API Gateway

This project is a proof of concept (PoC) Spring API Gateway. It includes a rate limiter and a circuit breaker. I used the tutorials listed at the bottom of this document while coding this project.

All the code is in DemoApplication.java. We have two routes, one based on host, the other is based on path.

The host route has the circuit breaker which returns the success response, or a fallback message if things take too long.

Circuit Breaker

The circuit breaker implementation will return an error response if the response takes too long. It should handle service exceptions as well.

To test this, run the following from the command line. This will return a successful response from http://httpbin.org (httpbin).

```
curl --dump-header - --header 'Host: www.circuitbreaker.com' http://localhost:8080/delay/0
```

Response ...

```
HTTP/1.1 200 OK
Date: Wed, 06 Dec 2023 20:15:59 GMT
Content-Type: application/json
Content-Length: 487
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Forwarded": "proto=http;host=www.circuitbreaker.com;for=\"127.0.0.1:56110\"",
    "Host": "httpbin.org",
    "User-Agent": "curl/8.1.2",
    "X-Amzn-Trace-Id": "Root=1-6570d67f-451fcfa73cbc804f506e391e",
    "X-Forwarded-Host": "www.circuitbreaker.com"
  },
  "origin": "127.0.0.1, 69.243.41.76",
  "url": "http://www.circuitbreaker.com/delay/0"
}
```

Now run this to trigger the fallback response from the Api gateway. This does so by creating a 3 second delayed response from httpbin.

```
curl --dump-header - --header 'Host: www.circuitbreaker.com' http://localhost:8080/delay/3
```

Response ...

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8
Content-Length: 21

Something went wrong.
```

Rate Limiter

To run the rate limiter, you'll need Redis running. See the Spring with Redis reference below as a guide. Note that you don't want to follow that exactly. Stick to the tutortial instructions.

In summary ...

```
docker run -p 16379:6379 -d redis:6.0 redis-server --requirepass "mypass"
```

Note that I didn't use a password. Add this to the pom.xml file ...

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
</dependency>
```

And the application.properties file ...

```
spring.data.redis.database=0
spring.data.redis.host=localhost
spring.data.redis.port=16379
spring.data.redis.password=mypass
spring.data.redis.timeout=60000
```

The key to making this work is creating a custom KeyResolver. The one in the PoC, SimpleClientAddressResolver, is based on the client's IP address.

To test the rate limiter, run the following from the command line. This will return a successful response from httpbin.

`curl http://localhost:8080/get`

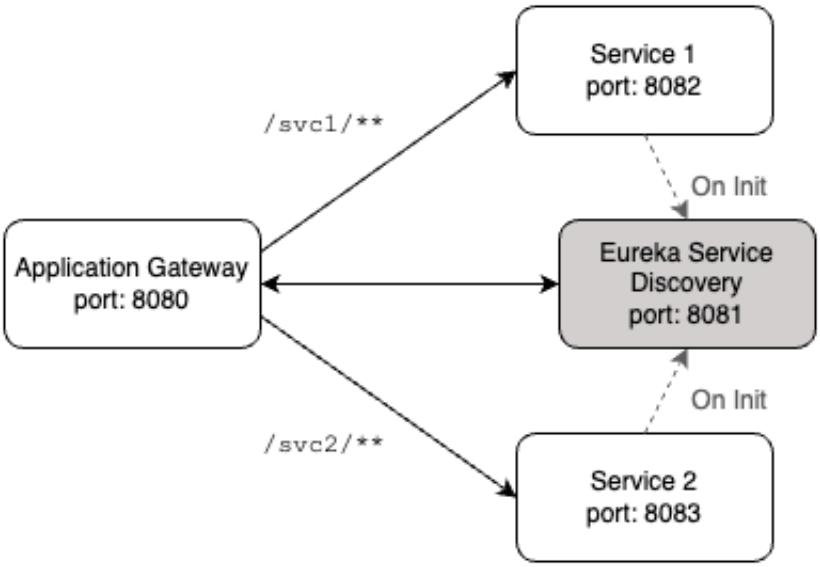
Response ...


```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Forwarded": "proto=http;host=\"localhost:8080\";for=\"127.0.0.1:56872\"",
    "Hello": "World",
    "Host": "httpbin.org",
    "User-Agent": "curl/8.1.2",
    "X-Amzn-Trace-Id": "Root=1-6570e2d5-1cb9d40a28fabd3b59228054",
    "X-Forwarded-Host": "localhost:8080"
  },
  "origin": "127.0.0.1, 69.243.41.76",
  "url": "http://localhost:8080/get"
}
```


Run the same command multiple times in a row, alteast twice per second. The first response should succeed. The second one should return a 429 error response. The command line won't show this to you. You can confirm by running `http://localhost:8080/get` in a browser with the network debugger open.

Eureka Discovery Service

The Proof of Concept does not use a discovery service. However, to do so would be fairly simple. Refer to the Eureka tutorial at the bottom of this document.



Each service, containing a unique host and port, needs a mapping in the gateway. The mapping between the gateway and the micro-service seems to be the application name.

Challenges

I encountered a few challenges that remain.

I was unable to route a request to a local service on a different port. I get a `404, Not Found` response.

I was also unable to connect to a non-local service using https. I get a `404, Page not found` response.

Tutorials

Circuit Breaker Tutorial

https://spring.io/guides/gs/gateway/#scratch

Rate Limiter Tutorial

https://www.baeldung.com/spring-cloud-gateway-rate-limit-by-client-ip

Spring Cloud Gateway + Netflix Eureka Example

https://www.javainuse.com/spring/cloud-gateway-eureka

Another Eureka example. This one has slightly different configuration which may be helpful.

https://www.geeksforgeeks.org/java-spring-boot-microservices-integration-of-eureka-and-spring-cloud-gateway/

References

Spring Cloud Circuit Breakers

https://spring.io/projects/spring-cloud-circuitbreaker

Spring with Redis

https://www.baeldung.com/spring-data-redis-properties