Netflix Eureka

Service Discovery

- Eureka is the Netflix Service Discovery Server and Client.
- The server can be configured and deployed to be highly available, with each server replicating state about the registered services to the others.

Authenticating with the Eureka Server

HTTP basic authentication will be automatically added to the eureka client if one of the eureka.client.serviceUrl.defaultZone URLs has credentials embedded in it http://user:password@localhost:8761/eureka)

Status Page and Health Indicator

The status page and health indicators for a Eureka instance default to "/info" and "/health" respectively, which are the default locations of useful endpoints in a Spring Boot Actuator application.

However, we can change the context-path as below:

```
eureka:
  instance:
    statusPageUrlPath: ${management.context-path}/info
    healthCheckUrlPath: ${management.context-path}/health
```

Eureka’s Health Checks

By default, Eureka uses the client heartbeat to determine if a client is up. Unless specified otherwise the Discovery Client will not propagate the current health check status of the application per the Spring Boot Actuator. Which means that after successful registration Eureka will always announce that the application is in 'UP' state.

This behaviour can be altered by enabling Eureka health checks, which results in propagating application status to Eureka.

application.yml

```
eureka:
  client:
    healthcheck:
      enabled: true
```

eureka.instance.leaseRenewalIntervalInSeconds

The initial registration is actually trigger by the first heartbeat: the client tries to send the first heartbeat and receives a "not found" answer from the server which means it doesn't know the instance. The client then immediately attempts to register the instance. This process only happens 30s (eureka.instance.leaseRenewalIntervalInSeconds) after startup

We can always speed-up the initial registration process by lowering the value of eureka.client.initialInstanceInfoReplicationIntervalSeconds. This parameter controls the initial delay before the client transmits changes made to the InstanceInfo (like the UP/DOWN status).

eureka.instance.leaseExpirationDurationInSeconds

If we want to detect "dead" instances quicker, we can set eureka.instance.leaseExpirationDurationInSeconds parameter.

The value is set to 90s by default, which means a lease is expired after 3 consecutive missing heartbeats.

The SELF PRESERVATION MODE is design to avoid poor network connectivity failure. Connectivity between Eureka instance A and B is good, but B is failed to renew its lease to Eureka server in a short period due to connectivity hiccups, at this time Eureka server can't simply just kick out instance B. If it does, instance A will not get available registered service from Eureka server despite B is available. So this is the purpose of SELF PRESERVATION MODE, and it's better to turn it on.

```
eureka:
    server:
        enableSelfPreservation: true
```

Using the EurekaClient

Spring DI can be used to inject the discovery client to access the Eureka service registry com.netflix.discovery.EurekaClient

```
@Autowired
private EurekaClient discoveryClient;

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

(OR)

We can also use org.springframework.cloud.client.discovery.DiscoveryClient which provides a simple API for discovery clients that is not specific to Netflix.

```
@Autowired
private DiscoveryClient discoveryClient;
```

Service Discovery: Eureka Server

Include Eureka Server

To include Eureka Server in your project use the starter with group org.springframework.cloud and artifact id spring-cloud-starter-netflix-eureka-server.

Run a Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder(Application.class).web(true).run(args);
    }

}
```

The server has a home page with a UI, and HTTP API endpoints per the normal Eureka functionality under /eureka/*.

High Availability, Zones and Regions

The Eureka server does not have a backend store, but the service instances in the registry all have to send heartbeats to keep their registrations up to date (in memory).

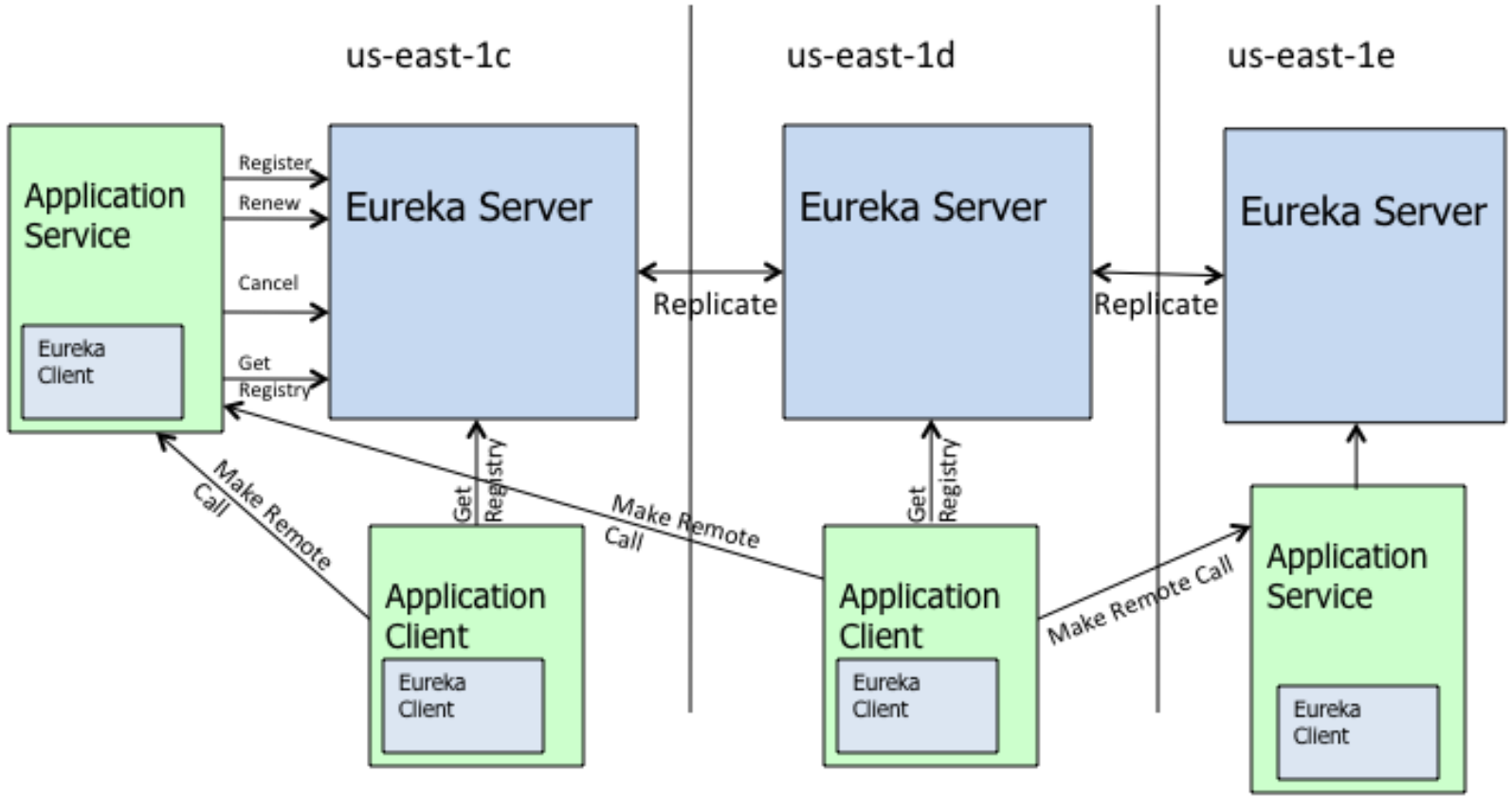
Clients also have an in-memory cache of eureka registrations (so they don’t have to go to the registry for every single request to a service).

Standalone Mode

The combination of the two caches (client and server) and the heartbeats make a standalone Eureka server fairly resilient to failure,

```
server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/
```


us-east-1c
us-east-1d
us-east-1e
Application Service
Eureka Client
Register
Renew
Cancel
Get Registry
Eureka Server
Eureka Server
Eureka Server
Replicate
Replicate
Make Remote Call
Get Registry
Make Remote Call
Get Registry
Application Client
Eureka Client
Application Client
Eureka Client
Make Remote Call
Application Service
Eureka Client

Peer Awareness

Eureka can be made even more resilient and available by running multiple instances and asking them to register with each other

application.yml (Two Peer Aware Eureka Servers).

```
---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: peer1
  client:
    serviceUrl:
      defaultZone: http://peer2/eureka/

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: peer2
  client:
    serviceUrl:
      defaultZone: http://peer1/eureka/
```

Prefer IP Address

Set eureka.instance.preferIpAddress to true and when the application registers with eureka, it will use its IP Address rather than its hostname.