Created by:
Tyree Solomon-Phillips
Collin Klenke

Eureka Properties

Basic Architecture

1.) **eureka.client.eurekaServiceUrlPollIntervalSeconds**
   a.) Used to configure the interval or amount of time in between property changes that the client might have made.
   b.) Used by the client
2.) **org.springframework.cloud.netflix.eureka.server.EurekaServerConfigBean**
   a.) For the Eureka server config. This implements **com.netflix.eureka.EurekaServerConfig**. All properties are prefixed by eureka.server.
   b.) Used by client config
3.) **org.springframework.cloud.netflix.eureka.EurekaInstanceConfigBean**
   a.) This indirectlyimplements com.netflix.appinfo.EurekaInstanceConfig. All properties are prefixed by eureka.instance.
   b.) Used by Eureka instance config

Eureka Instance and Client

1.) **eureka.instance.instanceId** is an instances identifier for Eureka or **eureka.instance.metadataMap.instanceId** if the first one is not present.
   a.) The instances find each other using **spring.application.name**
   b.) Used by the instance or client
2.) **registerWithEureka**
   a.) if set to “true” an instance registers with a Eureka server using a given URL
3.) **eureka.instance.leaseRenewalIntervalInSeconds**
   a.) used to configure the heartbeat intervals for Eureka clients (by default they are set to 30 seconds). If the server does not receive a heartbeat it waits 90 seconds (the wait time can be configured by **eureka.instance.leaseExpirationDurationInSeconds**.
4.) **eureka.client.heartbeatExecutorExponentialBackOffBound**

   a.) Sending a heartbeat is a asynchronous task and if that operation fails it backs off exponentially by a factor of two until a maximum delay of
   this value * eureka.instance.leaseRenewalIntervalInSeconds
   is reached using the above property (there is no limit to the number of retries for registering with Eureka.
5.) **com.netflix.appinfo.InstanceInfo**

a.) Used to update the instance information to the Eureka server. This is sent to the Eureka server at regular intervals, starting at 40 seconds after startup, which is configured by **eureka.client.initialInstanceInfoReplicationIntervalSeconds** and then every 30 seconds after startup, configured by **eureka.client.instanceInfoReplicationIntervalSeconds**

6.) **eureka.client.fetchRegistry**
   a.) if this is set to “true” the client fetches the Eureka server registry at startup and caches it locally. After that point, it will only fetch the delta (this can be turned off by setting **eureka.client.shouldDisableDelta** to false.
   b.) The registry fetch is an asynchronous task scheduled every 30 seconds, which can be configured by using **eureka.client.registryFetchIntervalSeconds.** If the fetch task fails it backs off exponentially by a factor of two until a maximum delay of **eureka.client.registryFetchIntervalSeconds * eureka.client.cacheRefreshExecutorExponentialBackOffBound** is reached (there is no limit to the number of retries for fetching the registry information.
7.) **ribbon.ServerListRefreshInterval**
   a.) used to refresh the Ribbon load balancer. Ribbon gets its information from the Eureka client and maintains it’s own local cache to avoid calling the client for every request (this cache is refreshed every 30 seconds).
8.) **com.netflix.discovery.DiscoveryClient**
   a.) Used to schedule client tasks, which Spring Cloud extends in **org.springframework.cloud.netflix.eureka.CloudEurekaClient**

Eureka Server

1.) **registerWithEureka**
   a.) used by Eureka server in peer aware mode and acts as a Eureka client and registers to a peer on a given URL.
2.) **eureka.server.numberRegistrySyncRetries**
   a.) When the eureka server starts up it tries to fetch all the registry information from the peer eureka nodes. This operation is retried 5 times for each peer. If for some reason this operation fails, the server does not allow clients to get the registry information for 5 min (this can be configured by **eureka.server.getWaitTimeInMsWhenSyncEmpty**
3.) eureka.server.enableSelfPreservation
   a.) Eureka has a concept called self-preservation and can be turned on or off by using this property.
   b.) From the Netflix Wiki: *When the Eureka server comes up, it tries to get all of the instance registry information from a neighboring node. If there is a problem getting the information from a node, the server tries all of the peers before it gives up. If the server is able to successfully get all of the instances, it sets the renewal threshold that it should be receiving based on that information. If any time, the*

renewals falls below the percent configured for that value, the server stops expiring instances to protect the current instance registry information.

Regions and Availability Zones

1.) **org.springframework.cloud.netflix.eureka.server.EurekaServerBootstrap**
   a.) Used to display the environment and data center values, which are set to test and default using **com.netflix.config.ConfigurationManager.**
   b.) The Eureka client prefers the same zone by default, which can be configured by **eureka.client.preferSameZone** from **com.netflix.discovery.endpoint.EndpointUtils.getServiceUrlsFromDNS**

Extras

I. Why Does It Take So Long to Register an Instance with Eureka?

1. Client Registration
   a. The first heartbeat happens 30s (described earlier) after startup so the instance doesn't appear in the Eureka registry before this interval.
2. Server Response Cache
   a. The server maintains a response cache that is updated every 30s (configurable by eureka.server.responseCacheUpdateIntervalMs). So even if the instance is just registered, it won't appear in the result of a call to the /eureka/apps REST endpoint. However, the instance may appear on the Eureka Dashboard just after registration. This is because the Dashboard bypasses the response cache used by the REST API. If you know the instanceId, you can still get some details from Eureka about it by calling /eureka/apps/<appName>/<instanceId>. This endpoint doesn't make use of the response cache.
   b. So, it may take up to another 30s for other clients to discover the newly registered instance.
3. Client Cache Refresh
   a. Eureka client maintain a cache of the registry information. This cache is refreshed every 30s (described earlier). So, it may take another 30s before a client decides to refresh its local cache and discover other newly registered instances.
4. LoadBalancer Refresh
   a. The load balancer used by Ribbon gets its information from the local Eureka client. Ribbon also maintains a local cache to avoid calling the client for every request. This cache is refreshed every 30s (configurable by ribbon.ServerListRefreshInterval). So, it may take another 30s before Ribbon can make use of the newly registered instance.

b. In the end, it may take up to 2min before a newly registered instance starts receiving traffic from the other instances.

II. High Availability (HA)

- The HA strategy seems to be one main Eureka server (server1) with backup(s) (server2).
- A client is provided with a list of Eureka servers through config (or DNS or /etc/hosts)
- Client attempts to connect to server1; at this point, server2 is sitting idle.
- In case server1 is unavailable, the client tries the next one from the list.
- When server1 comes back online, client goes back to using server1.

Of course server1 and server2 can be ran in peer-aware mode, and their registries replicated. But that's orthogonal to client registration.