(../..)

Purpose of this lab

- How to create a route service
- Estimated Time: 25 minutes

Setup

1. Download the zip file (route-service.zip). The zip file contains source code and jar ready for you to deploy (no building necessary). Copy the file to folder: `~/pivotal-cloud-foundry-developer-workshop/`. You will need to create this directory in your home directory.
2. Extract the the zip file to `~/pivotal-cloud-foundry-developer-workshop/route-service`.
3. Import applications into your IDE (IntelliJ).

Route Service Overview

1. Review the documentation on Route Services (http://docs.pivotal.io/pivotalcf/services/route-services.html).

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 (https://httpstatuses.com/429) (too many

requests). Rate limiting is very common in the API space and 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`.

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 rateLim
iter) {
        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_REQUES
TS);
        };
        RequestEntity<?> outgoing = getOutgoingRequest(incoming);
```

```
        logger.debug("Outgoing Request: {}", outgoing);

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

    private static RequestEntity<?> getOutgoingRequest(RequestEnti
ty<?> incoming) {
        HttpHeaders headers = new HttpHeaders();
        headers.putAll(incoming.getHeaders());
        URI uri = headers.remove(FORWARDED_URL).stream()
            .findFirst()
            .map(URI::create)
            .orElseThrow(() -> new IllegalStateException(String.fo
rmat("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.
4. 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.F
ORWARDED_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);
        }

        return requestsPerInterval > 3;
    }
}
```

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.

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-SNA
PSHOT.jar -m 512M --random-route --no-start
```

2. Create a Redis service instance.

   Pivotal Cloud Foundry:

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

   Pivotal Web Services:

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

3. Bind the service instance.

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

4. Start the application.

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

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 https://<R
   ATE-LIMITER-APP-ADDRESS>
   ```

   For Example:

   ```
   cf create-user-provided-service rate-limiter-service -r https://ro
   ute-service-random-route.cfapps.io
   ```

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

   ```
   cf bind-route-service <APPLICATION-DOMAIN> rate-limiter-service --
   hostname <APPLICATION-HOST>
   ```

   For Example:

   ```
   cf bind-route-service cfapps.io rate-limiter-service --hostname
   attendee-service-random-route
   ```

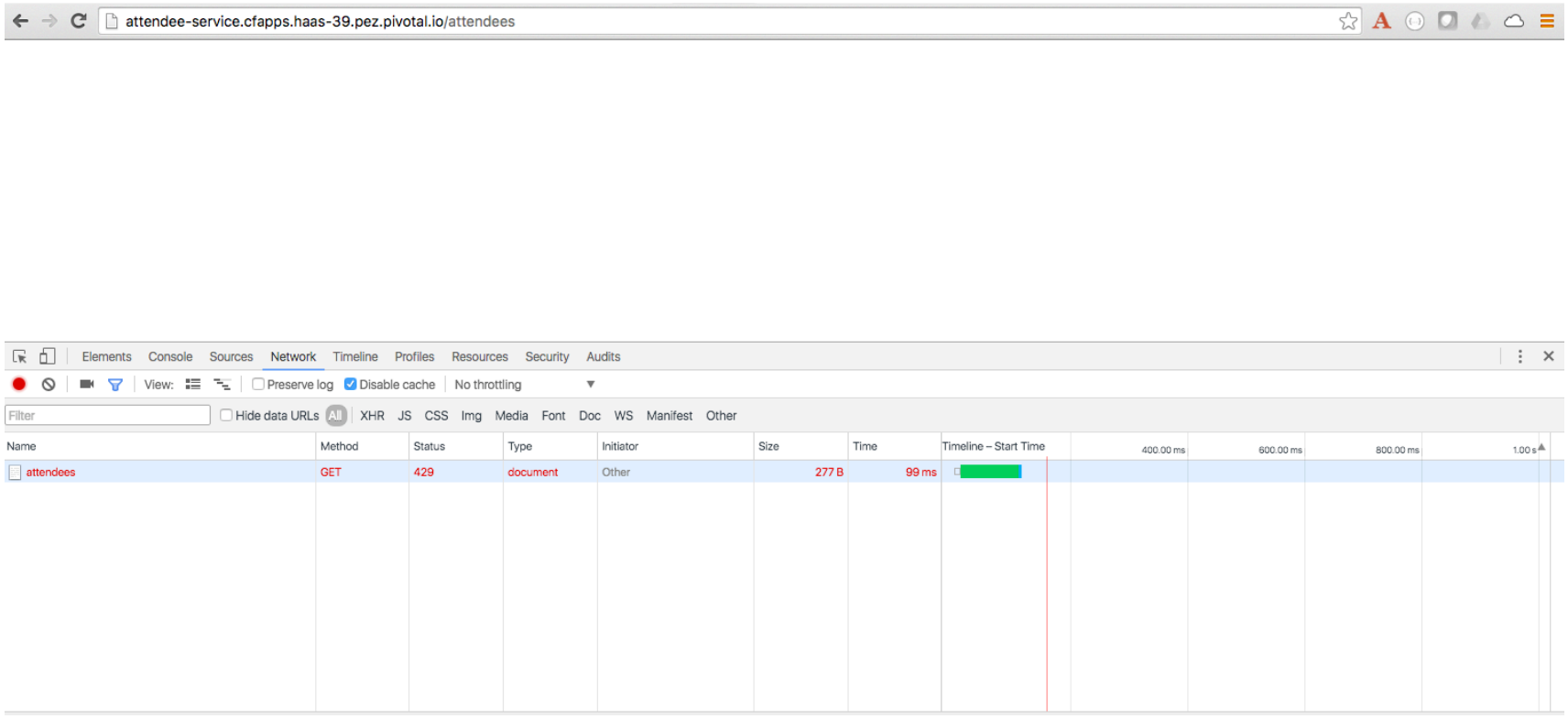
Observe the effects of the `rate-limiter-app`

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.



Questions

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

Clean up

1. Unbind the route service.

   ```
   cf unbind-route-service <APPLICATION-DOMAIN> rate-limiter-service
   --hostname <APPLICATION-HOST>
   ```

   For Example:

   ```
   cf unbind-route-service cfapps.io rate-limiter-service --hostname
   attendee-service-random-route
   ```

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
```

(https://pivotal.io)