Protecting Systems with Circuit Breakers

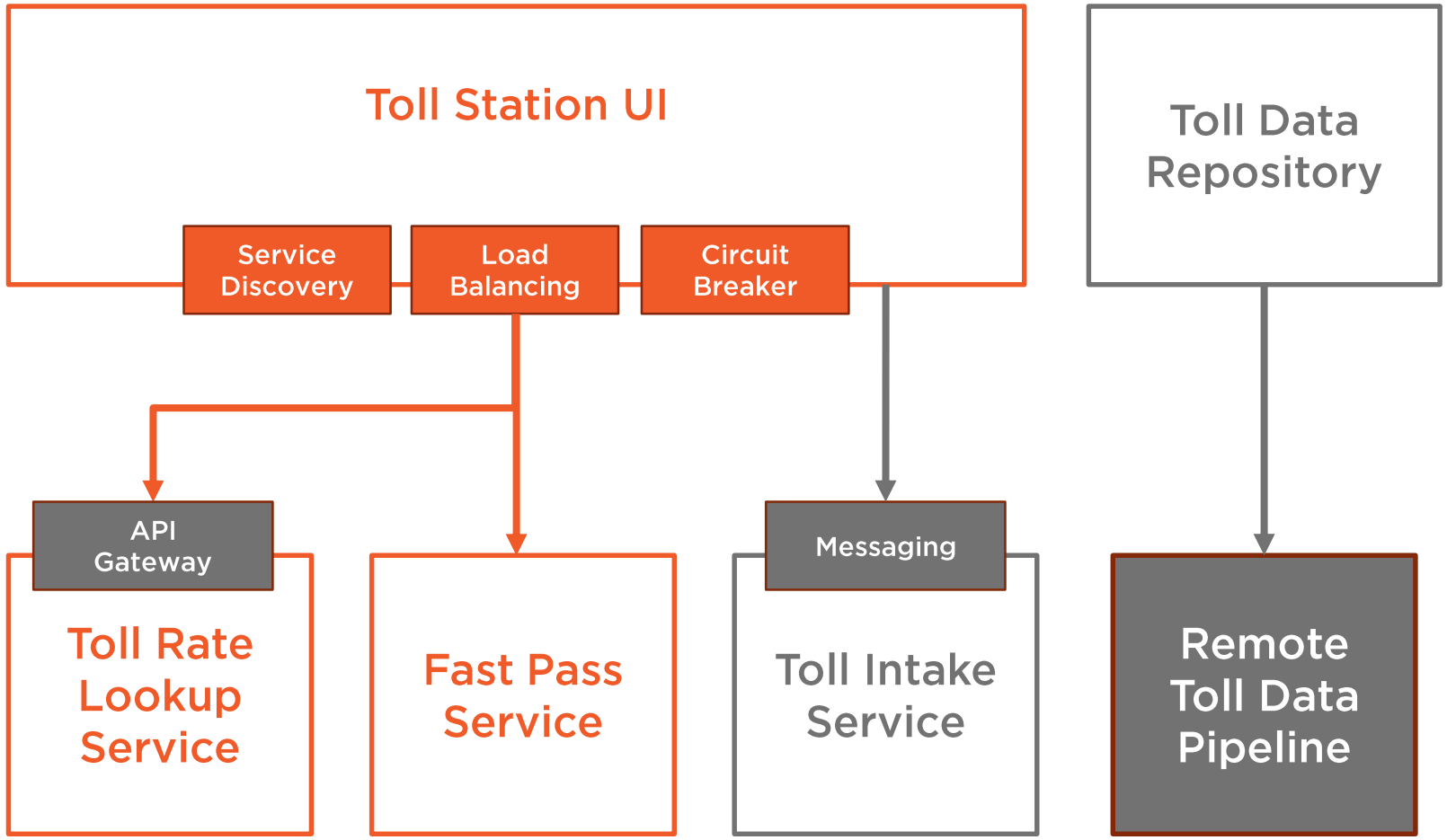
Richard Seroter
SENIOR DIRECTOR OF PRODUCT, PIVOTAL

@rseroter

Overview

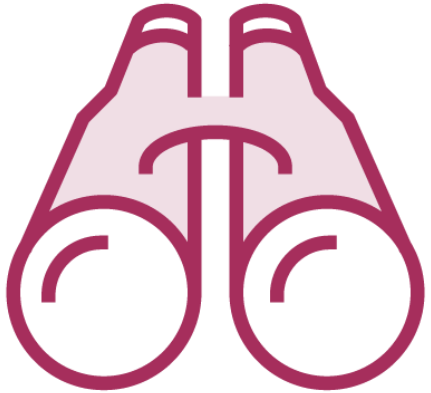
- Role of circuit breakers in microservices
- Problems with the status quo
- Describing Spring Cloud Hystrix
- Creating a Hystrix-protected service
- Using the Hystrix Dashboard
- What Turbine adds to Hystrix
- Summary

Capabilities That We Will Add in This Module

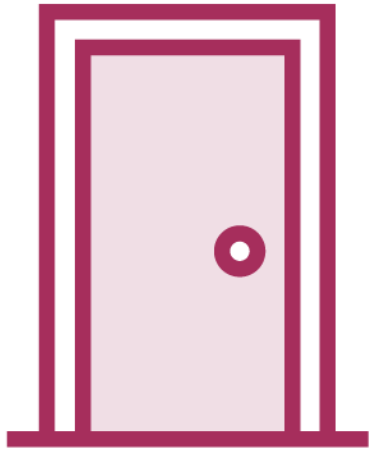


The Role of Circuit Breakers in Microservices

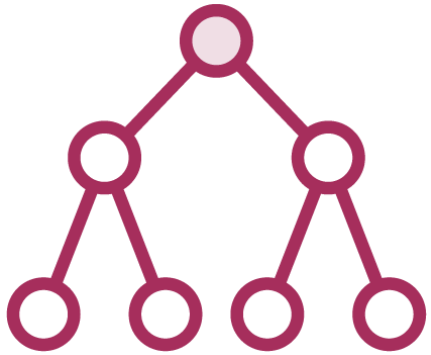
Circuit breakers protect an electrical circuit from damage



Watch for service faults in real-time



Circuit closes when successful request processed



Prevents cascading failures

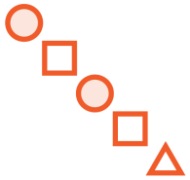
Problems with the Status Quo

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

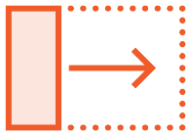
Spring Cloud Hystrix

Library for enabling
resilience in microservices.

What Hystrix Does



Supported patterns include bulkhead, fail fast, graceful degradation (e.g. 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.

How Spring Cloud Hystrix Works

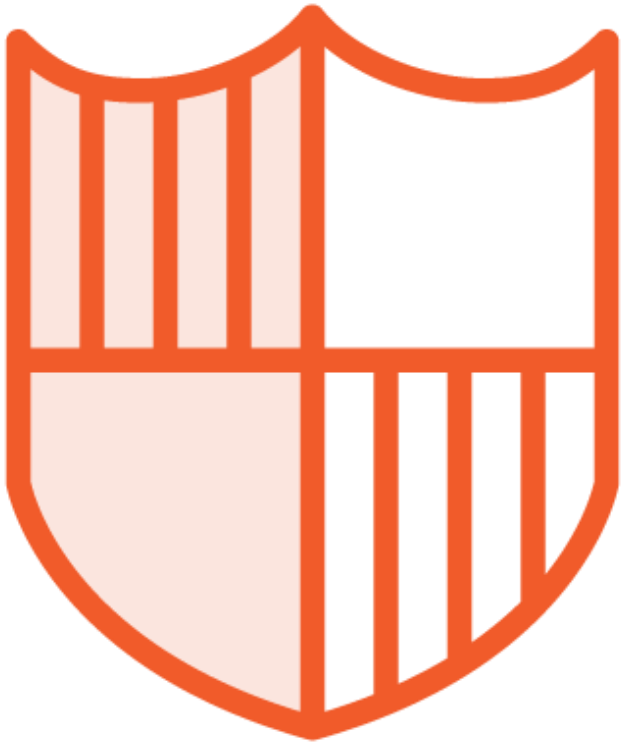
Circuit breaker via annotations at class, operation level

Hystrix manages the thread pool, emits metrics

Dashboard integrates with Eureka to look up services

Dashboard pulls metrics from instances or services

Creating a Hystrix-protected Service



Add **spring-cloud-starter-hystrix** dependency to calling service

Annotate class with @EnableCircuitBreaker annotation

Set up @HystrixCommand and define fallback method

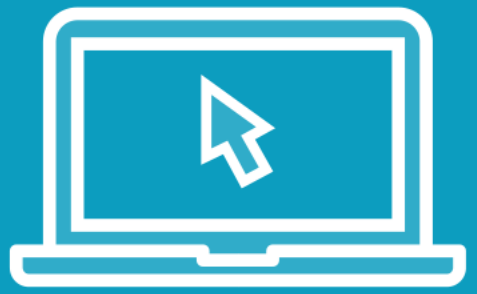
```
Circuit status - http://[host]:[port]/health

Metrics stream - http://[host]:[port]/hystrix.stream
```

Hystrix Stream and Endpoints

State of circuit comes from /health endpoint of calling application

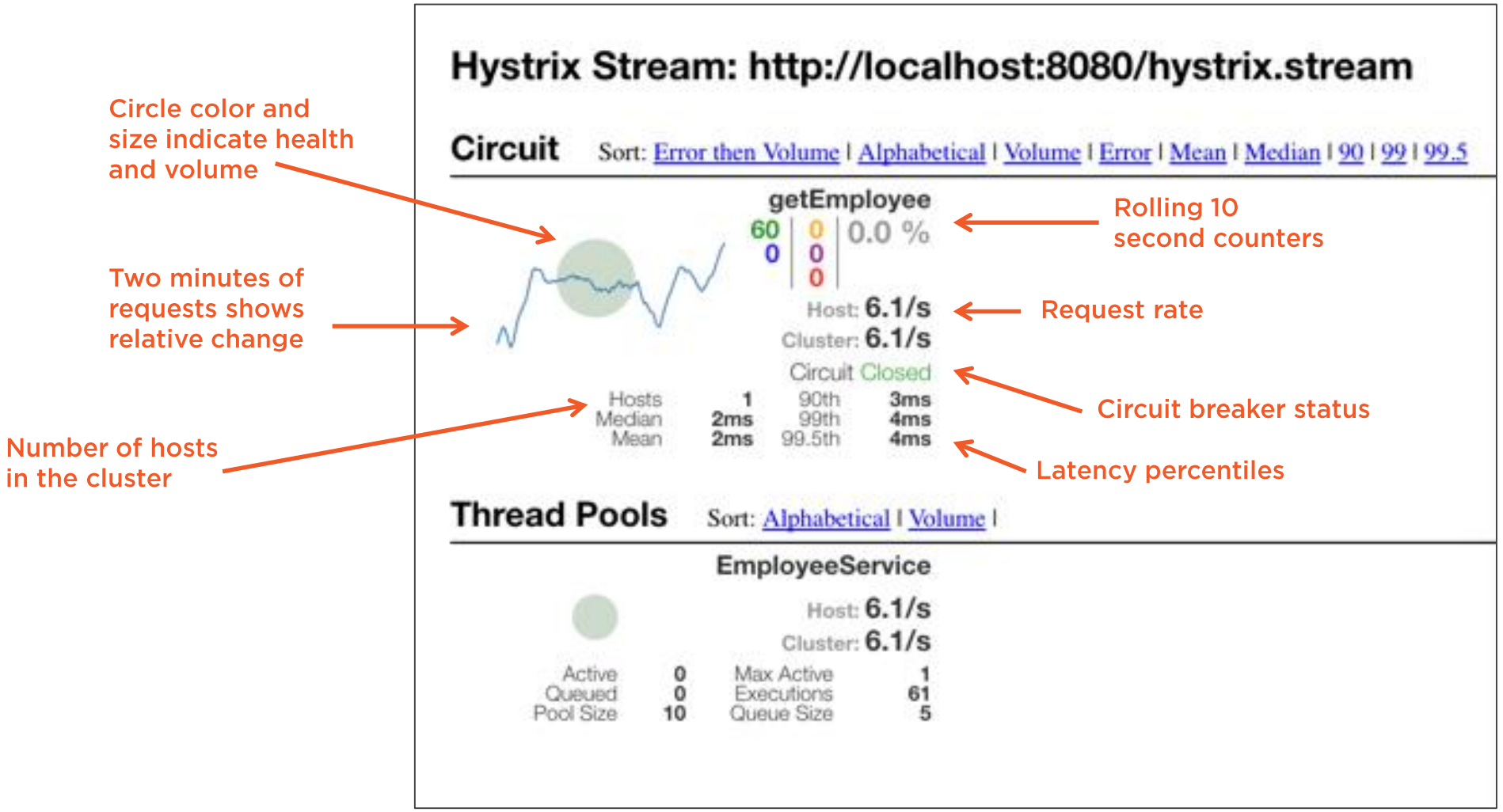
Hystrix metrics stream comes from actuator dependency

Demo



- Review existing Eureka-enabled “toll rate” and “fast pass” microservices
- Open existing client applications
- Add Hystrix dependency to projects
- Introduce Circuit Breaker to code
- Start up and call client applications
- Review circuit and metrics endpoints

What's Visible on the Hystrix Dashboard



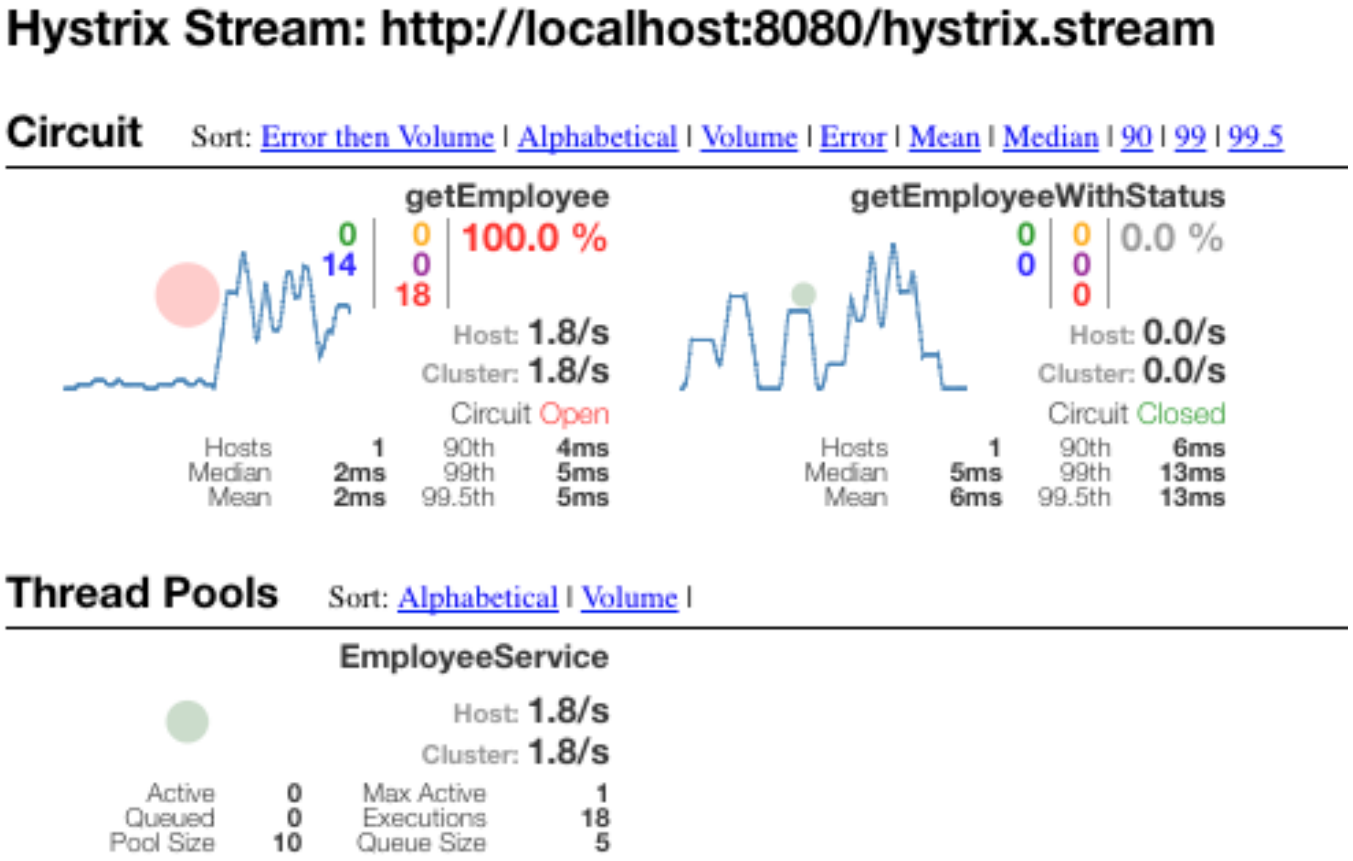
What does failure look like?

Can see failure in single circuit

See fine-grained view of what's happening

Important to not overreact

If all circuits show bad, probably a system problem, not a wholesale collapse!



Demo

Create new project from Spring Initializr

Choose Hystrix Dashboard dependency

Annotate main class

Start up Dashboard project

Load streams for toll rate billboard and fast pass console

Advanced Hystrix Configuration

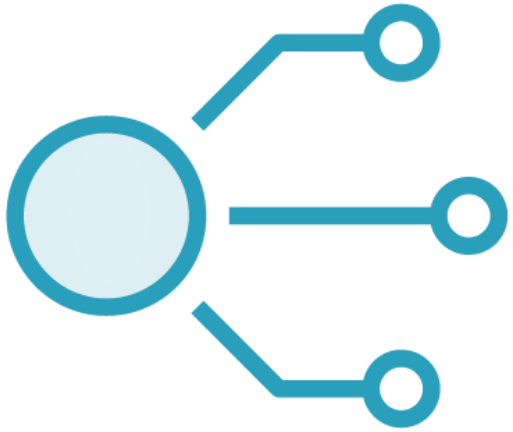
@HystrixProperty settings

Set command properties

Set thread pool properties

Use annotations or property files

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

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

Demo

- Update Hystrix Dashboard with Turbine dependency
- Set application properties
- Start up Dashboard and use Turbine endpoint
- Create new project from Spring Initializr
- Add Turbine Stream and RabbitMQ dependencies
- Add Hystrix Stream and RabbitMQ dependency to client application
- Start all projects and hit Turbine Stream endpoint from Dashboard

Summary

- Overview
- Role of circuit breakers in microservices
- Problems with the status quo
- Describing Spring Cloud Hystrix
- Creating a Hystrix-protected service
- Using the Hystrix Dashboard
- What Turbine adds to Hystrix