Service Registry

Ashok Koyi

Version 1.0.0-SNAPSHOT, 2019-03-18

Table of Contents

- Navigate to `service-registry` project
- Add `org.springframework.cloud:spring-cloud-starter-netflix-eureka-server` as a dependency as shown below

build.gradle

```
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
```

- Annotate `ServiceRegistryApplication` with `@EnableEurekaServer` as shown below

```
@SpringBootApplication
@EnableEurekaServer ①
public class ServiceRegistryApplication {
  public static void main(String[] args) {
    SpringApplication.run(ServiceRegistryApplication.class, args);
  }
}
```

① Enabling all components responsible for starting a service registry server using eureka

- Specify the port on which eureka server with the following configuration under `src/main/resources/application.yml`

application.yml

```
server:
  port: 8761 ①
eureka:
  client:
    register-with-eureka: false ②
    fetch-registry: false ③
```

① Port on which service registry server runs

② Since service registry server is running in standalone mode (no peer), we are asking not to register this with itself

③ Since service registry server is running in standalone mode (no peer), there is no point in fetching its own registry to know instances that are bound to a given `service-id`

- Now start `ServiceRegistryApplication` & observe that service registry server (eureka) is listening on port `8761`

Working with Service Registry Server APIs

- We can access the server APIs directly using the following endpoints

```
# Status of server
http -jv :8761/eureka/status

# Fetch all fetch all apps
http -jv :8761/eureka/apps
# Fetch all instances of a service id
http -jv :8761/eureka/apps/upstream-service
# Fetch instance info of an instance
http -jv :8761/eureka/apps/upstream-service/<instance id>
# Take an instance out of service by overriding status
http -v PUT :8761/eureka/apps/upstream-service/<instance
id>/status?value=OUT_OF_SERVICE
# Restore the old status of the instance
http -v DELETE :8761/eureka/apps/upstream-service/<instance id>/status

# Get instance info by instance id
http -jv :8761/eureka/instances/<instance id>
# Get all instance info by vip
http -jv :8761/eureka/vips/upstream-service
```

- By default, each service registry attempts to register with a peer to stay in HA mode and the default peer url is `http://localhost:8761/eureka`, which is itself under the service id `service-registry`. Since the same node cannot be a replica for itself, lets add the following changes to our configuration. With this configuration, we informed service registry not register itself as a client to itself

application.yml

```
spring:
  application:
    name: service-registry
server:
  port: 8761
eureka:
  server:
    response-cache-auto-expiration-in-seconds: 1 ①
    response-cache-update-interval-ms: 900 ②
  client:
    register-with-eureka: false
    fetch-registry: false
```

① Lets force the cache to be invalidated as we are in demo, not in prod. By default, in prod, cache refreshes every 3mins

② Lets keep cache update aswell in sync with expiration so that client will be aware of latest instance info

Peer mode: Service registry instance #1

application-service-registry1.yml

```
spring:
  profiles: service-registry1
  application:
    name: service-registry ①
server:
  port: 8761
eureka:
  instance:
    hostname: service-registry1 ②
  client:
    service-url:
      defaultZone: http://service-registry2:8762/eureka ③
    register-with-eureka: true
    fetch-registry: true
```

① Name with which to register with peer service registry. All nodes can use this to find other service registry instances

② Since we are running in the same node

③ Specifying the location of its peer. The peer will be used share instance info across zones

Now start `ServiceRegistryApplication` with `-Dspring.profiles.active=service-registry1` to start 1st instance of `service-registry` server on port `8761`

Peer mode: Service registry instance #2

application-service-registry2.yml

```
spring:
  profiles: service-registry2
  application:
    name: service-registry ①
server:
  port: 8762 ②
eureka:
  instance:
    hostname: service-registry2 ③
  client:
    service-url:
      defaultZone: http://service-registry1:8761/eureka ④
    register-with-eureka: true
    fetch-registry: true
```

① Even the peer will have the same name

② Avoiding port conflict on the same node by using a different port number

③ Avoid cookie conflict on the same node, useful in case security is enabled. Make sure `hosts` file has mapped `service-registry2` is mapped to `localhost`

④ Pointing to its peer

Now start `ServiceRegistryApplication` with `-Dspring.profiles.active=service-registry2` to 2nd instance of `service-registry` server on port `8762`

Now observe that each eureka server are peers to each other

Register micro service across all peers

- Create an application named `upstream-service`
- Make sure `build.gradle` contains

```
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-web' ①
  implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-
client' ②
}
```

① Adding all web related dependencies & autoconfiguration

② Adding service registry client as a dependency. This auto configures a `DiscoveryClient` that attempts to register with service registry server at `http://localhost:8761/eureka`

- Create `UpstreamServiceApplication` class in package `com.acme.upstream.service`

UpstreamServiceApplication.java

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

- Create `application.yml` under `src/main/resources`

application.yml

```
spring:
  application:
    name: upstream-service ①
```

① Default service id under which this service instance would be registered under within service registry

Start `UpstreamServiceApplication` an observe that this application is registered under the service id `upstream-service`