Routing Your Microservices Traffic

Richard Seroter
SENIOR DIRECTOR OF PRODUCT, PIVOTAL

@rseroter

Overview

Role of routing in microservices

Problems with the status quo

Describing Spring Cloud Ribbon

Configuring Ribbon

Customizing Ribbon

Describing Spring Cloud Zuul

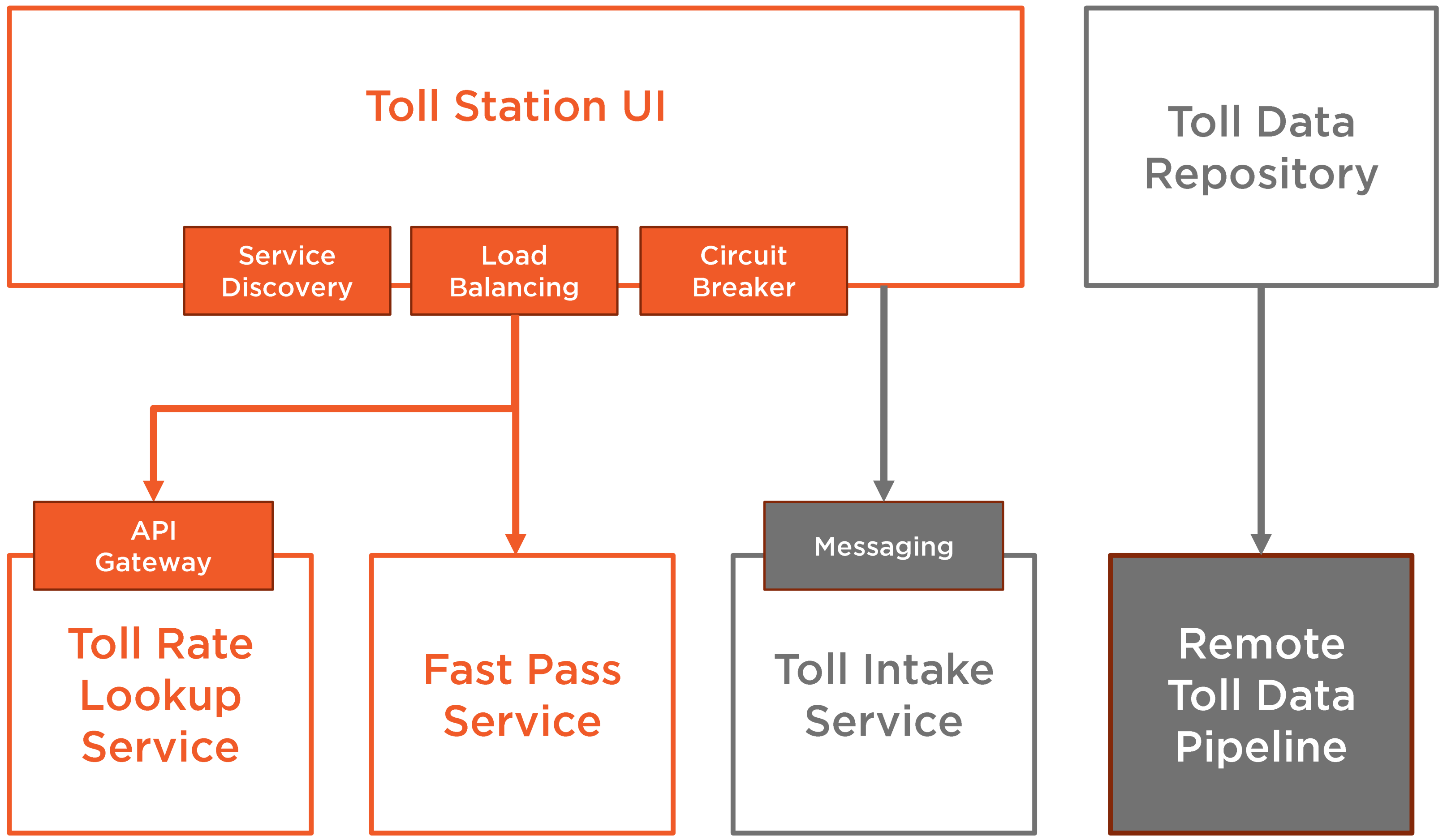
Creating a Zuul proxy

Exploring Zuul route configurations

Extending Zuul with filters

Summary

Capabilities That We Will Add in This Module



The Role of Routing in Microservices

Rapid decision making

Developer-centric options for public, private services

Address cross-cutting concerns

Offer data aggregation to limit chattiness

Problems with the Status Quo

Centralized load balancers, API gateways

Routing tech focused on public services

API granularity often at odds with client demands

Different performance, needs for different clients

Tools that don't account for constant change

Spring Cloud Ribbon

Client-side software load balancer.

Key Concepts

Ribbon offers: storage of server addresses (“server list”), server freshness checks (“ping”), and server selection criteria (“rules”).

Activate in code with @LoadBalanced, @RibbonClient annotations

Extend or override by using configuration classes

Configuring Ribbon in Your Applications

Ribbon listed under “Cloud Routing” on start.spring.io

[spring-cloud-starter-ribbon]

Provide list of servers in code, configuration or Eureka

Directly access client, or use @LoadBalanced RestTemplate

Built-in collection of behaviors and rules to use or deactivate

Demo

Open client application and see Ribbon dependency

Disable existing Eureka configuration

Add list of target servers to properties file

Add RibbonClient annotation

Run client application and see load balancing occur

How Ribbon and Eureka Work Together

Eureka simplifies server discovery

Server list comes from Eureka server

Ribbon delegates “ping”

Get back servers from same “zone” as client

Ribbon cache comes from Eureka client

Demo

Re-enable Eureka in client application

Update annotations

Test client application

```
@Configuration
public class DemoConfig {
 public IPing
  ribbonPing(IClientConfig config)
 {
  return new PingUrl();
 }

 public IRule
  ribbonRule(IClientConfig config)
 {

  return new
   WeightedResponseTimeRule();
 }
}
```

- Override behavior in @Configuration class
- Default Ping is NoOp
- Default Rule is ZoneAvoidance

```
ps-client.ribbon.NFLoadBalancerRuleClassName=
com.netflix.loadbalancer.WeightedResponseTime
Rule

ps-client.ribbon.NFLoadBalancerPingClassName=
com.netflix.loadbalancer.PingUrl

ps-client.ribbon.MaxAutoRetries=1

ps-client.ribbon.MaxAutoRetriesNextServer=1

ps-client.ribbon.ServerListRefreshInterval=
5000
```

◀ Override in properties

◀ Set retry behavior

◀ Set interval to refresh server list

Demo

Add Ribbon configuration to properties file of client application

Observe behavior of running application

Add class that changes ping and routing rules

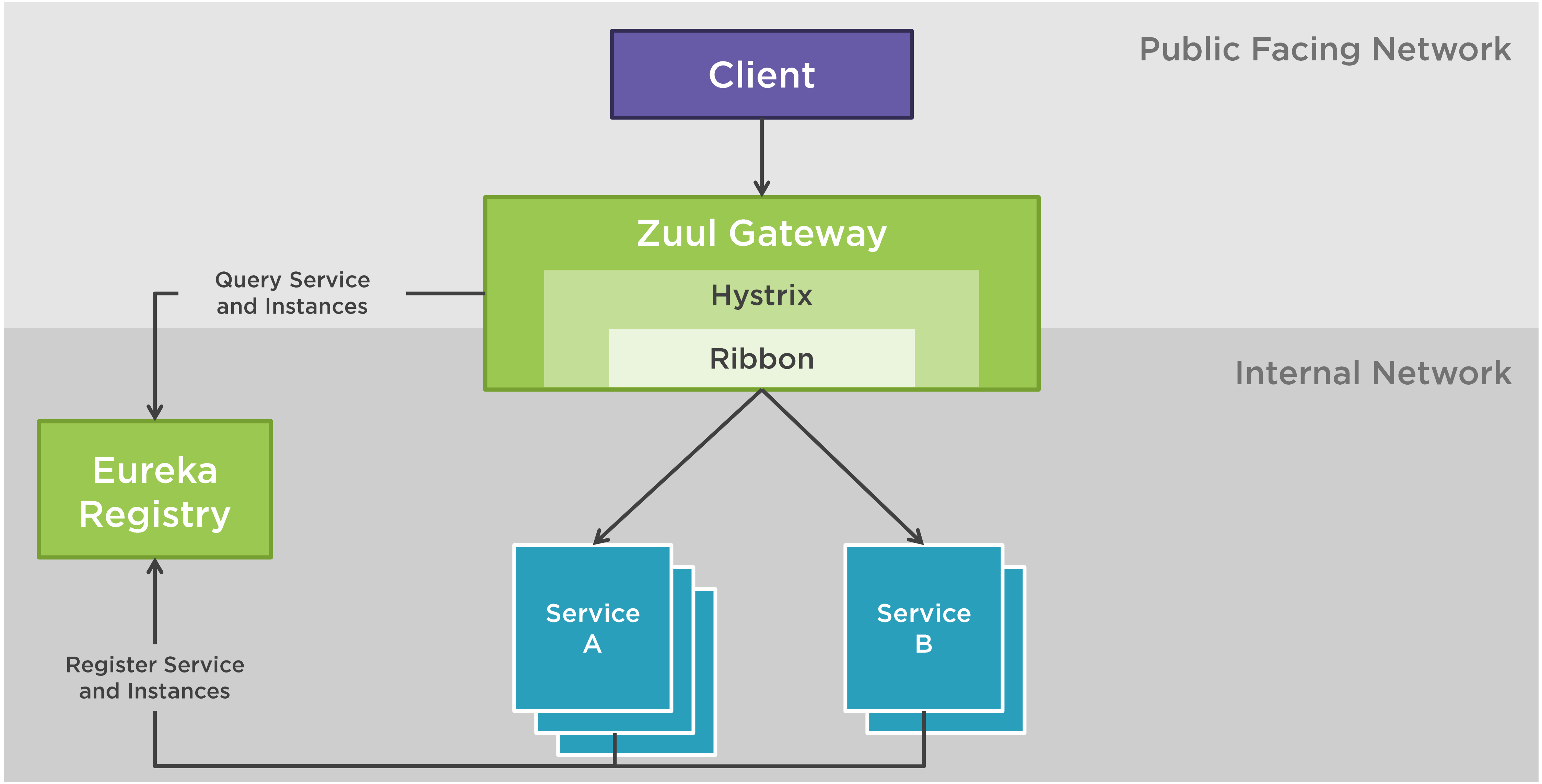
Update RibbonClient annotation

Observe behavior of running application

Spring Cloud Zuul

Embedded proxy for routing traffic in a microservices architecture.

How Zuul Works



Choosing a Spring Cloud Zuul Model

@ZuulProxy	@EnableZuulServer
Primed for reverse-proxy scenarios	“Blank” Zuul server
Proxy filters automatically added	Passthrough requests by default
Can integrate with Eureka, Ribbon	No service discovery
Additional /routes endpoint	Add filters manually

Creating a Zuul Proxy with Configurable Routes

Add actuator and spring-cloud-starter-zuul references

Optionally add Eureka for discovery

Backend location can be URL or service ID

Can ignore discovered services

Fine-grained control over path of route

Can trigger refresh of route configuration

```
zuul.routes.employees.path=/emps/*

zuul.routes.employees.url=http://server1:6551/employees
```

Configuring Routes – Fixed Endpoint

Define Zuul proxy path

Set “url” to endpoint

```
zuul.routes.employees.path=/emps/*

zuul.routes.employees.serviceId=employees

ribbon.eureka.enabled=false

employees.ribbon.listOfServers=server1,server2,server3
```

Configuring Routes – Load Balanced URLs

Define a service ID

Disable Eureka support in Ribbon

Set list of servers for Ribbon to load balance

Configuring Routes – Simple Discovery

If add Eureka to classpath, Zuul will automatically forward requests

```
zuul.ignoredServices=*

zuul.routes.employees.path=/emps/*
```

Configuring Routes – Ignore Discovered Services

“Ignored Services” tells Zuul to not automatically add services

In above example, all services are ignored <u>except</u> for ”employees” one

```
zuul.ignoredServices=*
zuul.routes.employees.path=/emps/*
zuul.routes.employees.serviceId=employees_service
```

Configuring Routes – Fine-Grained Route

The “employees_service” maps to Eureka-registered service name

Path reflects the proxy route

```
zuul.ignoredServices=*

zuul.routes.employees=/emps/*

zuul.prefix=/api
```

Configuring Routes – Adding Route Prefix

Can add, keep, or remove route prefixes

For above request, URL would be: http://[zuul-proxy]/api/emps/100

Demo

Create new project from Spring Initializr

Annotate class to turn into Zuul proxy

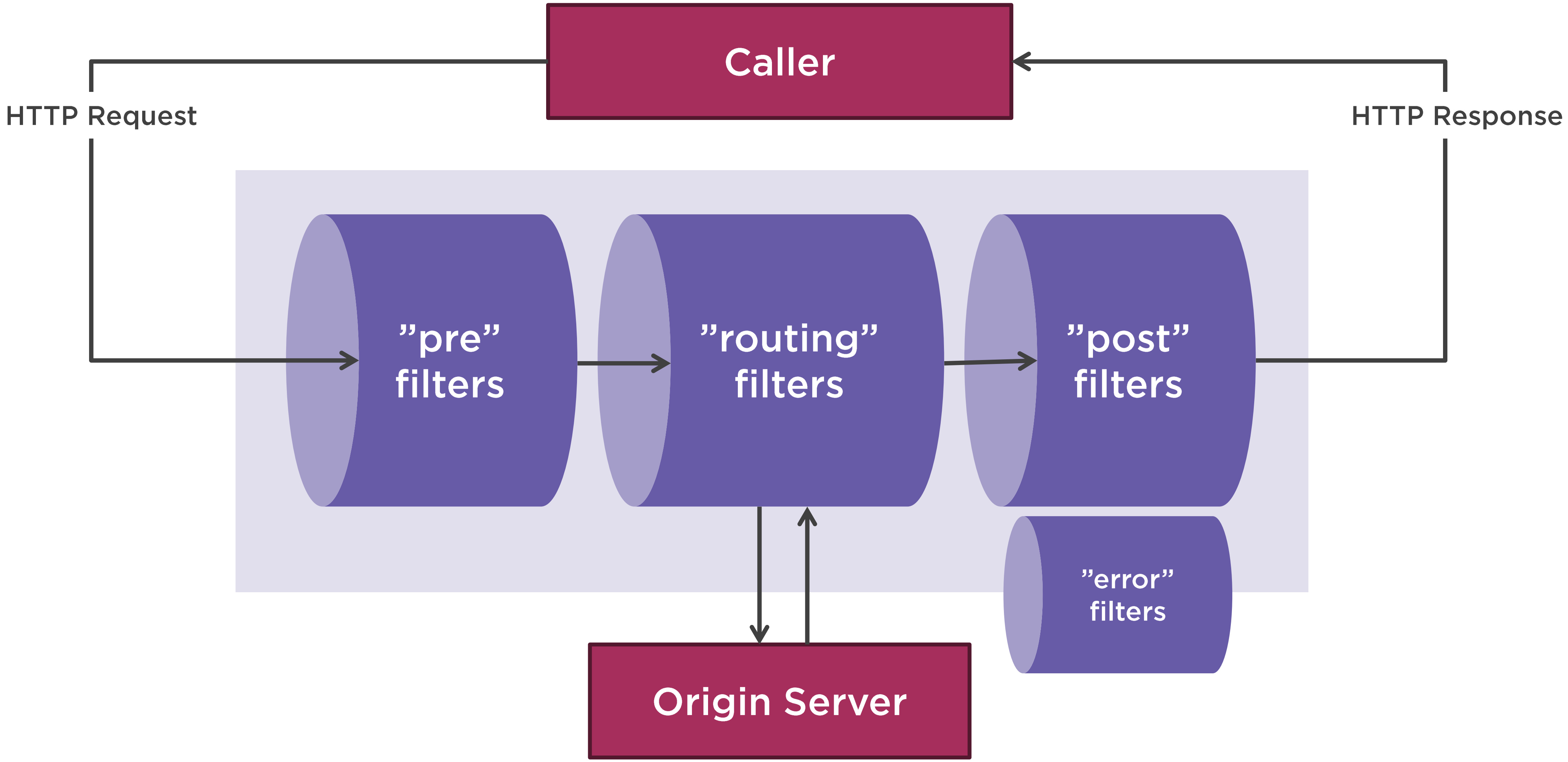
Set up with local URLs, no Eureka

Add Eureka with no whitelisting

Lock down allowable services and experiment with routes

Introduce prefix handling

About Zuul Filters and Stages



Working with Zuul Filters

Pre, Routing, Post, and Error filters

Wide range of standard filters automatically added

Filter has: type, execution order, execution criteria, and action

Filters share a RequestContext

Disable filters in property file

Create beans for Zuul to see custom filters

Demo

Create ”pre” filter that logs requests from specific callers

Create another “pre” filter that adds “start time” property to RequestContext

Create “post” filter that checks “start time” and logs the call duration

Add filter beans to Zuul proxy

Summary

Overview

Role of routing in microservices

Problems with the status quo

Describing Spring Cloud Ribbon

Configuring Ribbon

Customizing Ribbon

Describing Spring Cloud Zuul

Creating a Zuul proxy

Exploring Zuul route configurations

Extending Zuul with filters