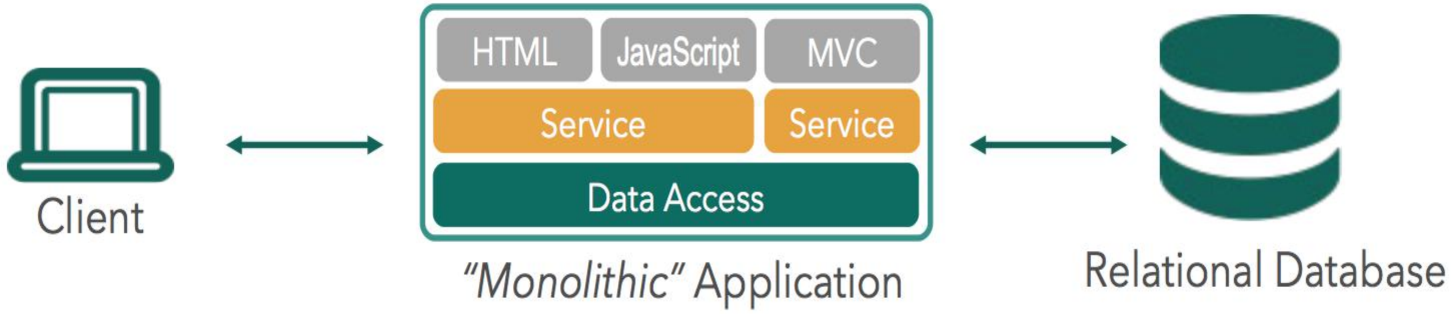
Spring Cloud Services

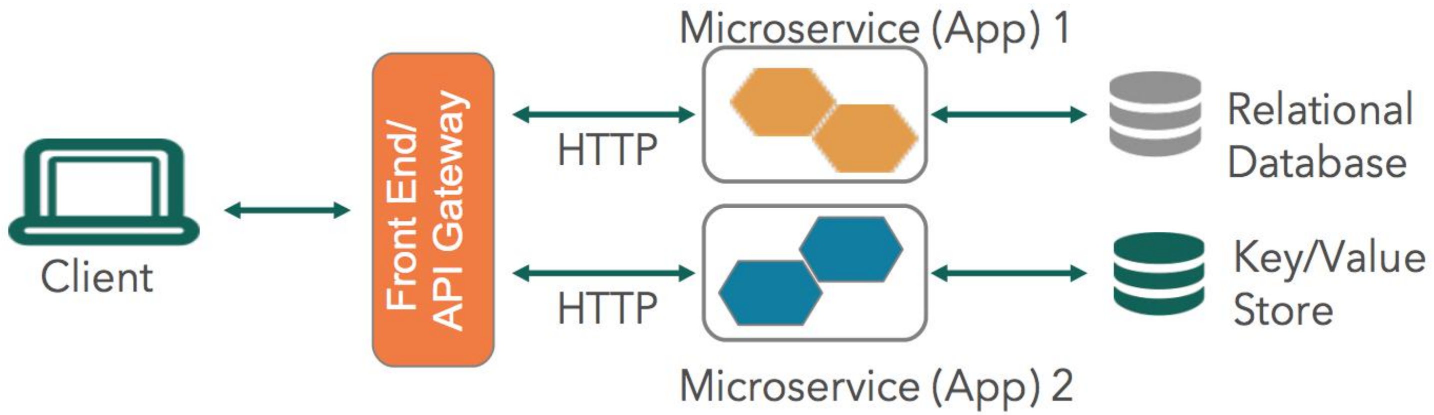
for Pivotal Cloud Foundry

Monolithic vs. Microservice Architecture

- Classic 3-tier application

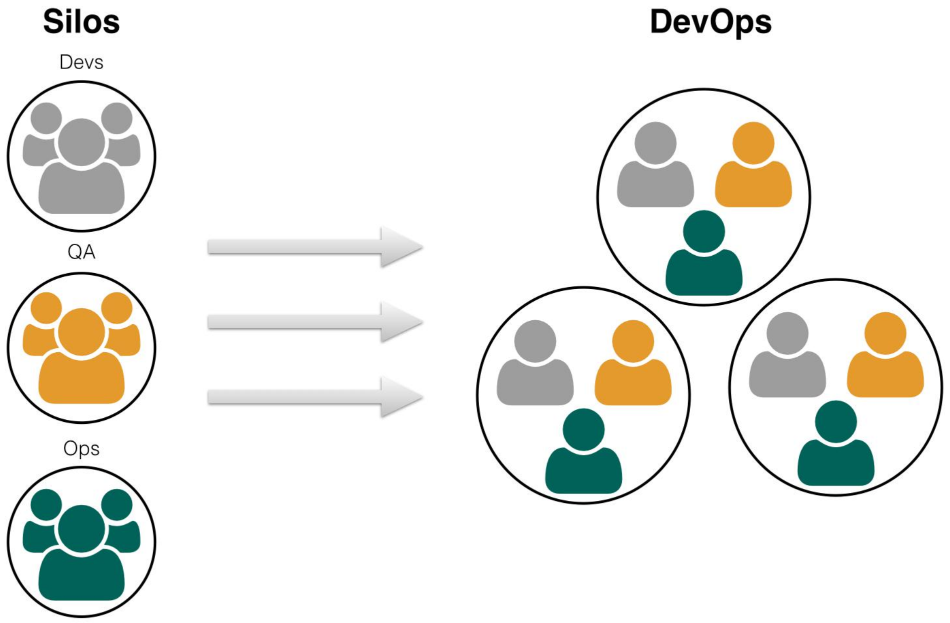


- Microservice architecture



Silos to DevOps

- Goal: Deliver value rapidly and safely
- Shared vocabulary, tools, and incentive structures
- Bureaucratic processes replaced with trust & accountability
- Common leadership



Spring Cloud

- **Spring Cloud Netflix**:
  - **Hystrix**: circuit breaker
  - **Eureka**: service discovery
  - **Ribbon**: client-side load balancer
  - **Feign**: declarative REST Client
  - **Zuul**: API proxy server
- **Spring Cloud Config Server**: configuration as a service
- **Spring Cloud Sleuth**: distributed tracing
- **Spring Cloud Contract**: facilitates contract testing

Spring Cloud Services (SCS)

A PCF managed service for deploying Spring Cloud infrastructure services on-demand in the cloud

Currently supported:

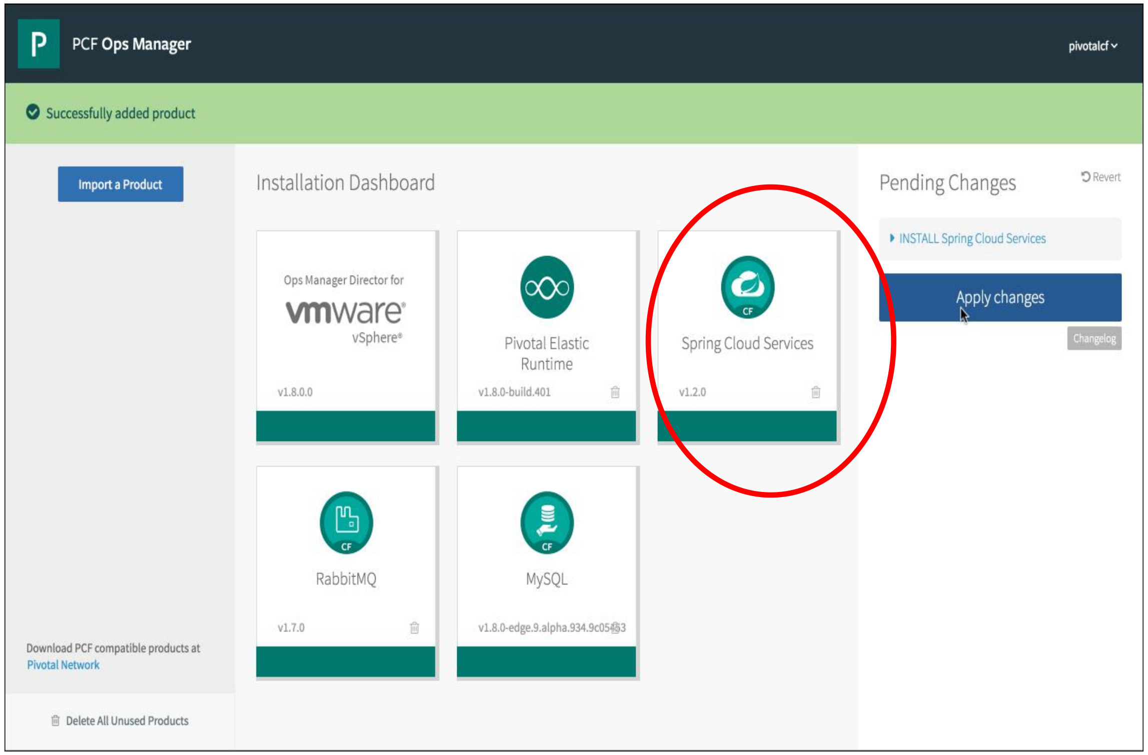
- *Config Server*
- *Service Registry*
- *Circuit Breaker*

https://docs.pivotal.io/spring-cloud-services

Installation

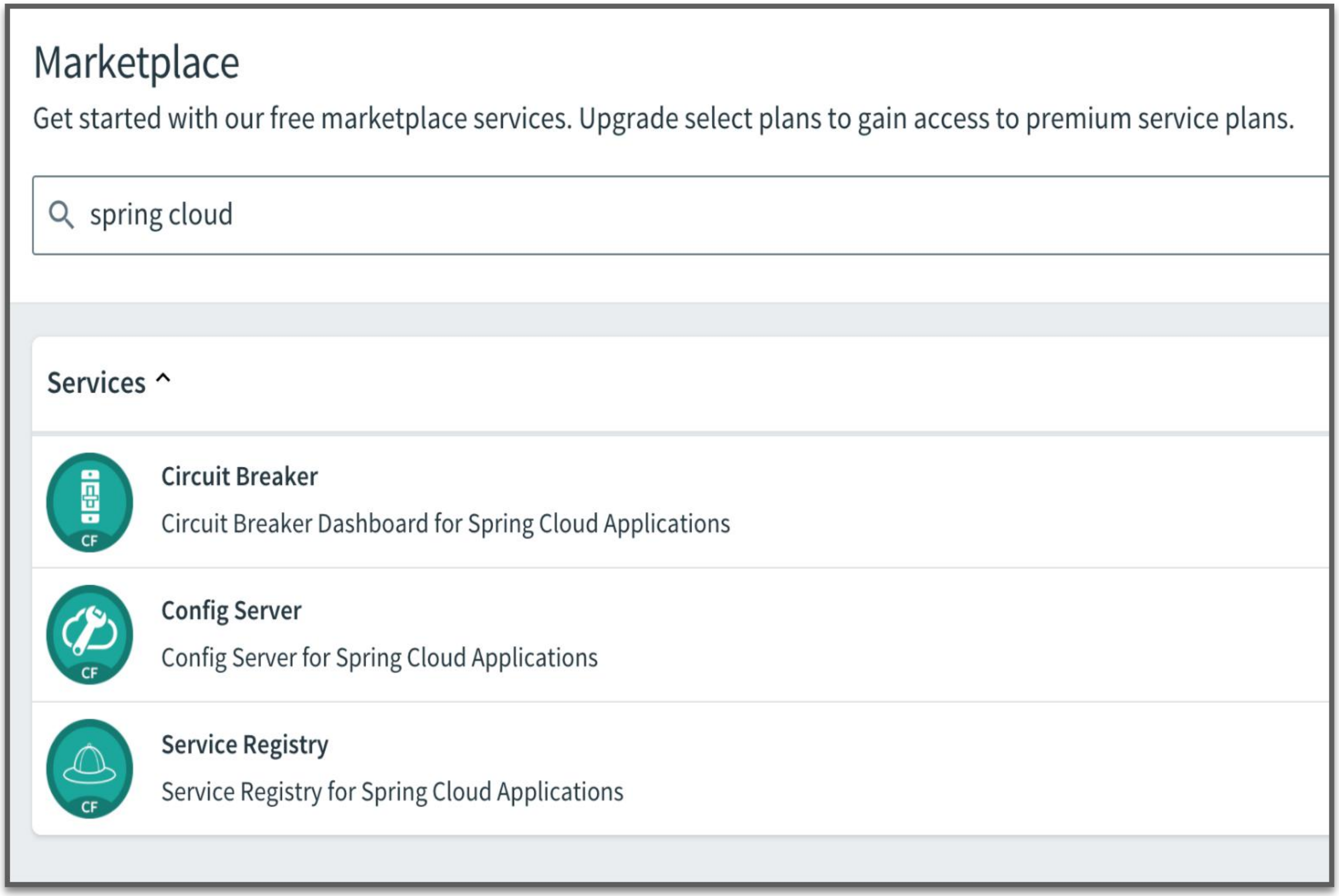
Spring Cloud Services is packaged as a PCF “tile", installed by a PCF administrator as a CF extension

The installation involves the deployment of the service broker and registration of services into the PCF marketplace


PCF Ops Manager
pivotalcf
Successfully added product
Import a Product
Installation Dashboard
Pending Changes
Revert
INSTALL Spring Cloud Services
Apply changes
Changelog
Ops Manager Director for vmware vSphere
v1.8.0.0
Pivotal Elastic Runtime
v1.8.0-build.401
Spring Cloud Services
v1.2.0
RabbitMQ
v1.7.0
MySQL
v1.8.0-edge.9.alpha.934.9c05853
Download PCF compatible products at Pivotal Network
Delete All Unused Products

PCF Operations Manager

Can verify if a PCF instance has the Spring Cloud Services installed by looking for the presence of these services in the Cloud Foundry Marketplace



Marketplace

Get started with our free marketplace services. Upgrade select plans to gain access to premium service plans.

spring cloud

Services ^

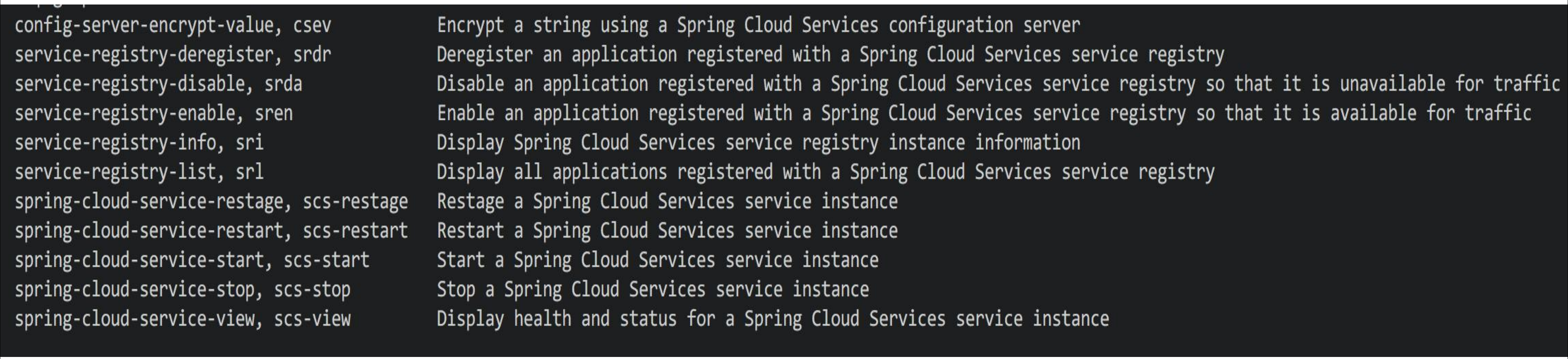
Circuit Breaker
Circuit Breaker Dashboard for Spring Cloud Applications

Config Server
Config Server for Spring Cloud Applications

Service Registry
Service Registry for Spring Cloud Applications

cf plugin for SCS

A plugin is available for the cf CLI, that provides the following commands:



```
config-server-encrypt-value, csev         Encrypt a string using a Spring Cloud Services configuration server
service-registry-deregister, srdr         Deregister an application registered with a Spring Cloud Services service registry
service-registry-disable, srda            Disable an application registered with a Spring Cloud Services service registry so that it is unavailable for traffic
service-registry-enable, sren             Enable an application registered with a Spring Cloud Services service registry so that it is available for traffic
service-registry-info, sri                Display Spring Cloud Services service registry instance information
service-registry-list, srl                Display all applications registered with a Spring Cloud Services service registry
spring-cloud-service-restage, scs-restage Restage a Spring Cloud Services service instance
spring-cloud-service-restart, scs-restart Restart a Spring Cloud Services service instance
spring-cloud-service-start, scs-start     Start a Spring Cloud Services service instance
spring-cloud-service-stop, scs-stop       Stop a Spring Cloud Services service instance
spring-cloud-service-view, scs-view       Display health and status for a Spring Cloud Services service instance
```

These commands can be useful for analysis and troubleshooting of provisioned services.

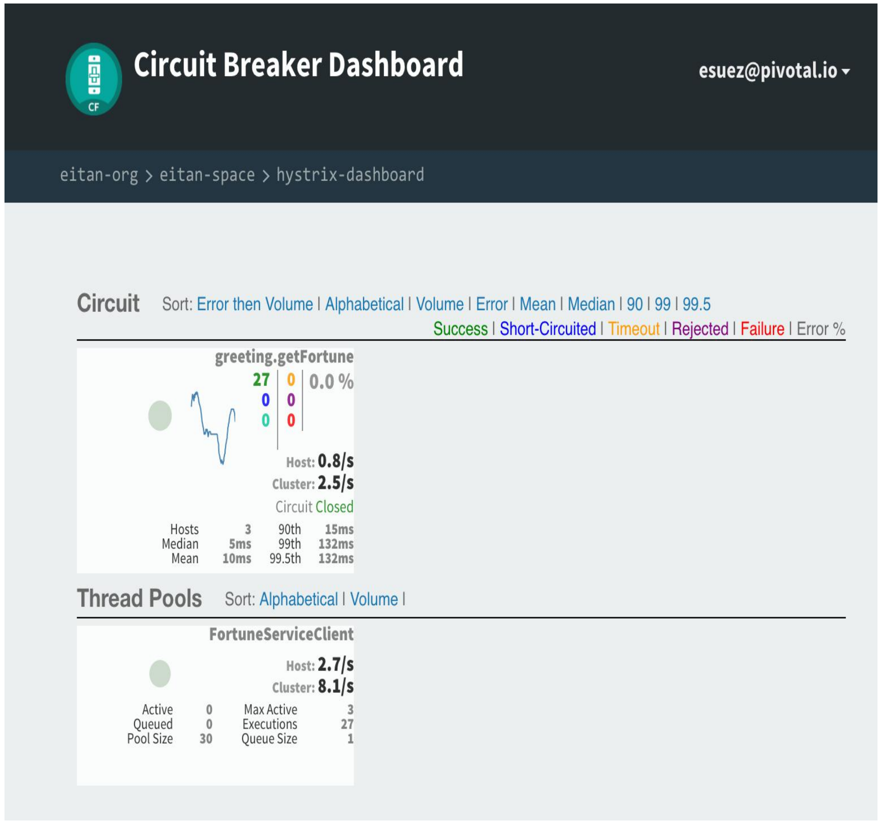
https://plugins.cloudfoundry.org/

Provisioning a Circuit Breaker

Example:

```
cf create-service p-circuit-breaker—dashboard standard cb-dashboard
```

After service has been provisioned,
the circuit breaker dashboard is accessible
directly from PCF *Apps Manager*



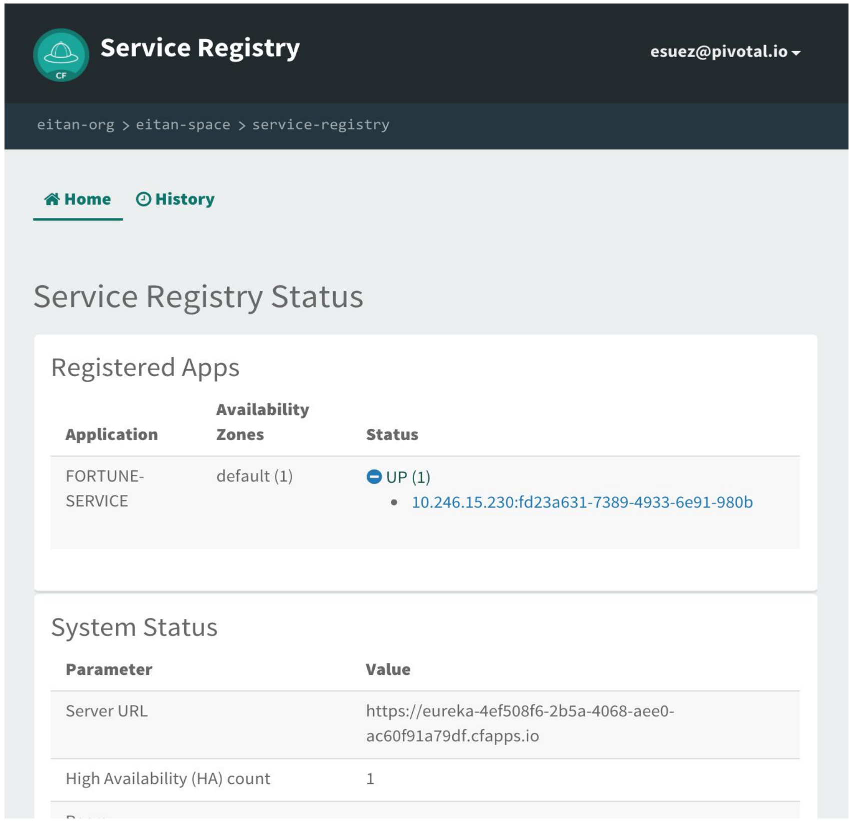
```
Circuit Breaker Dashboard                                   esuez@pivotal.io
eitan-org > eitan-space > hystrix-dashboard

Circuit   Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5
          Success | Short-Circuited | Timeout | Rejected | Failure | Error %

                 greeting.getFortune
                 27 | 0 | 0.0 %
                  0 | 0 |
                  0 | 0 |
                          Host: 0.8/s
                       Cluster: 2.5/s
                        Circuit Closed
  Hosts      3     90th   15ms
  Median   5ms     99th  132ms
  Mean    10ms   99.5th  132ms

Thread Pools   Sort: Alphabetical | Volume |

                 FortuneServiceClient
                          Host: 2.7/s
                       Cluster: 8.1/s
  Active      0   Max Active    3
  Queued      0   Executions   27
  Pool Size  30   Queue Size    1
```

Provisioning a Service Registry

Example:

```
cf create-service p-service-registry standard service-registry
```

After service has been provisioned,
the service registry dashboard is accessible
directly from PCF *Apps Manager*



Service Registry Status

Registered Apps

Application	Availability Zones	Status
FORTUNE-SERVICE	default (1)	UP (1) • 10.246.15.230:fd23a631-7389-4933-6e91-980b

System Status

Parameter	Value
Server URL	https://eureka-4ef508f6-2b5a-4068-aee0-ac60f91a79df.cfapps.io
High Availability (HA) count	1

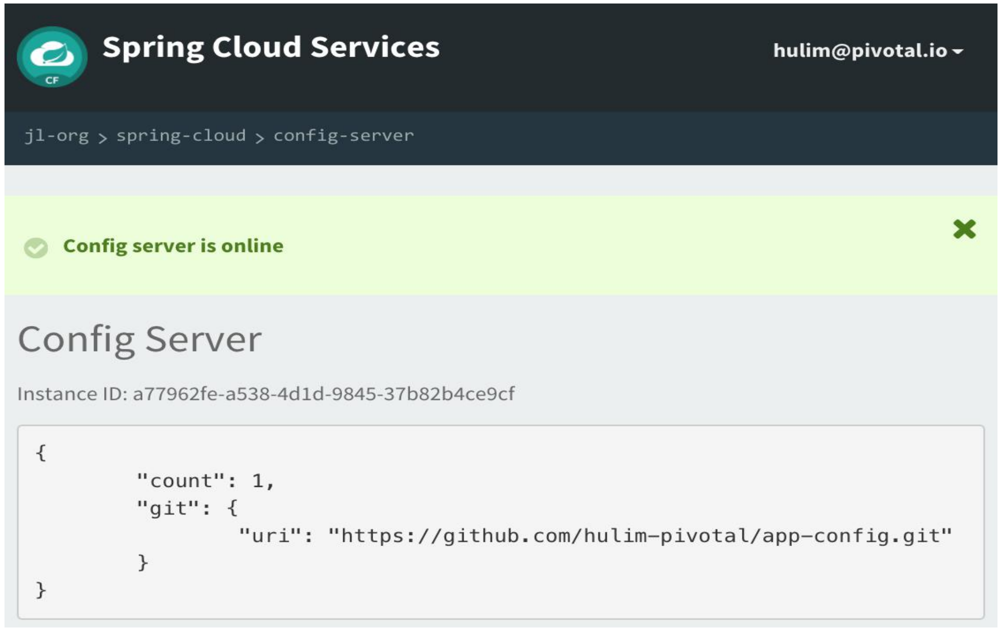
Provisioning a Config Server

- Example:

```
cf create-service p-config-server standard config-server -c config.json
```

- minimal *config.json* file contents:

```
{"git": {"uri": "https://github.com/<username>/config-repo.git" }}
```



Config server is online

Config Server

Instance ID: a77962fe-a538-4d1d-9845-37b82b4ce9cf

```
{
        "count": 1,
        "git": {
                "uri": "https://github.com/hulim-pivotal/app-config.git"
        }
}
```

Circuit Breakers

Netflix Hystrix

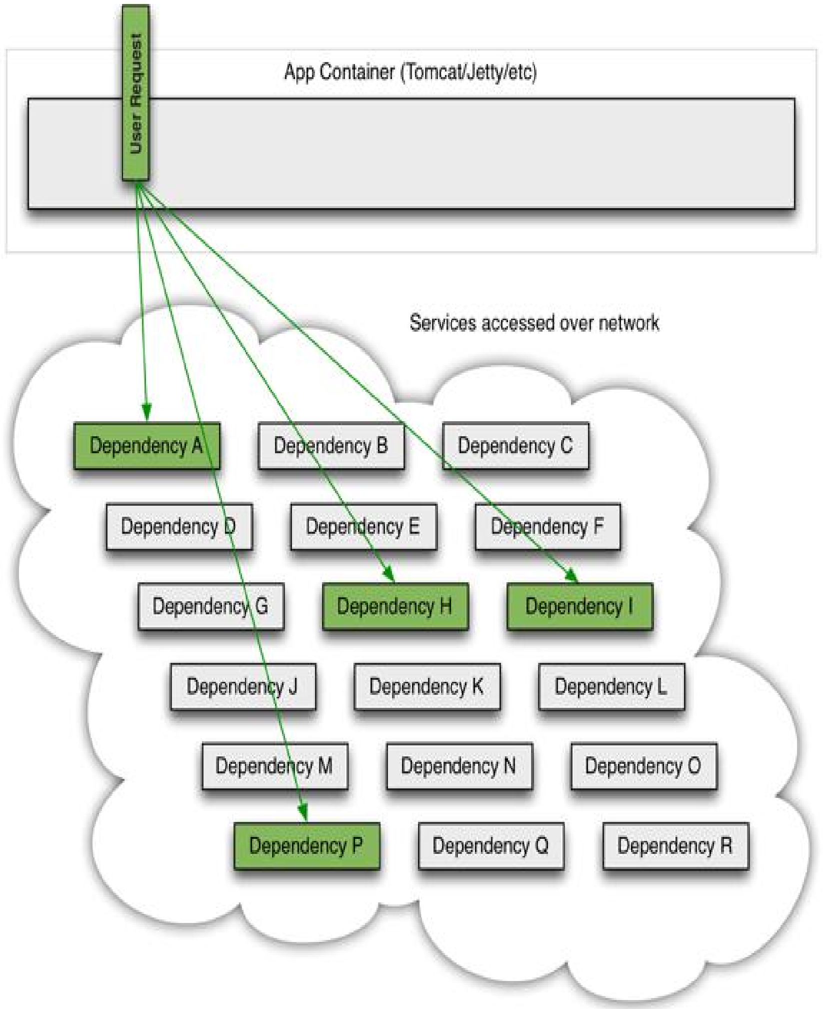
Motivation

“Applications in complex distributed architectures have dozens of dependencies, each of which will inevitably fail at some point.

If the host application is not isolated from these external failures, it risks being taken down with them.”

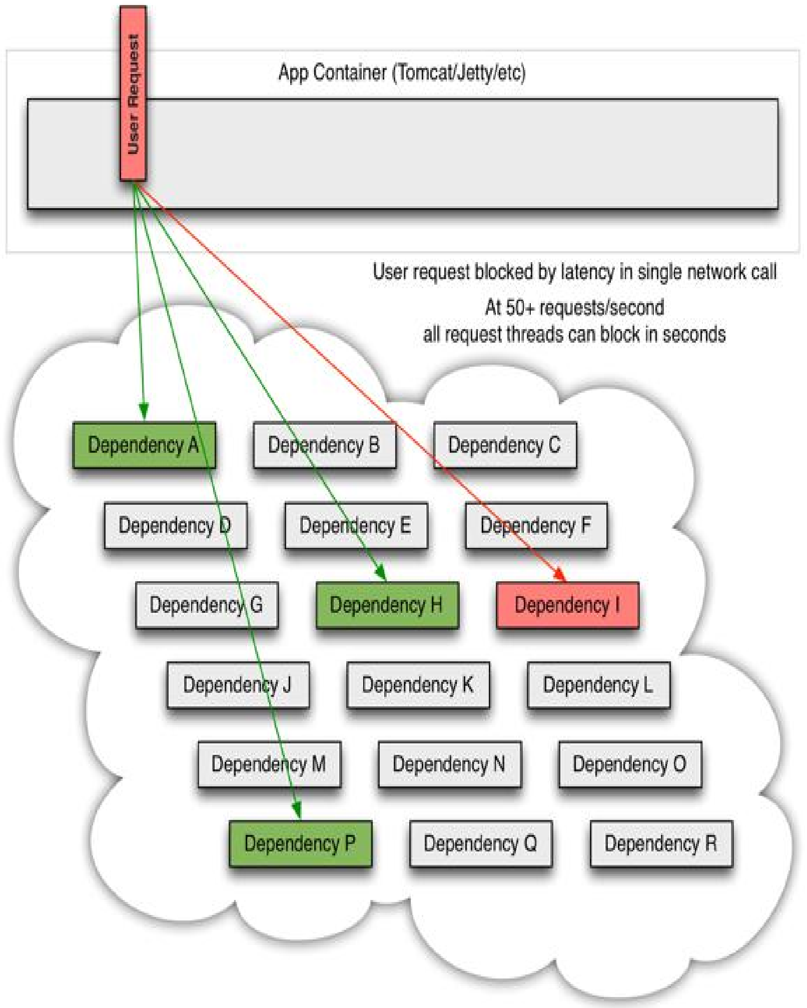
Services Dependency Scenario

- A typical application depending on a number of backing services
- All services are up and behaving normally
- Circuit is *Closed*



Failing Dependency

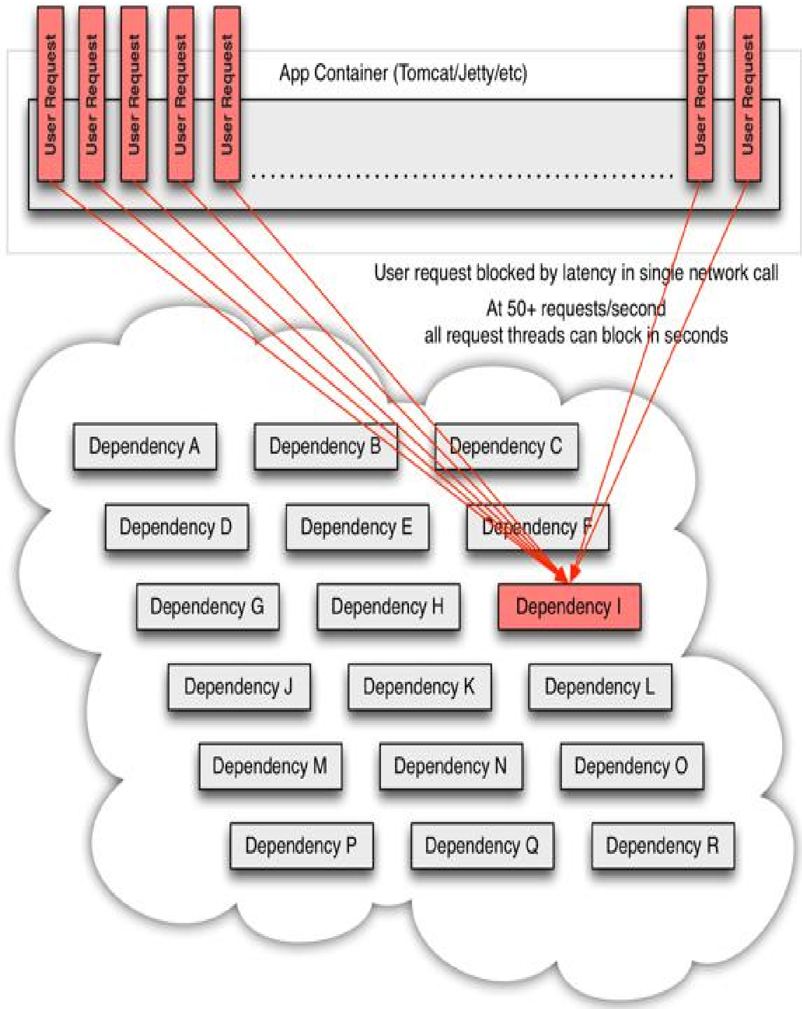
- A dependency begins misbehaves
- Response latency increases, tying up thread in calling application



Failure Cascades to Caller

- Calling application’s thread pool is exhausted waiting on misbehaving dependency

- Failure cascades to caller



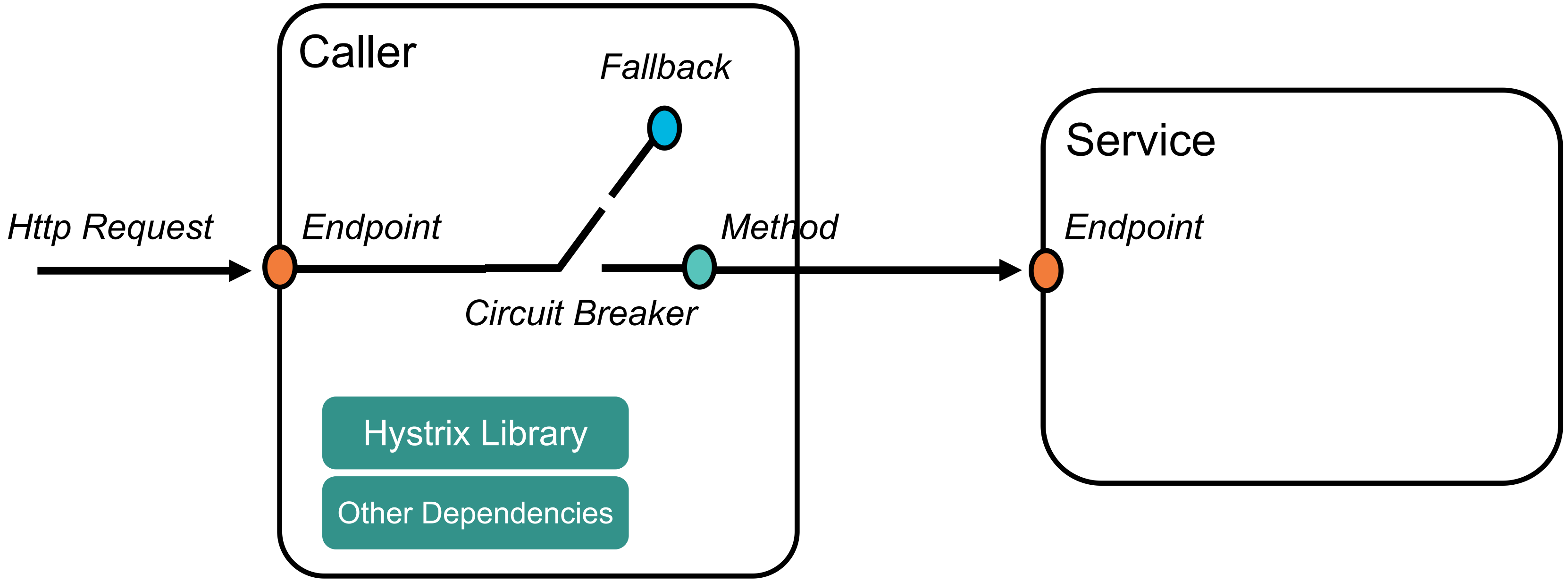
What is it for?

- Give protection & control over latency & failure from dependencies accessed via 3rd-party client libraries
- Stop cascading failures in a complex distributed system
- Fail fast and rapidly recover
- Fallback and gracefully degrade when possible
- Enable near real-time monitoring, alerting, operational control

Circuit Breaker isolates calls to other services

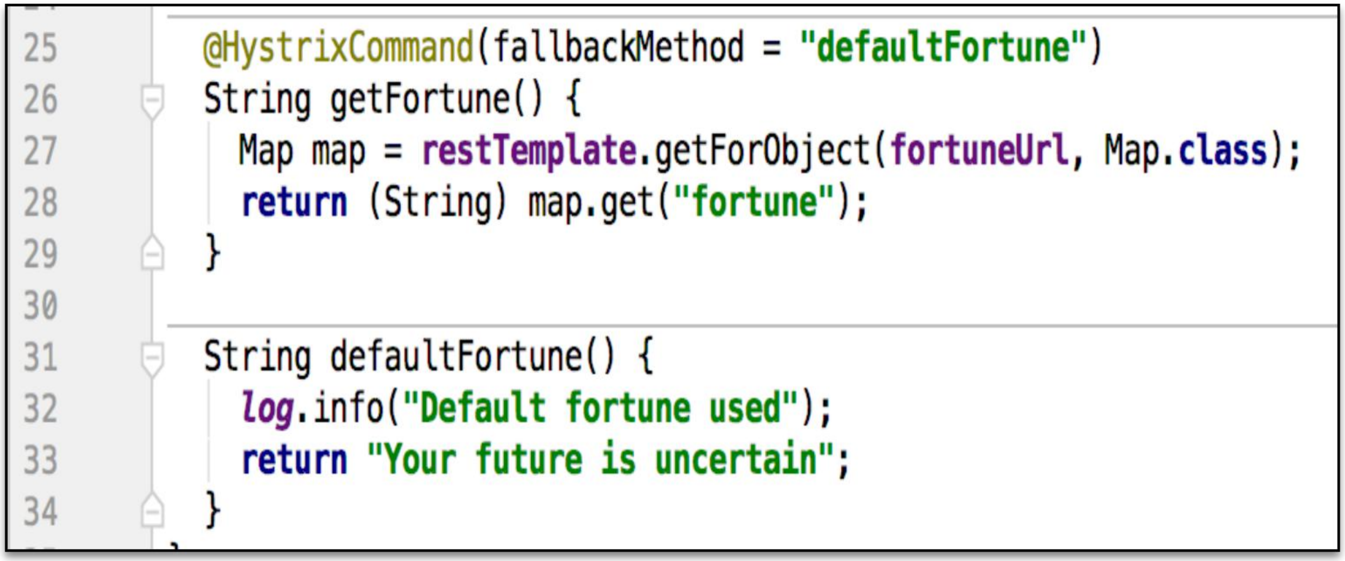
Application is isolated from a misbehaving backing service

When backing service health is restored, calling application will automatically reconfigure itself to call it once more

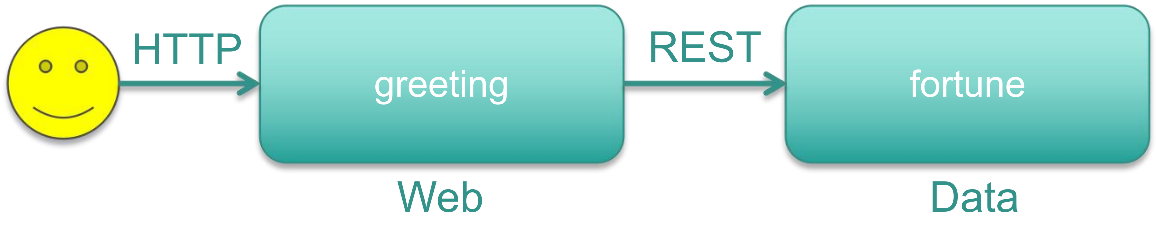


Annotating a service call

Add build dependency: spring-cloud-starter-hystrix



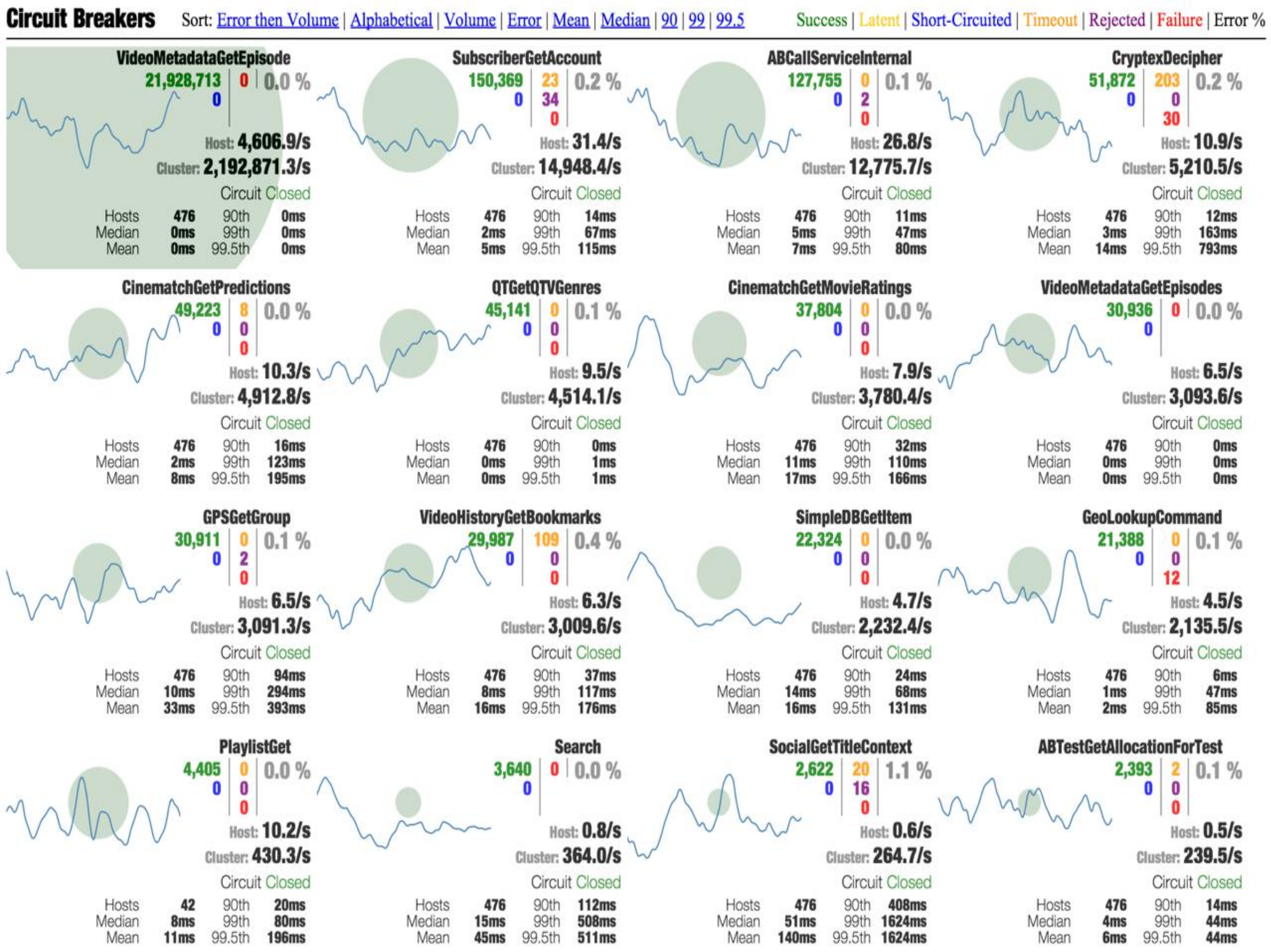
```
@HystrixCommand(fallbackMethod = "defaultFortune")
String getFortune() {
  Map map = restTemplate.getForObject(fortuneUrl, Map.class);
  return (String) map.get("fortune");
}

String defaultFortune() {
  log.info("Default fortune used");
  return "Your future is uncertain";
}
```



Configuration

	Default
execution.isolation.thread.timeoutInMilliseconds The time in milliseconds after which the caller will observe a timeout and walk away from the command execution	1000 ms
circuitBreaker.requestVolumeThreshold The minimum number of requests in a rolling window that will trip the circuit	20 requests min in rolling window
circuitBreaker.sleepWindowInMilliseconds The amount of time, after tripping the circuit, to reject requests before allowing attempts again to determine if the circuit should again be closed	5000 ms
circuitBreaker.errorThresholdPercentage The error percentage at or above which the circuit should trip open and start short-circuiting requests to fallback logic	50%
hystrix.threadpool.*HystrixThreadPoolKey*.maximumSize The maximum thread-pool size. This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands	10

Hystrix Dashboard



Circuit Breakers Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

Success | Latent | Short-Circuited | Timeout | Rejected | Failure | Error %

VideoMetadataGetEpisode
21,928,713 | 0 | 0.0 %
0
Host: 4,606.9/s
Cluster: 2,192,871.3/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

SubscriberGetAccount
150,369 | 23 | 0.2 %
0 | 34
0
Host: 31.4/s
Cluster: 14,948.4/s
Circuit Closed
Hosts 476 90th 14ms
Median 2ms 99th 67ms
Mean 5ms 99.5th 115ms

ABCallServiceInternal
127,755 | 0 | 0.1 %
0 | 2
0
Host: 26.8/s
Cluster: 12,775.7/s
Circuit Closed
Hosts 476 90th 11ms
Median 5ms 99th 47ms
Mean 7ms 99.5th 80ms

CryptexDecipher
51,872 | 203 | 0.2 %
0 | 0
30
Host: 10.9/s
Cluster: 5,210.5/s
Circuit Closed
Hosts 476 90th 12ms
Median 3ms 99th 163ms
Mean 14ms 99.5th 793ms

CinematchGetPredictions
49,223 | 8 | 0.0 %
0 | 0
0
Host: 10.3/s
Cluster: 4,912.8/s
Circuit Closed
Hosts 476 90th 16ms
Median 2ms 99th 123ms
Mean 8ms 99.5th 195ms

QTGetQTVGenres
45,141 | 0 | 0.1 %
0 | 0
0
Host: 9.5/s
Cluster: 4,514.1/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 1ms

CinematchGetMovieRatings
37,804 | 0 | 0.0 %
0 | 0
0
Host: 7.9/s
Cluster: 3,780.4/s
Circuit Closed
Hosts 476 90th 32ms
Median 11ms 99th 110ms
Mean 17ms 99.5th 166ms

VideoMetadataGetEpisodes
30,936 | 0 | 0.0 %
0
Host: 6.5/s
Cluster: 3,093.6/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms

GPSGetGroup
30,911 | 0 | 0.1 %
0 | 2
0
Host: 6.5/s
Cluster: 3,091.3/s
Circuit Closed
Hosts 476 90th 94ms
Median 10ms 99th 294ms
Mean 33ms 99.5th 393ms

VideoHistoryGetBookmarks
29,987 | 109 | 0.4 %
0 | 0
0
Host: 6.3/s
Cluster: 3,009.6/s
Circuit Closed
Hosts 476 90th 37ms
Median 8ms 99th 117ms
Mean 16ms 99.5th 176ms

SimpleDBGetItem
22,324 | 0 | 0.0 %
0 | 0
0
Host: 4.7/s
Cluster: 2,232.4/s
Circuit Closed
Hosts 476 90th 24ms
Median 14ms 99th 68ms
Mean 16ms 99.5th 131ms

GeoLookupCommand
21,388 | 0 | 0.1 %
0 | 0
12
Host: 4.5/s
Cluster: 2,135.5/s
Circuit Closed
Hosts 476 90th 6ms
Median 1ms 99th 47ms
Mean 2ms 99.5th 85ms

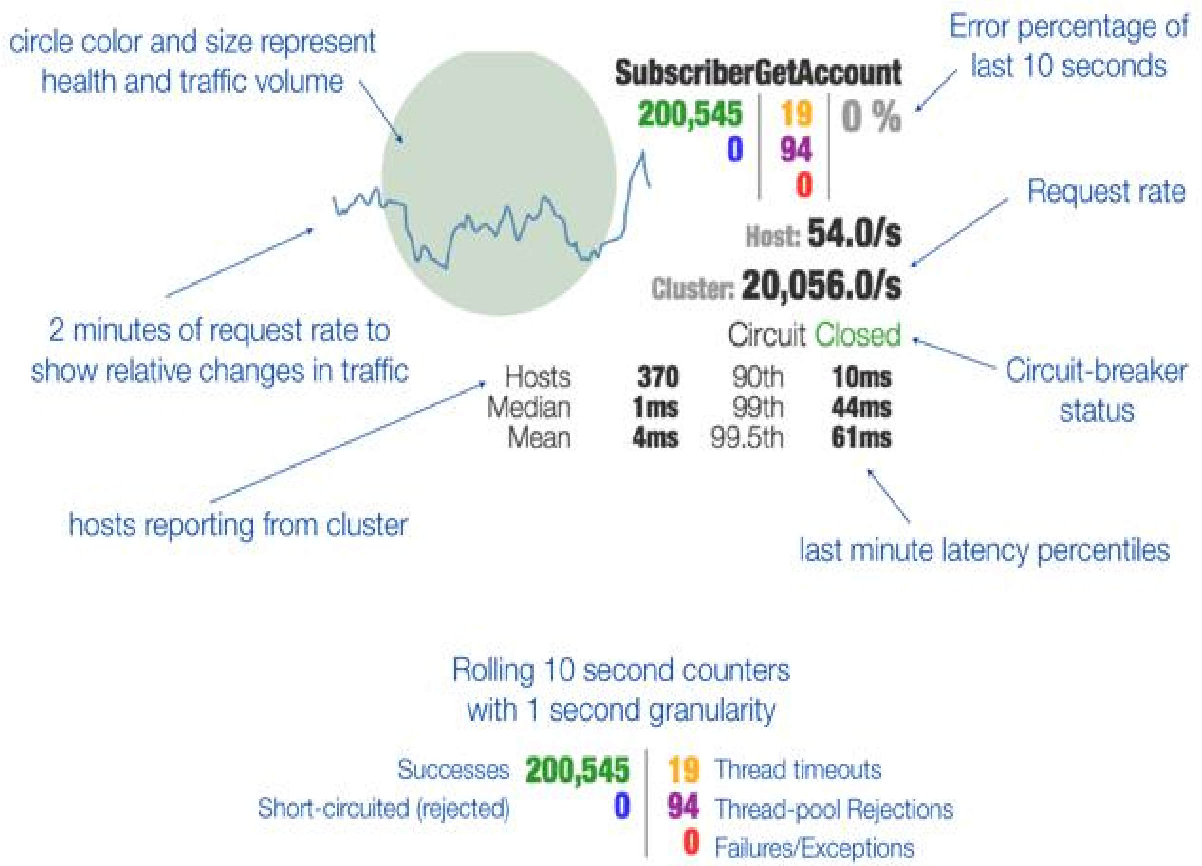
PlaylistGet
4,405 | 0 | 0.0 %
0 | 0
0
Host: 10.2/s
Cluster: 430.3/s
Circuit Closed
Hosts 42 90th 20ms
Median 8ms 99th 80ms
Mean 11ms 99.5th 196ms

Search
3,640 | 0 | 0.0 %
0
Host: 0.8/s
Cluster: 364.0/s
Circuit Closed
Hosts 476 90th 112ms
Median 15ms 99th 508ms
Mean 45ms 99.5th 511ms

SocialGetTitleContext
2,622 | 20 | 1.1 %
0 | 16
0
Host: 0.6/s
Cluster: 264.7/s
Circuit Closed
Hosts 476 90th 408ms
Median 51ms 99th 1624ms
Mean 140ms 99.5th 1624ms

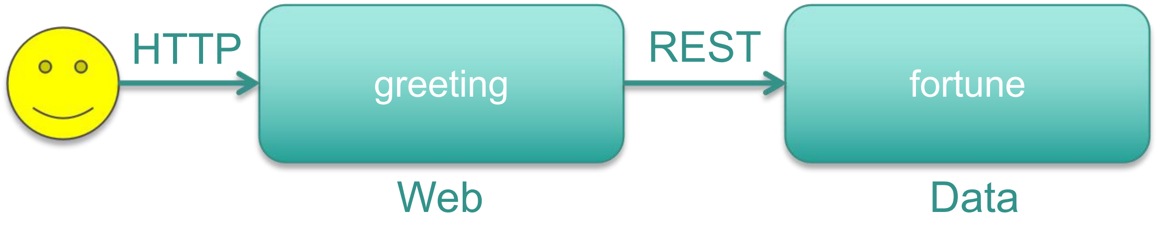
ABTestGetAllocationForTest
2,393 | 2 | 0.1 %
0 | 0
0
Host: 0.5/s
Cluster: 239.5/s
Circuit Closed
Hosts 476 90th 14ms
Median 4ms 99th 44ms
Mean 6ms 99.5th 44ms

Circuit Breaker Monitoring


circle color and size represent health and traffic volume
SubscriberGetAccount
200,545
19
0 %
0
94
0
Error percentage of last 10 seconds
Request rate
Host: 54.0/s
Cluster: 20,056.0/s
Circuit Closed
Circuit-breaker status
2 minutes of request rate to show relative changes in traffic
Hosts 370
Median 1ms
Mean 4ms
90th 10ms
99th 44ms
99.5th 61ms
hosts reporting from cluster
last minute latency percentiles
Rolling 10 second counters with 1 second granularity
Successes 200,545
Short-circuited (rejected) 0
19 Thread timeouts
94 Thread-pool Rejections
0 Failures/Exceptions

Lab / Demo

Hystrix Circuit Breakers



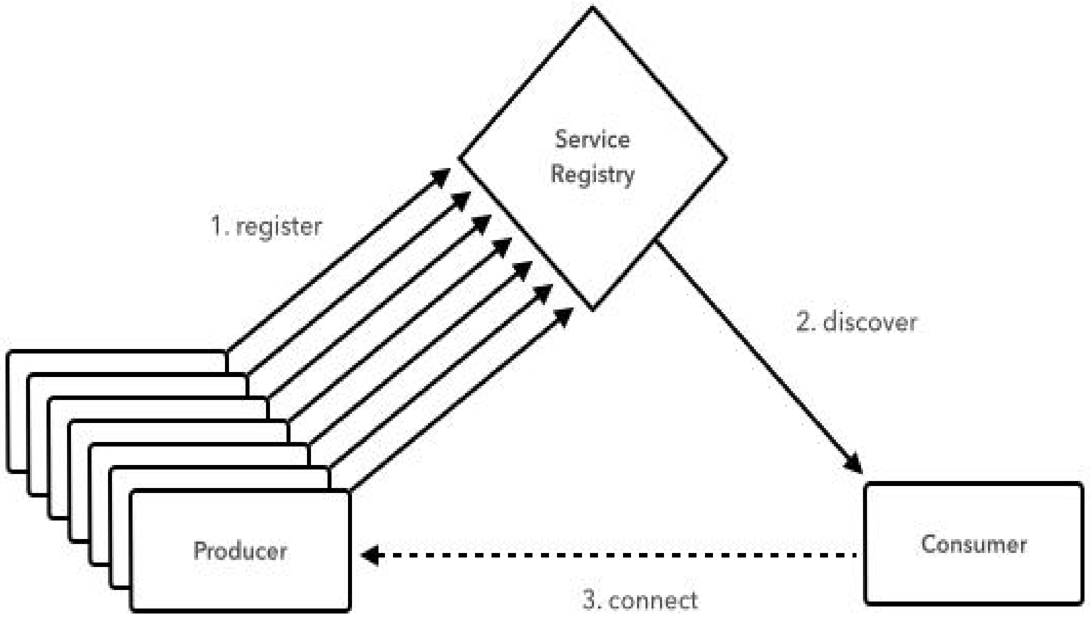
Service Discovery

Netflix Eureka

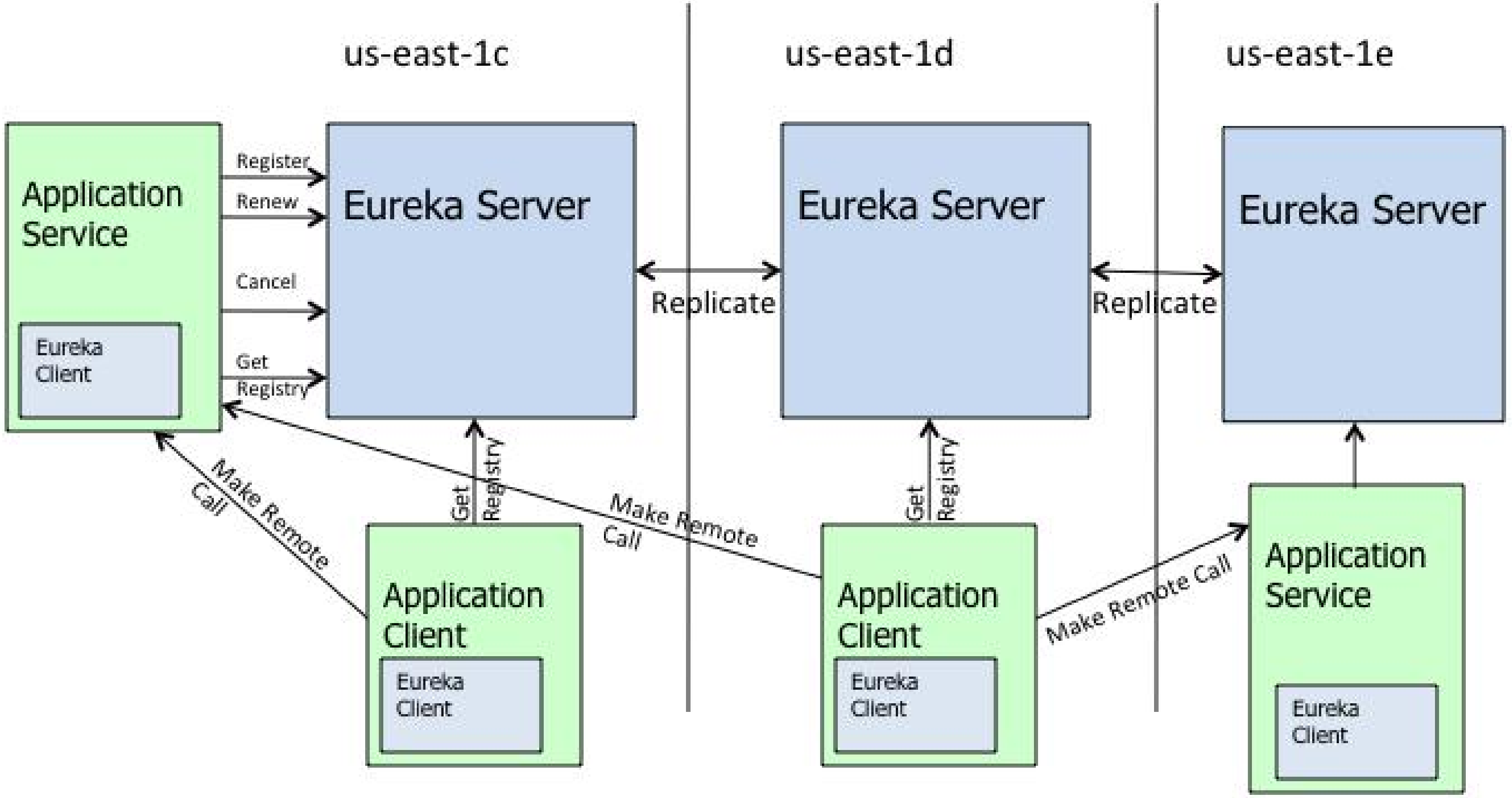
Service Registries

- A microservice architecture consists of many collaborating service instances that much know each others' address
- A cloud environment implies application instances that come and go, that are dynamically scaled
- Service registries provide dynamic application instance lookup capabilities
- Pattern prevalent in distributed systems: Service Locators, Membership Coordinators
- Examples: HashiCorp Consul, Apache ZooKeeper, Netflix Eureka

General Concept



Eureka Architecture



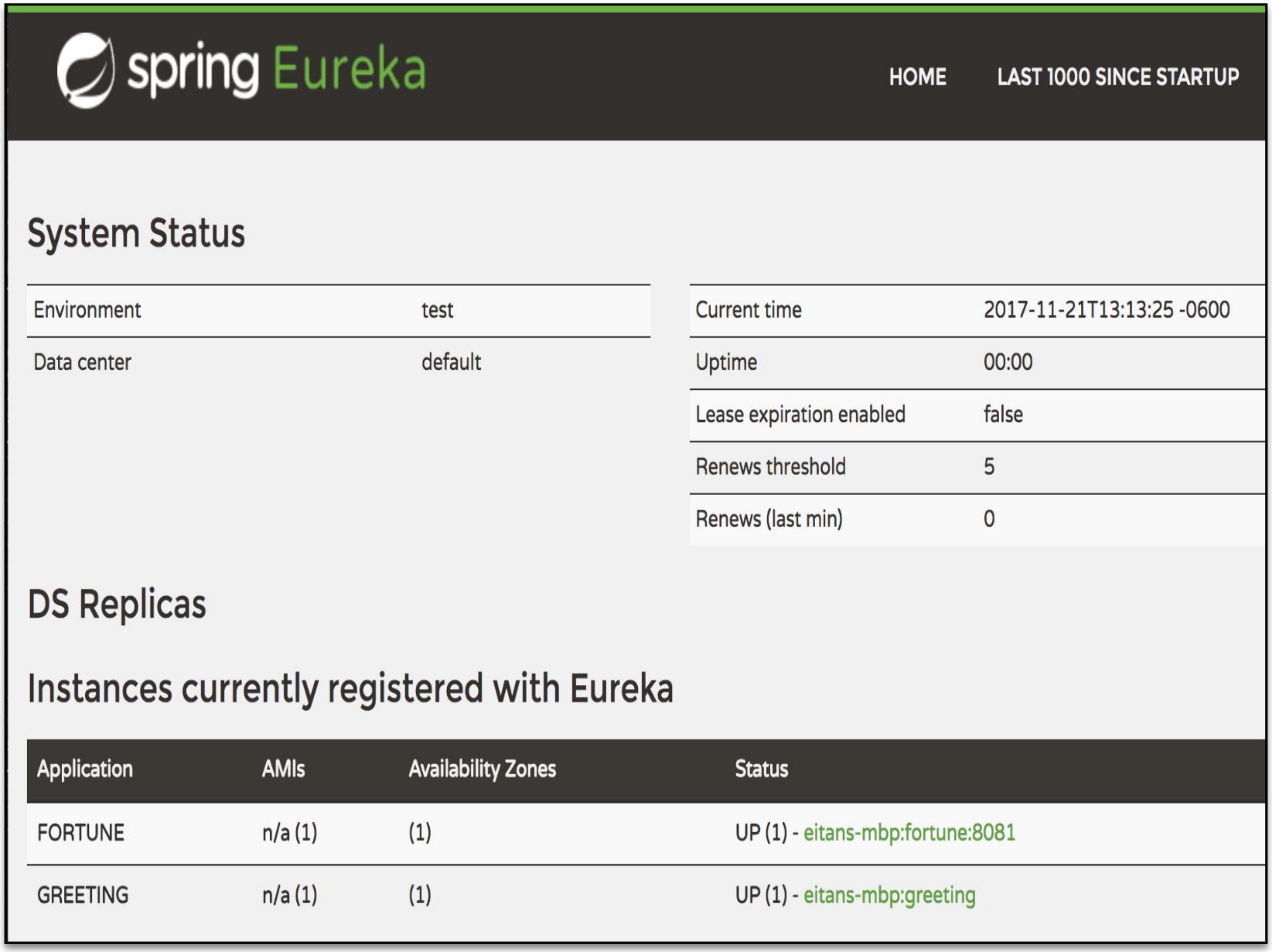
Renew Registration

- Services must periodically renew their registration, which would otherwise expire
- aka “Heartbeats”
- The configuration property eureka.instance.leaseRenewalIntervalInSeconds governs how often a service renews their registration

Fetch Registry

- Clients fetch a copy of the registry periodically
- An optimization, allows lookups to be performed directly against a cached copy
- eureka.client.fetchRegistry can be used to control whether to fetch the registry
- eureka.client.registryFetchIntervalSeconds controls how frequently to fetch a new copy

Eureka Dashboard



System Status

Environment	test
Data center	default

Current time	2017-11-21T13:13:25 -0600
Uptime	00:00
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	0

DS Replicas

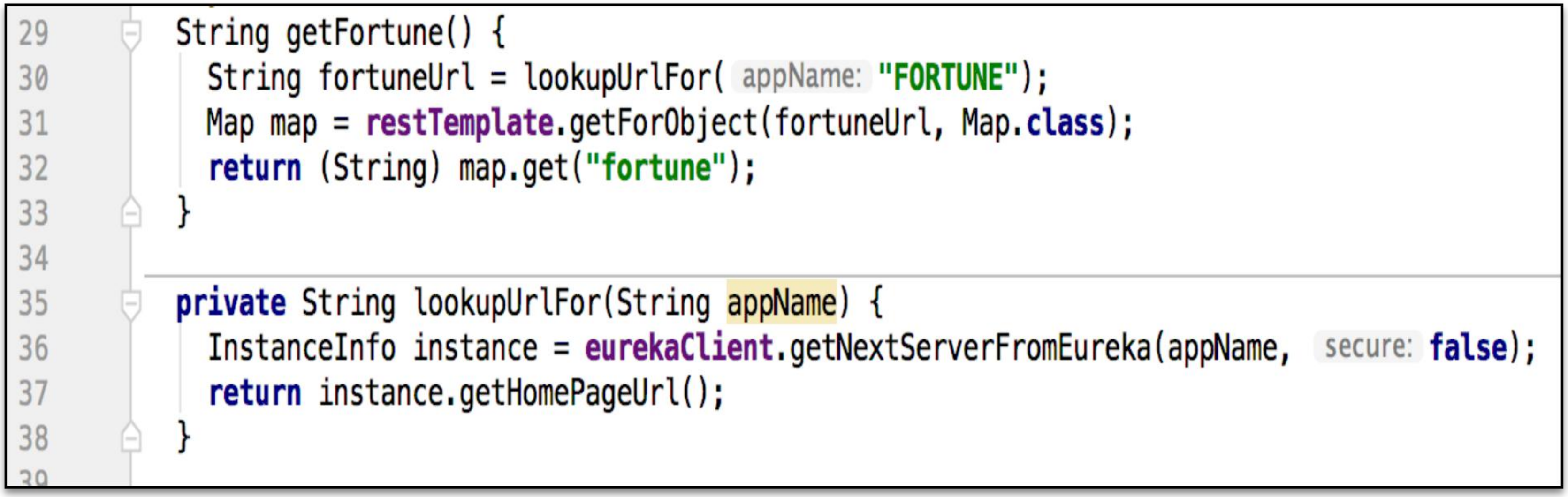
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
FORTUNE	n/a (1)	(1)	UP (1) - eitans-mbp:fortune:8081
GREETING	n/a (1)	(1)	UP (1) - eitans-mbp:greeting

Configuring a eureka instance or client

- Add build dependency: spring-cloud-starter-eureka
- Configure service with spring.application.name property
- Annotate Spring Boot Application class with @EnableDiscoveryClient
- Clients auto-wire a EurekaClient instance`

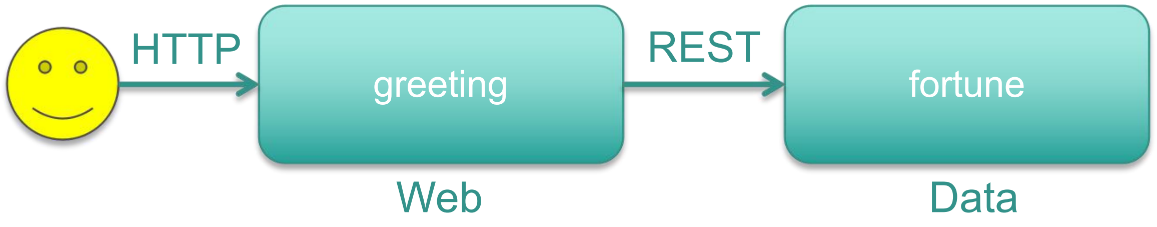
Eureka Lookup Example



```
String getFortune() {
  String fortuneUrl = lookupUrlFor( appName: "FORTUNE");
  Map map = restTemplate.getForObject(fortuneUrl, Map.class);
  return (String) map.get("fortune");
}

private String lookupUrlFor(String appName) {
  InstanceInfo instance = eurekaClient.getNextServerFromEureka(appName,  secure: false);
  return instance.getHomePageUrl();
}
```

Lab / Demo

Eureka Service Discovery



Client-side Load Balancer

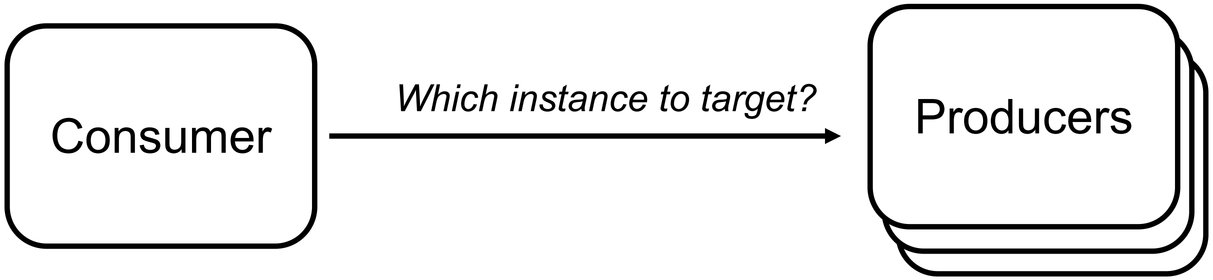
Netflix Ribbon

Traditional vs Microservice Load Balancing

- Traditional LB:
  - LB performed by dedicated appliance e.g. F5 or HAProxy
  - Configured manually
  - Entry point for HTTP requests from end users (public-facing)
  - Fronts monolithic server instances

- Microservice LB:
  - Embed LB logic in consumer (caller)
  - Configuration is dynamic and automatic
  - Not public-facing
  - Load balancing is between services (inter-service)

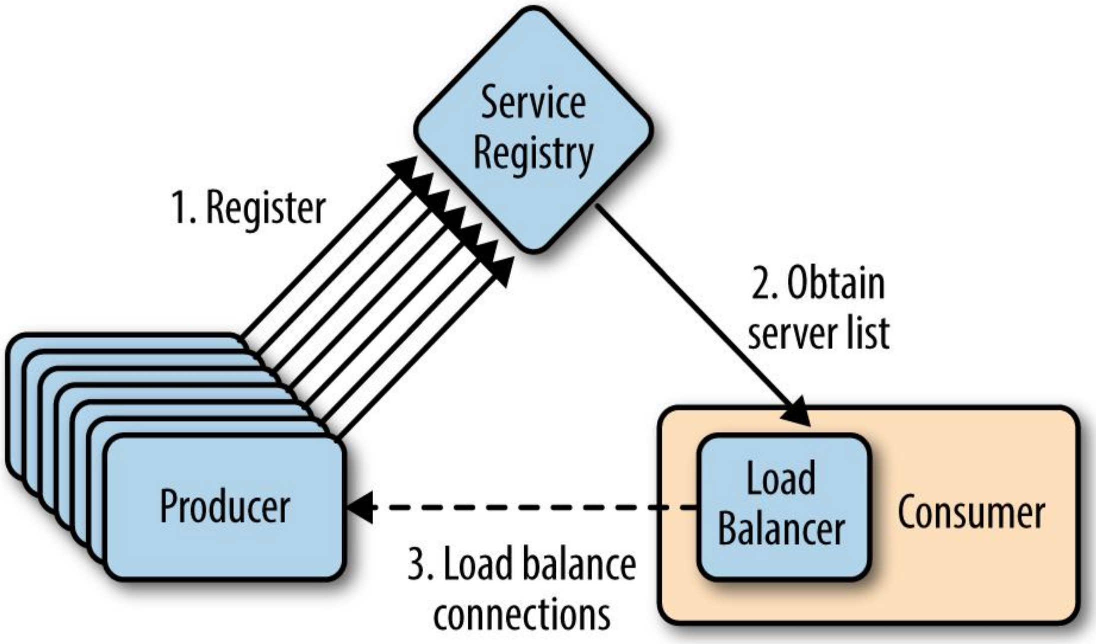
Service Instances are scaled out



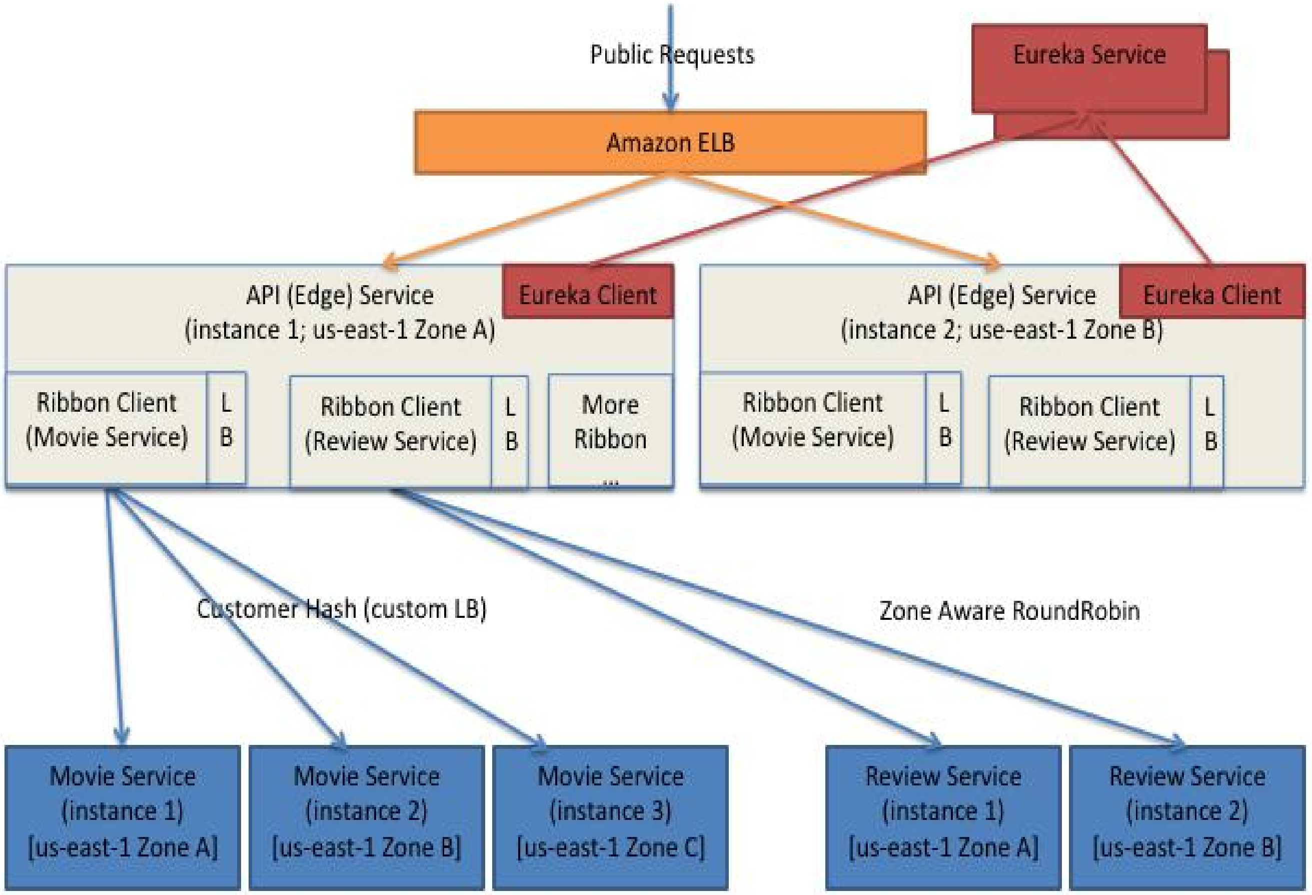
A eureka lookup yields multiple service instances for a given service name

Netflix Ribbon with Eureka

Ribbon provides several LB algos, runs in-process in the consumer, gets its list of producers from Eureka, and so does not require manual configuration of the server list



Inter-service Load Balancing


Public Requests
Eureka Service
Amazon ELB
API (Edge) Service (instance 1; us-east-1 Zone A)
Eureka Client
API (Edge) Service (instance 2; use-east-1 Zone B)
Eureka Client
Ribbon Client (Movie Service)
LB
Ribbon Client (Review Service)
LB
More Ribbon …
Ribbon Client (Movie Service)
LB
Ribbon Client (Review Service)
LB
Customer Hash (custom LB)
Zone Aware RoundRobin
Movie Service (instance 1) [us-east-1 Zone A]
Movie Service (instance 2) [us-east-1 Zone B]
Movie Service (instance 3) [us-east-1 Zone C]
Review Service (instance 1) [us-east-1 Zone A]
Review Service (instance 2) [us-east-1 Zone B]

Load Balancing Rule Options

- RoundRobinRule
- WeightedResponseTimeRule
- RandomRule
- BestAvailableRule
- AvailabilityFilteringRule

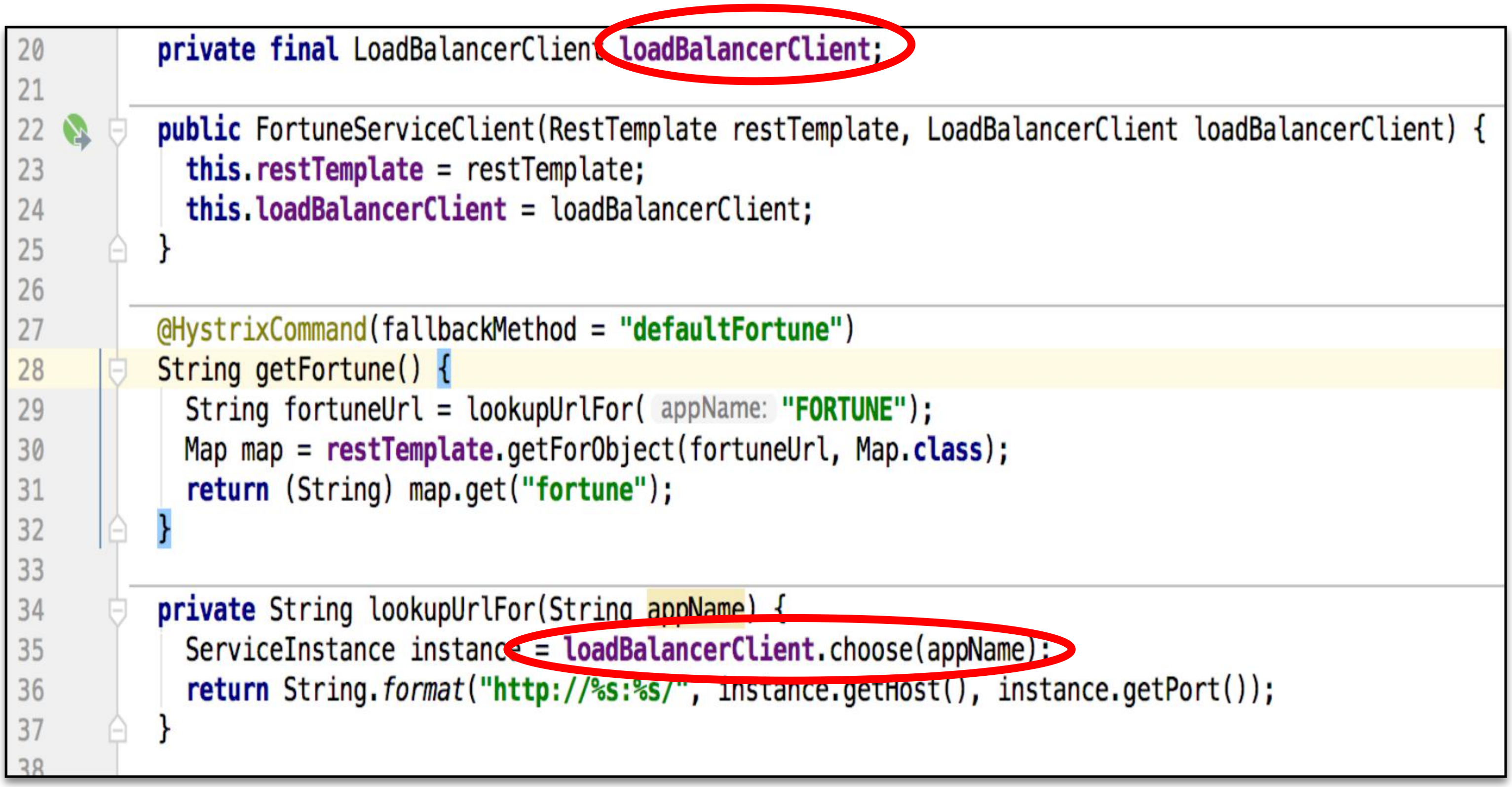
https://github.com/netflix/ribbon/wiki/working-with-load-balancers

Configuration

	Default
myclient.ribbon.ServerListRefreshInterval The time in milliseconds after which the caller will observe a timeout and walk away from the command execution	30 seconds
myclient.ribbon.NFLoadBalancerRuleClassName The implementation of the load balancing Rule (strategy)	*AvailabilityFiltering Rule*
myclient.ribbon.NFLoadBalancerPingClassName Strategy for pinging servers	*NoOpPing*
myclient.ribbon.MaxAutoRetriesNextServer *Max number of next servers to retry (excluding the first server)*	*1*

See: https://github.com/Netflix/ribbon/wiki/Getting-Started

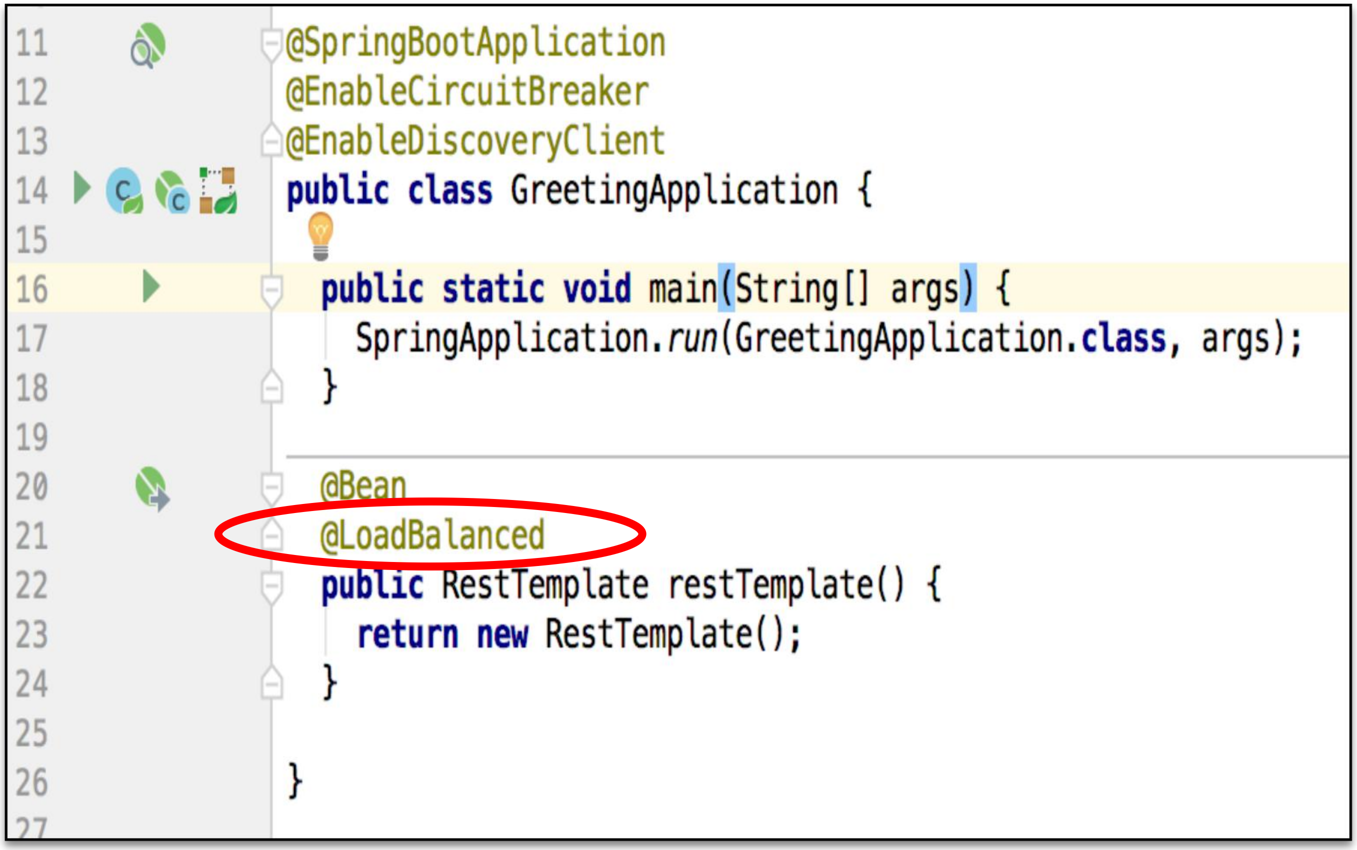
Ribbon Load Balancing Example



```java
private final LoadBalancerClient loadBalancerClient;

public FortuneServiceClient(RestTemplate restTemplate, LoadBalancerClient loadBalancerClient) {
  this.restTemplate = restTemplate;
  this.loadBalancerClient = loadBalancerClient;
}

@HystrixCommand(fallbackMethod = "defaultFortune")
String getFortune() {
  String fortuneUrl = lookupUrlFor( appName: "FORTUNE");
  Map map = restTemplate.getForObject(fortuneUrl, Map.class);
  return (String) map.get("fortune");
}

private String lookupUrlFor(String appName) {
  ServiceInstance instance = loadBalancerClient.choose(appName);
  return String.format("http://%s:%s/", instance.getHost(), instance.getPort());
}
```

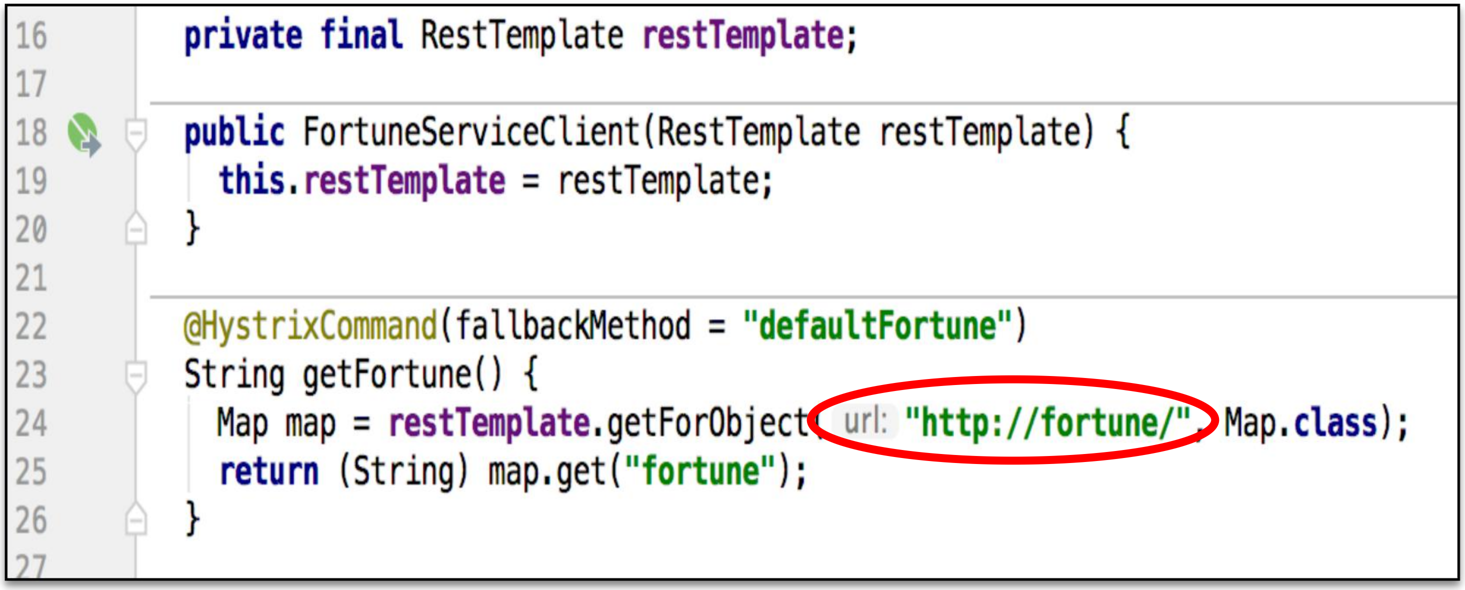
Basically, swap EurekaClient with LoadBalancerClient
API changes slightly: use the choose() method, which returns a ServiceInstance type

Alternative: @LoadBalanced RestTemplate



```java
@SpringBootApplication
@EnableCircuitBreaker
@EnableDiscoveryClient
public class GreetingApplication {

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

    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

}
```

RestTemplate Usage

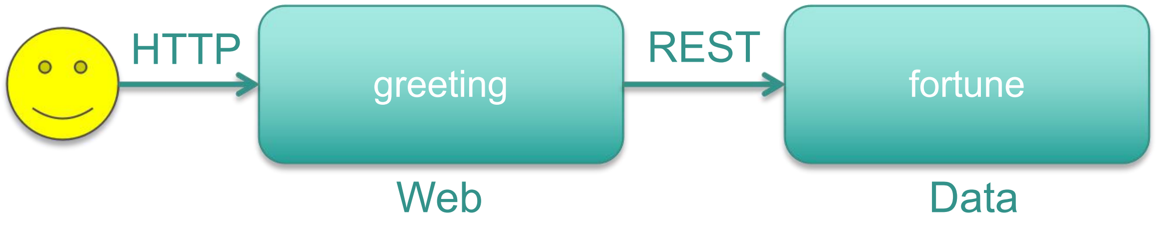


```
    private final RestTemplate restTemplate;

    public FortuneServiceClient(RestTemplate restTemplate) {
      this.restTemplate = restTemplate;
    }

    @HystrixCommand(fallbackMethod = "defaultFortune")
    String getFortune() {
      Map map = restTemplate.getForObject( url: "http://fortune/", Map.class);
      return (String) map.get("fortune");
    }
```

- URL encodes service name (as registered in Eureka)
- Replacement of key with actual service instance returned by load balancing strategy is performed automatically internally to the restTemplate API call (delegates to LoadBalancerClient)

Lab / Demo

Ribbon client-side LB



Declarative REST Client

Netflix Feign

Spring Cloud Feign

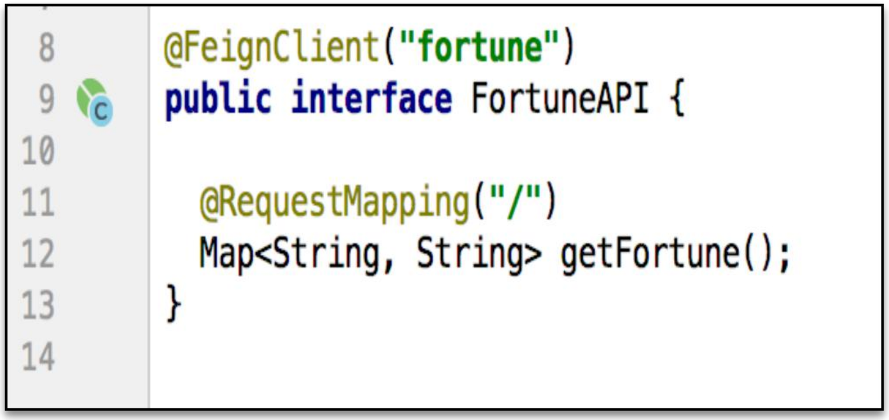
- Encapsulates details of REST API calls behind an interface
- Integrated with Eureka and Ribbon

Steps:

- Add dependency: spring-cloud-starter-feign
- Annotate Spring Boot app class with @EnableFeignClients
- Define the interface

Feign Interface: Simple Example

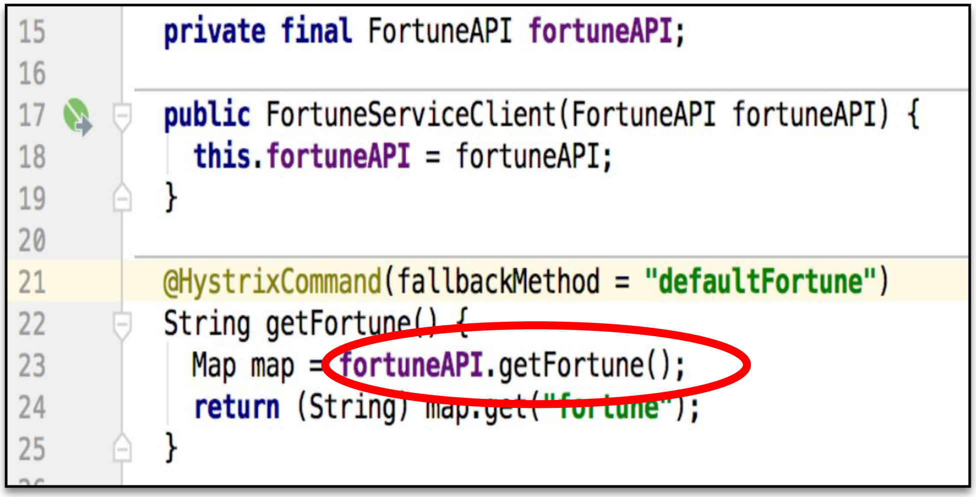
- Annotate class with @FeignClient and indicate the service id of the backing service this interface represents
- Map REST API calls to interface methods
- Annotate each method with the familiar @RequestMapping annotation



```
@FeignClient("fortune")
public interface FortuneAPI {

  @RequestMapping("/")
  Map<String, String> getFortune();
}
```

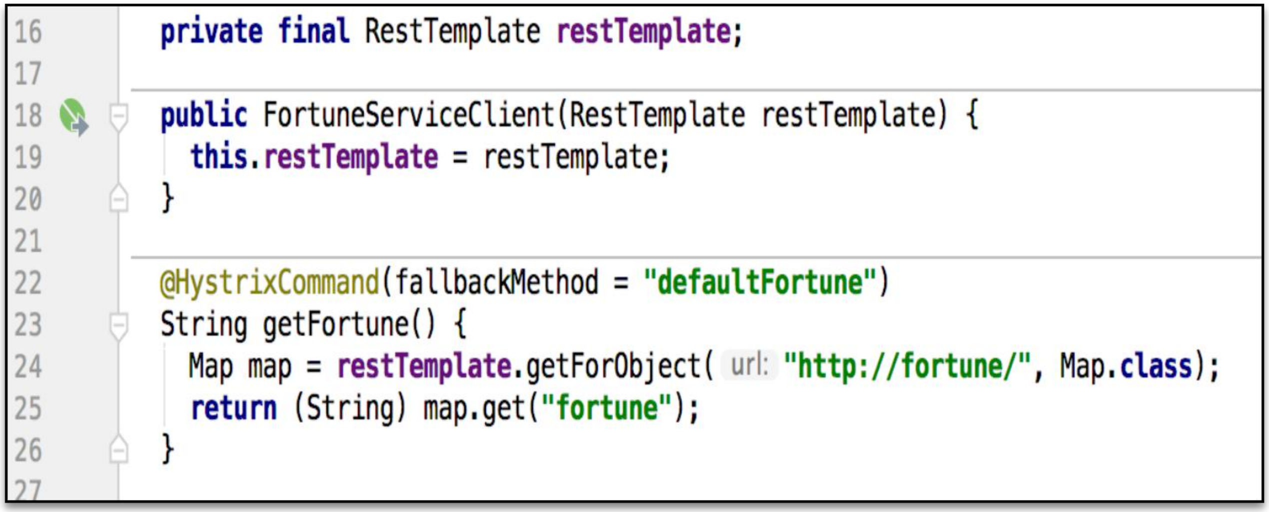
Usage

- Auto-wire the interface into any client class
- To make REST API call, invoke corresponding interface method instead
- Eureka URL lookup and Ribbon load balancing still take place
- Encapsulate a set of related REST API calls behind an interface

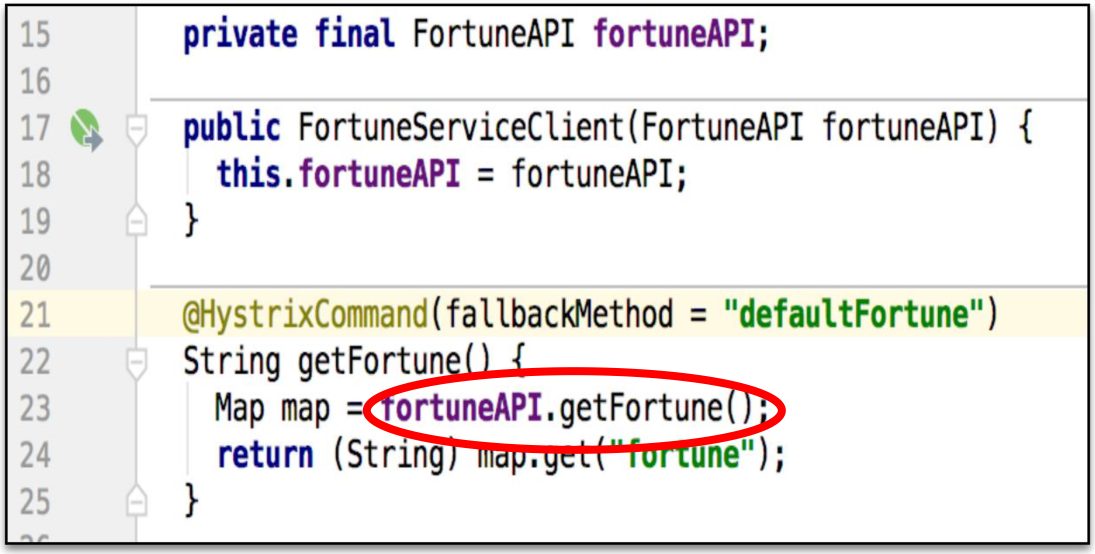


```java
    private final FortuneAPI fortuneAPI;

    public FortuneServiceClient(FortuneAPI fortuneAPI) {
      this.fortuneAPI = fortuneAPI;
    }

    @HystrixCommand(fallbackMethod = "defaultFortune")
    String getFortune() {
      Map map = fortuneAPI.getFortune();
      return (String) map.get("fortune");
    }
```

Contrast with RestTemplate



```java
private final RestTemplate restTemplate;

public FortuneServiceClient(RestTemplate restTemplate) {
  this.restTemplate = restTemplate;
}

@HystrixCommand(fallbackMethod = "defaultFortune")
String getFortune() {
  Map map = restTemplate.getForObject( url: "http://fortune/", Map.class);
  return (String) map.get("fortune");
}
```



```java
private final FortuneAPI fortuneAPI;

public FortuneServiceClient(FortuneAPI fortuneAPI) {
  this.fortuneAPI = fortuneAPI;
}

@HystrixCommand(fallbackMethod = "defaultFortune")
String getFortune() {
  Map map = fortuneAPI.getFortune();
  return (String) map.get("fortune");
}
```

Configuration as a Service

Spring Config Server

Spring Application Configuration

Traditionally:

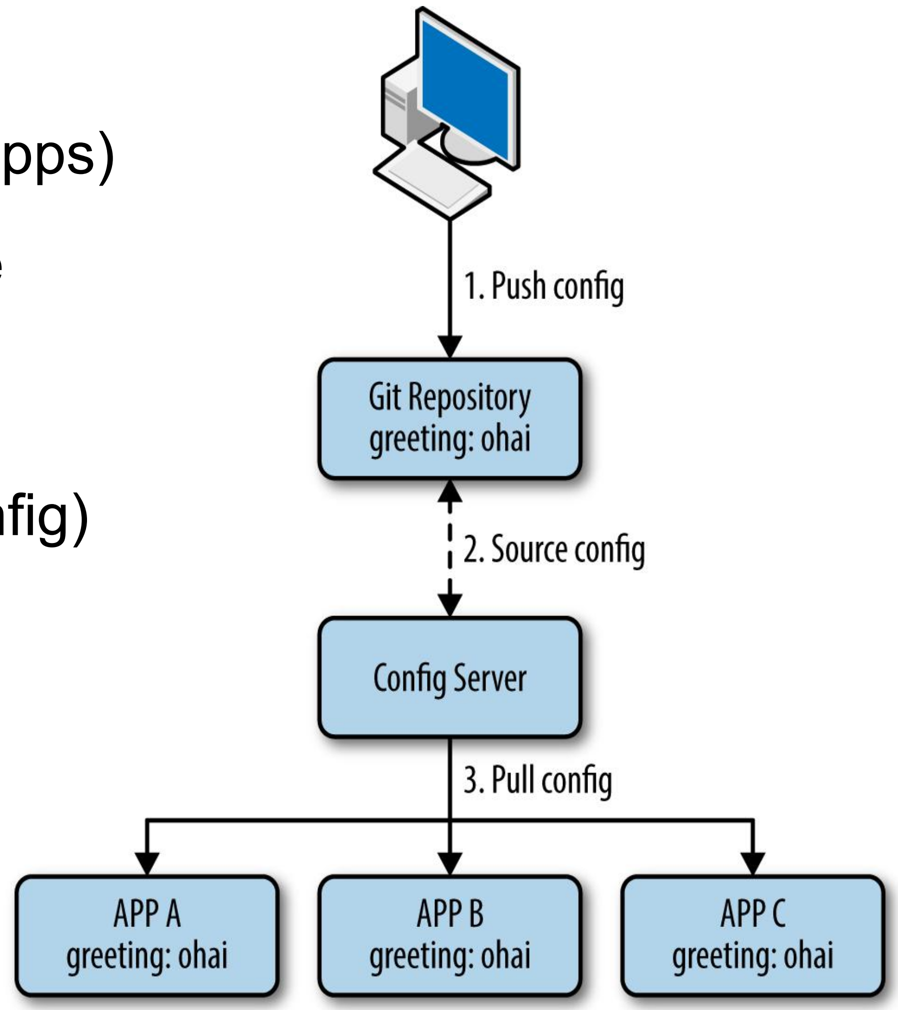
- configuration is stored with the application, and fetched from the classpath
- configuration in a .properties file under src/main/resources

Evolution:

- YAML files (.yml)
- Java system properties
- Environment variables

Spring Config Server: Concepts

- Externalized config (outside apps)
- Centralized config for multiple services & environments
- Config as a service (REST endpoints for reading app config)



Backends

- Supports multiple types of backends:
  - Git
  - Subversion
  - HashiCorp Vault
  - File System
  - JDBC

- Also supports a composite of backends

Backend File Naming

- Default Pattern:

  {application}-{profile}.[properties|yml]

- Example:

  spring.application.name: *greeting*
  spring.profiles.active: *qa*

  Configuration stored in a file: ***greeting-qa.yml***
  (or ***greeting-qa.properties***)

NOTE: The pattern can be customized via a configuration property named searchPaths

HTTP Service Endpoints

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

/{application}-{profile}.properties

/{label}/{application}-{profile}.properties

- *With the Git backend, {label} maps to a branch name.*
- *{label} is optional, and if not specified, defaults to master*

Example

spring.application.name: greeting

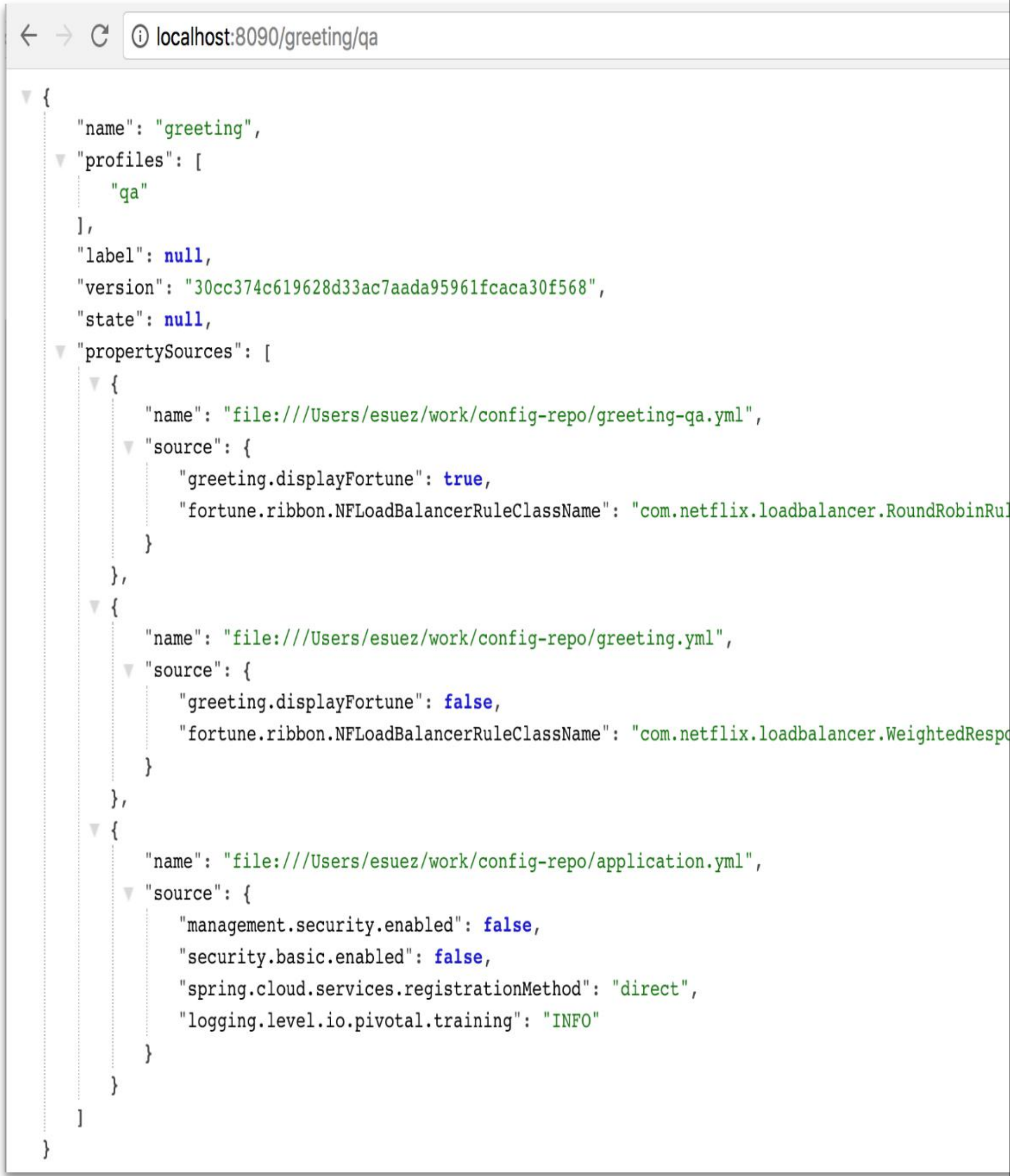
spring.profiles.active: qa

Endpoint ➔ http://{host}:{port}/greeting/qa

If any of these files exist, their config are returned as one JSON response:

> *application.yml (or .properties)*
> *application-qa.yml*
> *greeting.yml*
> *greeting-qa.yml*

Properties in more specifically-named files override those in more general file



localhost:8090/greeting/qa

```
{
    "name": "greeting",
    "profiles": [
        "qa"
    ],
    "label": null,
    "version": "30cc374c619628d33ac7aada95961fcaca30f568",
    "state": null,
    "propertySources": [
        {
            "name": "file:///Users/esuez/work/config-repo/greeting-qa.yml",
            "source": {
                "greeting.displayFortune": true,
                "fortune.ribbon.NFLoadBalancerRuleClassName": "com.netflix.loadbalancer.RoundRobinRul
            }
        },
        {
            "name": "file:///Users/esuez/work/config-repo/greeting.yml",
            "source": {
                "greeting.displayFortune": false,
                "fortune.ribbon.NFLoadBalancerRuleClassName": "com.netflix.loadbalancer.WeightedRespo
            }
        },
        {
            "name": "file:///Users/esuez/work/config-repo/application.yml",
            "source": {
                "management.security.enabled": false,
                "security.basic.enabled": false,
                "spring.cloud.services.registrationMethod": "direct",
                "logging.level.io.pivotal.training": "INFO"
            }
        }
    ]
}
```

Setting up the Server

- Add dependency: spring-cloud-config-server
- Annotate Spring Boot app class with @EnableConfigServer
- Example configuration of git backend to a public repository: *application.properties*:

```
spring.cloud.config.server.git.uri=https://github.com/{username}/config-repo.git
```

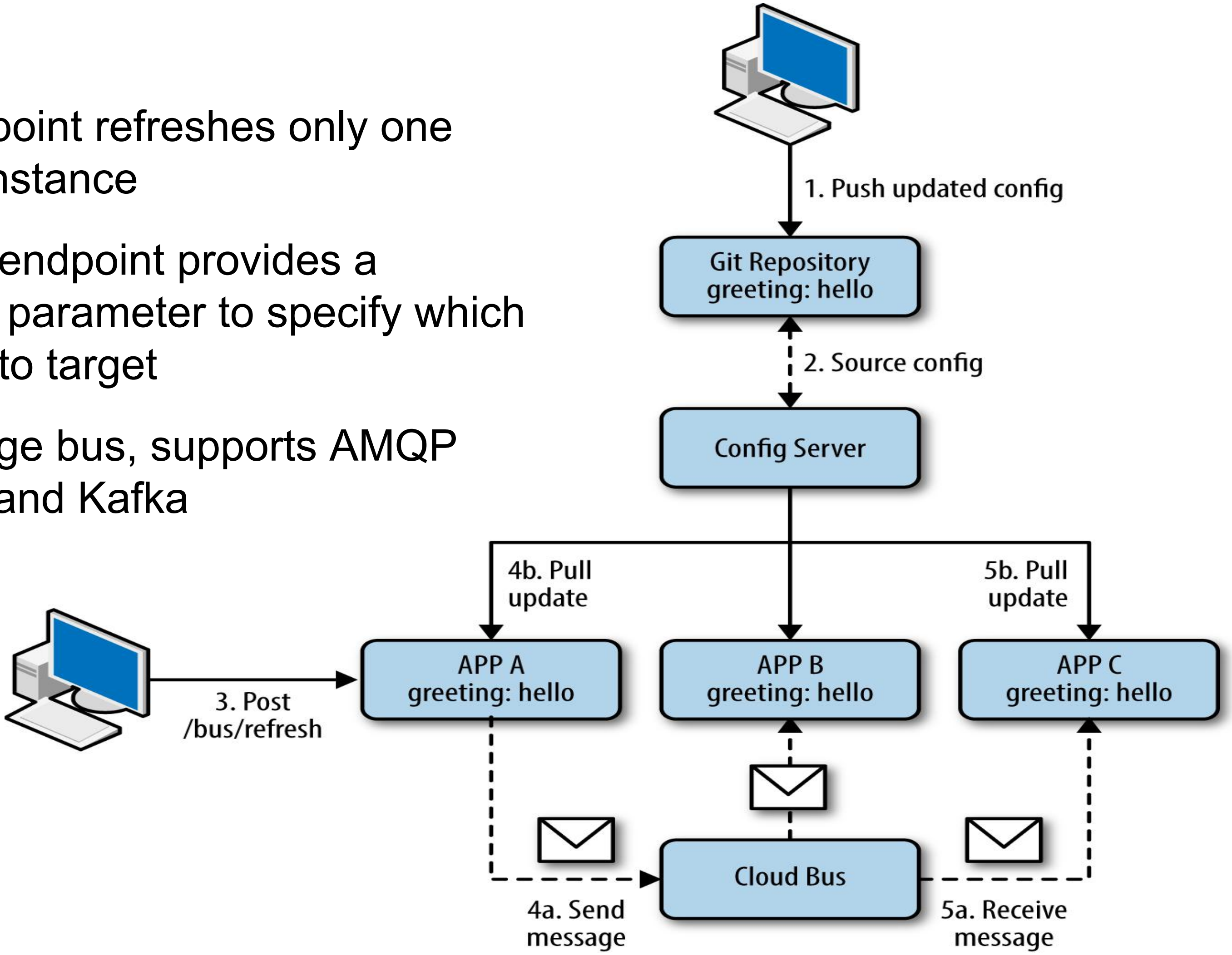
Many more configuration options exist for the server (consult project reference manual).

Configuring Clients

- Add dependency: spring-cloud-starter-config
- Set spring.cloud.config.uri for location of config server
- Set spring.application.name in *bootstrap.yml*,
  not *application.yml*
- Alternatively, set cloud.config.discovery.enabled
  to lookup Config Server via Eureka
  (if config server is registered with Eureka)

Config Server + Spring Cloud Bus

- /refresh endpoint refreshes only one application instance
- /bus/refresh endpoint provides a "destination" parameter to specify which applications to target
- Uses message bus, supports AMQP (RabbitMQ) and Kafka

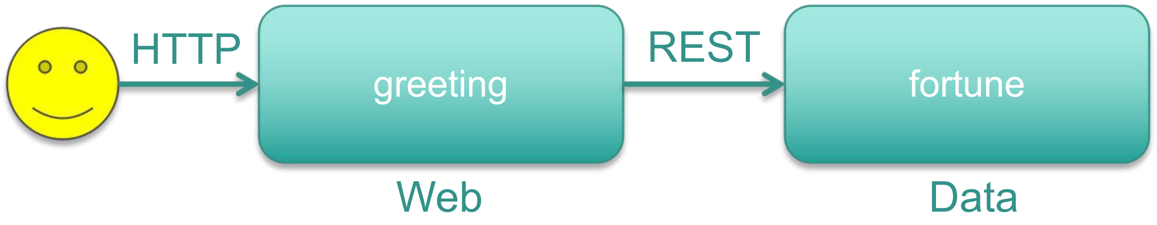


Benefits of Config Server

- All configuration is available in one place
- Separation of app dev lifecycle from configuration lifecycle
- Re-configure aspects of running apps without downtime (e.g. log level, feature toggles)
- Supports encryption of sensitive properties using various mechanisms (symmetric encryption, asymmetric key pair)
- Choice of Git backend provides complete configuration auditability

Lab / Demo

Spring Config Server



Pivotal®

Transforming How The World Builds Software

Feedback @ http://tinyurl.com/apj-eval

(course: Customized Session Not Shown in the Above Listing)