Spring Cloud 2

by Vlad Ungureanu

Software Architecture

- Hystrix
- Spring Hystrix
- Fault Tolerance Overview
- API Gateway
- Spring Zuul
- Gateway Overview

LEARN stuff IO

Hystrix

Latency and Fault Tolerance

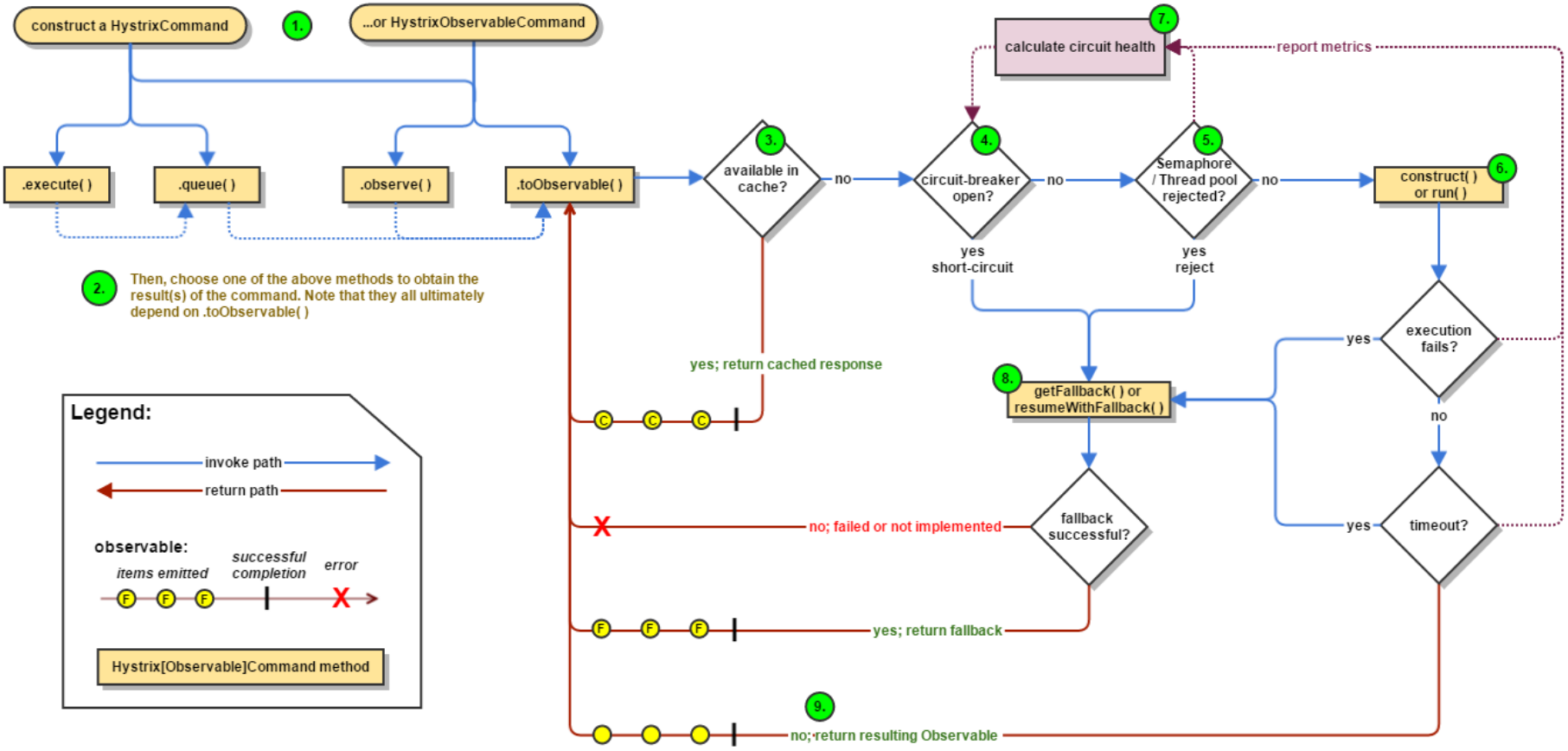
- Stop cascading failures. Fallbacks and graceful degradation. Fail fast and rapid recovery.
- Thread and semaphore isolation with circuit breakers.

Real-time Operations

- Real-time monitoring and configuration changes. Watch service and property changes take effect immediately as they spread across a fleet.
- Be alerted, make decisions, affect change and see results in seconds.

Concurrency

- Parallel execution. Concurrency aware request caching. Automated batching through request collapsing.


construct a HystrixCommand
1.
...or HystrixObservableCommand
.execute( )
.queue( )
.observe( )
.toObservable( )
2.
Then, choose one of the above methods to obtain the result(s) of the command. Note that they all ultimately depend on .toObservable( )
3.
available in cache?
no
yes; return cached response
4.
circuit-breaker open?
no
yes
short-circuit
5.
Semaphore / Thread pool rejected?
no
yes
reject
6.
construct( ) or run( )
7.
calculate circuit health
report metrics
8.
getFallback( ) or resumeWithFallback( )
execution fails?
yes
no
timeout?
yes
fallback successful?
no; failed or not implemented
yes; return fallback
9.
no; return resulting Observable
Legend:
invoke path
return path
observable:
items emitted
successful completion
error
Hystrix[Observable]Command method

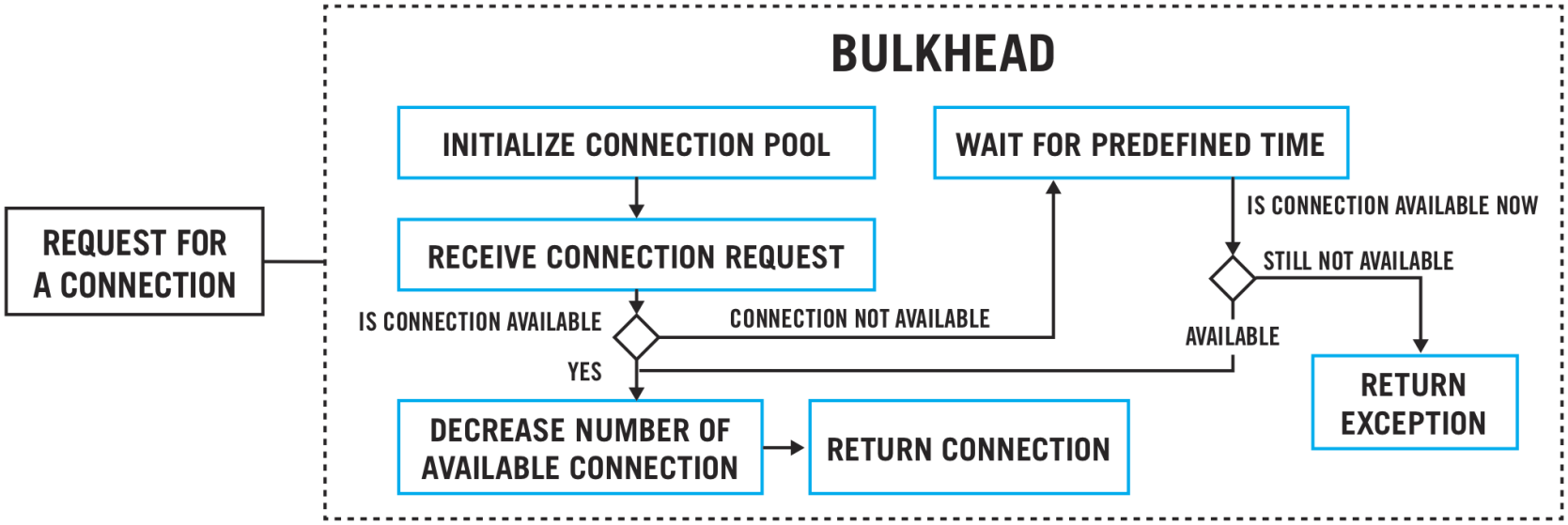
LEARN stuff IO

Hystrix

There are four ways you can execute the command, by using one of the following four methods of your Hystrix command object (the first two are only applicable to simple HystrixCommand objects and are not available for the HystrixObservableCommand):

- execute() — blocks, then returns the single response received from the dependency (or throws an exception in case of an error)
- queue() — returns a Future with which you can obtain the single response from the dependency
- observe() — subscribes to the Observable that represents the response(s) from the dependency and returns an Observable that replicates that source Observable
- toObservable() — returns an Observable that, when you subscribe to it, will execute the Hystrix command and emit its response

- Bulk Head Pattern

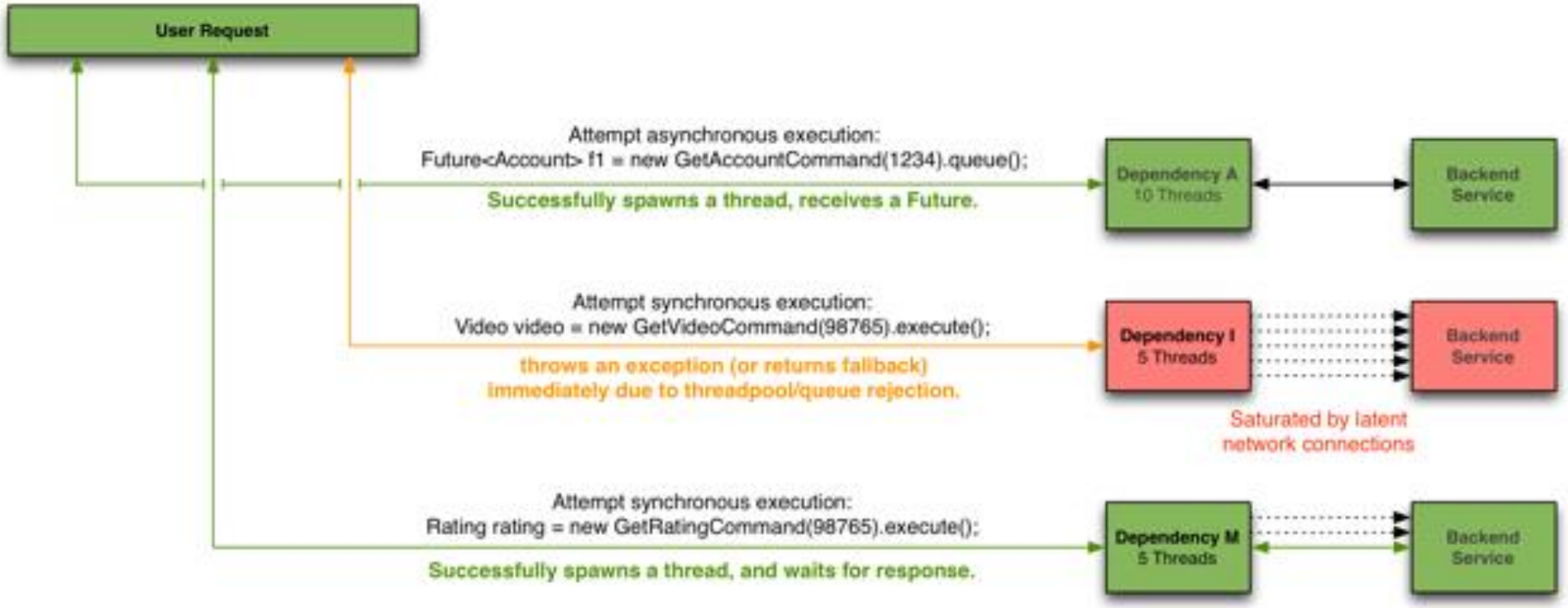


Hystrix

- Hystrix reports successes, failures, rejections, and timeouts to the circuit breaker, which maintains a rolling set of counters that calculate statistics.

- It uses these stats to determine when the circuit should “trip,” at which point it short-circuits any subsequent requests until a recovery period elapses, upon which it closes the circuit again after first checking certain health checks.

- Hystrix Threads and Thread Pools



Spring Hystrix

- Build a service in Spring Boot that will perform an operation
- Build a Hystrix project in Spring Boot
- Exposing a service in the Hystrix project that will call the first second
- Define a fallback method
- If needed, use configuration to enrich behavior

Spring Hystrix

```
@RestController
@EnableCircuitBreaker
public class SpringMicroservicesHystrixApplication {
   @Autowired
    private RestTemplate restTemplate;
    @RequestMapping("/startClient")
    @HystrixCommand(fallbackMethod="failover")
    public List<String> startClient(@RequestParam long time) throws InterruptedException{
        return this.restTemplate.getForObject("http://localhost:8888/service", List.class);
    }
    public List<String> failover(long time){
       return Arrays.asList("Default1", "Default2");
    }
}
```

Spring Hystrix

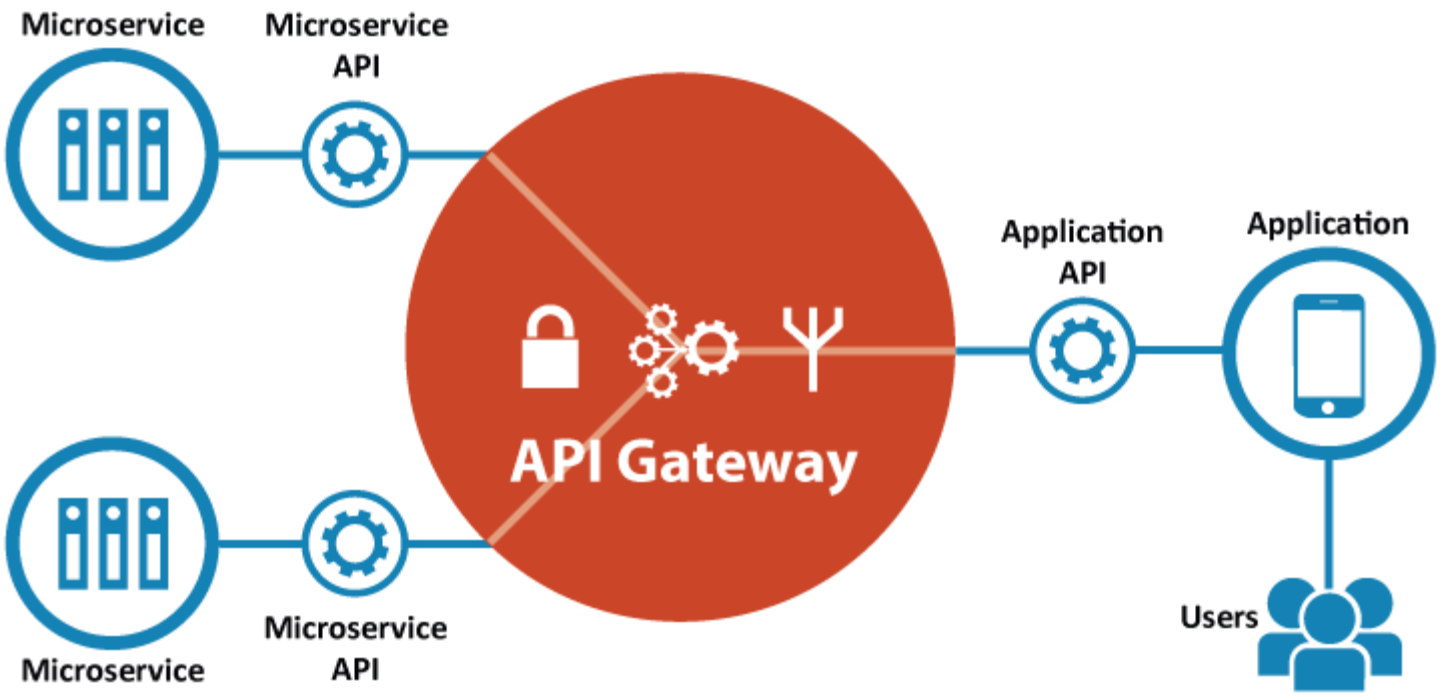
```
@SuppressWarnings("unchecked")
@RequestMapping("/startClient")
@HystrixCommand(fallbackMethod="failover", commandProperties={
      @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",
       value="500")
})
public List<String> startClient(@RequestParam long time) throws
InterruptedException{return this.restTemplate.getForObject("http://localhost:8888/service",
List.class);
}

public List<String> failover(long time){
   return Arrays.asList("Default1", "Default2");
}
```

Fault Tolerance Overview

- Fault tolerance is **not a "silver bullet"** solution to your problems
- This is a "use with care" solution
- There are **significantly fewer cases in which fault tolerance is a better approach** then standard exception management
- Faulty fail over mechanism can create business logic issues or considerably lower performance rather than improve it
- Fault tolerance is one of those cases in development where "**quality**" should always be treated with more importance then "**quantity**"

- API Gateway


Microservice
Microservice API
Application API
Application
API Gateway
Microservice
Microservice API
Users

API Gateway

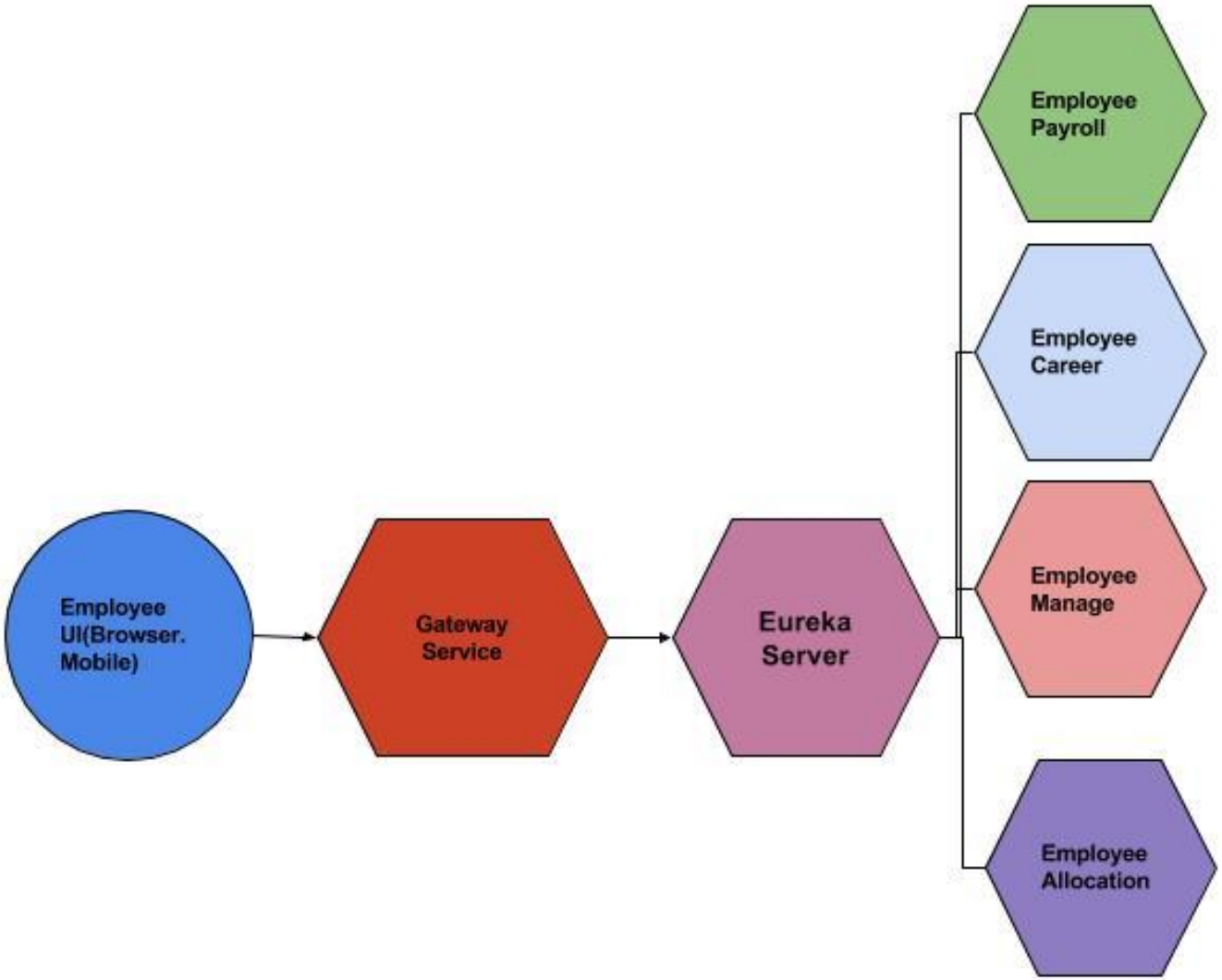
- A major benefit of using an API Gateway is that it encapsulates the internal structure of the application.

- Rather than having to invoke specific services, clients simply talk to the gateway. The API Gateway provides each kind of client with a specific API.

- This reduces the number of round trips between the client and application. It also simplifies the client code.

API Gateway

- It is yet another highly available component that must be developed, deployed, and managed.
- There is also a risk that the API Gateway becomes a development bottleneck. Developers must update the API Gateway in order to expose each micro-service's endpoints.
- It is important that the process for updating the API Gateway be as lightweight as possible.
- Otherwise, developers will be forced to wait in line in order to update the gateway.
- Despite these drawbacks, however, for most real-world applications it makes sense to use an API Gateway.



Zuul Enabled Edge Service

Zuul Responsabilities

- Apply micro-service authentication and security in the gateway layer to protect the actual services
- We can do micro-services insights and monitoring of all the traffic that are going in to the ecosystem by enabling some logging to get meaningful data and statistics at the edge in order to give us an accurate view of production.
- Dynamic Routing can route requests to different backend clusters as needed.
- We can do runtime stress testing by gradually increasing the traffic to a new cluster in order to gauge performance in many scenarios e.g. cluster has new H/W and network setup or that has new version of production code deployed.
- We can do dynamic load shedding i.e. allocating capacity for each type of request and dropping requests that go over the limit.
- We can apply static response handling i.e. building some responses directly at the edge instead of forwarding them to an internal cluster for processing.

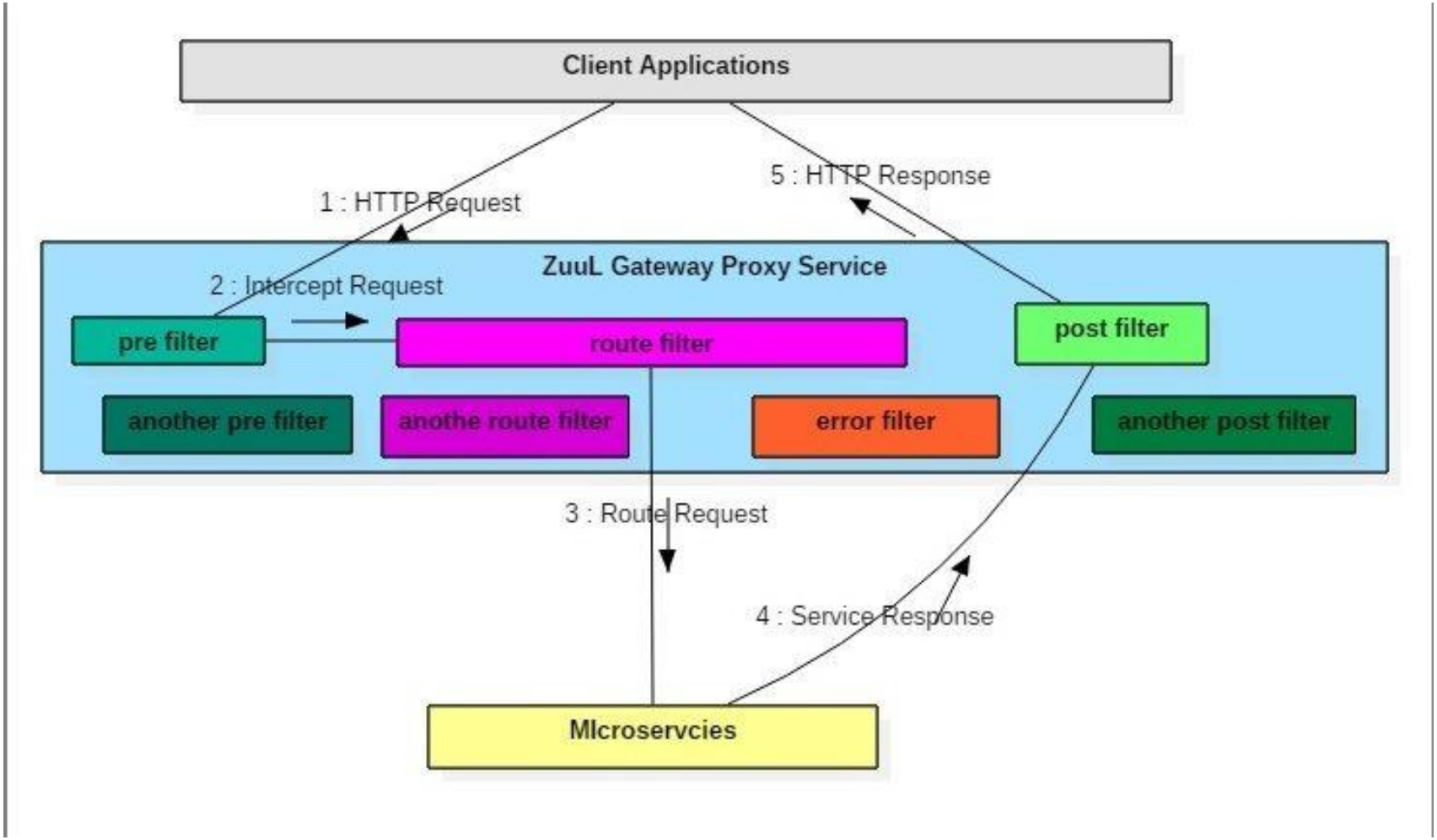
Spring Zuul

```
@EnableZuulProxy
@EnableDiscoveryClient
public class ZuulConfig {
        // main declaration
}
// in application.properties
server.port=8762
// app name
spring.application.name=zuul-server
eureka.instance.preferIpAddress=true
// use Eureka for finding services
eureka.client.registerWithEureka=true
eureka.client.fetchRegistry=true
// default zone
eureka.serviceurl.defaultzone=http://localhost:8761/eureka/
```

Spring Zuul

- Configuration for Zuul:

  zuul.routes.**apipath**.url=http://localhost:7777
  zuul.routes.**apipath**.path=/**apipath**/**
  zuul.prefix=/V1


Client Applications
1 : HTTP Request
5 : HTTP Response
ZuuL Gateway Proxy Service
2 : Intercept Request
pre filter
route filter
post filter
another pre filter
anothe route filter
error filter
another post filter
3 : Route Request
4 : Service Response
Mlcroservcies

Spring Zuul

```
public class MyZuulFilter extends ZuulFilter {
    @Override
    public Object run() {
        System.out.println("This request has passed through the custom Zuul Filter...");
        return null;
    }
    @Override
    public boolean shouldFilter() {
        return true;
    }
    @Override
    public int filterOrder() {
        return 1;
    }
    @Override
    public String filterType() {
        return "pre";
    }
}
```

Gateway Overview

- Can become a single point of failure for the system
- Business logic could creep into the gateway
- Filters should always be lightweight in order not to slow down the system
- Considering the gateway environment, filters should not have a decorator behavior for requests

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com

This is a free course from LearnStuff.io
– not for commercial use –