Load Balancing with Netflix Ribbon

Spring Cloud

- **Spring Cloud Contract**: supports and facilitates contract testing
- **Spring Cloud Netflix**: umbrella project offering a number of Netflix services and libraries adapted for Spring:
  - Hystrix, Eureka, **Ribbon**, Feign, Zuul

- **Spring Cloud Config Server**: configuration as a service
- **Spring Cloud Sleuth**: distributed tracing

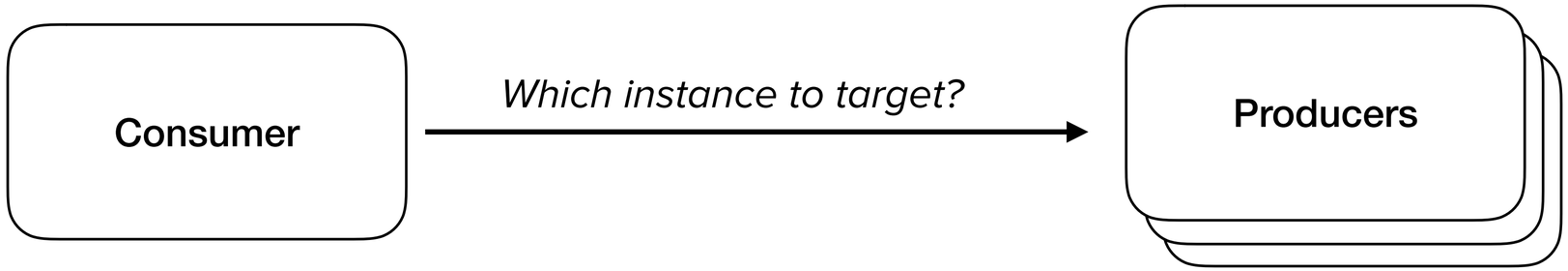
Traditional Load Balancing

- Traditionally, load balancing is performed by a dedicated appliance: either an F5 or a software component such as HAProxy
- Configured manually
- Entry point for HTTP requests from end users. i.e. Public-facing
- Fronts monolithic server instances

Load Balancing in a Microservice Architecture

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

Service Instances are scaled out

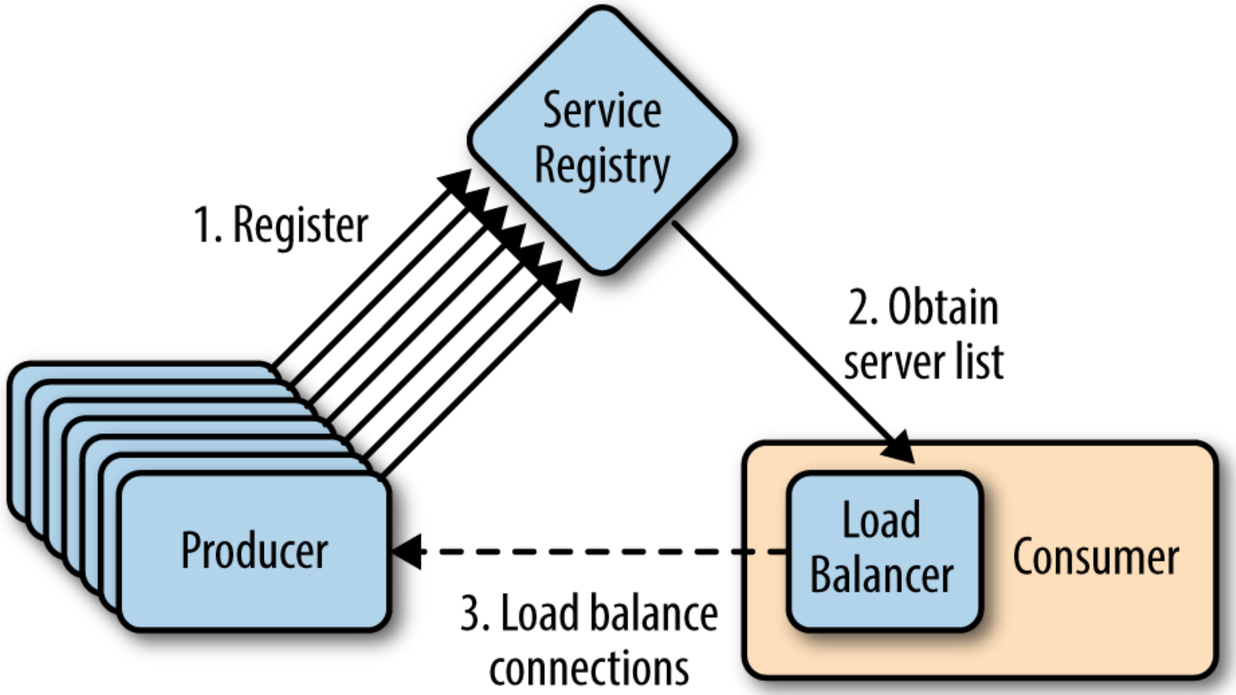


A eureka lookup yields multiple service instances for a given service name

Netflix Ribbon

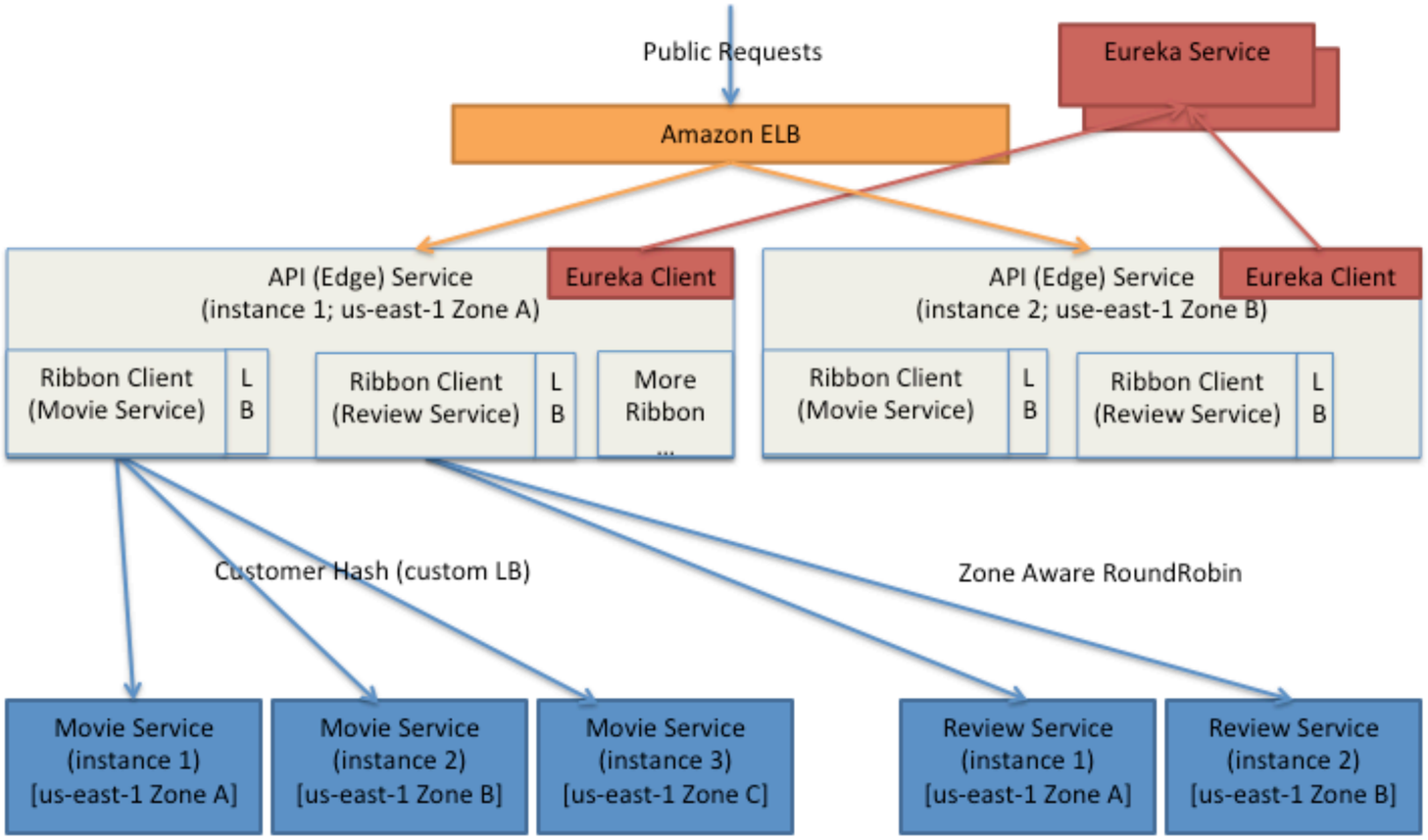
- A library that implements load balancing algorithms “out of the box”
- Added as a dependency, runs in-process in the consumer (caller)
- Automatically integrates with eureka to get the list of urls to load-balance across for each instance of a given service
- Configurable: choice of load balancing algorithms

Works in Concert with Eureka



Ribbon 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



Components of a Ribbon Load Balancer

- Rule - a logic component to determine which server to return from a list
- Ping - a component running in background to ensure liveness of servers
- ServerList - this can be static or dynamic. If it is dynamic (as used by DynamicServerListLoadBalancer), a background thread will refresh and filter the list at certain interval

Load Balancing Rule Options

- RoundRobinRule
- WeightedResponseTimeRule
- RandomRule
- BestAvailableRule
- AvailabilityFilteringRule

See: 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)	*Availability--FilteringRule*
myclient.ribbon.NFLoadBalancerPingClassName Strategy for pinging servers	*NoOpPing*
myclient.ribbon.MaxAutoRetriesNextServer Max number of next servers to retry (excluding the first server)	1

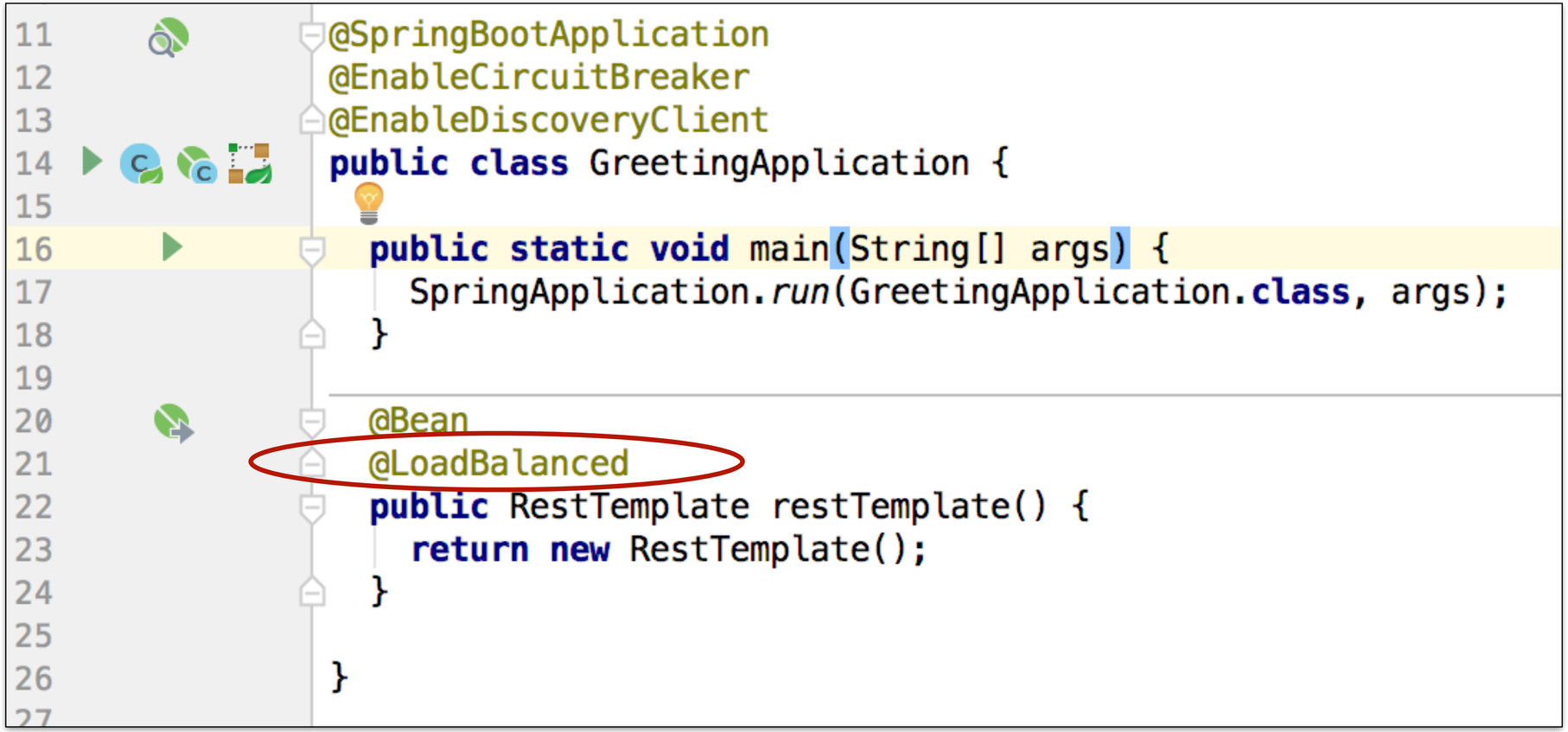
See: https://github.com/Netflix/ribbon/wiki/Getting-Started

Ribbon Load Balancing Example

```
20      private final LoadBalancerClient loadBalancerClient;
21
22      public FortuneServiceClient(RestTemplate restTemplate, LoadBalancerClient loadBalancerClient) {
23        this.restTemplate = restTemplate;
24        this.loadBalancerClient = loadBalancerClient;
25      }
26
27      @HystrixCommand(fallbackMethod = "defaultFortune")
28      String getFortune() {
29        String fortuneUrl = lookupUrlFor( appName: "FORTUNE");
30        Map map = restTemplate.getForObject(fortuneUrl, Map.class);
31        return (String) map.get("fortune");
32      }
33
34      private String lookupUrlFor(String appName) {
35        ServiceInstance instance = loadBalancerClient.choose(appName);
36        return String.format("http://%s:%s/", instance.getHost(), instance.getPort());
37      }
38
```

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

Alternative: @LoadBalanced RestTemplate



```
11  @SpringBootApplication
12  @EnableCircuitBreaker
13  @EnableDiscoveryClient
14  public class GreetingApplication {
15
16    public static void main(String[] args) {
17      SpringApplication.run(GreetingApplication.class, args);
18    }
19
20    @Bean
21    @LoadBalanced
22    public RestTemplate restTemplate() {
23      return new RestTemplate();
24    }
25
26  }
27
```

RestTemplate Usage

```
16      private final RestTemplate restTemplate;
17
18      public FortuneServiceClient(RestTemplate restTemplate) {
19        this.restTemplate = restTemplate;
20      }
21
22      @HystrixCommand(fallbackMethod = "defaultFortune")
23      String getFortune() {
24        Map map = restTemplate.getForObject( url: "http://fortune/", Map.class);
25        return (String) map.get("fortune");
26      }
27
```

- 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)