CLOUDFOUNDRY

Pivotal Cloud Foundry Developer

Lab Instructions

Application Deployment using Pivotal Cloud Foundry

Version 1.11.a

Pivotal

Copyright Notice

Pivotal

Table of Contents

Chapter 1. Route Service

Estimated Time: 25 minutes

1.1. Preface

Route services can be put to use in many ways: to cache responses from back-end services, to control access to an application, to audit how an application is accessed, and more.

In this lab you'll deploy a route service that will allow you to rate-limit access to the attendee-service application. In the process you'll learn yet more cf CLI commands, namely the `create-user-provided-service` command with a new flag (`-r`), and the `bind-route-service` command.

Don't forget to examine the source code to understand how this route service is implemented.

1.2. Exercises

1.2.1. Setup

1. `route-service.zip` is included in your lab files. It contains source code and a jar ready for you to deploy (no building necessary).
   - If you don't have this file, download it from zheee, and copy it to `pivotal-cloud-foundry-developer-workshop/`
2. Extract the the zip file to `pivotal-cloud-foundry-developer-workshop/route-service`.
3. OPTIONAL STEP - Import applications into your IDE such as Spring Tool Suite (STS).

STS Import Help

- Select menu:File[Import…]
- Then select `Maven` -> `Existing Maven Projects`.
- On the `Import Maven Project` page, browse to the directory where you extracted the zip.
- Then push the `Next` button, and then click `Finish`.

1.2.2. Route Service Overview

1. Review the documentation on Route Services.

1.2.3. Scenario

Route services can be used for a number of things such as logging, transformations, security and rate limiting.

Our `rate-limiter-app` application will do a couple of things. It will log incoming and outgoing requests. It will also impose a rate limit. No more than 3 requests per 15 seconds. Rate limited requests will be returned with a HTTP status code 429 (too many requests). Rate limiting is very common in the API space. Rate limiting protects your API from being overrun. The `rate-limiter-app` application will keep its state in Redis.

The `attendee-service` service exposes a RESTful API, so we will front it with our `rate-limiter-app`.

1.2.4. Implementing `rate-limiter-app`

1. Review the following file: … `/pivotal-cloud-foundry-developer-workshop/route-service/src/main/java/org/cloudfoundry/example/Controller.java`.

```
@RestController
final class Controller {
    static final String FORWARDED_URL = "X-CF-Forwarded-Url";
    static final String PROXY_METADATA = "X-CF-Proxy-Metadata";
    static final String PROXY_SIGNATURE = "X-CF-Proxy-Signature";

    private final static  Logger logger = LoggerFactory.getLogger(Controller.class);

    private final RestOperations restOperations;
    private RateLimiter rateLimiter;

    @Autowired
    Controller(RestOperations restOperations,  RateLimiter rateLimiter) {
        this.restOperations = restOperations;
        this.rateLimiter = rateLimiter;
    }

    @RequestMapping(headers = {FORWARDED_URL, PROXY_METADATA, PROXY_SIGNATURE})
    ResponseEntity<?> service(RequestEntity<byte[]> incoming) {
        logger.debug("Incoming Request: {}", incoming);
        if(rateLimiter.rateLimitRequest(incoming)){
            logger.debug("Rate Limit imposed");
                return new ResponseEntity<>(HttpStatus.TOO_MANY_REQUESTS);
        };
        RequestEntity<?> outgoing = getOutgoingRequest(incoming);
        logger.debug("Outgoing Request: {}", outgoing);

        return this.restOperations.exchange(outgoing, byte[].class);
    }

    private static RequestEntity<?> getOutgoingRequest(RequestEntity<?> incoming) {
        HttpHeaders headers = new HttpHeaders();
        headers.putAll(incoming.getHeaders());
        URI uri = headers.remove(FORWARDED_URL).stream()
            .findFirst()
            .map(URI::create)
            .orElseThrow(() -> new IllegalStateException(String.format("No %s header present", FORWARDED_URL)));

        return new RequestEntity<>(incoming.getBody(), headers, incoming.getMethod(), uri);
    }
}
```


What's happening?

The `service` method is where the `rate-limiter-app` application handles incoming requests.

1. Any request with the `X-CF-Forwarded-Url`, `X-CF-Proxy-Metadata`, and `X-CF-Proxy-Signature` headers gets handled by the `service` method.

2. Log the `incoming` request.

3. Check the `rateLimiter` to see if the number of requests has exceeded the rate limit threshold. If the threshold is exceeded return a HTTP status code 429 (too many requests). If the threshold is not exceeded remove the `FORWARDED_URL` header, log the `outgoing` request, and send the `outgoing` request to the downstream application.

2. Review the following file: `pivotal-cloud-foundry-developer-workshop/route-service/src/main/java/org/cloudfoundry/example/RateLimiter.java`.

```
@Component
public class RateLimiter {
    private final static  Logger logger = LoggerFactory.getLogger(RateLimiter.class);
    private final String KEY = "host";

    @Autowired
    private StringRedisTemplate redisTemplate;

    @Scheduled(fixedRate = 15000)
    public void resetCounts() {
        redisTemplate.delete(KEY);
        logger.debug("Starting new 15 second interval");
    }

    public boolean rateLimitRequest(RequestEntity<?> incoming)  {
        String forwardUrl = incoming.getHeaders().get(Controller.FORWARDED_URL).get(0);
        URI uri;
        try {
            uri = new URI(forwardUrl);
        } catch (URISyntaxException e) {
            logger.error("error parsing url", e);
            return false;
        }

        String host = uri.getHost();
        String value = (String)redisTemplate.opsForHash().get(KEY, host);
          int requestsPerInterval = 1;

        if (value == null){
            redisTemplate.opsForHash().put(KEY, host, "1");
        }
        else{
            requestsPerInterval = Integer.parseInt(value) + 1;
            redisTemplate.opsForHash().increment(KEY, host, 1);
        }

        if(requestsPerInterval > 3)
            return true;
        else
            return false;
    }
}
```


What's happening?

The `rateLimitRequest` method determines whether a request should be rate limited.

1. Increment the request count by host.
2. Return `true` if request should be rate limited (`requestsPerInterval > 3`).
3. Return `false` if request should not be rate limited (`requestsPerInterval <= 3`).

The `resetCounts` method deletes the Redis `KEY` every 15 seconds, which resets the counts by deleting all the state.

Note

This is an example implementation for lab purposes only. A proper rate limiting service would need to uniquely identify the client. That can be accomplished via an API key, the `X-Forwarded-For` header, or other approaches.

1.2.5. Push `rate-limiter-app`

1. Push `rate-limiter-app`:

```
$> cd .../pivotal-cloud-foundry-developer-workshop/route-service/
$> cf push rate-limiter-app -p ./target/route-service-1.0.0.BUILD-SNAPSHOT.jar -m 768M --random-route --no-start
```

2. Create a Redis service instance. Do ONE of the following:

Option A: Pivotal Web Services

In PWS, the marketplace service for Redis is called "rediscloud".

```
cf create-service rediscloud 30mb redis
```

Option B: Pivotal Cloud Foundry

Pivotal provides a redis managed service named "p-redis".

```
cf create-service p-redis shared-vm redis
```


3. Bind the service instance.

```
cf bind-service rate-limiter-app redis
```

4. Start the application.

```
cf start rate-limiter-app
```

1.2.6. Create a Route Service and Bind it to a Route

1. Create a user provided service. Let's call it `rate-limiter-service`.

```
cf create-user-provided-service rate-limiter-service -r {{ratelimiter_baseurl}}
```

2. Bind the `rate-limiter-service` to the `attendee-service` route.

```
cf bind-route-service {{domain_name}} rate-limiter-service --hostname {{attendee_service_hostname}}
```

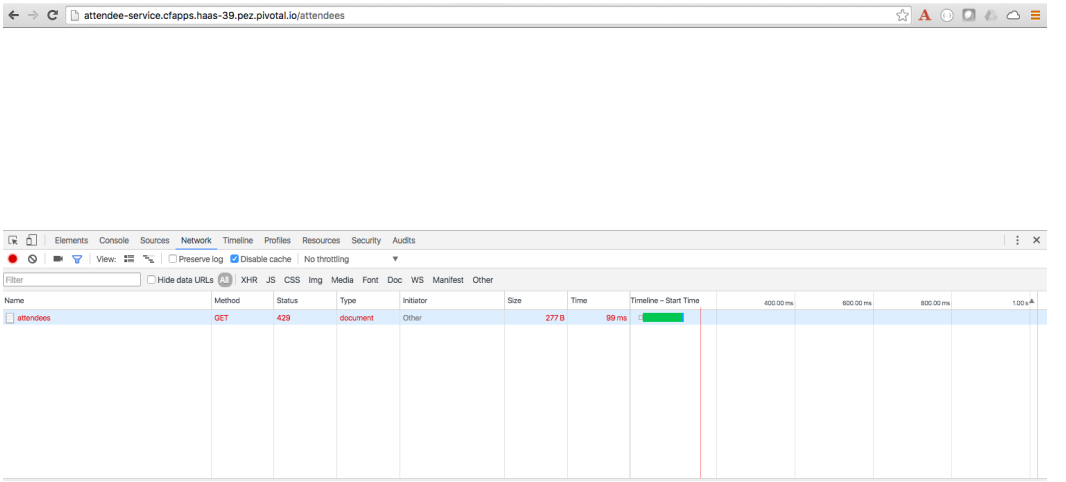
1.2.7. Observe the effects of the `rate-limiter-app`

1. Tail the logs of the `rate-limiter-app` application.

```
cf logs rate-limiter-app
```

2. Choose a client of your preference, but one that can show HTTP status code. Hit an `attendee-service` endpoint (e.g. `/attendees`) several times and see if you can get the rate limit to trigger. Observe the logs.

   Pic below is using Chrome with the Developer Tools.



1.2.8. Questions

- What are the key headers used to implement route services (Service Instance Responsibilities)?
- How would you apply route services in your environment?

1.2.9. Clean up

1. Unbind the route service.

```
cf unbind-route-service {{domain_name}} rate-limiter-service --hostname {{attendee_service_hostname}}
```

2. Delete `rate-limiter-service` service instance.

```
cf delete-service rate-limiter-service
```

3. Unbind `redis` service instance from the app.

```
cf unbind-service rate-limiter-app redis
```

4. Delete the `redis` service instance.

```
cf delete-service redis
```


5. Delete the `rate-limiter-app` app.

```
cf delete rate-limiter-app
```