Microservice Registration with Spring cloud using Netflix Eureka

Netflix component **Eureka** for service registry and discovery.

When we start a project, we usually have all the configurations in the properties file.  
As more and more services are developed and deployed, adding and modifying these properties become more complex. Some services might go down, while some the location might change. This manual changing of properties may create issues.  
Eureka Service Registration and Discovery helps in such scenarios. As all services are registered to the Eureka server and lookup done by calling the Eureka Server, any change in service locations need not be handled and is taken care of.

Analogy – Travel booking in Capgmeini.

Amazon Seller and Buyer

Ex. Employee Producer, Employee Consumer, Eureka Server

|  |  |
| --- | --- |
| Name | Description |
| spring.application.name | Unique name for a Eureka server service. |
| eureka.client.serviceUrl.defaultZone | It consults with other Eureka servers to sync the service registry. As it is in standalone mode, I am giving the local server address. |
| server.port | In which port the server will be bound. |
| eureka.client.register-with-eureka | This determines if this server registers itself as a client; as I said earlier, the Eureka server is also acting as a client so that it can sync the registry. The value being false means it prevents itself from acting as a client. |
| eureka.client.fetch-registry | Does not register itself in the service registry. |

## **Eureka Self-Preservation**

What is self preservation in Eureka?

**Self**-**preservation** is a feature where **Eureka** servers stop expiring instances from the registry when they do not receive heartbeats (from peers and client microservices) beyond a certain threshold.

During the start-up, the clients trigger a **REST call with the Eureka server to self-register to the server's instance registry.** When a graceful shutdown occurs after use, the clients trigger another REST call so that the server can wipe out all the data related to the caller.

To handle ungraceful client shutdowns the server expects heartbeats from the client at specific intervals. This is called *renewal*. If the server stops receiving the heartbeat for a specified duration, then it will start evicting the stale instances.

The mechanism that**stops evicting the instances when the heartbeats are below the expected threshold**is called *self-preservation*. This might happen in the case of a poor network partition, where the instances are still up, but just can't be reached for a moment or in the case of an abrupt client shutdown.

And when the server activates self-preservation mode it holds the instance eviction until the renewal rate is back above the expected threshold.

Configuration Details:

* *eureka.server.enable-self-preservation*: Configuration for disabling self-preservation – the default value is *true*
* *eureka.server.expected-client-renewal-interval-seconds*: The server expects client heartbeats at an interval configured with this property – the default value is *30*
* *eureka.instance.lease-expiration-duration-in-seconds*: Indicates the time in seconds that the Eureka server waits since it received the last heartbeat from a client before it can remove that client from its registry – the default value is *90*
* *eureka.server.eviction-interval-timer-in-ms*: This property tells the Eureka server to run a job at this frequency to evict the expired clients – the default value is *60* *seconds*
* *eureka.server.renewal-percent-threshold*: Based on this property, the server calculates the expected heartbeats per minute from all the registered clients – the default value is *0.85*
* *eureka.server.renewal-threshold-update-interval-ms*: This property tells the Eureka server to run a job at this frequency to calculate the expected heartbeats from all the registered clients at this minute – the default value is *15 minutes*

In most cases, the default configuration is sufficient. But for specific requirements, we might want to change these configurations. **Utmost care needs to be taken** in those cases to avoid unexpected consequences like **wrong renew threshold calculation or delayed self-preservation mode activation**.

**Configurations (with defaults)**

eureka.instance.lease-renewal-interval-in-seconds = 30

Indicates the frequency the client sends heartbeats to server to indicate that it is still alive. It’s not advisable to change this value since self-preservation assumes that heartbeats are always received at intervals of 30 seconds.

eureka.instance.lease-expiration-duration-in-seconds = 90

Indicates the duration the server waits since it received the last heartbeat before it can evict an instance from its registry. This value should be greater than lease-renewal-interval-in-seconds. Setting this value too long impacts the precision of actual heartbeats per minute calculation, since the liveliness of the registry is dependent on this value. Setting this value too small could make the system intolerable to temporary network glitches.

eureka.server.eviction-interval-timer-in-ms = 60 \* 1000

A scheduler is run at this frequency which will evict instances from the registry if the lease of the instances are expired as configured by lease-expiration-duration-in-seconds. Setting this value too long will delay the system entering into self-preservation mode.

eureka.server.renewal-percent-threshold = 0.85

This value is used to calculate the expected heartbeats per minute

eureka.server.renewal-threshold-update-interval-ms = 15 \* 60 \* 1000

A scheduler is run at this frequency which calculates the expected heartbeats per minute

eureka.server.enable-self-preservation = true

self-preservation can be disabled if required.

### Making sense of configurations

Eureka server enters self-preservation mode if the actual number of heartbeats in last minute is less than the expected number of heartbeats per minute.

#### Expected number of heartbeats per minute

We can see the means of calculating expected number of heartbeats per minute threshold. Netflix code assumes that heartbeats are always received at intervals of 30 seconds for this calculation.

Suppose the number of registered application instances at some point in time is N and the configured renewal-percent-threshold is 0.85.

* Number of heartbeats expected from one instance / min = 2
* Number of heartbeats expected from N instances / min = 2 \* N
* Expected minimum heartbeats / min = **2 \* N \* 0.85**

Since N is a variable, **2 \* N \* 0.85**is calculated in every 15 minutes by default (or based on renewal-threshold-update-interval-ms).

### Prefer IP Address

In some cases, it is preferable for Eureka to advertise the IP Adresses of services rather than the hostname. Set eureka.instance.preferIpAddress to true and when the application registers with eureka, it will use its IP Address rather than its hostname.

Eureka.instance.preferIpAddress=true