Service Discovery with Netflix Eureka

Objectives

Key objectives of this chapter

- Service Discovery in Microservices
- Netflix Eureka
- Eureka Server
- Eureka Client
- Eureka and the AWS Ecosystem

1.1 Service Discovery in Microservices

- In a dynamic environment the collection of service endpoints may change frequently due to:
  - Service failure
  - Auto-scaling
  - Normal service lifecycle (updates, maintenance, retirement)
- At *cloud scale* service discovery must be
  - Highly available
  - Fault tolerant
  - Low latency

1.2 Load Balancing in Microservices

- Microservices are *fine-grained*
- Individual services may be scaled *elastically*
- Traditional load balancers are not well-suited to a microservices environment
  - Configured with known IP addresses and/or hostname

1.3 Netflix Eureka

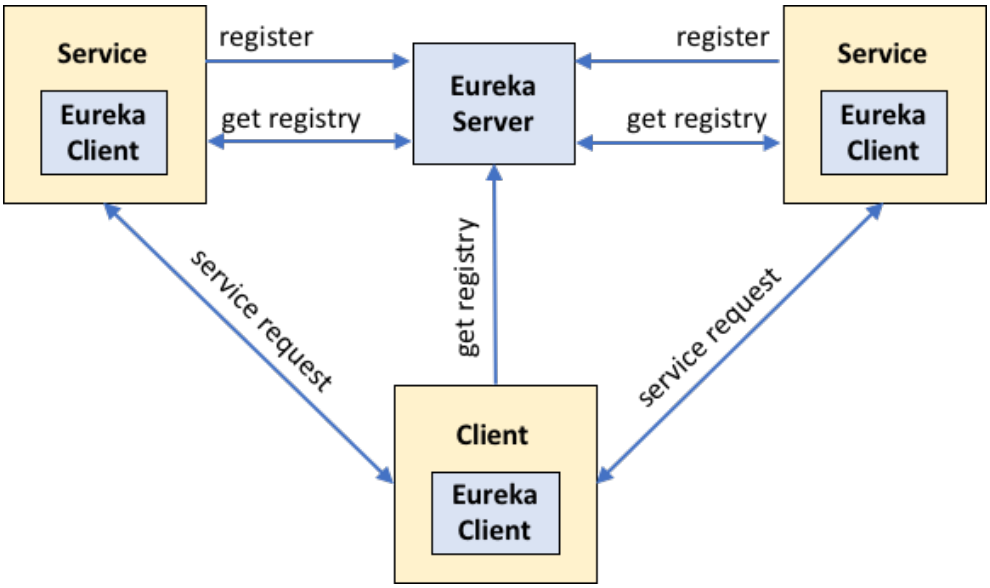
- Developed by Netflix
- Open source service discovery and load balance solution
  - Internally Netflix wraps Eureka with a proprietary load balance framework that provides more sophisticated load-balancing
- Designed to work in Amazon's AWS cloud
  - May be deployed outside of AWS
- Supports cloud scale applications
- Used internally by microservice applications
  - Not designed for *edge* or client-facing service discovery or load balance

Note

Netflix's internal load balance solution does more than *round-robin* requests. The Netflix solution takes service and network state (and more) into consideration in routing requests.

Even though Amazon and Netflix compete in the streaming video space, Netflix is deployed on Amazon's AWS cloud.

1.4 Eureka Architecture



Note

In this drawing the services and the client all get the registry from Eureka. Services may be clients of other services and clients may be services as well.

Not all communications are shown.

1.5 Communications in Eureka

- Register
  - Clients/Services register with Eureka
- Renew
  - Heartbeat
  - Every 30s
  - Servers that fail to send three consecutive heartbeats (90s) are

removed

- Get Registry
  - Clients/Services get registry info and cache it locally
  - Registry deltas retrieved at client every 30s
- Cancel
  - Client informs server to remove it from the registry

1.6 Time Lag

- Heartbeats and Registry updates are sent at a rate of 0.33 Hz (every 30 seconds)
- Changes may take up to 2 minutes to propagate within the system

0:00	Service fails
0:30	Eureka server misses 3 heartbeats
1:30	Server cached registry expires
2:00	Clients receive registry updates

Note

This is the worst case scenario for time lag.

1.7 Eureka Deployment

- Eureka should not be a *single point of failure*
  - Multiple Eureka servers should be deployed in production environments
- Peer Eureka servers communicate the same way that Eureka clients and servers do
- At startup Eureka servers attempt to get current registry info from existing servers
  - If necessary the server attempts to retrieve registry info from multiple peers
- A server that cannot communicate with its peers will try to preserve registry information longer than usual
  - In this case clients may receive outdated registry copies
  - Clients must be able to gracefully handle receiving info for *dead* nodes from Eureka

Note

In case Eureka sends outdated information, clients should set short service request timeouts and failover quickly.

1.8 Peer Communication Failure between Servers

- When a server stops receiving heartbeats from other servers
  - The server enters *self-preservation* mode
  - Preserves current registry state
  - Registry info may be *stale*

- If a service registers with a server in *self-preservation* mode
  - Only clients connected to this server node will acknowledge the new registration
  - Different clients may have different registries
- When peer communication resumes the servers will normalize and synchronize their registries

1.9 Eureka Server Configuration

- SpringBootApplication
- Add the @EnableEurekaServer annotation
- Configure application.properties
  - Specify listen port
  - Disable *normal* eureka client registration
  - Disable *normal* registry fetch

```
server.port=9999
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

1.10 Eureka Client/Service

- SpringBootApplication
- Add the @EnableDiscoveryClient annotation
- Use the @Autowired annotation to inject the DiscoveryClient

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

@RestController
class ServiceInstanceRestController {
  @Autowired
  private DiscoveryClient discoveryClient;
```

1.11 Eureka Client Properties

- Configured in bootstrap.properties
  - This is the first property file loaded during startup
- Specify application name

```
spring.application.name=foo-client
```

1.12 Spring Cloud DiscoveryClient Interface

- Access to the local copy of the registry

- Methods
  - String description()
    - Returns the name/description of the implementation
  - List<ServiceInstance> getInstances(String serviceId)
    - Returns a list of ServiceInstances containing the information to access *each* instance of the specified service
  - List<String> getServices()
    - Returns a list of service names
    - Names will only be in the list once even if multiple instances are deployed

1.13 ServiceInstance JSON


```
[{"host":"192.168.10.102","port":8080,"metadata":
{"management.port":"8080"},"uri":"http://192.168.10.102:8080","secure
":false,"serviceId":"FOO-SERVICE","instanceInfo":
{"instanceId":"192.168.10.102:foo-service:8080","app":"Foo-
SERVICE","appGroupName":null,"ipAddr":"192.168.10.102","sid":"na","ho
mePageUrl":"http://192.168.10.102:8080/","statusPageUrl":"http://192.
168.10.102:8080/info","healthCheckUrl":"http://192.168.10.102:8080/he
alth","secureHealthCheckUrl":null,"vipAddress":"foo-
service","secureVipAddress":"foo-service", "countryId":1,
"dataCenterInfo"{"@class":"com.netflix.appinfo.InstanceInfo$DefaultDa
taCenterInfo","name":"MyOwn"},"hostName":"192.168.10.102","status":"U
P","leaseInfo":
{"renewalIntervalInSecs":30,"durationInSecs":90,"registrationTimestam
p":1518591759032,"lastRenewalTimestamp":1518592028960,"evictionTimest
amp":0,"serviceUpTimestamp":1518591758520},"isCoordinatingDiscoverySe
rver":false,"metadata":
{"management.port":"8080"},"lastUpdatedTimestamp":1518591759032,"last
DirtyTimestamp":1518591758443,"actionType":"ADDED","asgName":null,"ov
erriddenStatus":"UNKNOWN"}}]
```


Note

This is a lot of information but if you examine it closely you will see some important information

- Directory info: host port URI
- renewalIntervalInSecs: rate at which heartbeats are sent
- durationInSecs: length of time the serve will wait before assuming a service is offline

1.14 ServiceInstance Interface

- ServiceInstance has methods to extract service information
- Information accessed through ServiceInstance is used to access the service
  - Host
  - Port
  - isSecure – HTTPS/TLS
  - URI
  - Metadata
  - etc

1.15 What about Services

- Services are also clients of Eureka and are configured as shown on the precious slides
- In a microservices environment individual services are often clients of other services

1.16 Eureka and the AWS Ecosystem

- Eureka was designed to run in Amazon AWS
- Eureka clusters are deployed within a single AWS *region*
- Multi-region deployment
  - Each region has its own Eureka cluster
  - Eureka does not replicate registry information across regions
- Each *zone* within a region should have at least on Eureka server
- In the case of server failure, clients can get registry updates across zones

1.17 Summary

In this module we examined:

- Service Discovery in Microservices
- Netflix Eureka
- Eureka Server
- Eureka Client
- Eureka Service Registration
- Eureka and the AWS Ecosystem