# Locating Services at Runtime Using Service Discovery

**Richard Seroter**
SENIOR DIRECTOR OF PRODUCT, PIVOTAL

@rseroter

## Overview

Role of service discovery in microservices

Problems with the status quo

Describing Spring Cloud Eureka

Creating a Eureka Server

Registering services with Eureka

Discovering services with Eureka

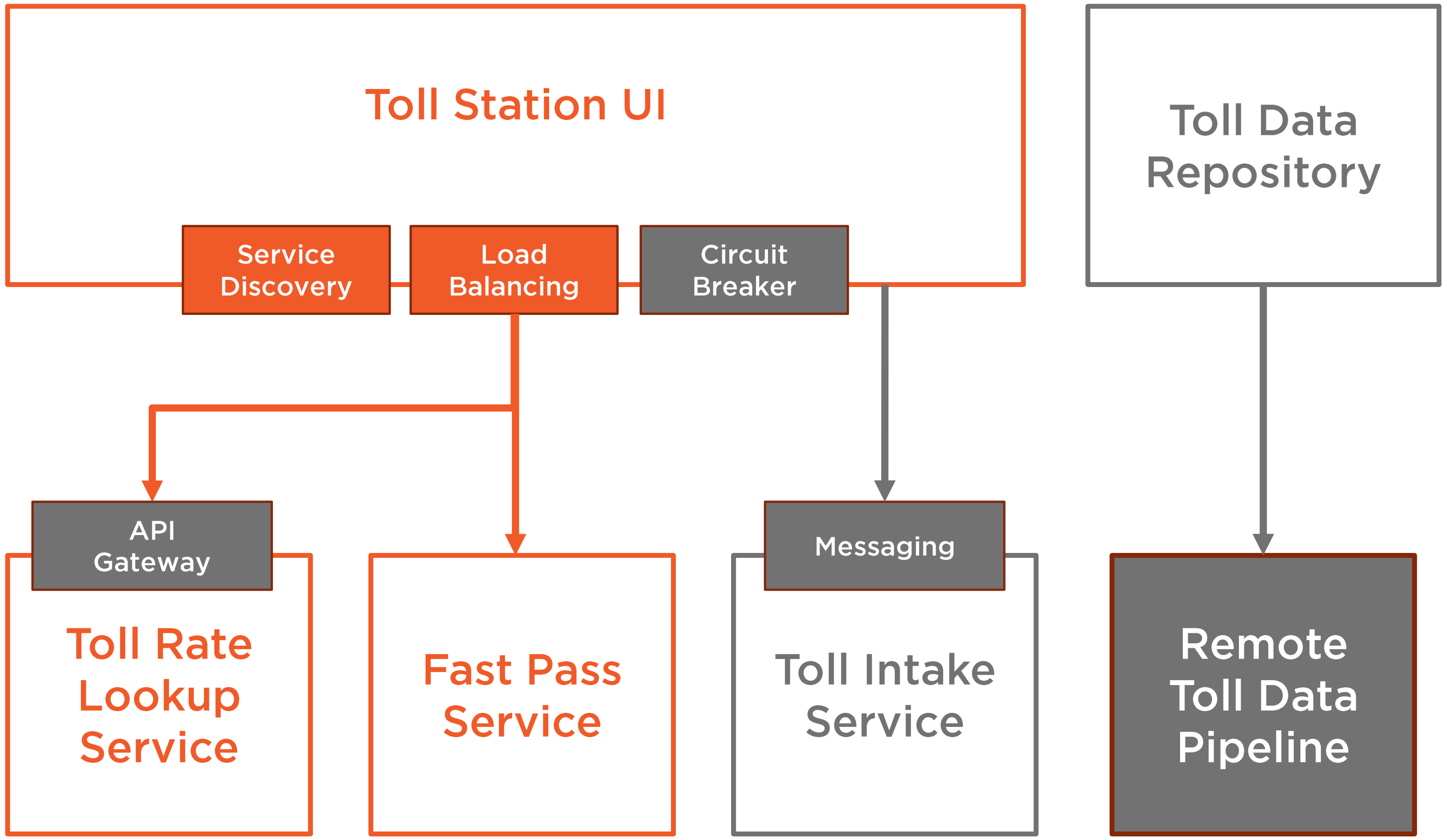
Configuring health information

Reviewing the high availability setup

Options for advanced configuration

Summary

# Capabilities That We Will Add in This Module



# The Role of Service Discovery in Microservices

Recognize the dynamic environment

Have a live view of healthy services

Avoid hard-coded references to service location

Centralized list of available services

## 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Outdated configuration management DBs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

## Spring Cloud Eureka

Registry that acts as a phone-book for services.

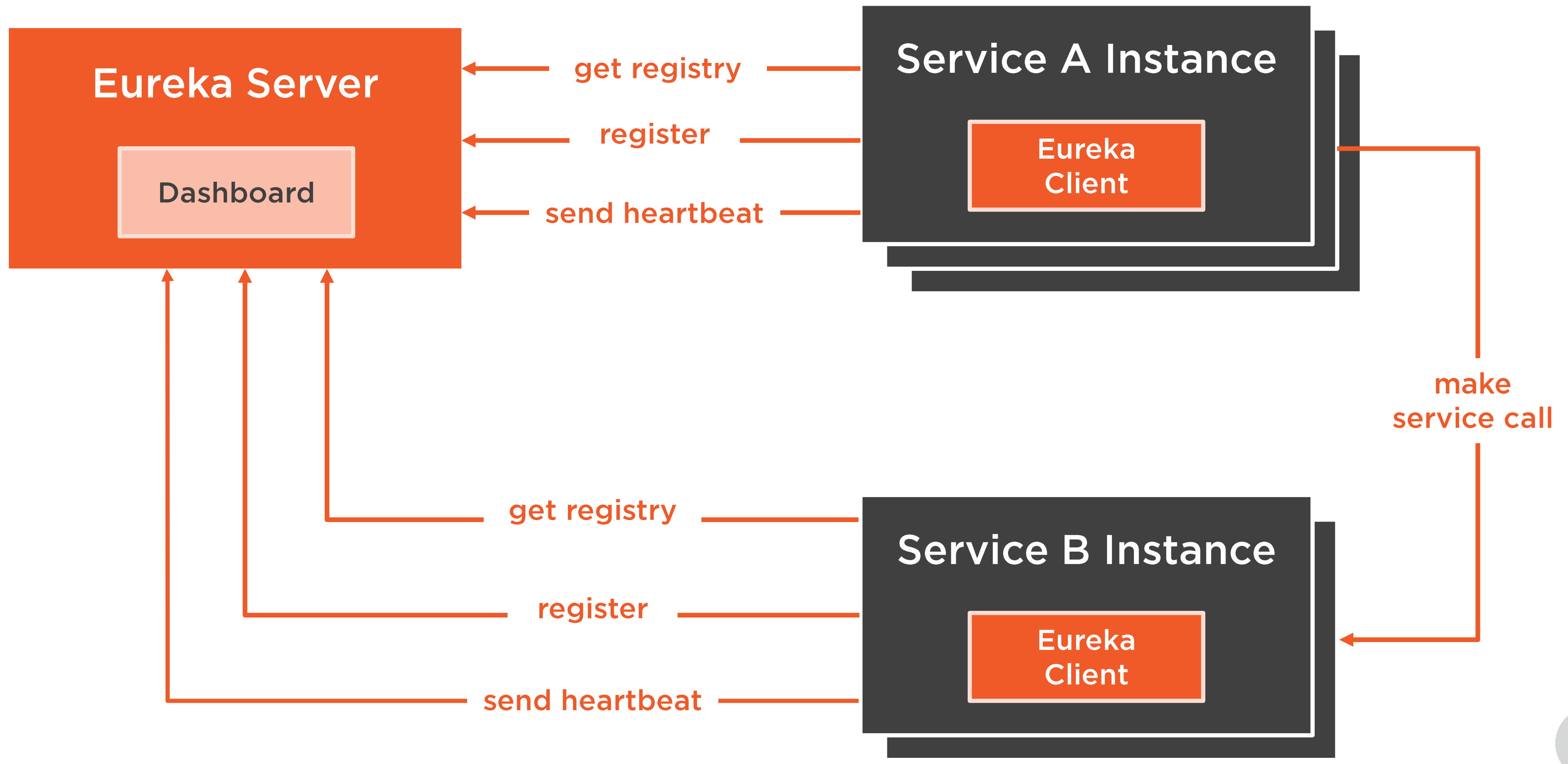
## The History of Eureka

First released by Netflix OSS team in 2012

Used for middle-tier load balancing

Integrated into many other Netflix projects

# Components of a Eureka Environment



# Creating a Eureka Server

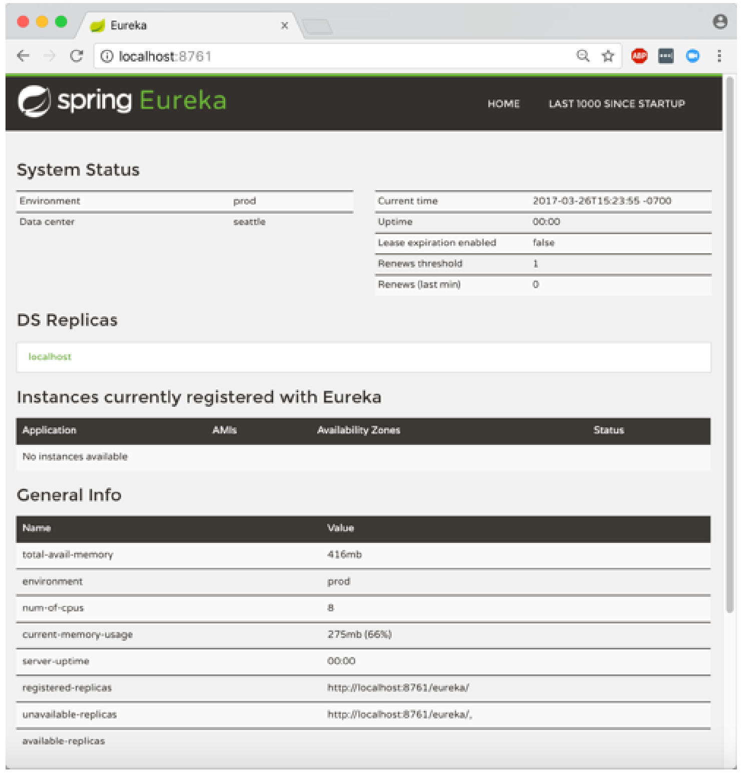
**Add spring-cloud-starter-eureka-server**

**Standalone or clustered configuration**

**@EnableEureka Server annotation**

**Numerous configuration options**

# Using the Eureka Dashboard



System Status

| | |
|---|---|
| Environment | prod |
| Data center | seattle |

| | |
|---|---|
| Current time | 2017-03-26T15:23:55 -0700 |
| Uptime | 00:00 |
| Lease expiration enabled | false |
| Renews threshold | 1 |
| Renews (last min) | 0 |

DS Replicas

localhost

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| No instances available | | | |

General Info

| Name | Value |
|---|---|
| total-avail-memory | 416mb |
| environment | prod |
| num-of-cpus | 8 |
| current-memory-usage | 275mb (66%) |
| server-uptime | 00:00 |
| registered-replicas | http://localhost:8761/eureka/ |
| unavailable-replicas | http://localhost:8761/eureka/, |
| available-replicas | |

Enabled by default

Shows environment info

Lists registered services and instances

View service health

## Demo

Start a new project via Spring Initializr

Add Eureka Server dependency

Annotate primary class

Set application properties

Start server and view dashboard

## Registering a Service 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

```
@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.

## Demo

Open existing “toll rate” microservice

Add project dependency on Eureka

Annotate primary class

Add bootstrap and application properties

Start up microservice and see in registry

Start a second instance and see in registry

Do same sequence with “fast pass customer” microservices

# Discovering a Service 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

## Demo

Open “toll rate billboard” application

Add dependency on Eureka

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

## Configuring Service Health Information

Heartbeat doesn't convey health

Possible to include health information

Can extend and create own health check

## Demo

Return to “toll rate” microservice and add a custom health check

Start up microservice and wait for error

See service taken out of rotation by Eureka

## High Availability Architecture for Eureka

Built-in “self preservation” model

Native support for peer to peer registry replication

Use DNS in front of Eureka cluster

Recommended to have one Eureka cluster in each Zone

## Advanced Configuration Options

Dozens and dozens of configuration flags

Set cache refresh intervals

Set timeouts

Set connection limits

Set service metadataMap

Override default service, health endpoints

Define replication limits, timeout, retries

## Summary

Overview

Role of service discovery in microservices

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