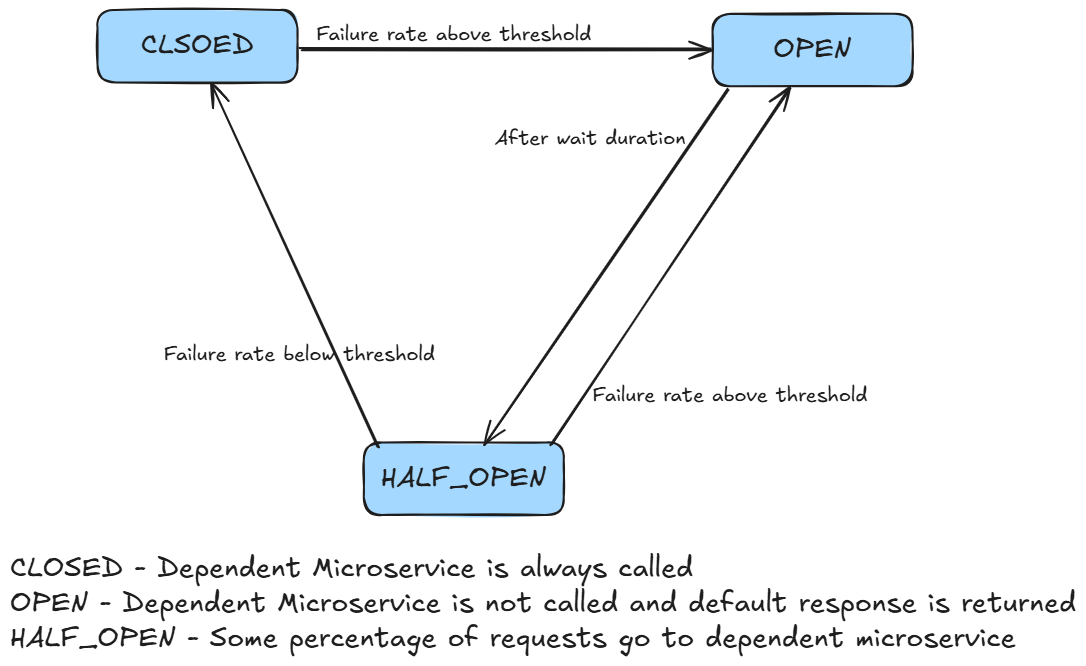
Circuit breaker



@CircuitBreaker

@RateLimiter – to limit no of requests for a given time window

@BulkHead – how many concurrent requests are allowed at a time



**Docker**

Docker commands

docker container run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker container run -d -p 5000:5000 in28min/hello-world-java:0.0.1.RELEASE

docker container run -d -p 5000:5000 in28min/hello-world-python:0.0.1.RELEASE

docker container ls

docker image ls

docker container stop cc

docker container run -d -p 5001:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker container run -d -p 5002:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker container run -p 5003:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker container run -p 5003:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker --version

docker container ls

docker build -t in28min/hello-world-docker:v1 .

docker image list

docker run -d -p 5000:5000 in28min/hello-world-docker:v1

docker build -t in28min/hello-world-docker:v2 .

docker container run -d -p 5000:5000 in28min/hello-world-docker:v2

docker build -t in28min/hello-world-docker:v3 .

docker container run -d -p 5000:5000 in28min/hello-world-docker:v3

docker build -t in28min/hello-world-docker:v4 .