Understanding

Microservices with Spring Cloud

—*Nandakumar Purohit*

Microservices with Spring Cloud

Agenda

- What is Netflix OSS
- What is Spring Cloud Eureka
- Locate & Consume Microservices
- Protecting Microservices with Circuit Breakers
- Routing Requests using Load Balancing & Micro Proxies
- Stitching Services together using Messaging
- Building Data Pipelines

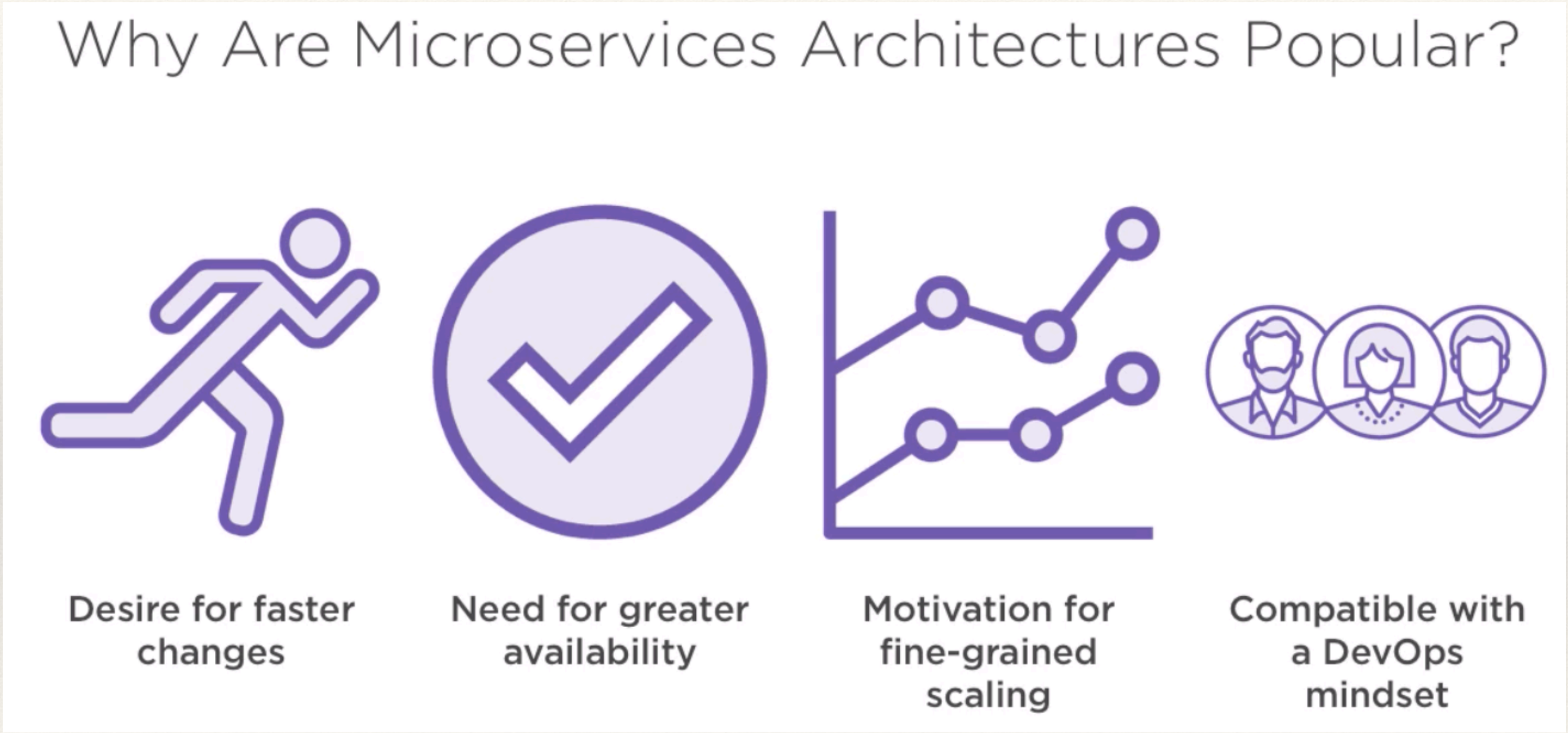
Microservices with Spring Cloud

Quick Look

- Why are Microservices Architectures so popular?
- Core Characteristics of Microservices
- Coordination challenges that emerge with Microservices
- Microservices with Spring Cloud

Microservices with Spring Cloud

Quick Look



Microservices with Spring Cloud

Core Characteristics of Microservices

Components exposed as services	Tied to a specific domain	Loosely coupled
Built to tolerate failure	Delivered continuously via automation	Built and run by independent teams

Microservices with Spring Cloud

Coordination Challenges with Microservices

- How do you locate services when hosts change as services get updated or scaled?
- How do you reduce single points of failure in a distributed architecture?
- What can you do to prevent cascading failures when one service starts misbehaving?
- Where should you perform load balancing of dynamic services?
- How can you dynamically adjust the routing tier?
- Whats a good way to introduce loose coupling to an architecture?

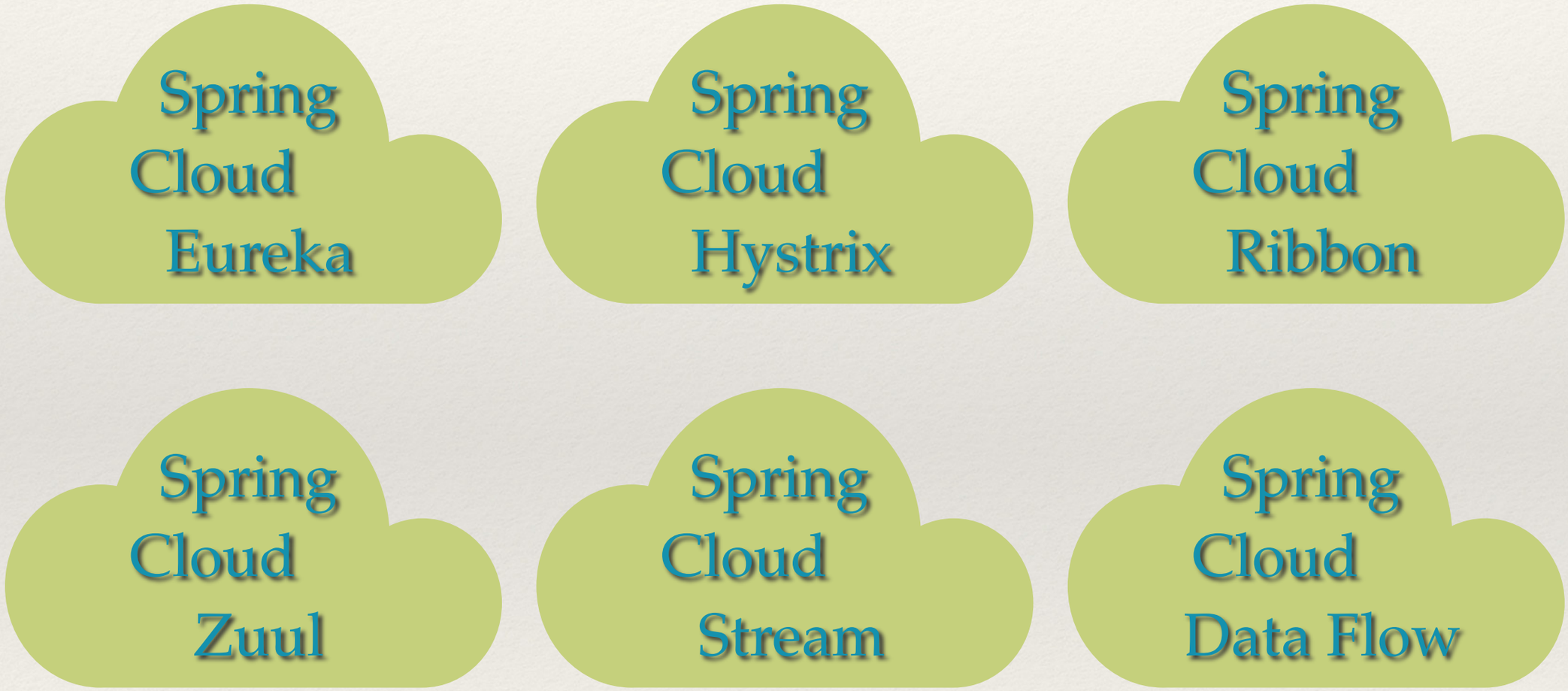
Microservices with Spring Cloud

Microservices Scaffolding with Spring Cloud

- Released in March 2015
- Build Common distributed systems patterns
- Open Source Software
- Optimized for Spring Applications
- Run Anywhere
- Includes Netflix OSS Technology

Microservices with Spring Cloud

Spring Cloud Projects



Microservices with Spring Cloud

What is Netflix OSS

- Role of Service Discovery
- Problems with the Status Quo
- Describing Spring Cloud Eureka
- Creating a Eureka Server
- Registering Services with Eureka
- Discovering Services with Eureka
- Configuring Health information
- Reviewing the High Availability Setup
- Options for Advanced Configuration

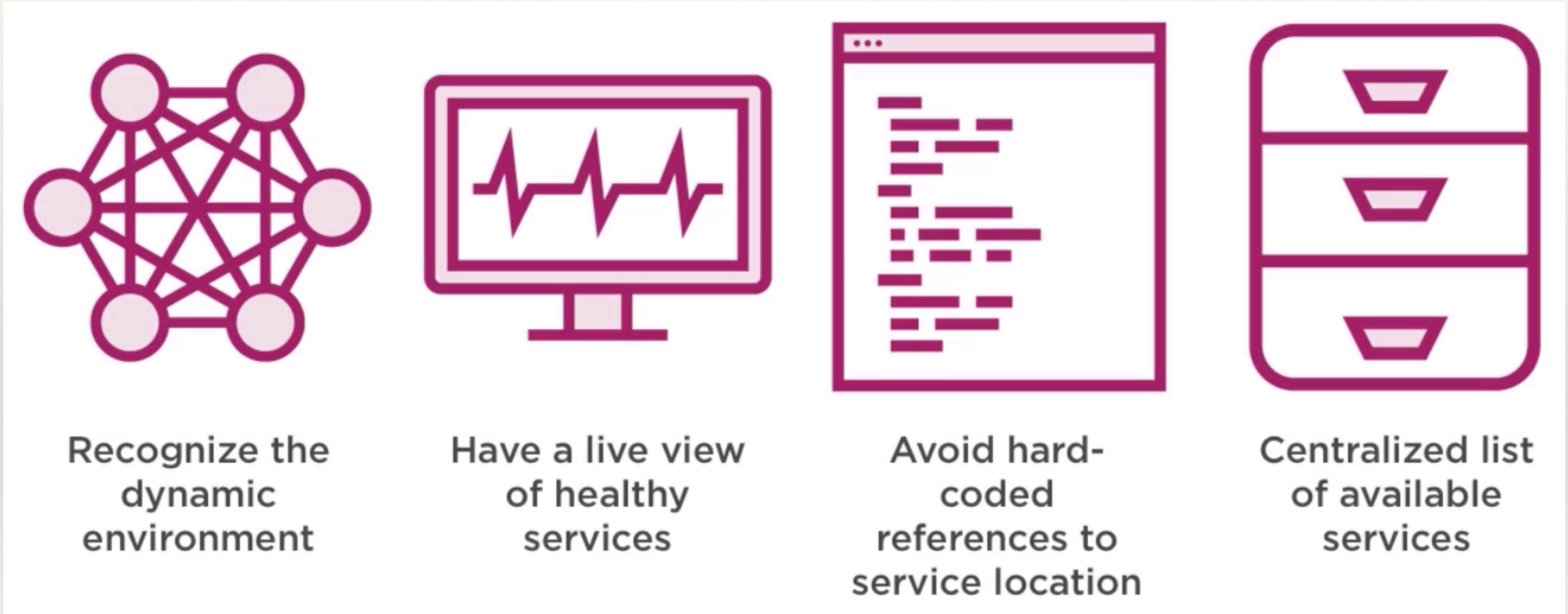
Microservices with Spring Cloud

Service Discovery

- Role of Service Discovery
- Problems with the Status Quo
- Describing Spring Cloud Eureka
- Creating a Eureka Server
- Registering Services with Eureka
- Discovering Services with Eureka
- Configuring Health information
- Reviewing the High Availability Setup
- Options for Advanced Configuration

Microservices with Spring Cloud

Role of Service Discovery



Microservices with Spring Cloud

Problems with the Status Quo

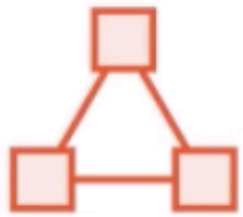
- Outdated configuration management DBs
- Simplistic HTTP 200 health checks
- Limited load balancing for middle-tier
- DNS is insufficient for Microservices
- Registries can be single points of failure

Microservices with Spring Cloud

Introducing Spring Cloud Eureka

First released by Netflix OSS team in 2012

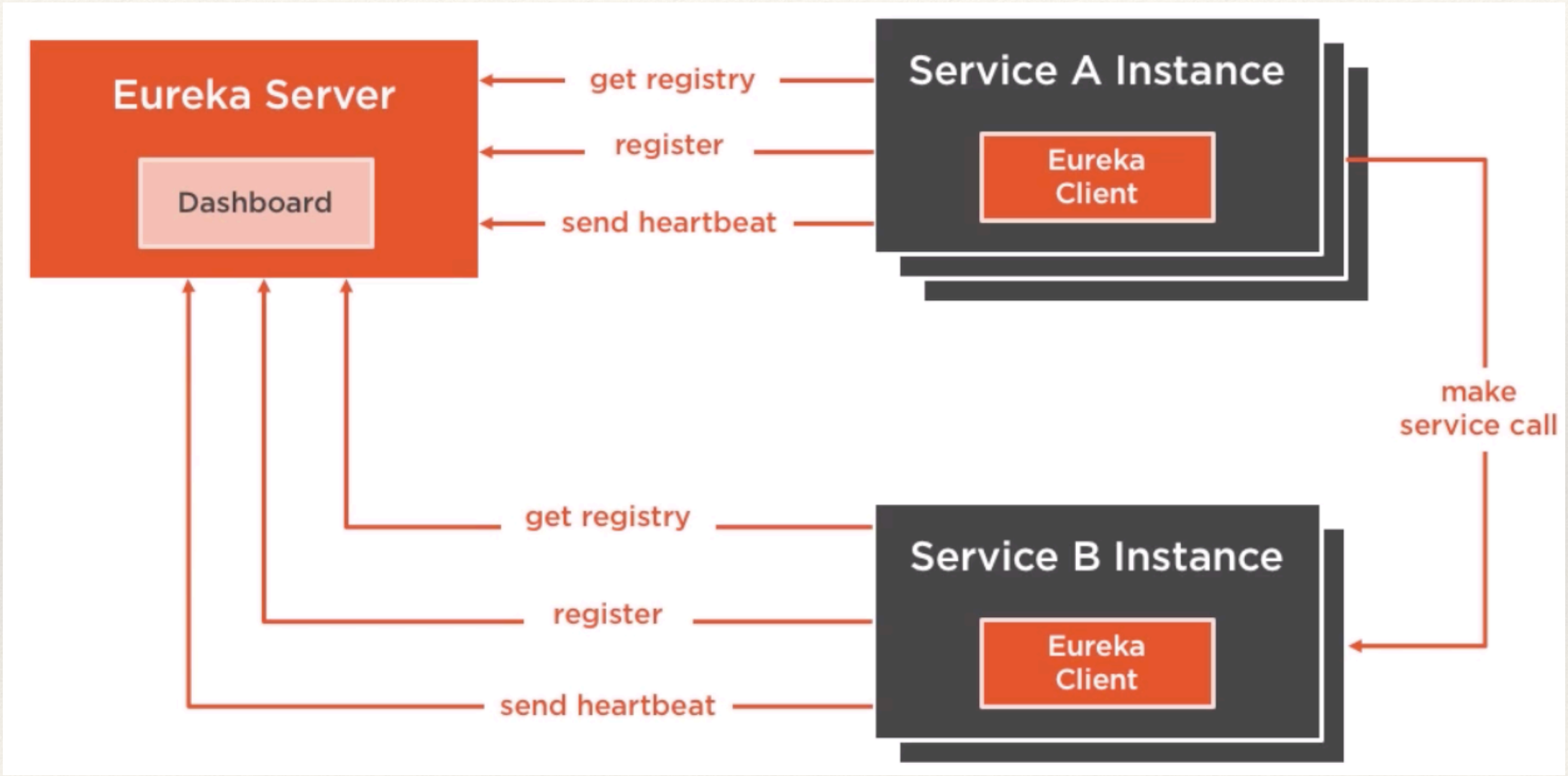
Used for middle-tier load balancing



Integrated into many other Netflix projects

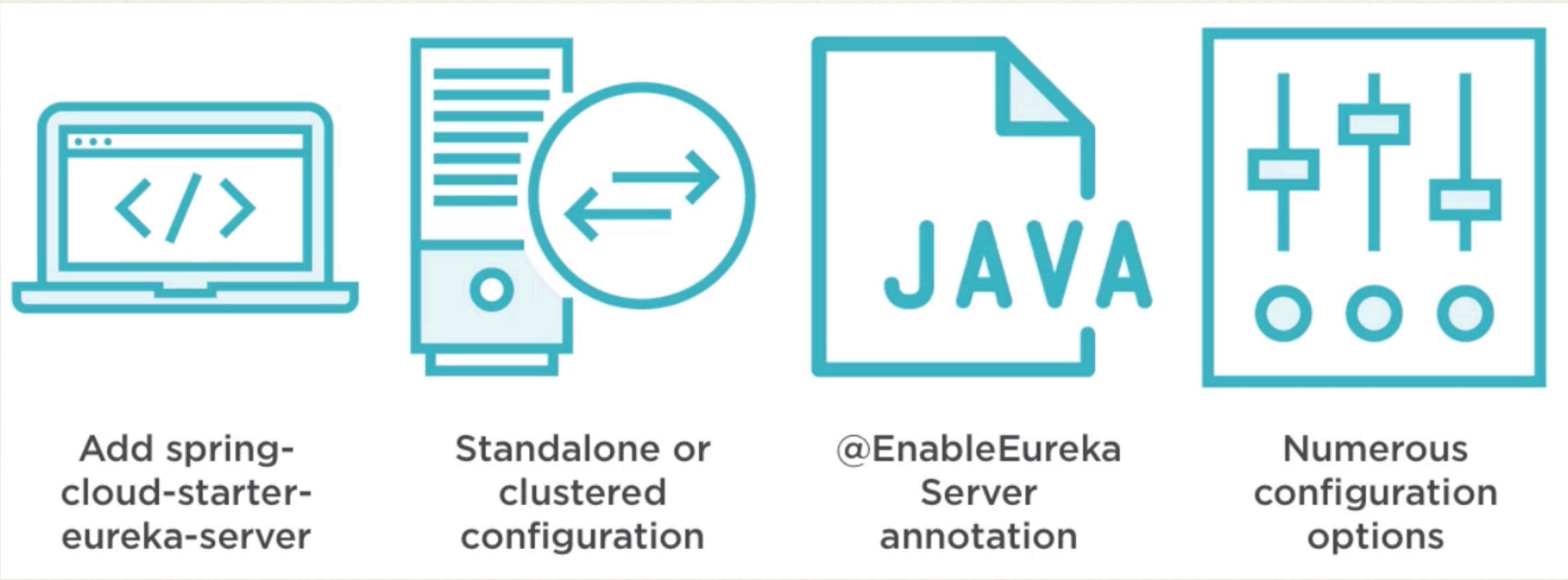
Microservices with Spring Cloud

Components of Eureka Environment



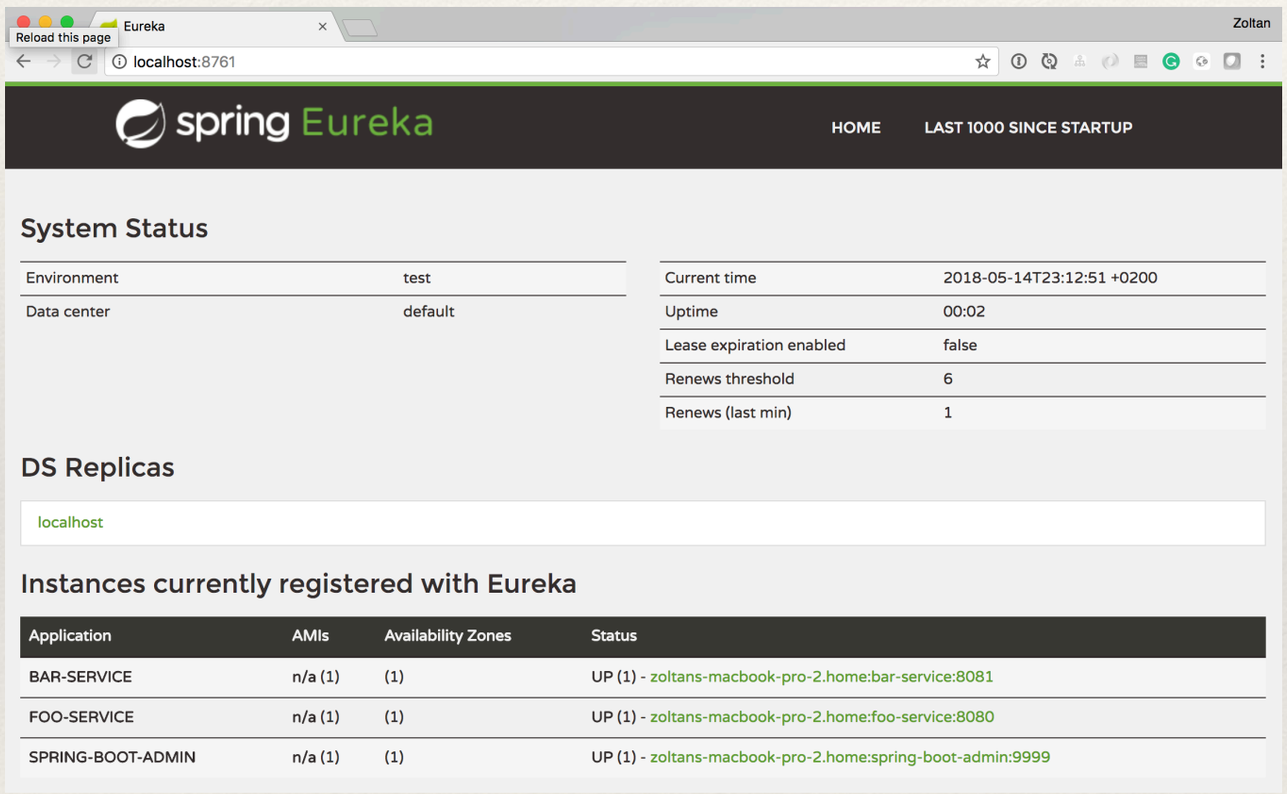
Microservices with Spring Cloud

Creating a Eureka Server



Microservices with Spring Cloud

Eureka Dashboard



- Enabled by default
- Shows environment info
- Lists registered services & instances
- View Service health

Microservices with Spring Cloud

Use Case - Creating a Eureka Server

- Create a New Project (Spring Starter / Initializr)
- Add Eureka Server dependency
- Annotate Primary Class
- Set Application Properties
- Start Server and view Dashboard

Microservices with Spring Cloud

Service Registration with Eureka

Eureka in classpath leads to registration	Service name, host info sent during bootstrap	@EnableDiscoveryClient and @EnableEurekaClient
Sends heartbeat every 30 seconds	Heartbeat can include health status	HTTP or HTTPS supported

Microservices with Spring Cloud

Service Registration with Eureka

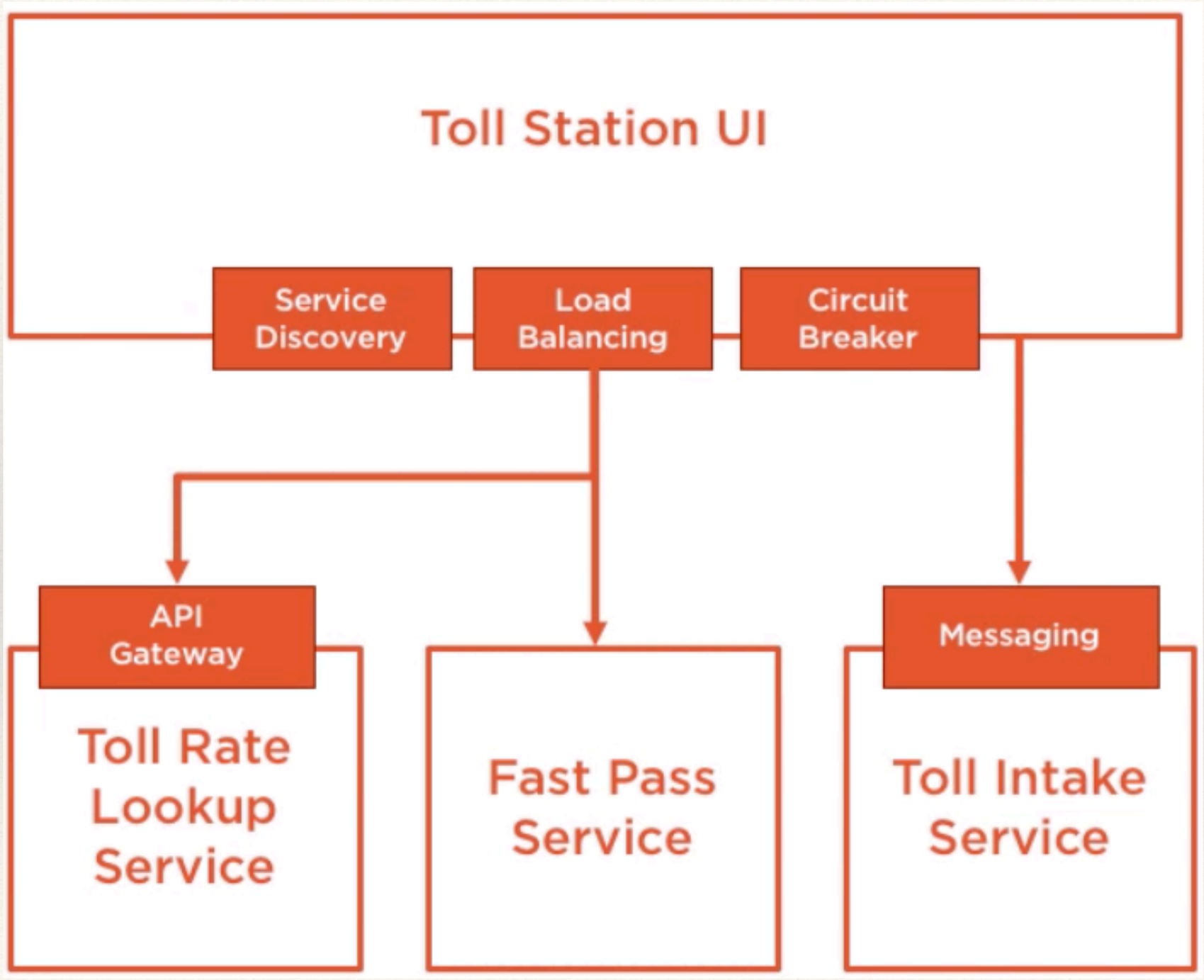
```
@EnableEurekaClient
@RestController
@SpringBootApplication
public class
PsPlaceholderEurekaServiceApplication {


 public static void main(String[] args) {
 SpringApplication.run(
  PsPlaceholderEurekaServiceApplication.class,
  args);
}
 @RequestMapping("/")
 public String SayHello() {
 System.out.println(”hi there!");
 return "hello from Spring Boot!";
 }
}
```

- Single annotation needed
- Configuration for app name, server location, health checks, and more

Microservices with Spring Cloud

Use Case Application Architecture



Microservices with Spring Cloud

Use Case - Registering Multiple Services with Eureka

- Open exiting “toll rate” Microservice
- Add project dependency on Eureka
- Annotate Primary class
- Add Bootstrap and application properties
- Start up Microservice and observe in registry
- Start a second instance and observe in registry
- Repeat above steps for Fast Pass Service Microservice

Microservices with Spring Cloud

Service Discovery with Eureka

@EnableDiscoveryClient and @EnableEurekaClient	Client works with local cache	Cache refreshed, reconciled regularly
Manually load balance, or use Ribbon	Can prefer talking to registry in closest Zone	May take multiple heartbeats to discover new services

Microservices with Spring Cloud

Use Case - Discovering and Load Balancing Services

- Open "**toll rate billboard**" application
- Add Dependency on Eureka
- Add bootstrap.properties
- Update application properties
- Annotate Primary class
- Add Load Balanced RestTemplate
- Replace hard-coded URL with registry lookup
- Test out "toll rate billboard" application
- Repeat with "**fast pass console**" application

Microservices with Spring Cloud

Configuring Health Check

- Heartbeat doesn't convey health
- Possible to include health information
- Can extend and create own health check

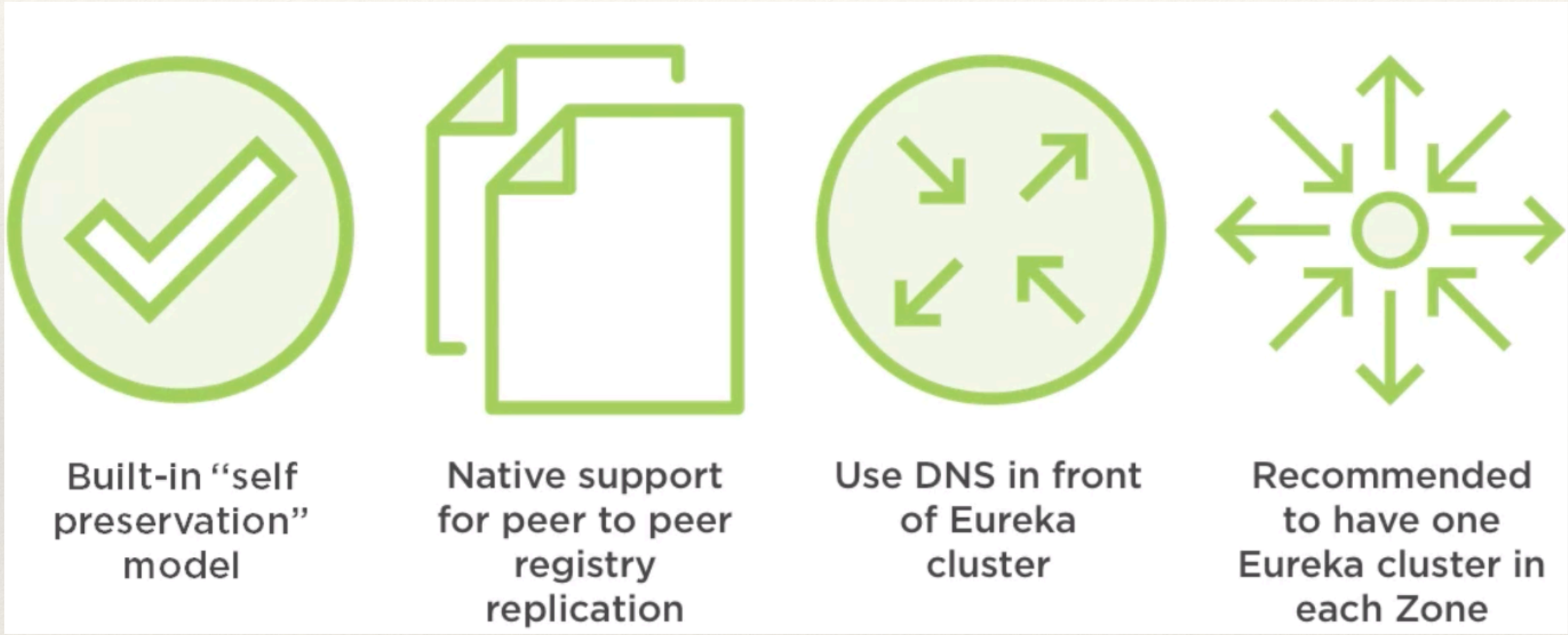
Microservices with Spring Cloud

Use Case - Adding Custom Health Check

- Return to “toll rate” Microservice & add a custom health check
- Start up Microservice & wait for error
- See service taken out of rotation by Eureka
- application.properties
  - eureka.client.healthcheck.enabled = true

Microservices with Spring Cloud

High Availability Architecture for Eureka



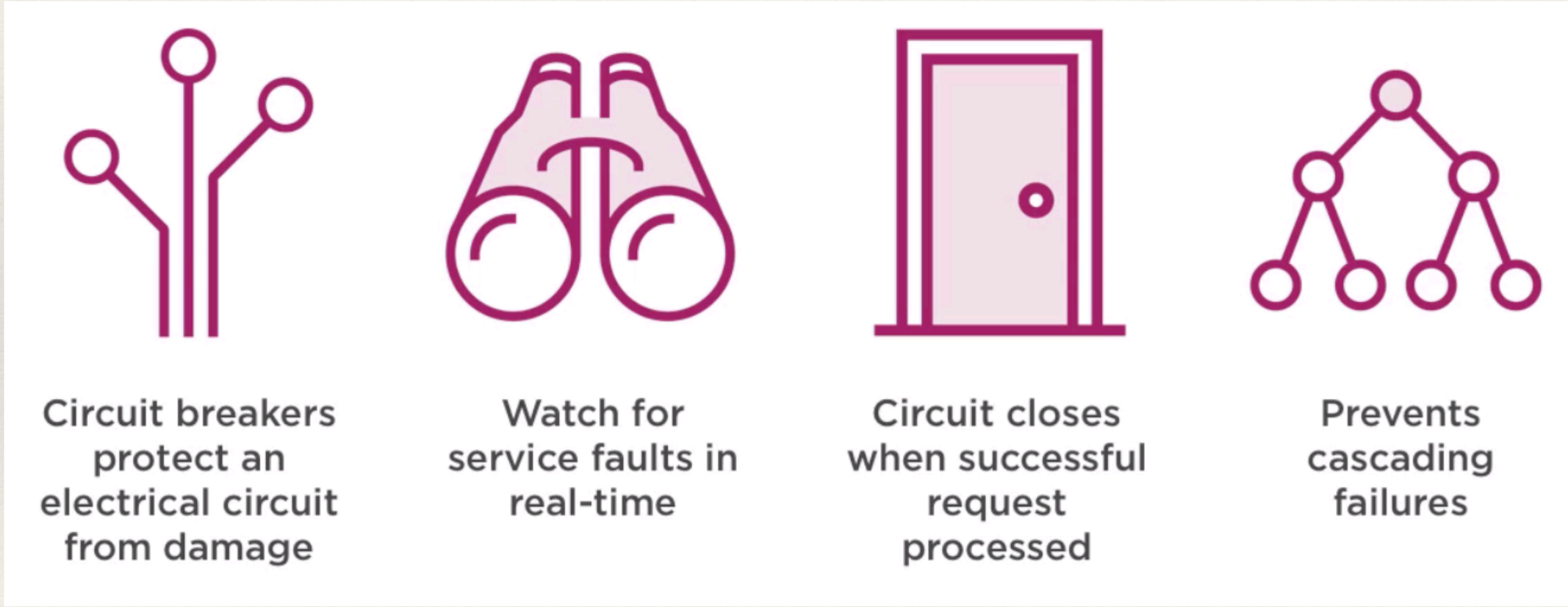
Microservices with Spring Cloud

Advanced Configuration Options

- Dozens & dozens of configuration flags
- Set cache refresh intervals
- Set timeouts
- Set connection limits
- Set service metadata maps
- Override default service, health endpoints
- Define replication limits, timeout, retries

Microservices with Spring Cloud

The Role of Circuit Breakers in Microservices



Microservices with Spring Cloud

Circuit Breakers - Problems with the Status Quo

- Major dependence on server to be resilient
- Load balancers are network calls too
- Hard to detect and recover via automation
- Solutions can be intrusive to codebase or add significant overhead
- Resilience engineering often not part of service logic or behavior

Microservices with Spring Cloud

Circuit Breakers - Spring Cloud Hystrix

Hystrix is a library for enabling Resilience in Microservices

- Supported patterns include bulkhead, fail fast, graceful degradation (Ex: Fail silently with fallback response)
- Hystrix wraps calls to external dependencies and monitors metrics in real time.
- Invokes failover method when encountering exceptions, timeouts, thread pool exhaustion or too many previous errors
- Hystrix periodically sends request through to see if service has recovered

Microservices with Spring Cloud

Spring Cloud Hystrix - How does it work?

- Circuit breaker via Annotations at class, operation level
- Hystirx manages the thread pool, emits metrics
- Dashboard integrates with Eureka to lookup services
- Dashboard pulls metrics from instances or services

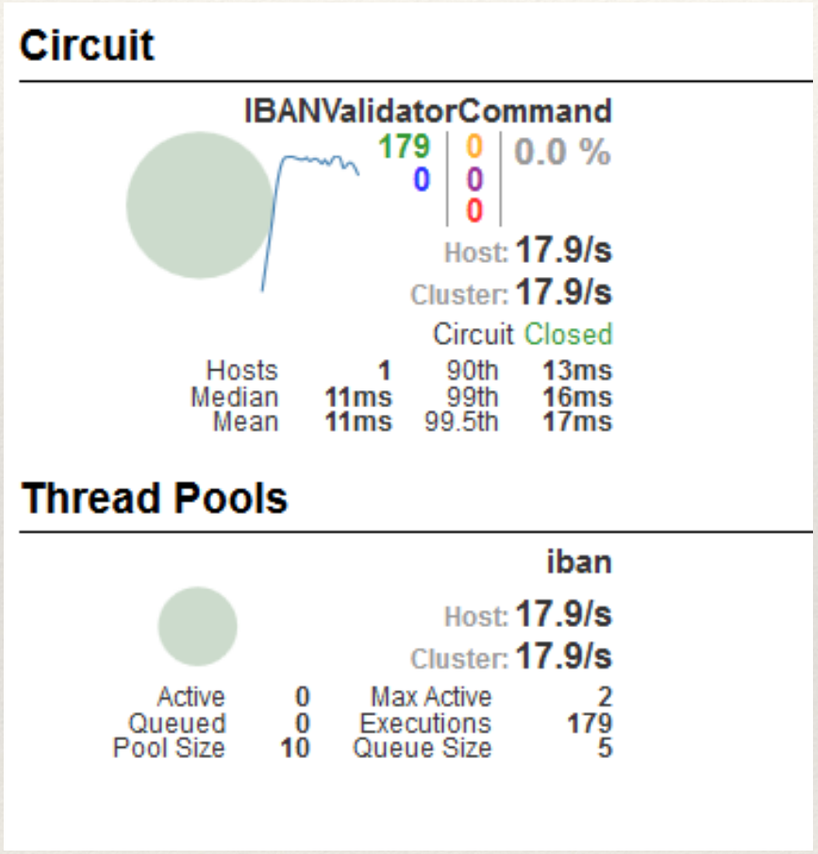
Microservices with Spring Cloud

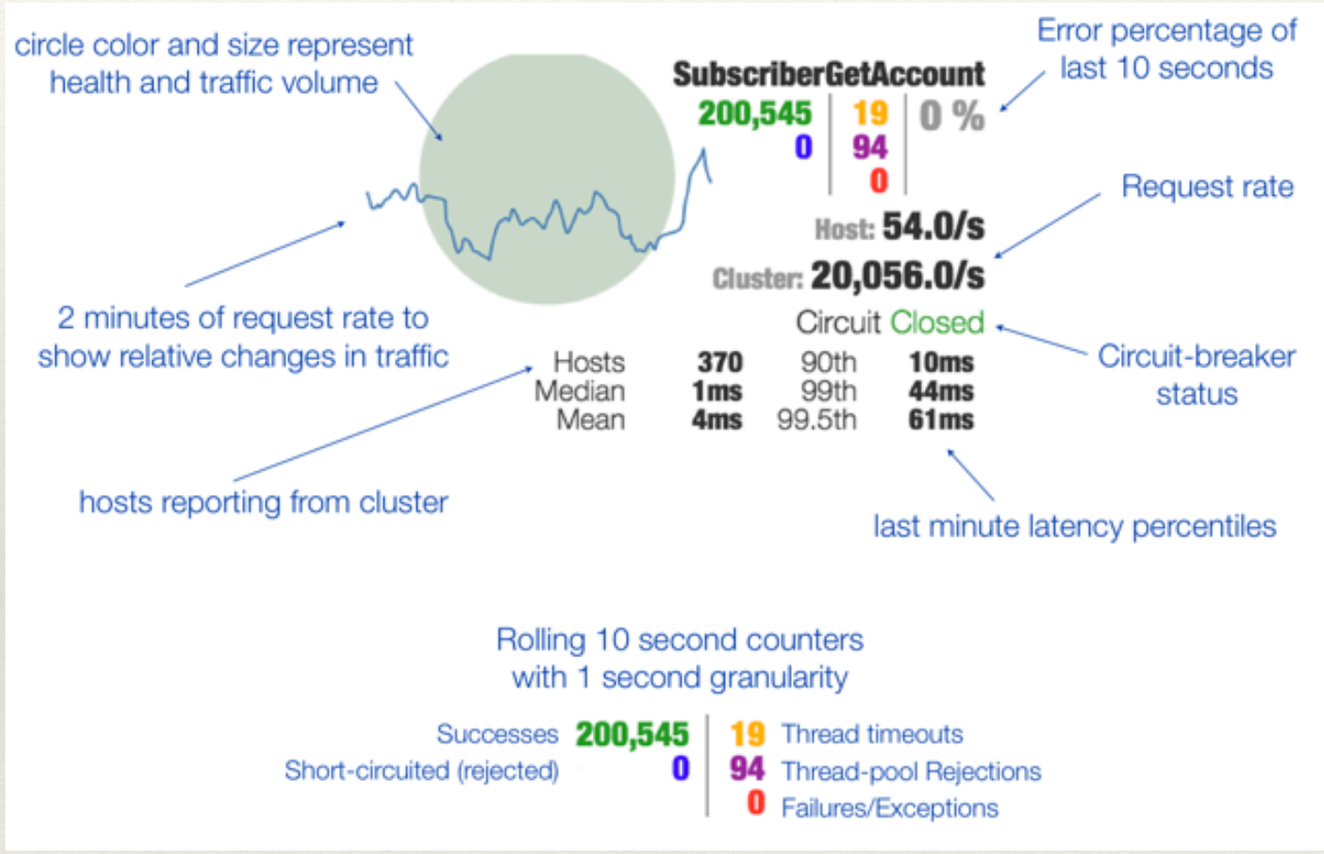
Use Case - Adding Hystrix to Microservices

- Review existing Eureka-enabled Microservices
- Open existing client applications
- Add Hystrix dependency to client applications
- Introduce Circuit Breaker to code
- Startup & Call client applications
- Review Circuit & Metrics endpoints

Microservices with Spring Cloud

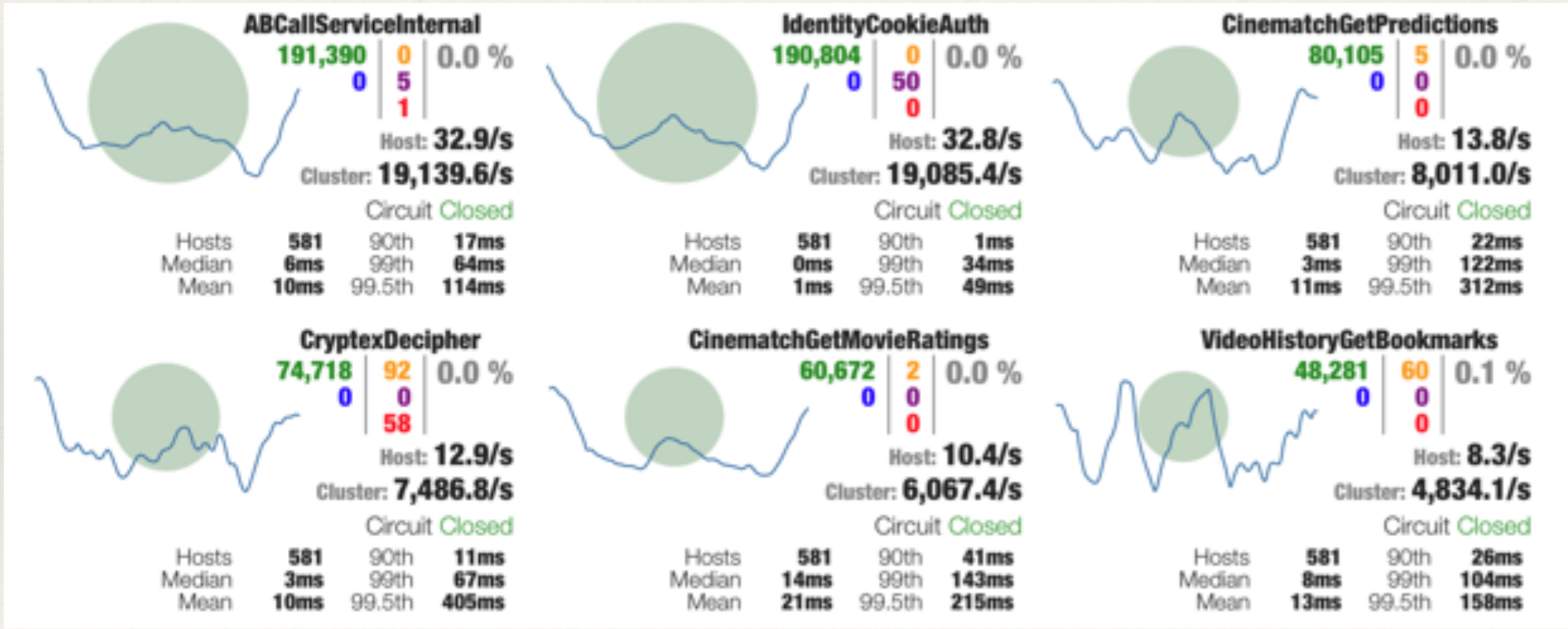
Hystrix Dashboard





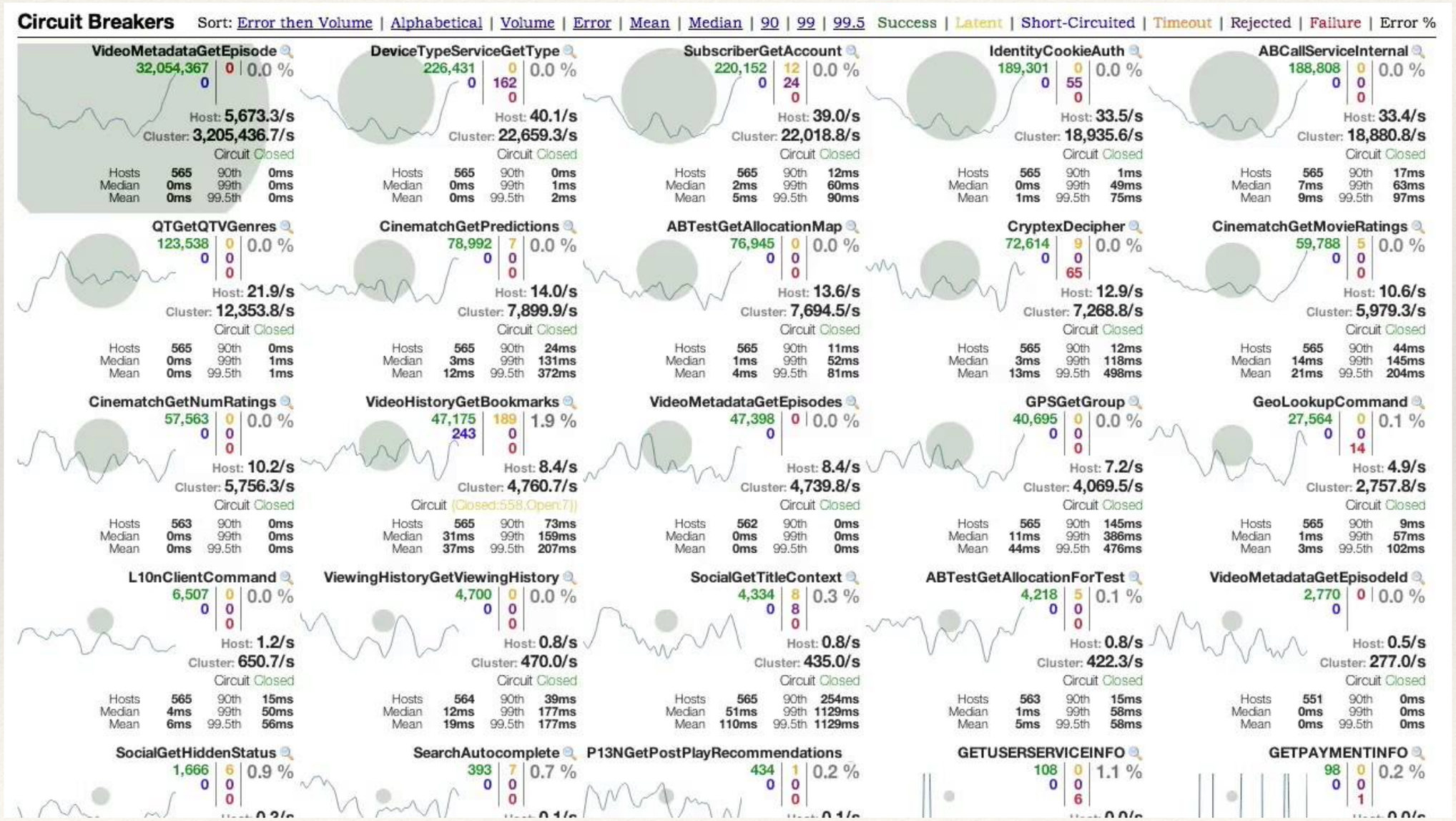
Microservices with Spring Cloud

Hystrix Dashboard



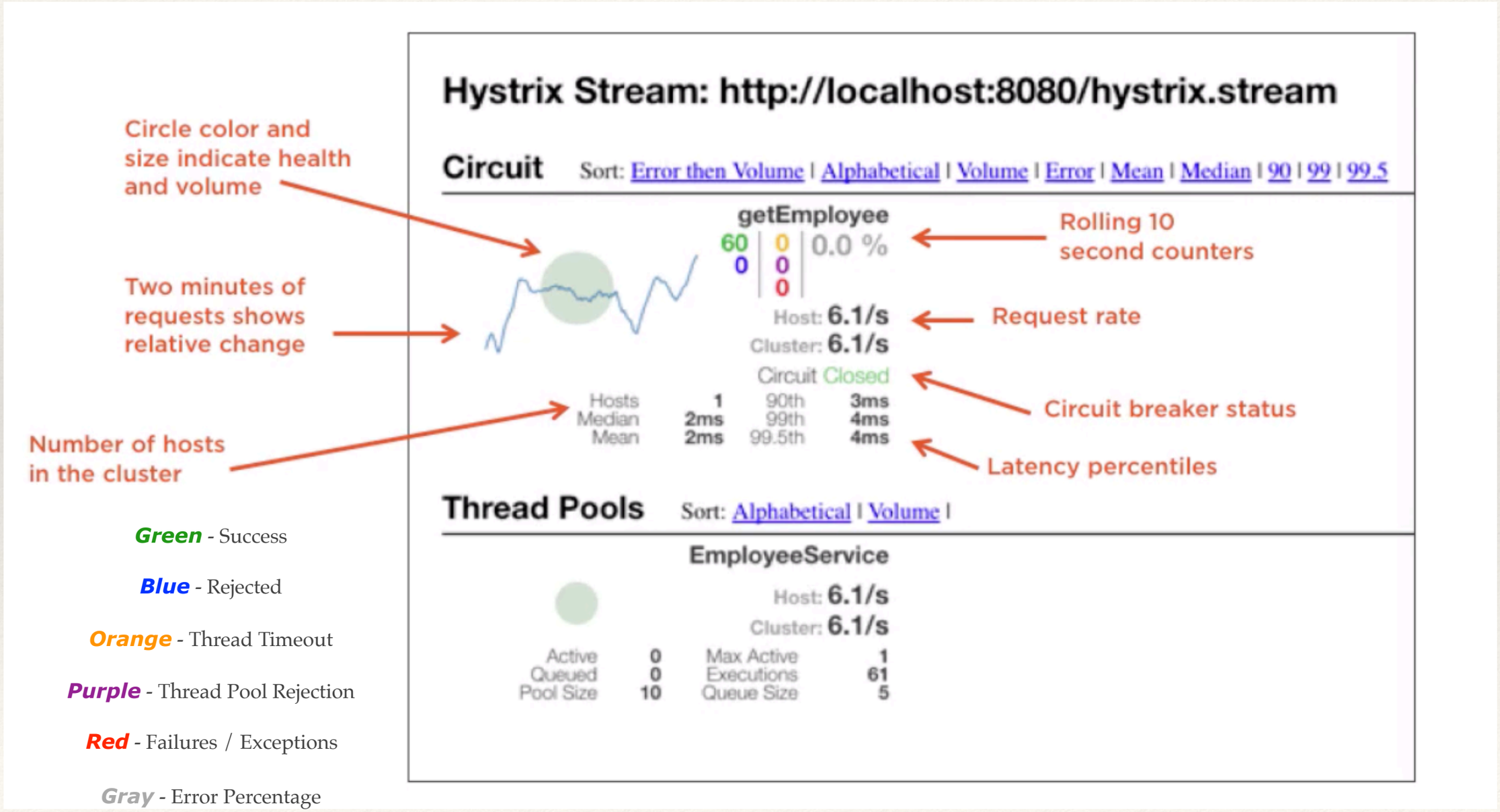
Microservices with Spring Cloud

Hystrix Dashboard



Microservices with Spring Cloud

Hystrix Dashboard - What is visible?



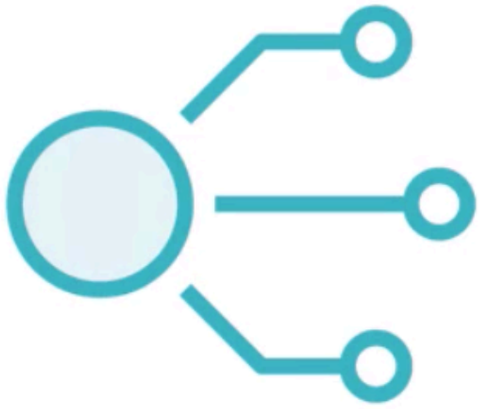
Microservices with Spring Cloud

Use Case - Setting Up Hystrix Dashboard

- Create a new Project via Spring Starter
- Select Hystrix Dashboard dependency
- Annotate Main class
- Start up the Hystrix Dashboard Application
- Load streams for toll rate billboard & fast pass console

Microservices with Spring Cloud

What does Turbine add to Hystrix?



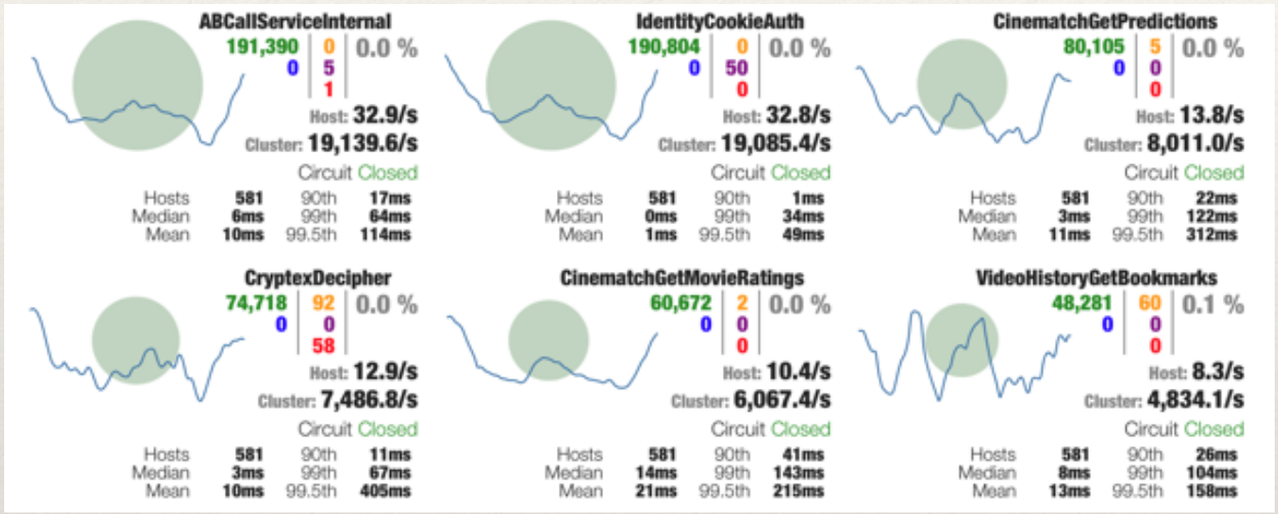
Combine metrics from multiple service instances

Integrates with Eureka to pull instance info

Turbine Stream uses messaging to aggregate service metrics

Microservices with Spring Cloud

Using Turbine Stream



Server-side

- Standalone Spring Boot App
- Add spring-cloud-starter-turbine stream
- Add spring-cloud-starter-stream-*

Client-side

- Add spring-cloud-starter-hystrix-stream
- Add spring-cloud-starter-stream-*

Dashboard

- Point to http://host:port of Turbine App

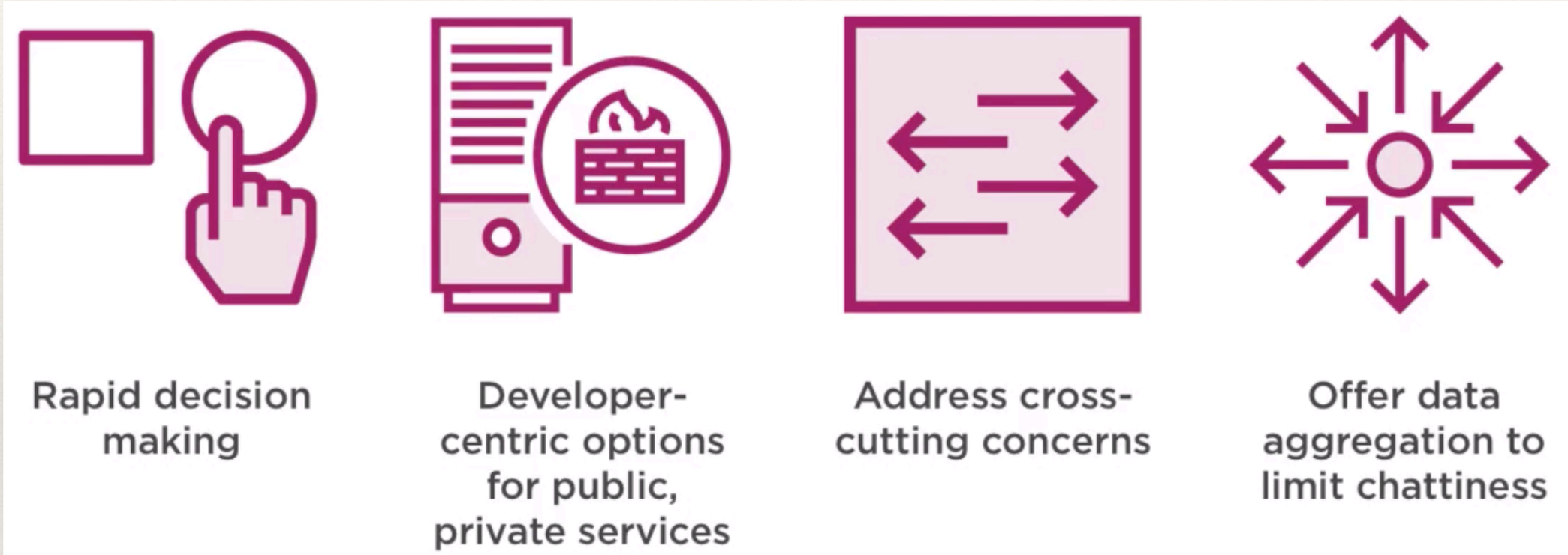
Microservices with Spring Cloud

Traffic Routing of Microservices

- Role of routing in Microservices
- Describing Spring Cloud Ribbon
- Configuring Ribbon
- Customizing Ribbon
- Describing Spring Cloud Zuul
- Creating a Zuul Proxy
- Exploring Zuul route configurations
- Extending Zuul with filters

Microservices with Spring Cloud

The Role of Routing in Microservices



Health info

Load statistics

Surgical routing

Meaningful routing

Caching

Rate limiting

Authenticating

Logging

API Proxy

Fine-grained services

Microservices with Spring Cloud

Traffic Routing of Microservices

Spring Cloud Ribbon

Client-side software Load Balancer

Microservices with Spring Cloud

Spring Cloud Ribbon

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

Microservices with Spring Cloud

Configuring Ribbon in Microservices

Ribbon listed under "Cloud Routing" on start.spring.io

[spring-cloud-starter-ribbon]

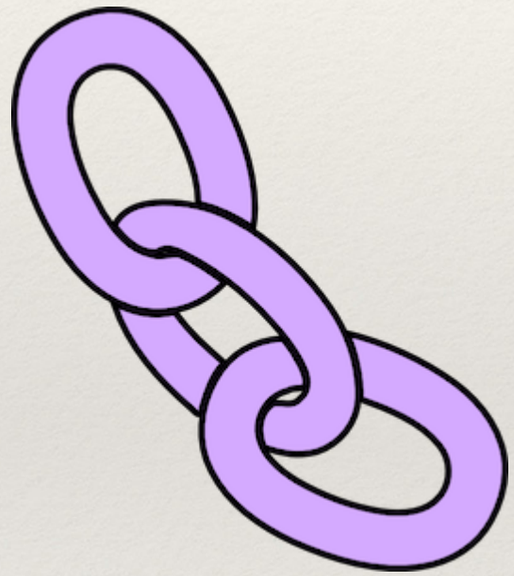
Provide list of servers in code, configuration or Eureka

Directly access client, or use @LoadBalanced RestTemplate

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

Microservices with Spring Cloud

How Ribbon & 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

Microservices with Spring Cloud

Use Case - Ribbon & Eureka

- Re-enable Eureka in client application
- Update Annotations
- Test client application

Microservices with Spring Cloud

Spring Cloud Zuul

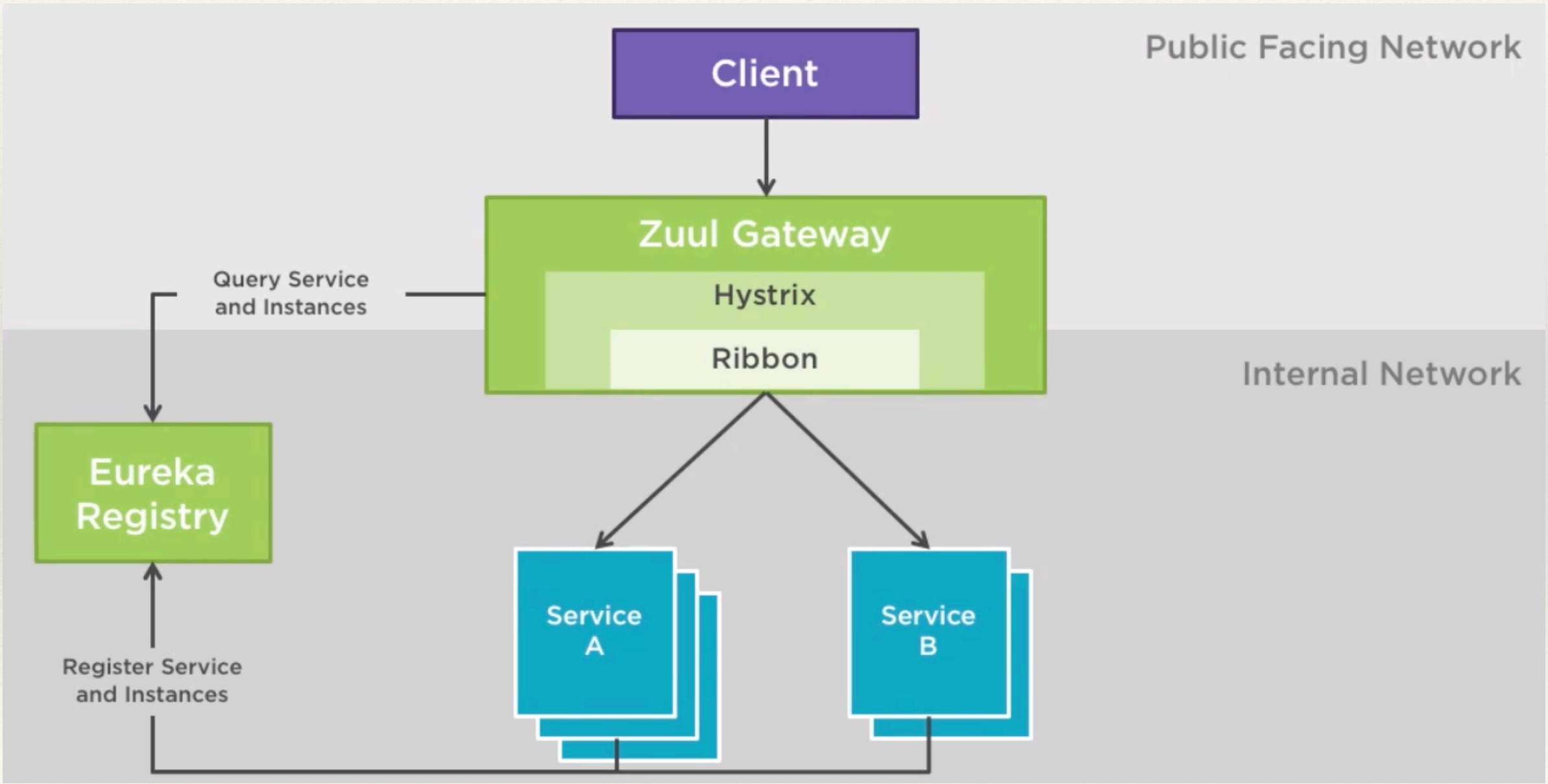
Embedded proxy for routing traffic in a

Microservices Architecture

Microservices with Spring Cloud

Spring Cloud Zuul

How Zuul Works



Microservices with Spring Cloud

Spring Cloud Zuul

Choosing a Spring Cloud Zuul Model

@ZuulProxy

Primed for reverse-proxy scenarios

Proxy filters automatically added

Can integrate with Eureka, Ribbon

Additional / routes endpoint

@EnableZuulServer

"Blank" Zuul Server

Passthrough requests by default

No service discovery

Add filters manually

Microservices with Spring Cloud

Spring Cloud Zuul

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

Microservices with Spring Cloud

Spring Cloud Zuul

Creating a Zuul Proxy with Configurable Routes

```
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

Microservices with Spring Cloud

Use Case - Spring Cloud Zuul

- Create new project from Spring Initializr
- Annotate class to turn into Zuul Proxy
- Setup with local URLs, No Eureka
- Add Eureka with no whitelisting
- Lock down allowable services and experiment with routes
- Introduce prefix handling

Microservices with Spring Cloud

Spring Cloud Zuul

About Zuul Filters & Stages