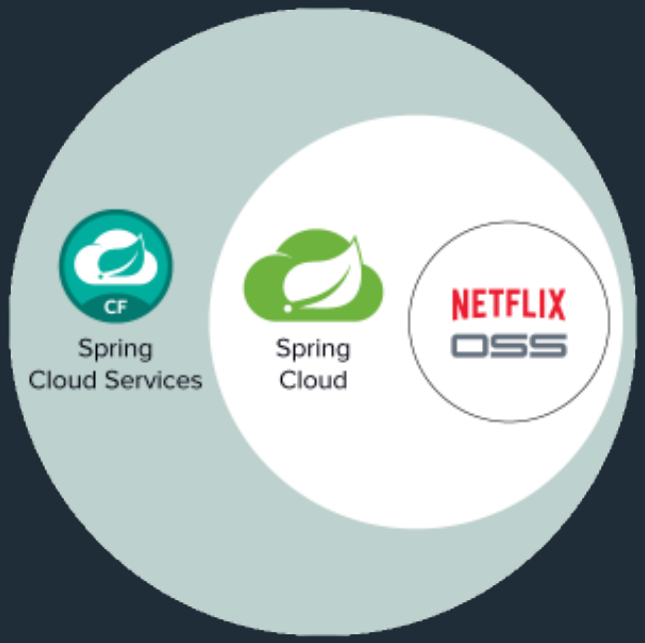
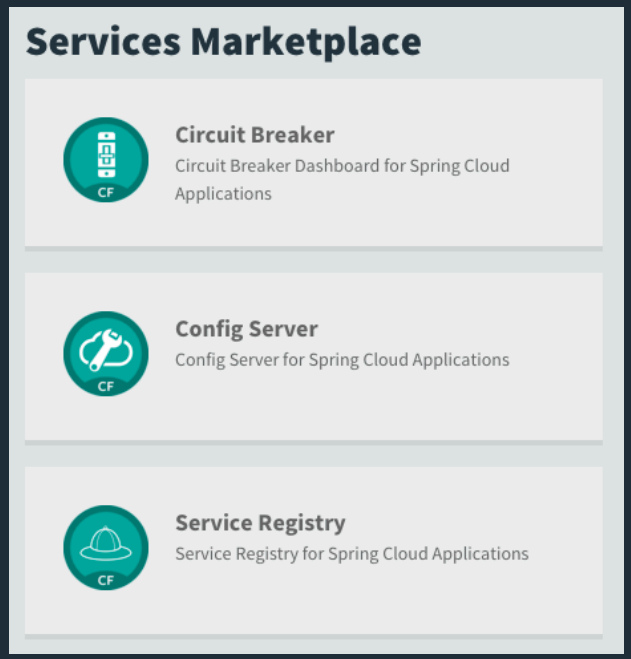
Pivotal

Spring Cloud

Reshmi Krishna

Spring Cloud Services





Pivotal™

Spring Cloud Services

Spring Cloud Services

Service Registry

- Service Registration and Discovery via Netflix OSS Eureka
- Registration via CF Route

Config Server

- Git URL for Config Repo provided via Service Dashboard (post-provisioning)
- Single tenant, scoped to CF space

Circuit Breaker

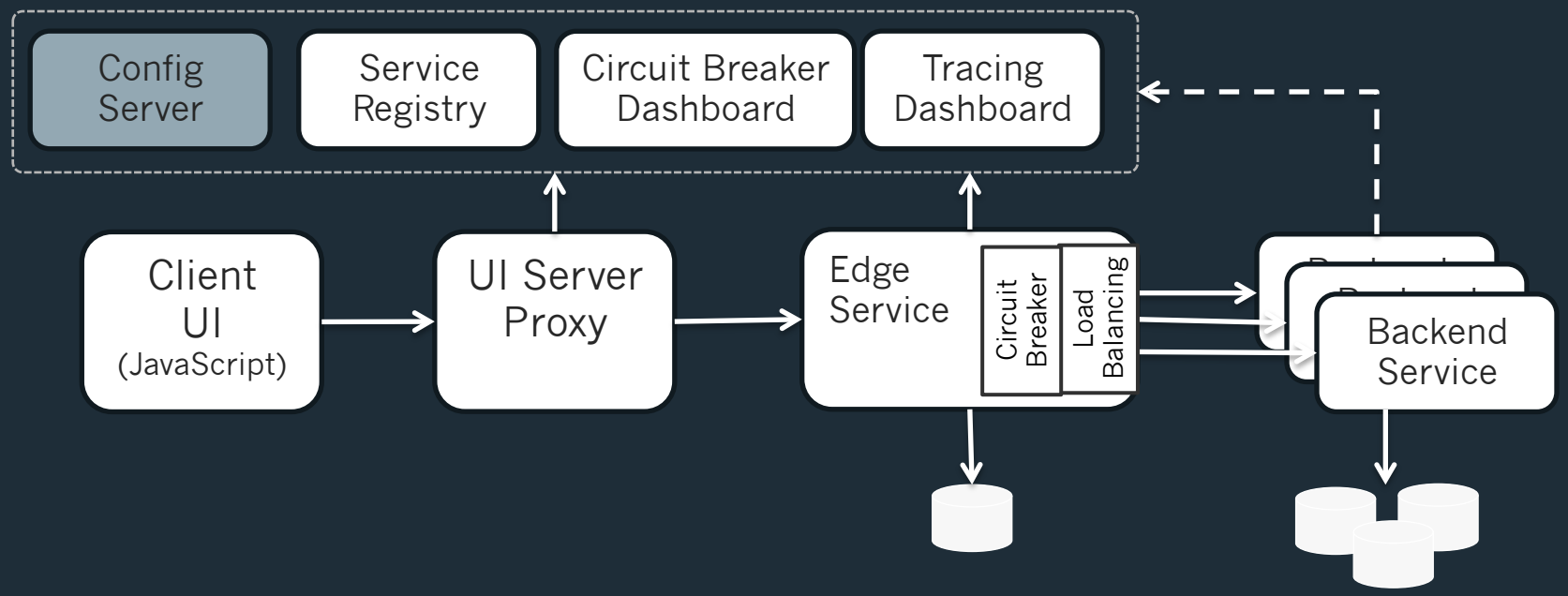
- Netflix OSS Turbine + Hystrix Dashboard
- Aggregation via AMQP (RabbitMQ)

Pivotal™

Pivotal

Spring Cloud Config Server

Configuration Management



Pivotal

Config in a Spring Context

Spring has provided several approaches to setting config

Still, gaps exist:

- Changes to config require restarts
- No audit trail
- Config is de-centralized
- No support for sensitive information

Pivotal™

Config in a 12 Factor Context

- 12 Factor application design states that configuration should be kept in OS environment variables.

In Pivotal Cloud Foundry, this is accomplished in the following ways:

- Using the cf set-env command
- Using a manifest with env: sections

Pivotal™

Challenges When Using Env. Variables

- Managing many env variables can be a challenge
- Pivotal Cloud Foundry uses immutable containers
- Any configuration change requires restarting the app
- If you want zero downtime then do blue/green deployments

Pivotal™

More Demanding Config Use Cases

- Debugging a production issue
- Report all configuration changes made to a production system to support regulatory audits
- Toggle features on/off in a running application

Pivotal™

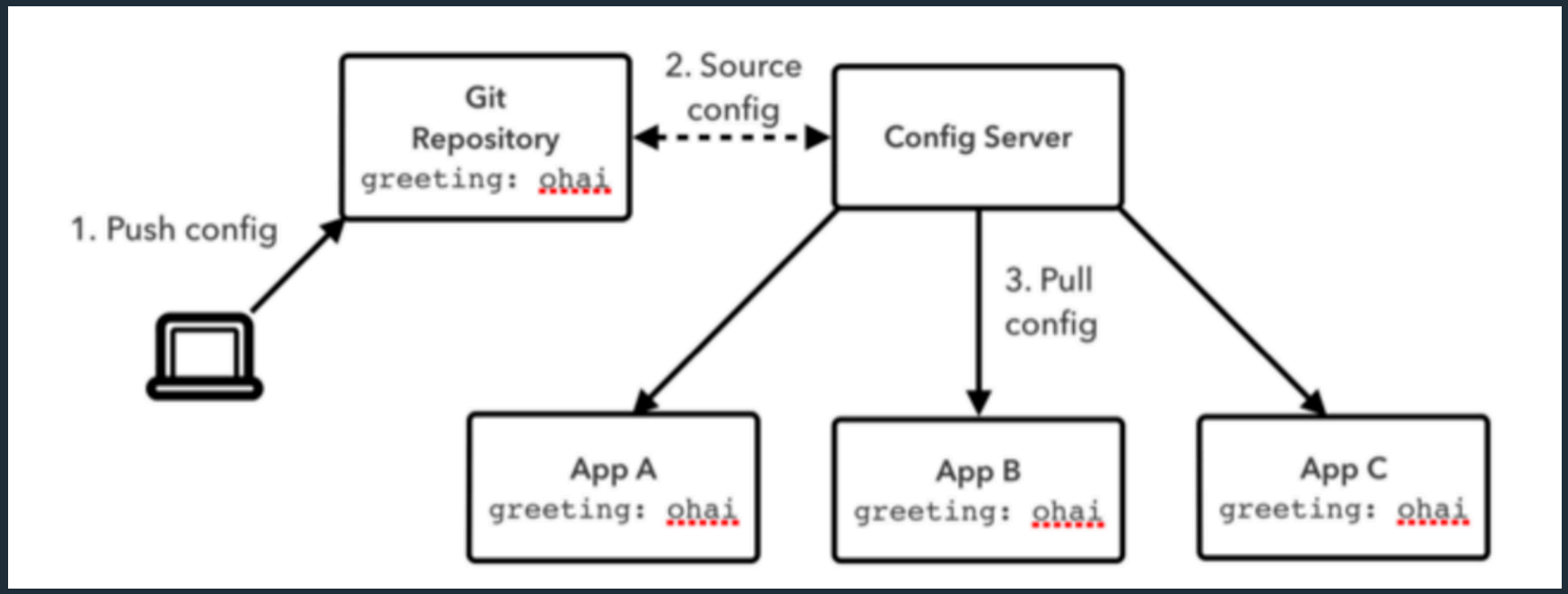
Externalized Configuration

Configuration Mgmt approach should support:

- Versioning
- Auditability
- Encryption
- Refresh without restart

Pivotal™

Configuration Flow



Pivotal

Spring Cloud Config Server

- The Server provides an HTTP, resource-based API for external configuration
- Bind to the Config Server and initialize Spring Environment
- Embeddable using @EnableConfigServer
- Include spring-cloud-config-server dependency

Pivotal™

Embedded Server

The server is easily embeddable in a Spring Boot application using the @EnableConfigServer annotation.

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServer {
  public static void main(String[] args) {
    SpringApplication.run(ConfigServer.class, args);
  }
}
```

Pivotal

Server Config

- Application configuration data is stored in a backend

```yaml
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/spring-cloud-samples/config-repo.git
```

Pivotal

Server Endpoints

Config server exposes config on the following endpoints:

```
/{application}/{profile}/[{label}]
/{application}-{profile}.yml
/{label}/{application}-{profile}.yml
/{application}-{profile}.properties
/{label}/{application}-{profile}.properties
```

Pivotal™

Configuration Files

spring.application.name=foo
spring.active.profiles=dev
Repo Files :
foo-dev.yml - app and profile specific
foo.yml - app specific
application-dev.yml – shared and profile specific
application.yml - shared between all clients

Pivotal

Client Application

```
@SpringBootApplication
@RestController
public class ClientConfigApplication {

    public static void main(String[] args) {
        SpringApplication.run(ClientConfigApplication.class, args);
    }

    @Value("${greeting}") //<-- configuration injected from environment
    private String greeting;

    @RequestMapping("/greeting")
    String greeting() {
        return String.format("%s World", greeting);
    }

}
```

Pivotal™

Include Dependency

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Pivotal™

Client Config

Sample bootstrap.yml

```yaml
spring:
  cloud:
    config:
      uri: http://my-config-server.io/
```

spring.cloud.config.uri defaults to http://localhost:8888

Pivotal™

Refreshable Application Components

What can be refreshed in a Spring Application?

- Loggers logging.level.*
- @ConfigurationProperties beans
- Beans with @RefreshScope annotation

Pivotal™

@RefreshScope

```
@SpringBootApplication
@RestController
@RefreshScope // <-- Add RefreshScope annotation
public class ClientApplication {

    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class, args);
    }

    @Value("${greeting}")
    private String greeting;

    @RequestMapping("/greeting")
    String greeting() {
        return String.format("%s World", greeting);
    }

}
```

Pivotal

Refreshing Configuration

Two step process

1) Update Repository

2) Send a request to the application(s)

- Send a POST request to the refresh endpoint in the client app to fetch updated config values:
- http://127.0.0.1:8080/refresh
- Requires the Actuator dependency on the classpath (pom.xml).

Pivotal™

Spring Cloud Bus

Leverage Spring Cloud Bus pub/sub notification with RabbitMQ

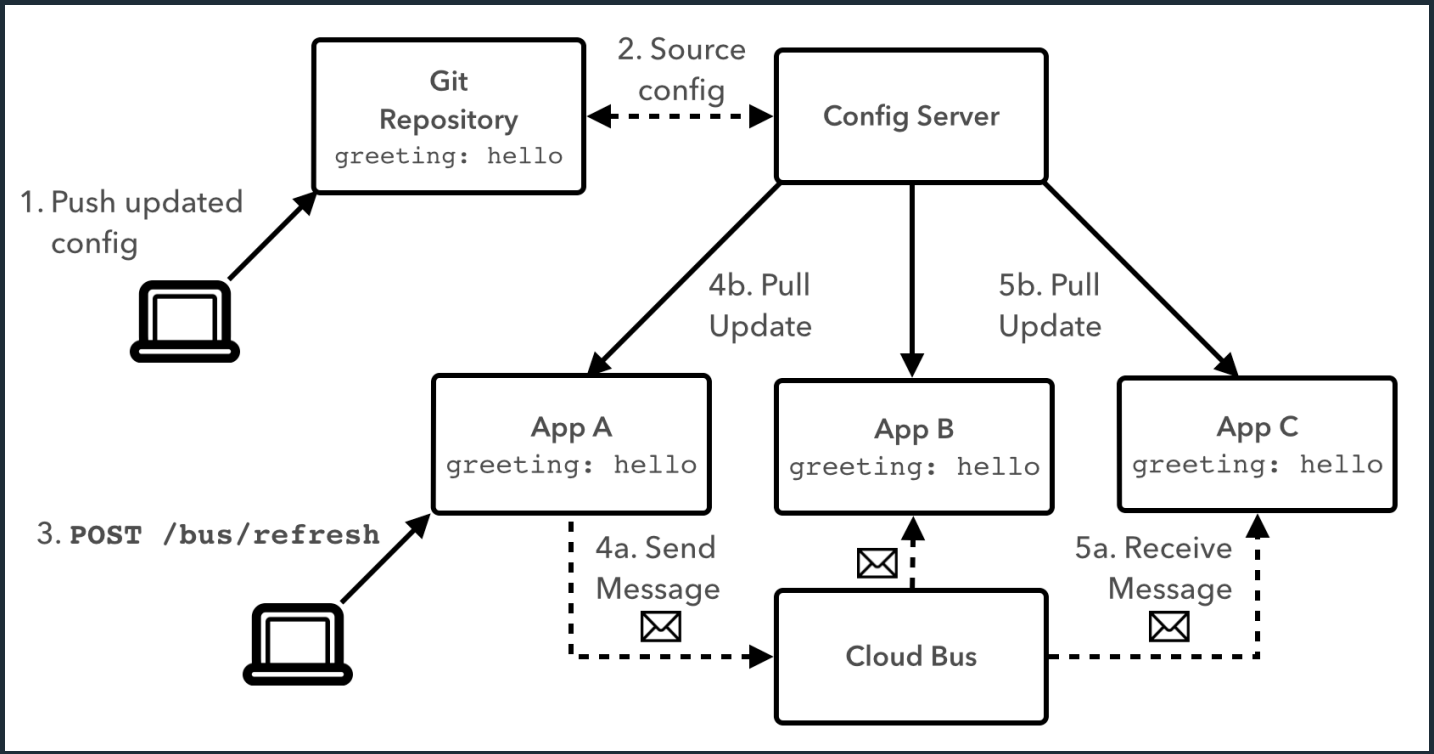
Send a POST request to the refresh endpoint to fetch updated config values:

http://127.0.0.1:8080/bus/refresh

Requires the Cloud Bus AMQP dependency on the classpath (pom.xml)

Pivotal™

Cloud Bus Diagram



Pivotal

Spring Cloud Services

- Brings Spring Cloud to Pivotal Cloud Foundry
- Includes: Config Server, Service Registry & Circuit Breaker Dashboard services

Pivotal™

Spring Cloud Services: Config Server

1) Include dependency:

```
<groupId>io.pivotal.spring.cloud</groupId>
<artifactId>spring-cloud-services-starter-config-client</artifactId>
```

2) Create a Config Server service instance

3) Configure the service instance with a Git Repository

4) Bind the service to the app

Pivotal™

Spring Cloud Services: Config Server

Config Server

Instance ID: b2620e4e-6339-4dfd-a7a8-76656d576616

Configuration Source

- (•) Git
- ( ) Subversion

Git URI

https://github.com/myorg/configurations

Git Branch (default is 'master')

Pivotal

Cloud Bus in Pivotal Cloud Foundry

Include dependency:

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
```

2) Create a RabbitMQ service instance

3) Bind the service to the app

Pivotal

Pivotal

Spring Cloud: Service Discovery

Challenges

- Service Discovery is one of the key tenets of a microservice based architecture.
- In distributed systems, application dependencies cease to be a method call away
- Trying to hand configure each client or use some form of convention can be very difficult to do and can be very brittle

Pivotal™

Where have we been?

How have we discovered services in the past?

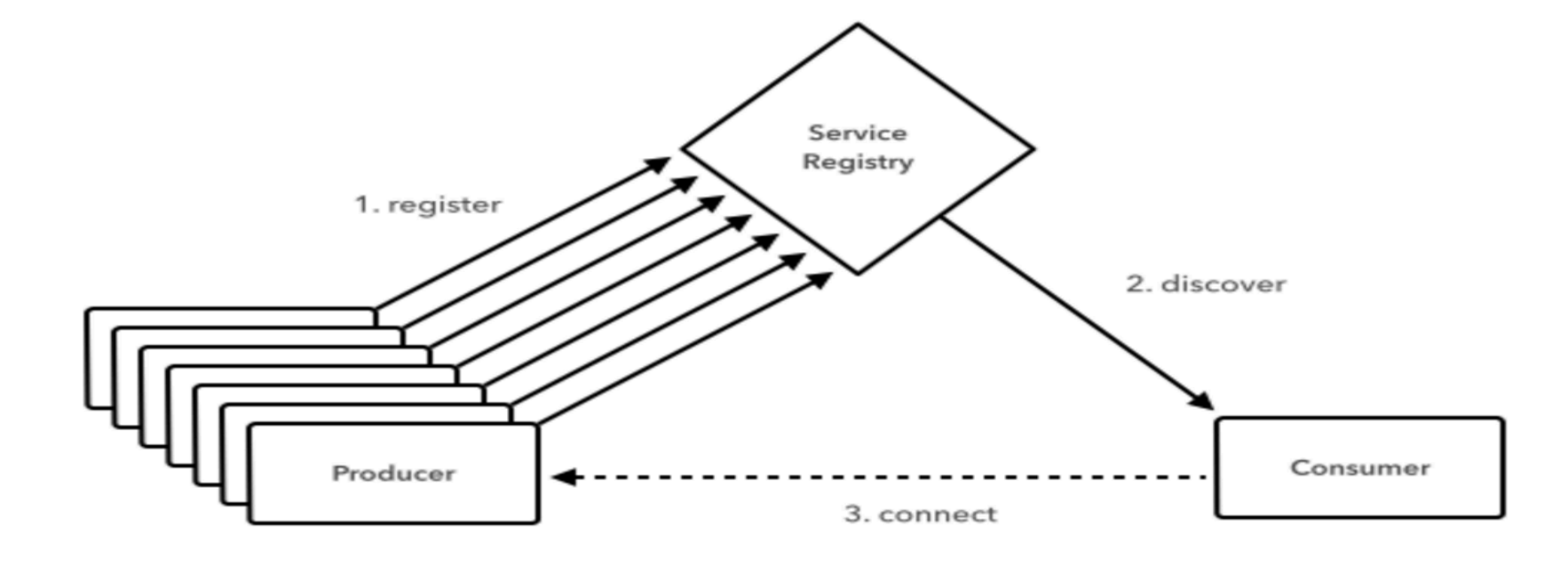
Service Locators

Dependency Injection

Service Registries

Pivotal™

Service Discovery with Spring Cloud



Pivotal

Include Dependency

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

Pivotal™

Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }
}
```

Pivotal™

Eureka Client : Include Dependency

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Pivotal™

Client Application

```
@SpringBootApplication
@EnableDiscoveryClient
public class GreetingServiceApplication {
public static void main(String[] args) {
 SpringApplication.run(GreetingServiceApplication.class, args);
    }
}
```

Pivotal™

Configuration required to locate the Eureka.

application.yml

```yml
spring:
  application:
    name: fortune-service
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Pivotal

Registering With Eureka

- When a client registers with Eureka, it provides meta-data about itself
- Eureka receives heartbeat messages from each instance belonging to a service.
- If the heartbeat fails over a configurable timetable, the instance is normally removed from the registry.

Pivotal™

Discovery Client

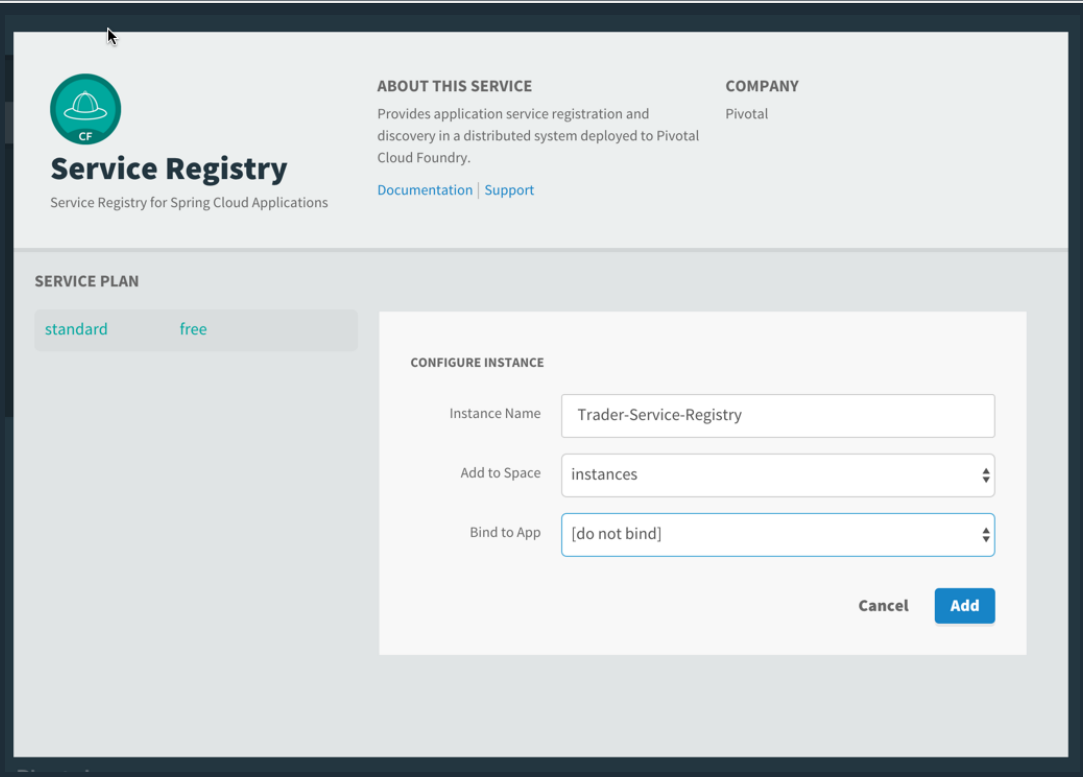
```
@Autowired
private DiscoveryClient discoveryClient;

public String serviceUrl() {
    InstanceInfo instance =
      discoveryClient.getNextServerFromEureka("STORES", false);
    return instance.getHomePageUrl();
}
```

Pivotal™

Spring Cloud Services: Service Registry

- Automated deployment of server component
- Security-optimized Eureka service instance.
- Bind into CF client application(s)



Pivotal

Spring Cloud Services: Service Registry

1) Add dependency

```
<dependency>
  <groupId>io.pivotal.spring.cloud</groupId>
  <artifactId>spring-cloud-services-starter-service-registry</artifactId>
</dependency>
```

2) Create a Service Registry service instance

3) Bind the service to the app

Pivotal™

Spring Cloud: Client-side Load Balancing

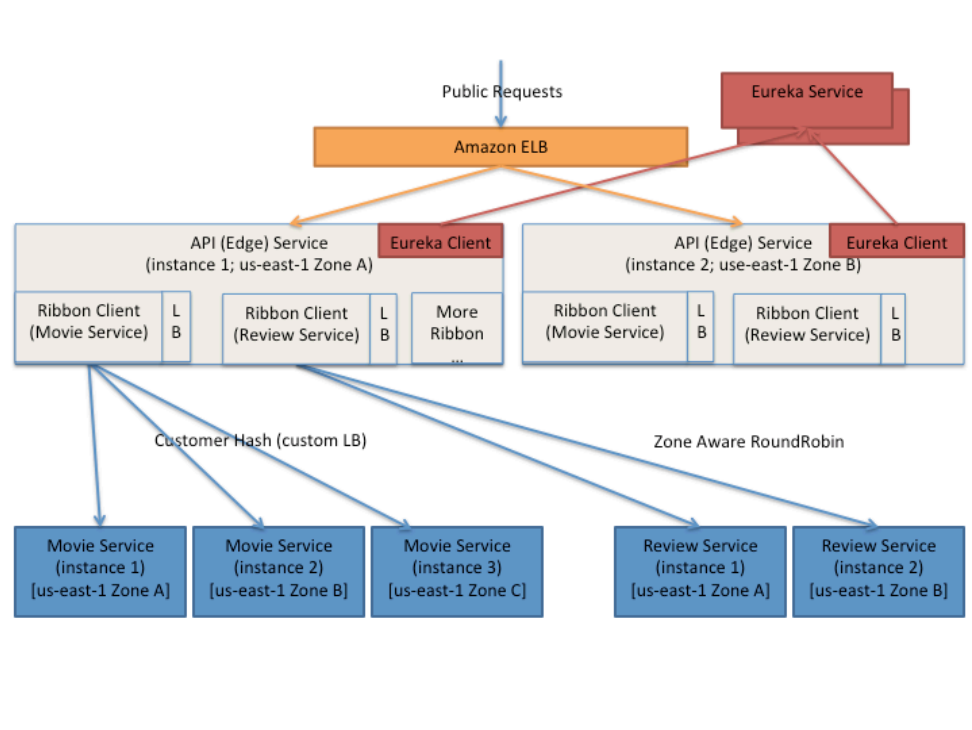
- Eureka *only* provides registry + discovery
- Ribbon is a *client side LB* providing control over the behavior of HTTP and TCP clients
  - Pick right LB algorithm for client application + extensible algorithms
  - At least 1 less hop for client requests
  - Cloud-aware patterns (zones, circuit breakers, etc.)
  - No additional setup, just deploy apps
- Zuul is JVM-based router and proxy commonly paired with Ribbon to create API gateways and reverse proxies

Pivotal™

Microservice API Gateways

Netflix uses Zuul and Ribbon for

- Authentication
- Stress Testing
- Canary Testing
- Dynamic Routing
- Service Migration
- Load Shedding
- Security
- Static Response handling
- Active/Active management



Pivotal

HOW??

```
@Autowired LoadBalancerClient loadBalancer;

public void doStuff() {
   ServiceInstance instance = loadBalancer.choose("stores");
   URI storesUri = URI.create(String.format("http://%s:%s",
                      instance.getHost(), instance.getPort()));
   // Do some stuff…
}
```

```
public Portfolio accountLookup(String[acctId) {
   Portfolio p = restTemplate.getForObject(
      "http://portfolio-service/portfolio/{accId}",
      Portfolio.class
      acctId);
   return p;
}
```

MAGIC!!

Pivotal

HOW??

```
@SpringBootApplication
@EnableZuulProxy
@EnableDiscoveryClient
public class MyAPIGateway {

   public static void main(String[] args) {
     SpringApplication.run(MyAPIGateway.class, args);
   }
}
```

MAGIC!!

application.yml

```
zuul:
   routes:
     users:
       path: /myusers/**
       serviceId: users_service
```

- API proxy will be created at /myusers
- Ribbon/Zuul creates load balancer for Eureka service “users_service”
- All requests are executed in a Hystrix command

Pivotal

Pivotal

Spring Cloud – Circuit Breakers and Fault Tolerance

Fault Tolerance

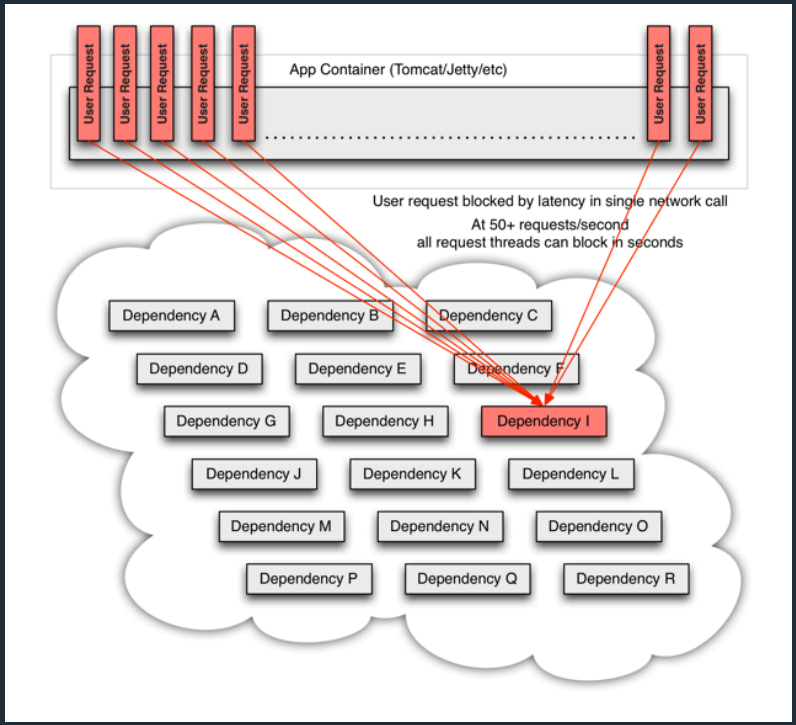
- One failure must not cause a cascading failure across the entire system.
- For example, for an application that depends on 30 services where each service has 99.99% uptime, here is what you can expect:

```
99.99^30 = 99.7% uptime
0.3% of 1 billion requests = 3,000,000 failures
2+ hours downtime/month if all dependencies have 99.99%
```

- Reality is *generally* worse.
- Source: https://github.com/Netflix/Hystrix/wiki

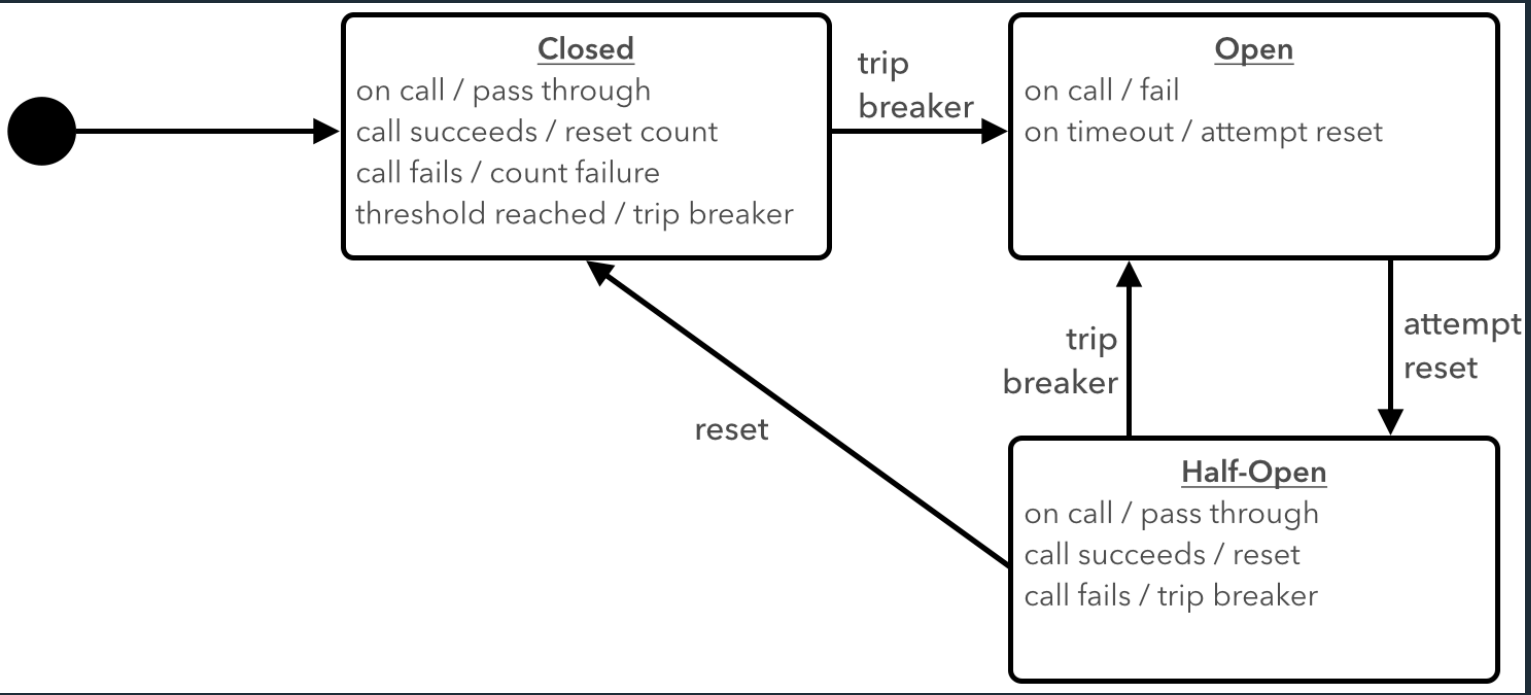
Pivotal

Distributed Systems Failures



Pivotal

Circuit Breaker Pattern



Pivotal

Implementing Circuit Breakers

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker   ← MAGIC!!
public class MyClientApp{
   public static void main(String[] args) {
     SpringApplication.run(MyClientApp.class, args);
   }
}
```

```
<dependency>
   <groupId>io.pivotal.spring.cloud</groupId>
   <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

Pivotal™

@HystrixCommand

```
@Service
public class FortuneService {

   @HystrixCommand(fallbackMethod = "defaultFortune",
      commandProperties = {
         @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds”, value="500")
      })
   public String getFortune() {
     return restTemplate.getForObject("http://fortune-service", String.class);
   }

   public String defaultFortune() {
     logger.debug("Default fortune used.");
     return "This fortune is no good. Try another.";
   }
 }
```

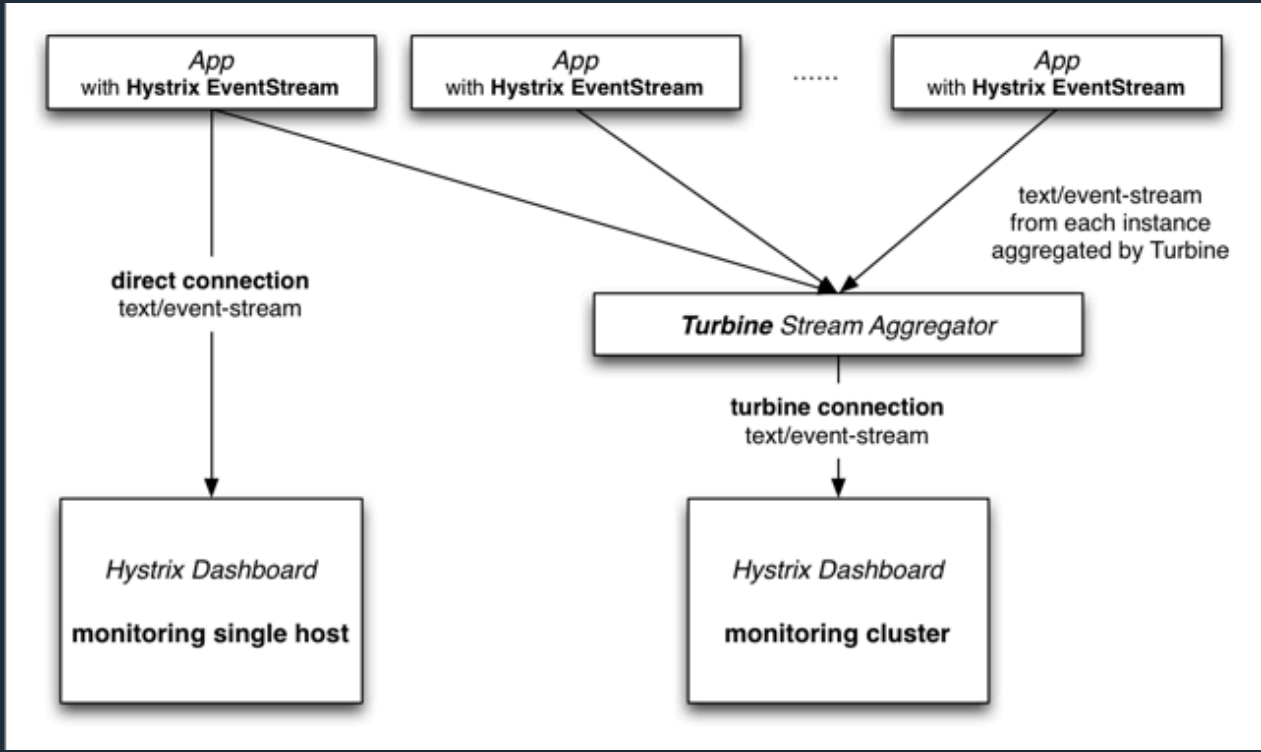
Pivotal

@HystrixCommand Metrics

- **Hystrix publishers real-time metrics for each `@HystrixCommand`**
  - Informational and Status (isCircuitOpen)
  - Cumulative and Rolling Event Counts (countExceptionsThrown & rollingCountExceptionsThrown)
  - Latency Percentiles (latencyExecute_percentile_995)
  - Latency Percentiles: End-to-End Execution (latencyTotal_percentile_5)
  - Property Values (propertyValue_circuitBreakerRequestVolumeThreshold)
- Published to /hystrix.stream endpoint & boot actuator metrics
- Individual /hystrix.streams aggregated via Turbine and published via /turbine.stream or AMQP.

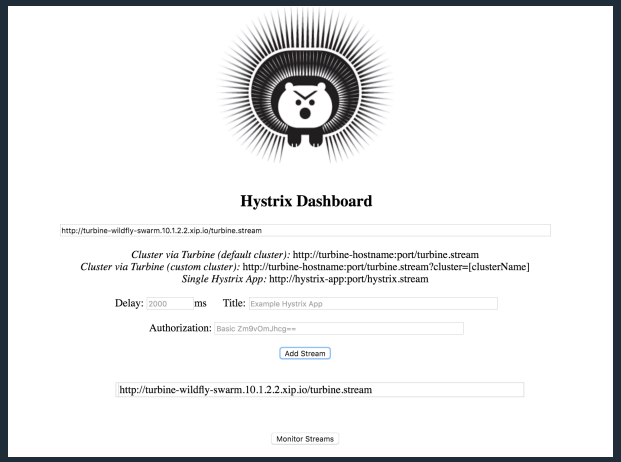
Pivotal™

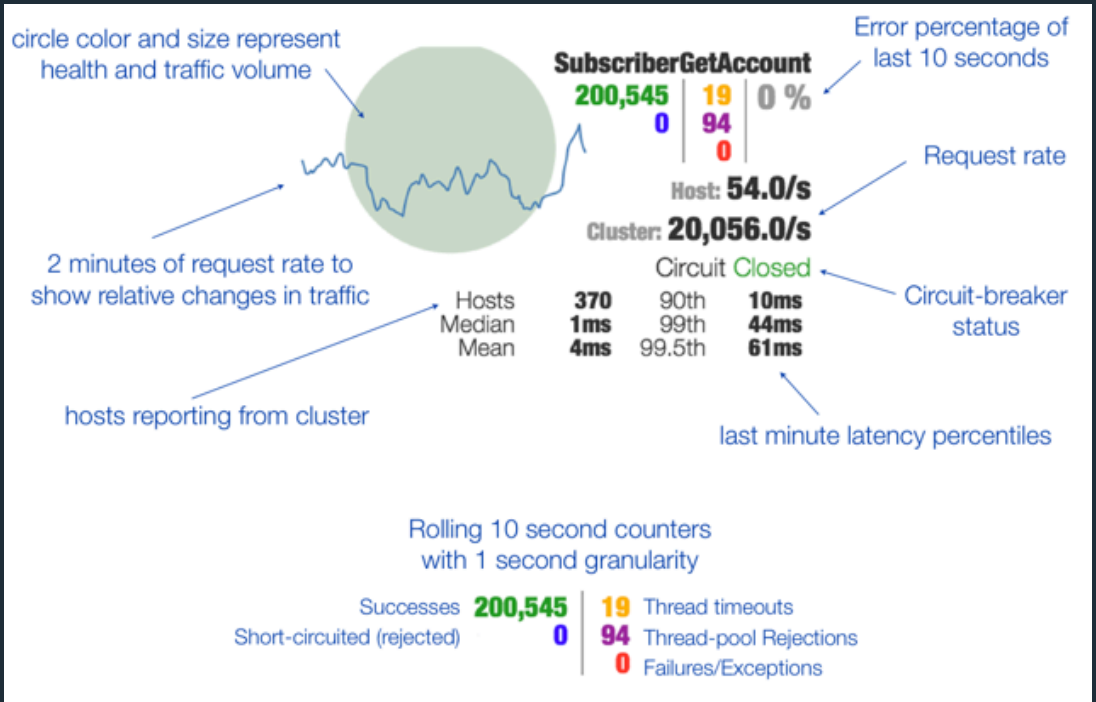
Hystrix Metrics With Turbine



Pivotal

Hystrix Dashboard



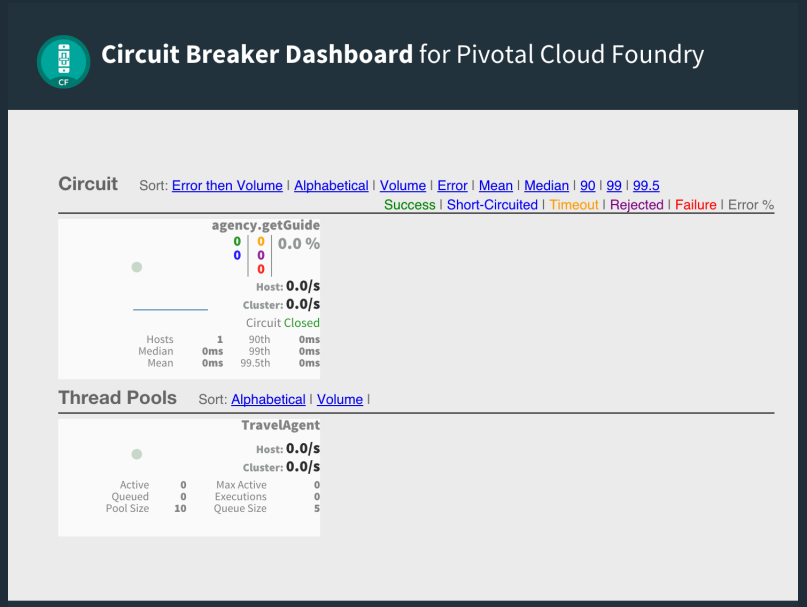


Pivotal

Spring Cloud Services: Hystrix Dashboard

- Automated deployment dashboard + Turbine + RabbitMQ
- Bind service into app
- Include starter dependency in app

```
<dependency>
   <groupId>io.pivotal.spring.cloud</groupId>
   <artifactId>spring-cloud-services-starter-circuit-
               breaker</artifactId>
</dependency>
```



Pivotal