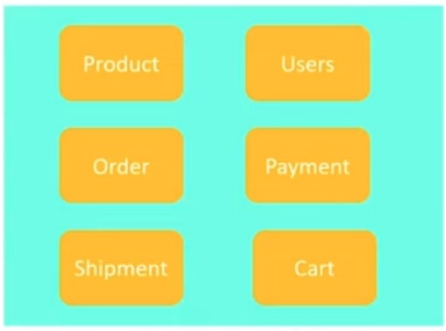
Micro Services(using SpringBoot)

Before Microservices

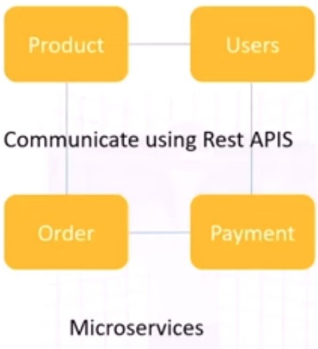
- Monolithic architecture: multiple components are combined in single large app.
- Single Code Base
- Deployed in single bundle
- Change in one service then whole app is redeployed
- Building problem: developers has to communicate
- Problem in scale
- Cumbersome over time



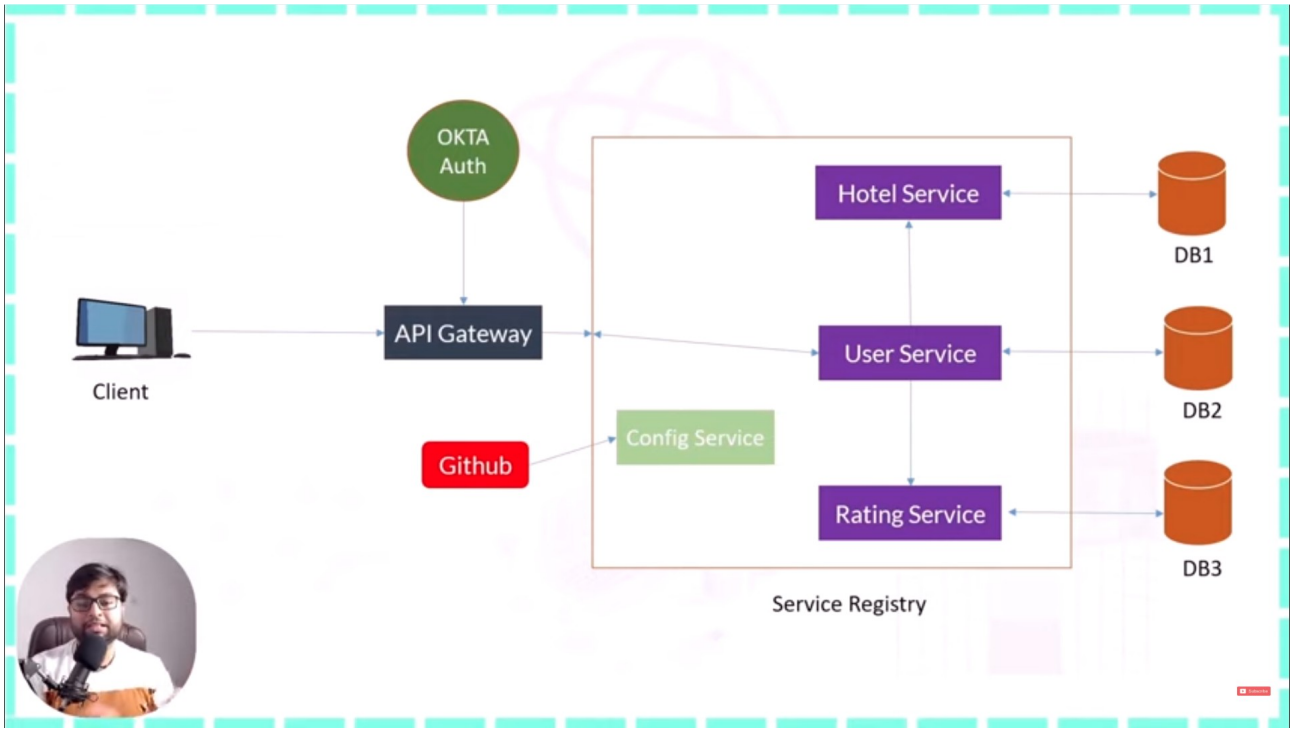
Application using Monolithic

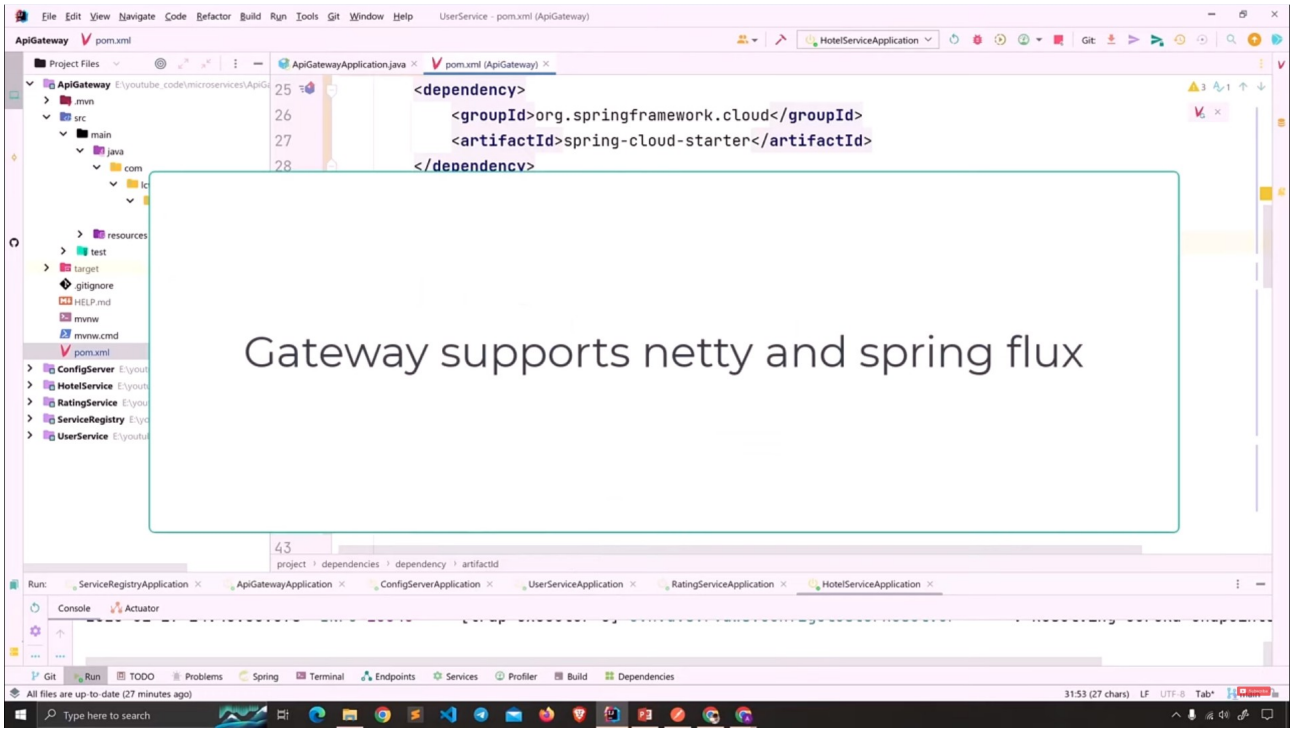
Microservices

- Large apps are divide into small parts
- Different codebase
- Each module managed independently
- Different tech stack
- Handling microservices is complex



Microservices


OKTA
Auth
Hotel Service
DB1
API Gateway
User Service
DB2
Client
Config Service
Github
Rating Service
DB3
Service Registry


Gateway supports netty and spring flux

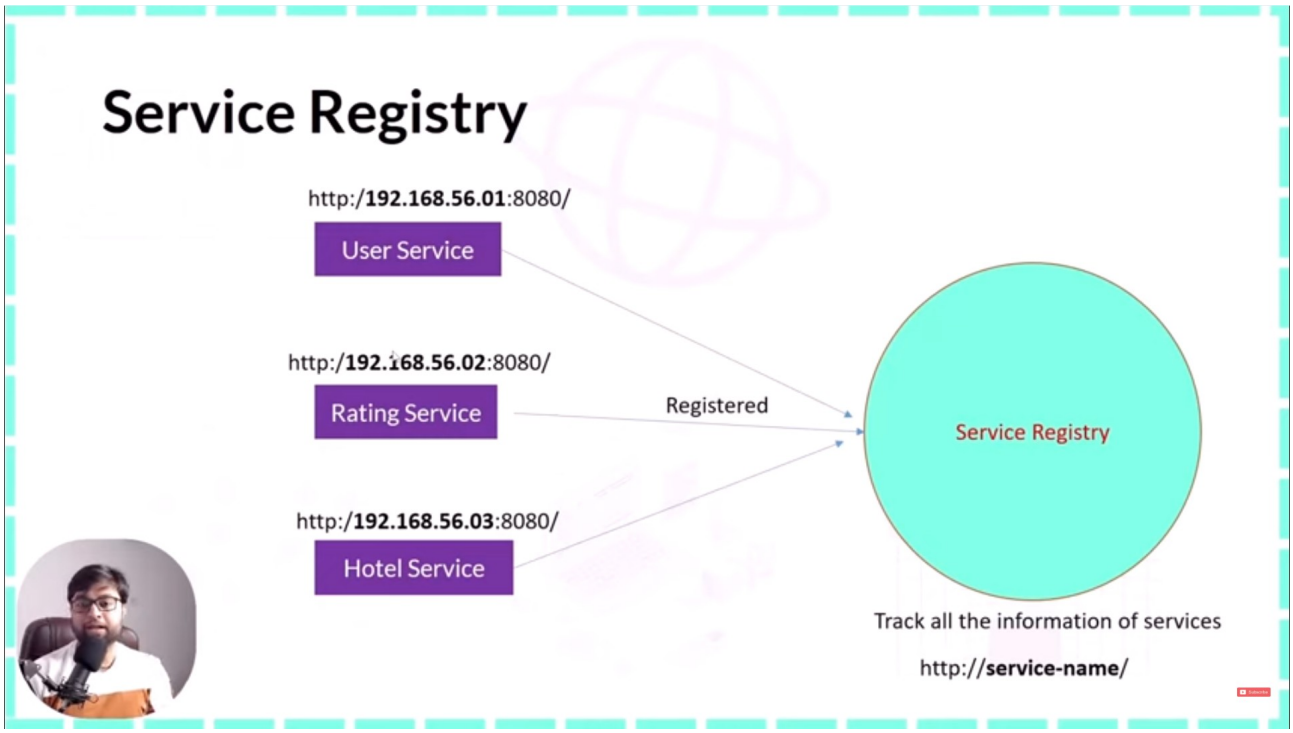
If you include the starter, but you do not want the gateway to be enabled, set `spring.cloud.gateway.enabled=false`.

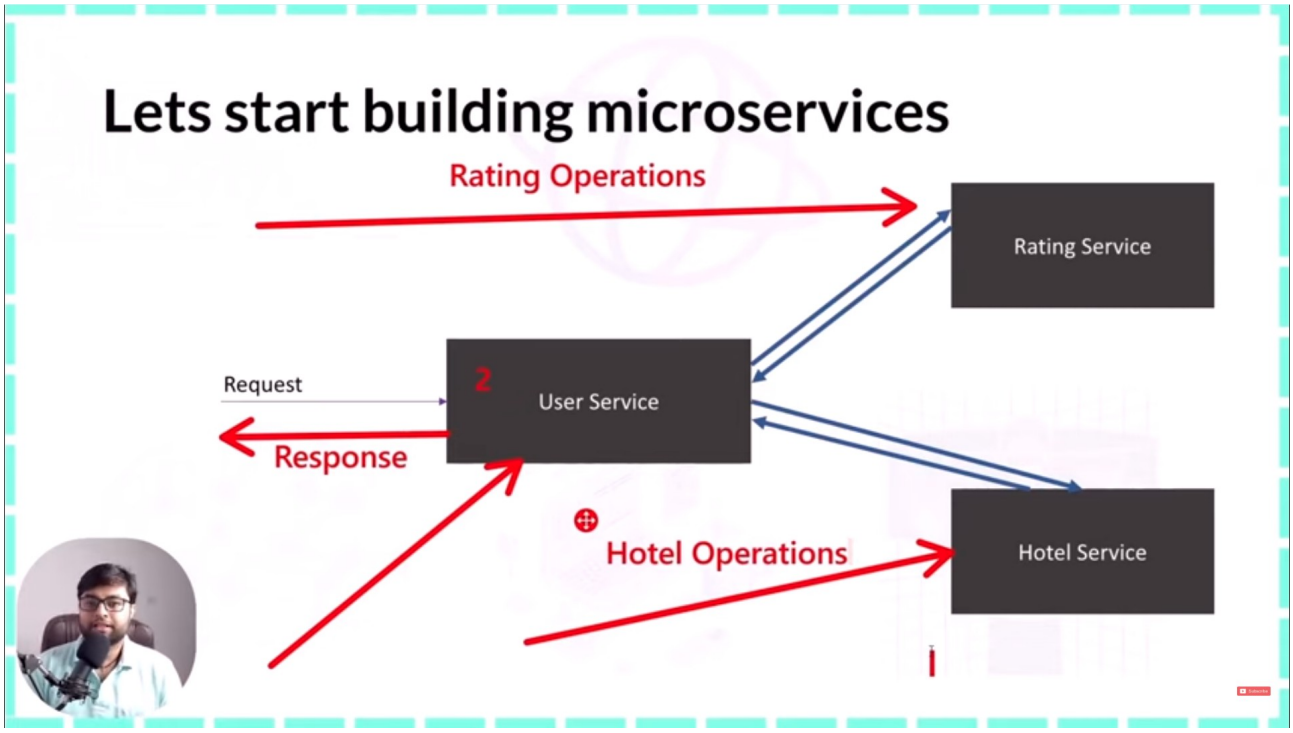
Spring Cloud Gateway is built on Spring Boot 2.x, Spring WebFlux, and Project Reactor. As a consequence, many of the familiar synchronous libraries (Spring Data and Spring Security, for example) and patterns you know may not apply when you use Spring Cloud Gateway. If you are unfamiliar with these projects, we suggest you begin by reading their documentation to familiarize yourself with some of the new concepts before working with Spring Cloud Gateway.

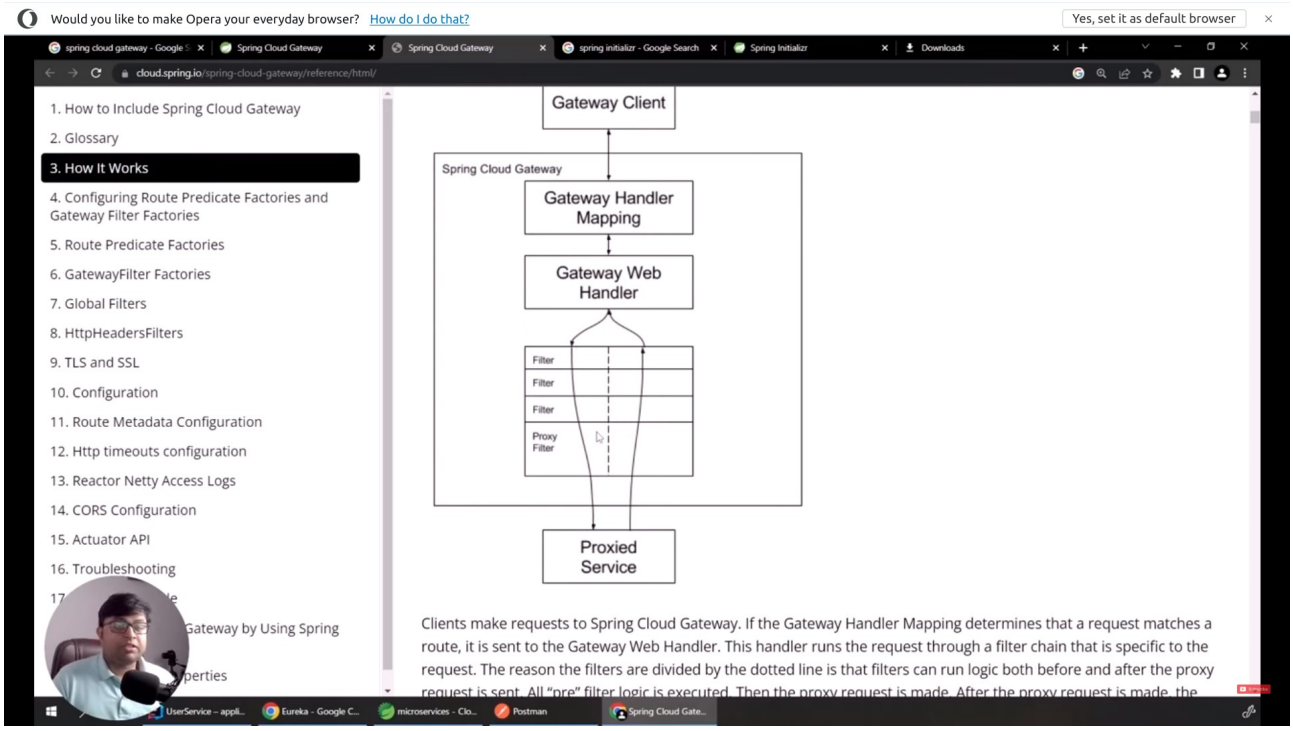
Spring Cloud Gateway requires the Netty runtime provided by Spring Boot and Spring Webflux. It does not work in a traditional Servlet Container or when built as a WAR.

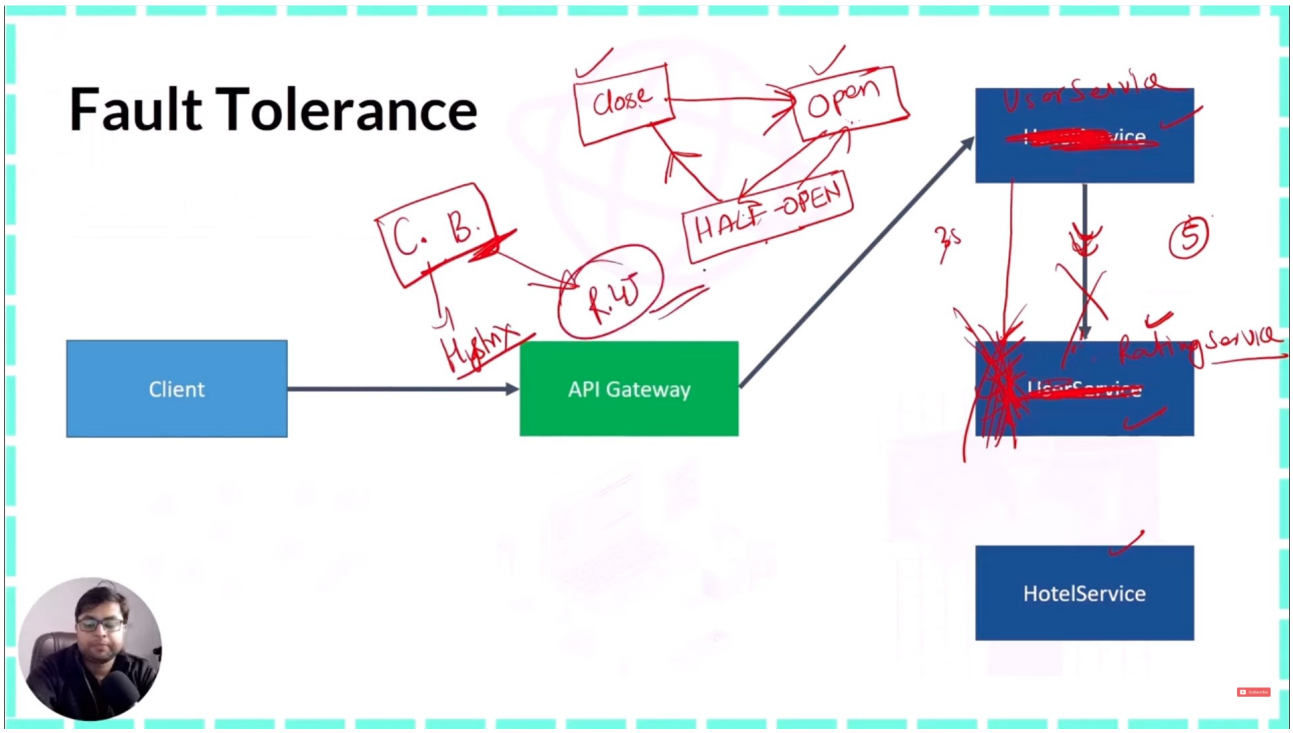
2. Glossary

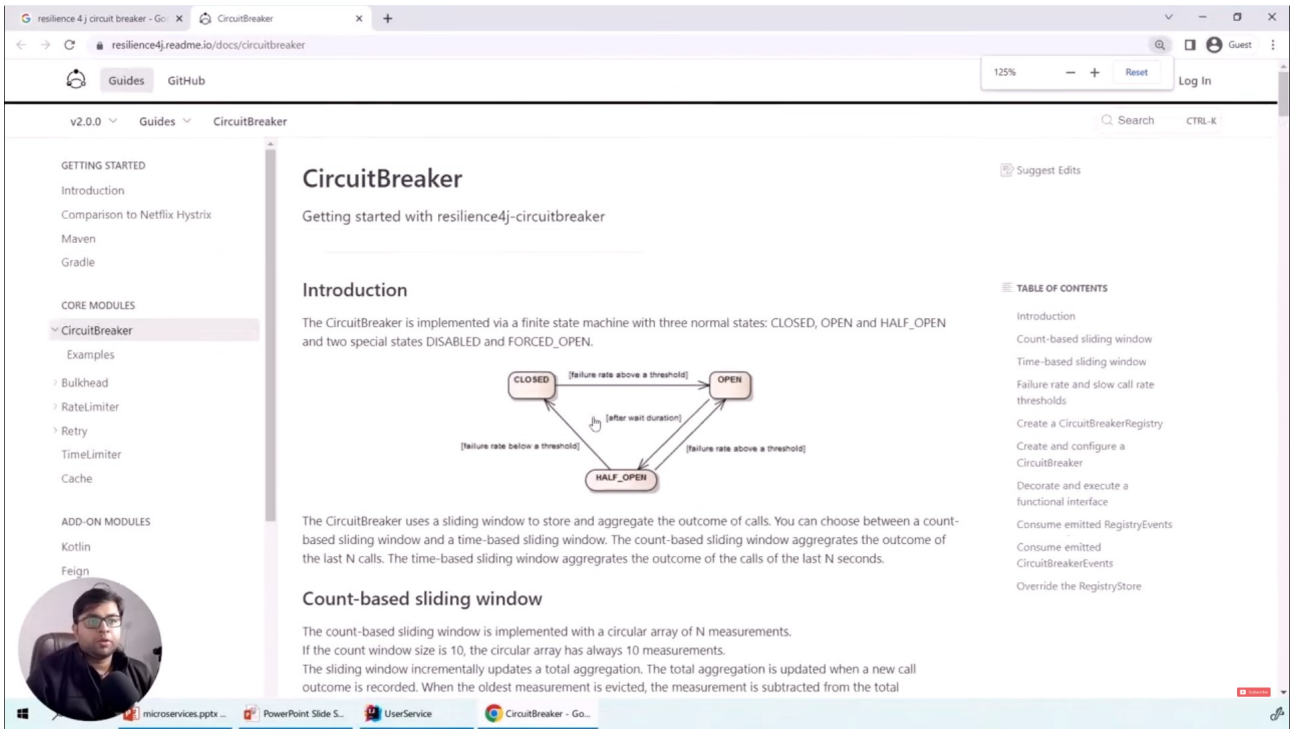
- **Route**: The basic building block of the gateway. It is defined by an ID, a destination URI, a collection of predicates, and a collection of filters. A route is matched if the aggregate predicate is true.
- **Predicate**: This is a Java 8 Function Predicate. The input type is a Spring Framework `ServerWebExchange`. This lets you match on anything from the HTTP request, such as headers or parameters.




Lets start building microservices
Rating Operations
Rating Service
Request
2
User Service
Response
Hotel Operations
Hotel Service


Would you like to make Opera your everyday browser? How do I do that?
Yes, set it as default browser
cloud.spring.io/spring-cloud-gateway/reference/html/
1. How to Include Spring Cloud Gateway
2. Glossary
3. How It Works
4. Configuring Route Predicate Factories and Gateway Filter Factories
5. Route Predicate Factories
6. GatewayFilter Factories
7. Global Filters
8. HttpHeadersFilters
9. TLS and SSL
10. Configuration
11. Route Metadata Configuration
12. Http timeouts configuration
13. Reactor Netty Access Logs
14. CORS Configuration
15. Actuator API
16. Troubleshooting
Gateway Client
Spring Cloud Gateway
Gateway Handler Mapping
Gateway Web Handler
Filter
Filter
Filter
Proxy Filter
Proxied Service
Clients make requests to Spring Cloud Gateway. If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler. This handler runs the request through a filter chain that is specific to the request. The reason the filters are divided by the dotted line is that filters can run logic both before and after the proxy request is sent. All "pre" filter logic is executed. Then the proxy request is made. After the proxy request is made, the

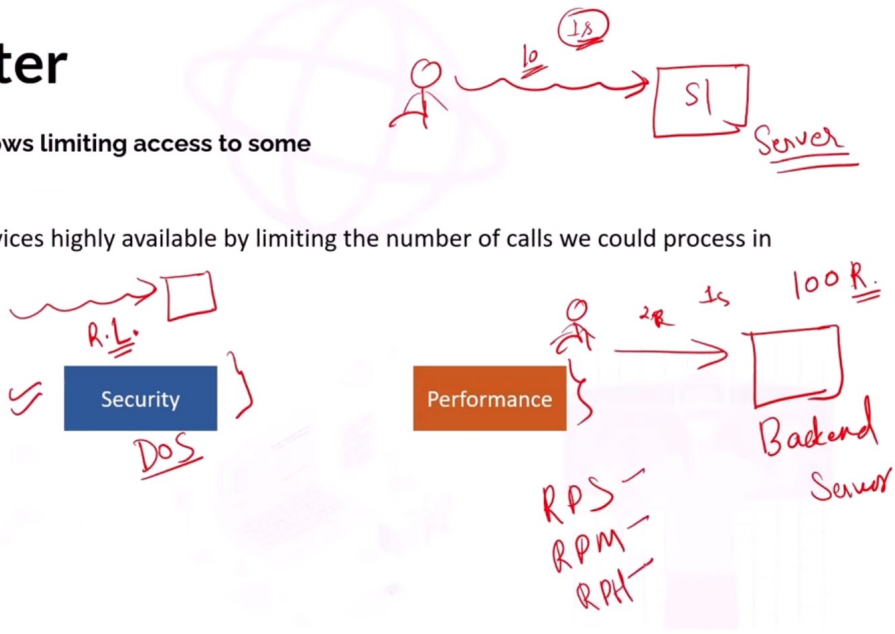

Fault Tolerance
Close
Open
HALF -OPEN
C. B.
R.T
Hystrix
Client
API Gateway
UserService
3s
5
HotelService

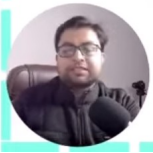

resilience 4 j circuit breaker - Go
CircuitBreaker
resilience4j.readme.io/docs/circuitbreaker
Guest
Guides
GitHub
125%
Reset
Log In
v2.0.0
Guides
CircuitBreaker
Search
CTRL-K
GETTING STARTED
Introduction
Comparison to Netflix Hystrix
Maven
Gradle
CORE MODULES
CircuitBreaker
Examples
Bulkhead
RateLimiter
Retry
TimeLimiter
Cache
ADD-ON MODULES
Kotlin
Feign
CircuitBreaker
Getting started with resilience4j-circuitbreaker
Suggest Edits
Introduction
The CircuitBreaker is implemented via a finite state machine with three normal states: CLOSED, OPEN and HALF_OPEN and two special states DISABLED and FORCED_OPEN.
CLOSED
[failure rate above a threshold]
OPEN
[after wait duration]
[failure rate below a threshold]
[failure rate above a threshold]
HALF_OPEN
The CircuitBreaker uses a sliding window to store and aggregate the outcome of calls. You can choose between a count-based sliding window and a time-based sliding window. The count-based sliding window aggregates the outcome of the last N calls. The time-based sliding window aggregates the outcome of the calls of the last N seconds.
Count-based sliding window
The count-based sliding window is implemented with a circular array of N measurements.
If the count window size is 10, the circular array has always 10 measurements.
The sliding window incrementally updates a total aggregation. The total aggregation is updated when a new call outcome is recorded. When the oldest measurement is evicted, the measurement is subtracted from the total
TABLE OF CONTENTS
Introduction
Count-based sliding window
Time-based sliding window
Failure rate and slow call rate thresholds
Create a CircuitBreakerRegistry
Create and configure a CircuitBreaker
Decorate and execute a functional interface
Consume emitted RegistryEvents
Consume emitted CircuitBreakerEvents
Override the RegistryStore
microservices.pptx
PowerPoint Slide S...
UserService
CircuitBreaker - Go...

Rate Limiter

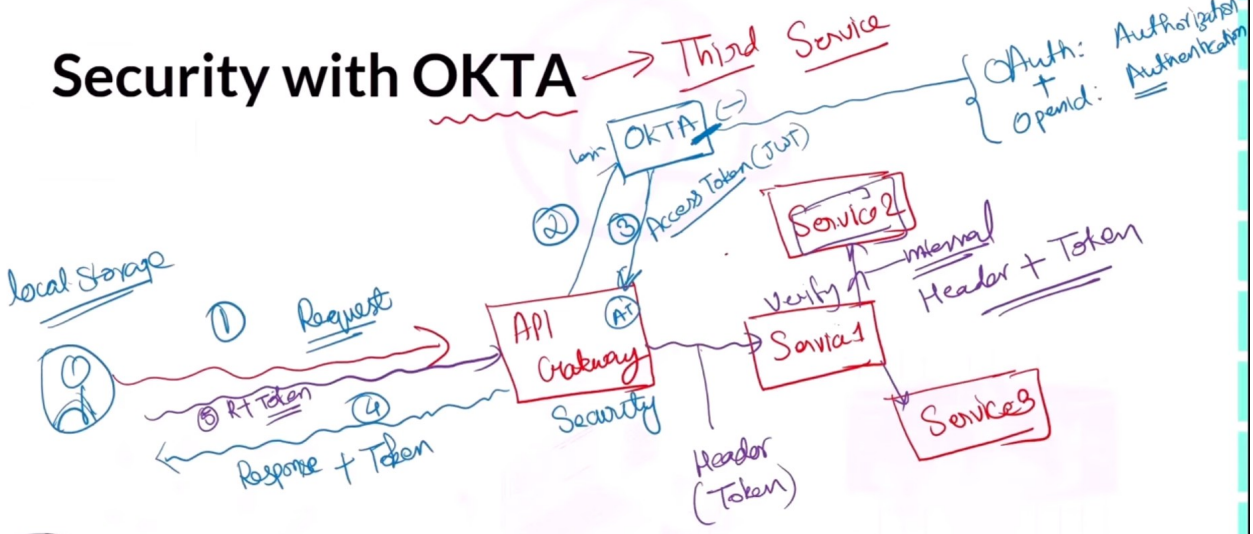
This functionality allows limiting access to some service.

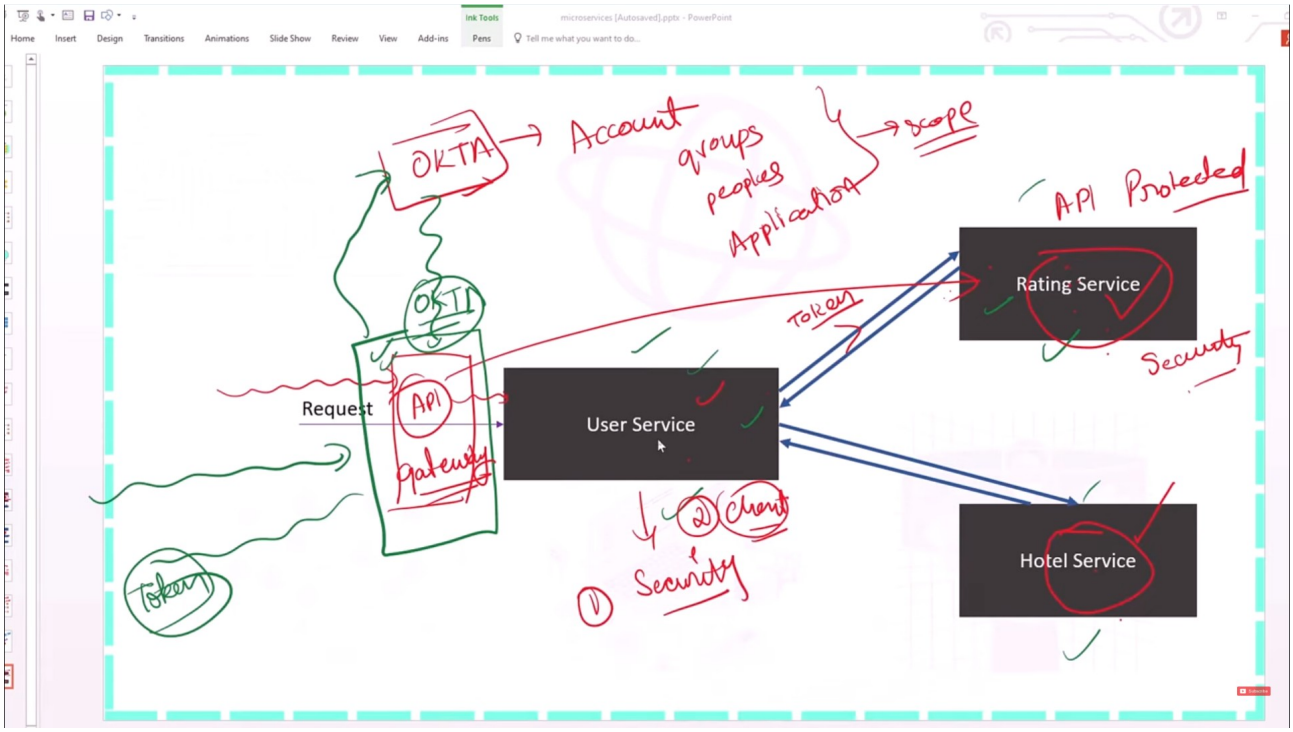
Rate Limiter make services highly available by limiting the number of calls we could process in specific windows.





Security with OKTA




Home
Insert
Design
Transitions
Animations
Slide Show
Review
View
Add-ins
Ink Tools
Pens
Tell me what you want to do...
OKTA
Account
groups
peoples
Applications
scope
API Protected
Rating Service
Token
Security
Request
API
gateway
User Service
Hotel Service
Token
2 client
1 Security