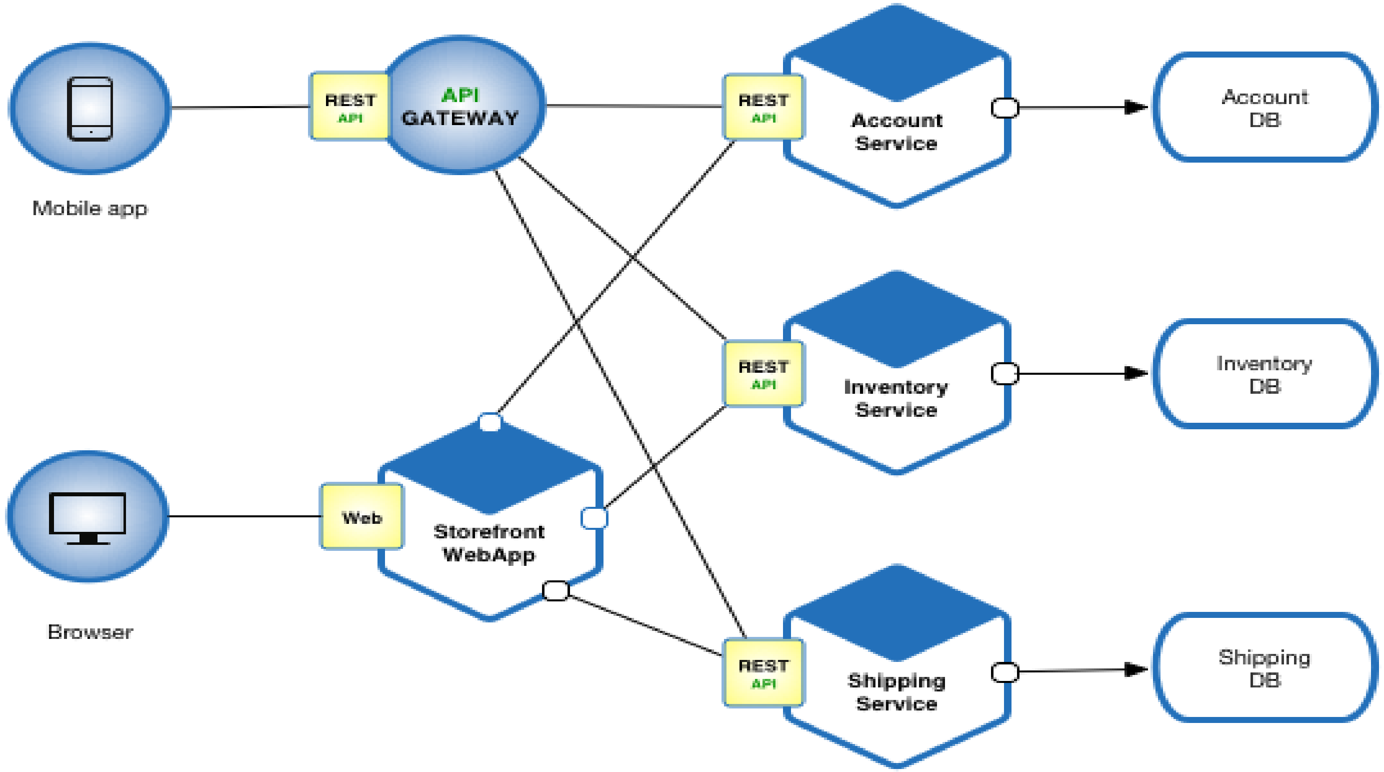
# MICROSERVICES ARCHITECTURE

1. API GATEWAY-ZUUL
2. LOAD BALANCER-RIBBON

# ESSENCE OF MICROSERVICES ARCHITECTURE

- Pattern Language for large/complex applications.
- Decompose Monoliths into fine-grained services.
- Deploy and scale the 'micro' services independently.
- High Maintainability and Testability



# API Gateway

# PROBLEM

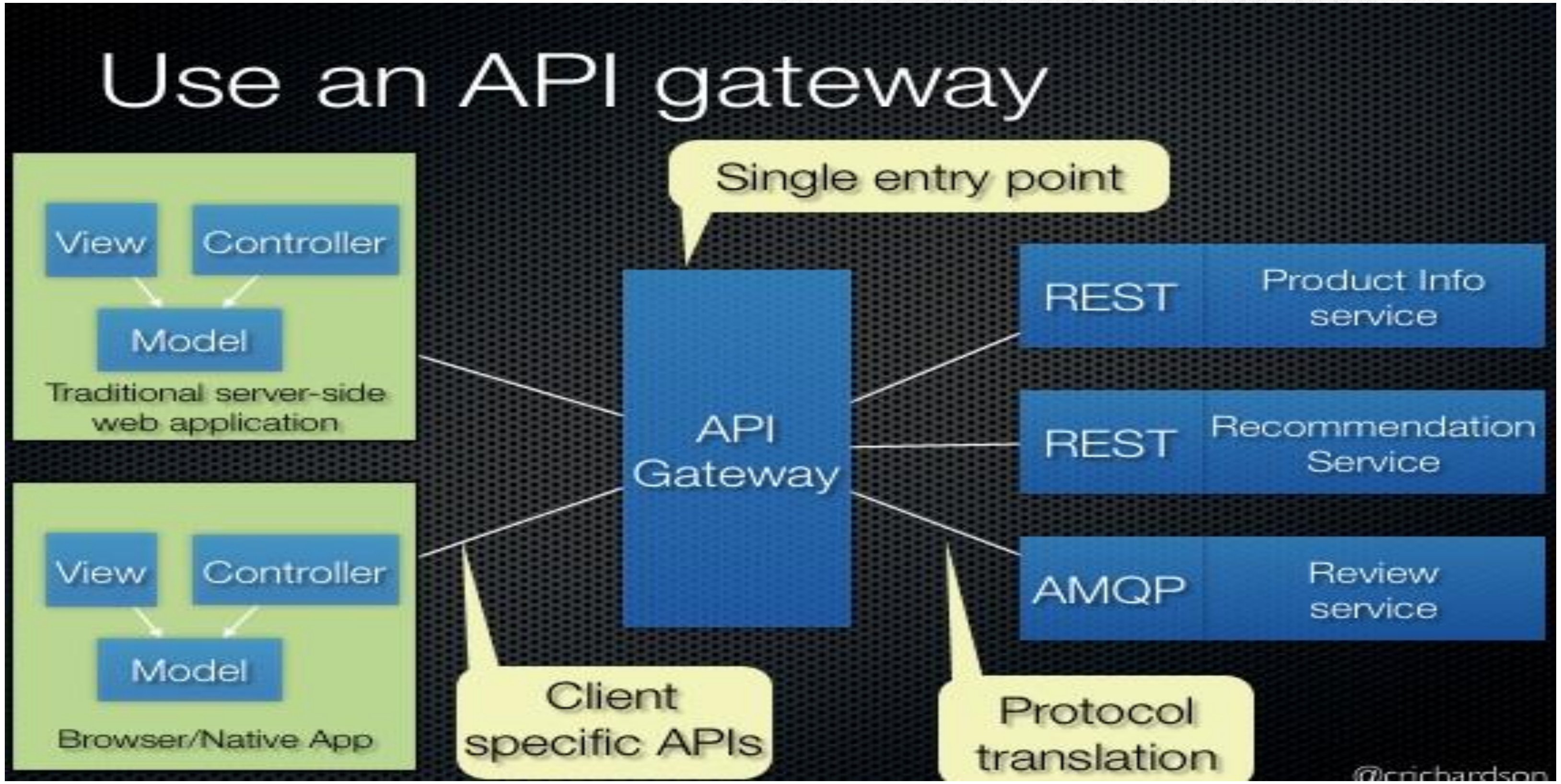
How do the clients of a Microservices-based application access the individual services?

# FORCES

- Need to fetch data from multiple services
- Different clients need different data
- Network performance is different for different types of clients
- Dynamic change of service instances (host + port)
- Diverse set of protocols (http, https etc)

# SOLUTION

**API Gateway** as the single-entry point for all the clients.



Use an API gateway

# BENEFITS

- Provides an optimal API for each client
- Reduces the number of requests or roundtrips
- Translates the standard protocols to whatever protocols used internally

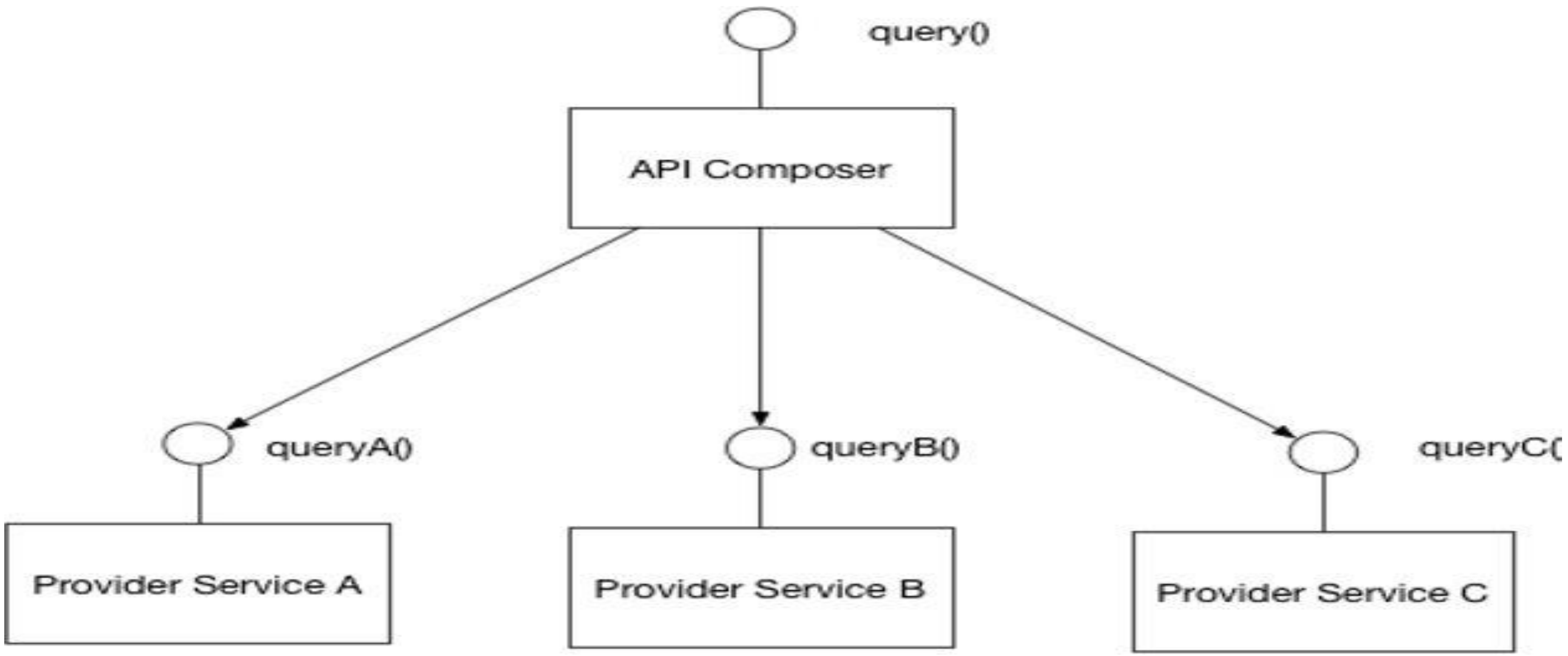
# DRAWBACKS

- Increased complexity
  - Yet another part that must be developed, deployed and managed
- Increased response time
  - Additional network hop however the cost is insignificant for most of the applications

# USAGE

- It is also a Micro service
- Must use **Client-side/Server-side Discovery pattern** to route the requests to available service instances
- May implement Security i.e., Authorization etc.,
- Use **Circuit Breaker Pattern** to invoke the services
- API Gateway implements often the **API Composition pattern**.

# API Composition Pattern

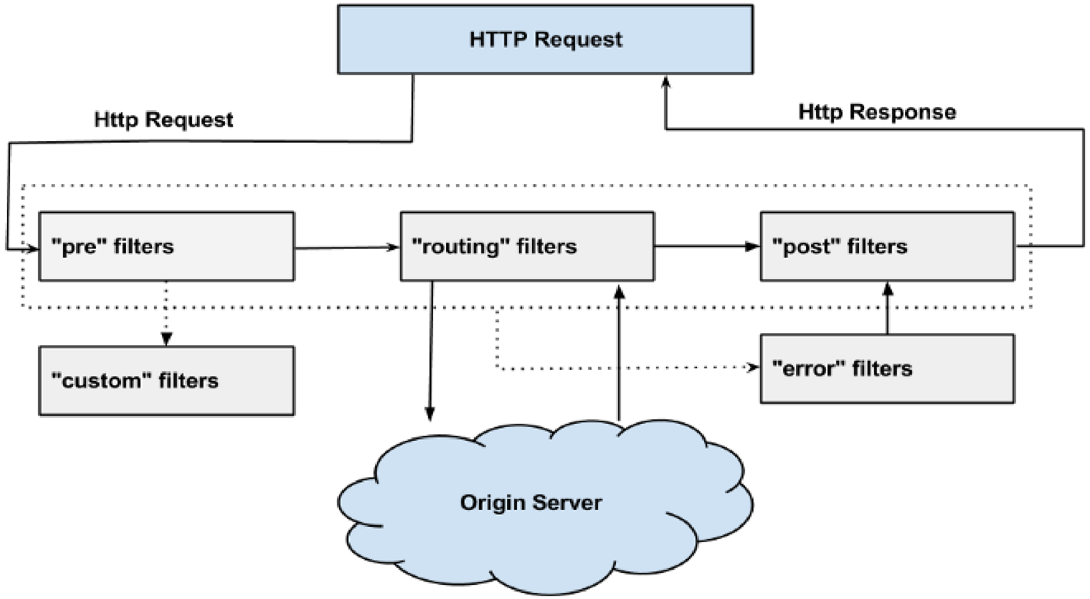


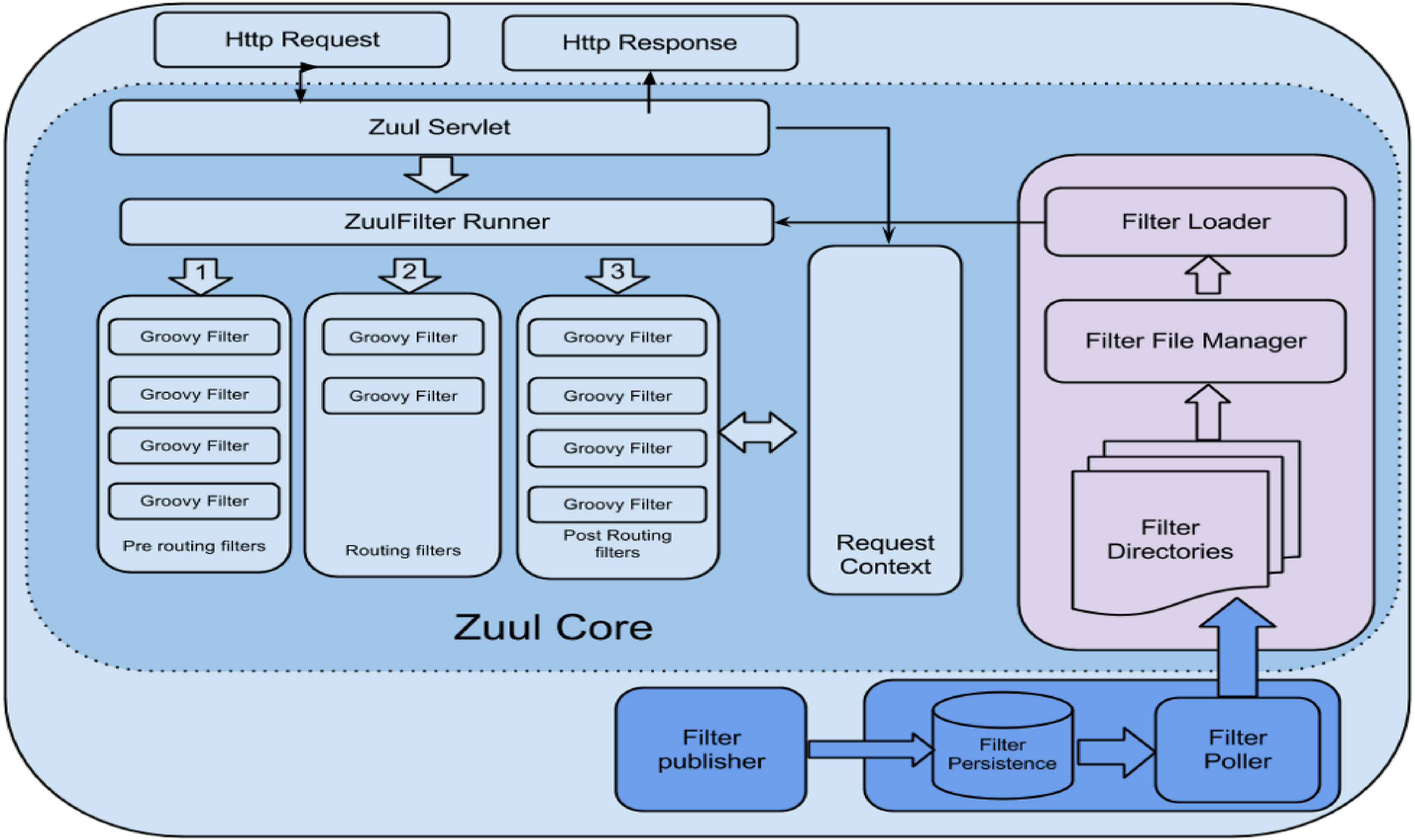
# ZUUL and its Capabilities

- Zuul is an API Gateway or an Edge Server developed by Netflix
- Zuul is capable of,
  - Authentication and Security
  - Insights and Monitoring
  - Dynamic Routing
  - Stress Testing
  - Load Shedding
  - Static Response Handling

# Other Benefits by Zuul

- **Filters** : A way of extending the Zuul's Default functionality
- Divided into 4 types during the request-routing process
  - `pre` — filters run before the request is routed.
  - `route` — filters can handle the actual routing of the request.
  - `post` — filters run after the request has been routed.
  - `error` — filters run if an error occurs in course of handling the request.
  - **Custom filters** for static response handling.

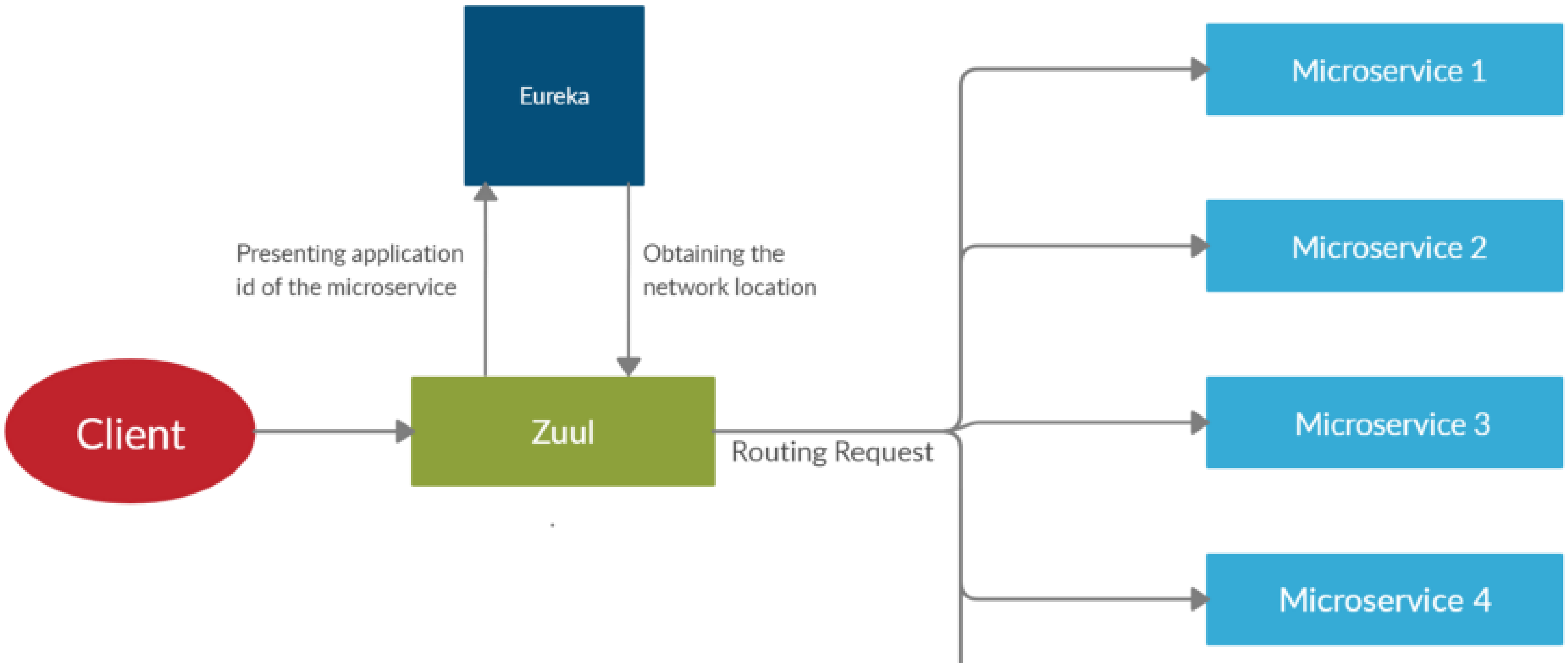



Http Request
Http Response
Zuul Servlet
ZuulFilter Runner
1
2
3
Groovy Filter
Groovy Filter
Groovy Filter
Groovy Filter
Pre routing filters
Groovy Filter
Groovy Filter
Routing filters
Groovy Filter
Groovy Filter
Groovy Filter
Groovy Filter
Post Routing filters
Request Context
Filter Loader
Filter File Manager
Filter Directories
Zuul Core
Filter publisher
Filter Persistence
Filter Poller

# Netflix Zuul OSS Components

- Zuul-core  -  library containing the set of core features
- Zuul-netflix – extension library for ribbon, eureka, hystrix etc.,
- Zuul-filters - (pre, post, route, error)
- Zuul-webapp-simple – sample app built on zuul-core
- Zuul-netflix-webapp – sample app built on all above

# Integration with Eureka



# Implementing Zuul Gateway with Spring Cloud

# Load Balancing

# PROBLEM

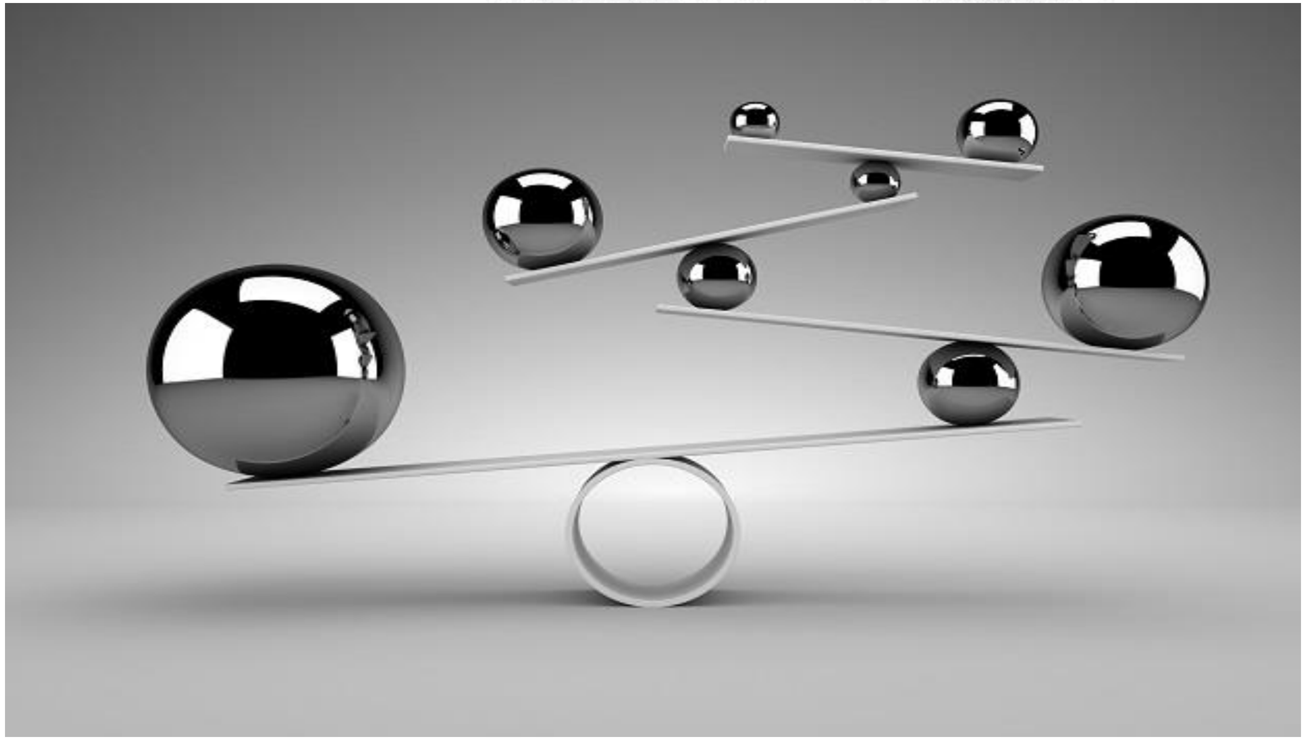
How Microservices can handle load, security and remains available?

# FORCES

- The amount of high traffic sometimes results in complex production issues.
- We do not know the service-instance failures.
- Security of exposed API's.
- On-demand scalability to address load increases.

# SOLUTION

- **Load Balancer:** To keep secure and controlled environment without losing the Development agility

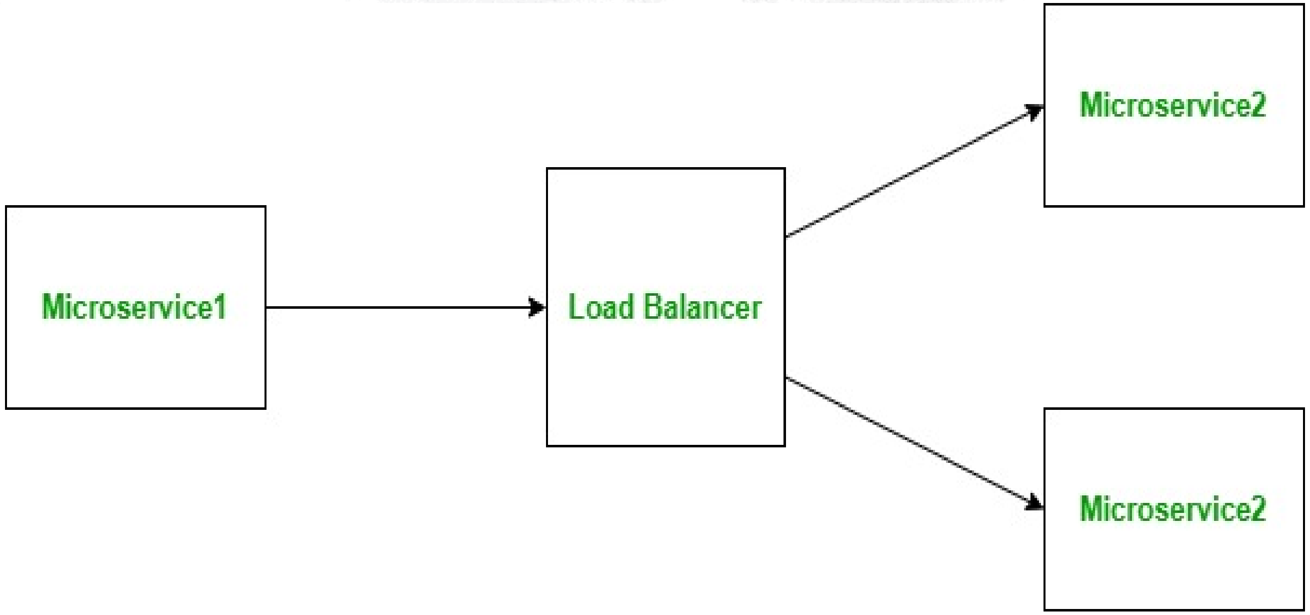


# USAGE

- Efficiently distributing the incoming network traffic across a group of backend servers (multiple instances of the service).
- 2 types of Load balancing
  - Server side
  - Client side

# Server-side Load balancing

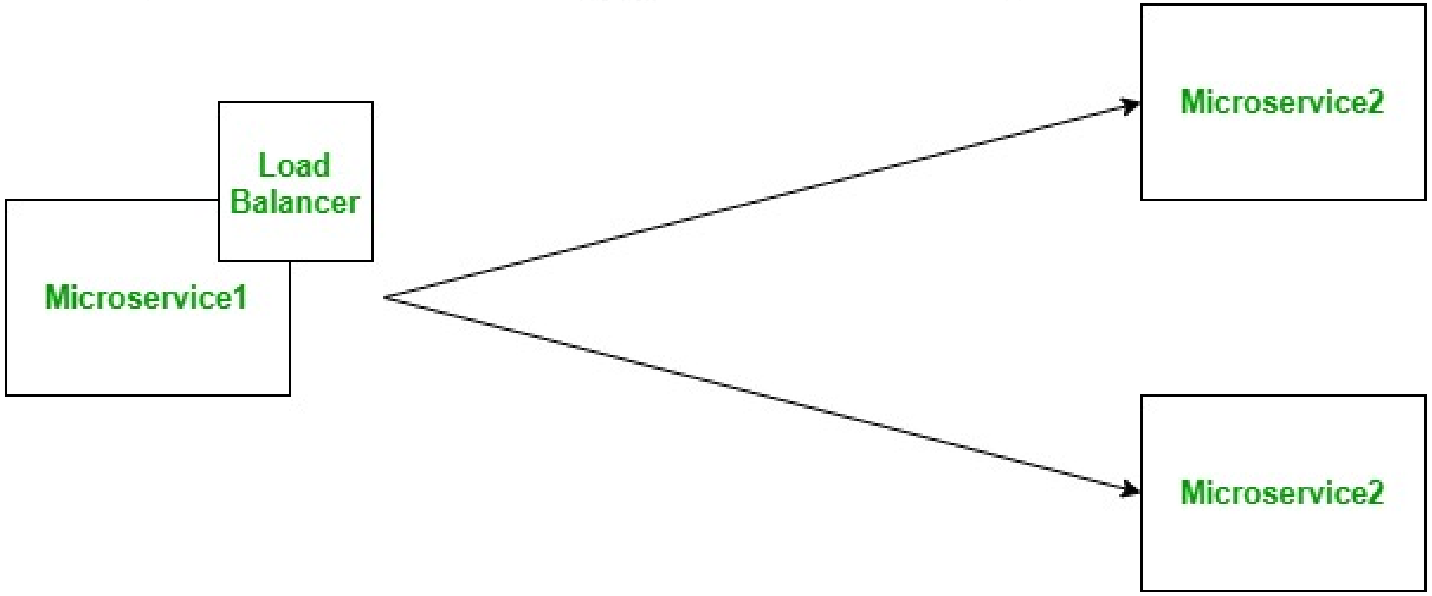
- Service Instances are deployed on multiple servers and then a load balancer is put in front of them.
- Also called as **Hardware load balancer.**
- Ex: Citrix, Cisco LB's
- Disadvantages:
  - Single point of failure
  - Network latency



*Server side load balancing*

# Client-side Load balancing

- The instances of the service are deployed on multiple servers.
- Load balancer's logic is part of the client itself.
- Also called as **Software Load balancer.**
- Ex: Ribbon etc.,
- Disadvantage:
  - The load balancer's logic is mixed up with the microservice code.



*Client side load balancing*

# Ribbon and its Capabilities

- Netflix's Ribbon is an Inter Process Communication (remote procedure calls) library with built in **client side(software) load balancer** and is a part of Netflix Open Source Software (Netflix OSS).
- Ribbon API works based on the concept called **“Named Client”**.
- Ribbon is capable of,
  - Load Balancing
  - Fault Tolerance
  - Configurable Load balancing rules (by default round-robin rule)
  - Supporting multiple protocols

# Implementing Ribbon Load Balancer with Spring Cloud

# Queries???