Envoy global rate limiting

E' possibile configurare *Envoy* in modo che interagisca con un servizio RPC di *rate limiting* (in particolare di tipo *gRPC*).
Questo servizio deciderà, sulla base della sua configurazione, se è il caso o meno di limitare una *request* proveniente da un *Envoy Proxy*.

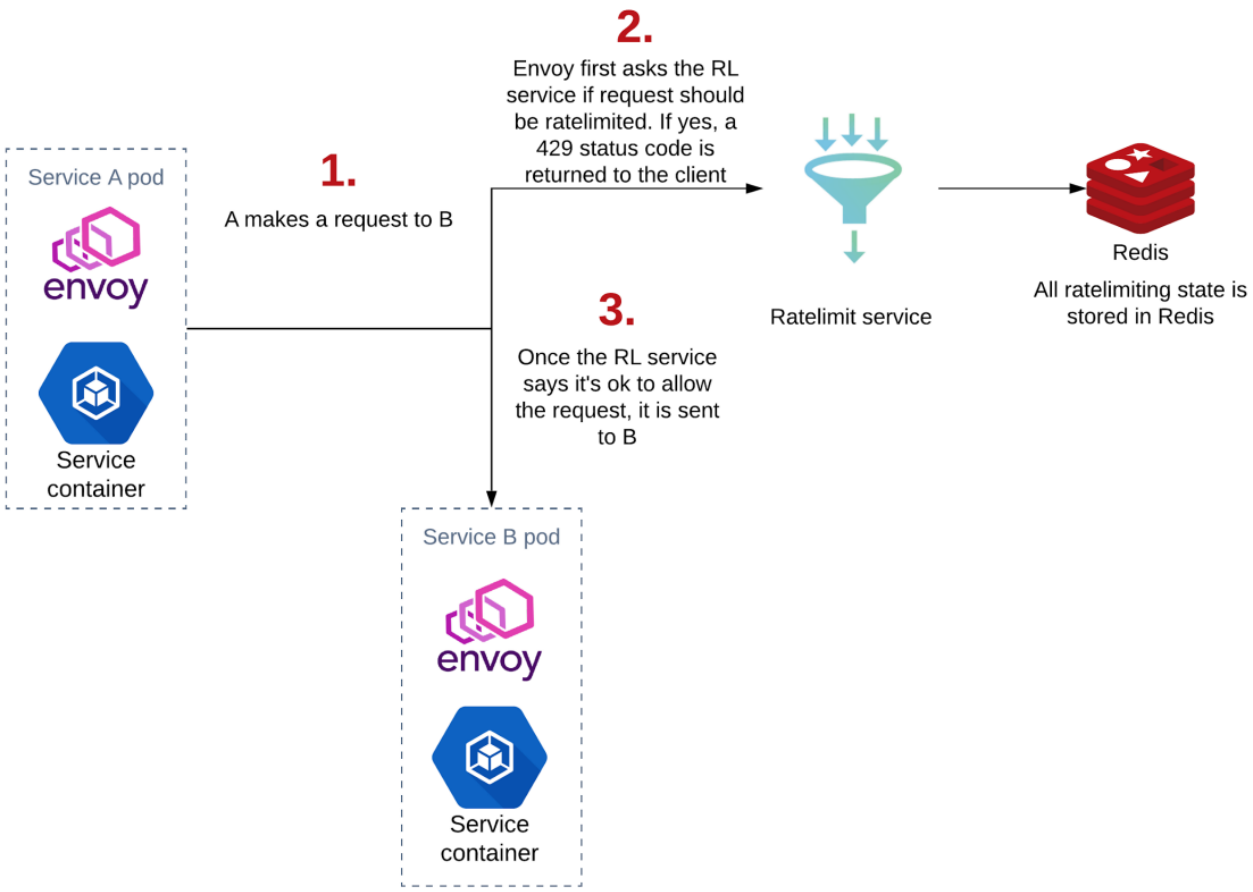
https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/other_features/global_rate_limiting#arch-overview-global-rate-limit

Per il servizio di *rate limiting, Envoy* fornisce un'implementazione *Open Source* di riferimento scritta in *Go*, che fa uso di un *Server Redis*:

https://github.com/envoyproxy/ratelimit

Le informazioni presenti sulla documentazione *Envoy*, sia per quanto riguarda l'architettura della soluzione, sia riguardo le configurazioni da effettuare, non sono però di semplice comprensione.

Tinder, all' envoycon del 15-10-2020, illustra la propria implementazione durante l'intervento di *Yuki Sawa* (Software Engineer, Tinder) dal titolo *"How Tinder implemented Envoy global rate limiting at scale"*



https://static.sched.com/hosted_files/envoycon2020/fc/envoycon2020-slides-final.pdf

Un *Envoy Proxy* viene quindi dispiegato come *sidecar-container* all'interno dei *PODs* che gestiscono i diversi microservizi presenti in un *Kubernetes Cluster*.
Per Le *request* provenienti dai *service-container*, *Envoy* contatterà il *Rate Limit Service,* che fornirà in risposta al proxy uno *status code 429 (Too Many Requests)* nel caso di limitazione della richiesta (200 altrimenti).
Lo stato delle *request* limitate verrà salvato all'interno di un *Redis Server.*

Altri articoli presenti sulla piattaforma *medium.com* aiutano nella comprensione dell' *Envoy Global rate limiting:*

https://eng.lyft.com/announcing-ratelimit-c2e8f3182555

https://medium.com/dm03514-tech-blog/sre-resiliency-bolt-on-sidecar-rate-limiting-with-envoy-sidecar-5381bd4a1137

https://xuorig.medium.com/understanding-envoyproxys-rate-limiting-cb3e00c9be2d

https://salmaan-rashid.medium.com/envoy-global-rate-limiting-helloworld-9d909dc318cb

Envoy Proxy - ratelimit configuration

Envoy consente il *ratelimit* sia a livello TCP che HTTP, attraverso due tipologie di filtri che è possibile applicare ai *Listener* definiti su un *Envoy Proxy*, rispettivamente, *Network Level Filter e HTTP Level Filter.*

All'interno della configurazione del proxy, dovrà essere specificata sia la configurazione del *Cluster*, ovvero delle risorse di *upstream* che il Proxy può contattare:

```
clusters:

  ...

  - name: <rate_limit_service_cluster_name>
    type: strict_dns
    connect_timeout: 0.25s
    lb_policy: round_robin
    hosts:
    - socket_address:
        address: <rate_limit_service_ip>
        port_value: <rate_limit_service_port>
```

sia la configurazione del *Rate Limit Service:*

```
rate_limit_service:
    grpc_service:
        envoy_grpc:
            cluster_name: <rate_limit_service_cluster_name>
        timeout: 0.25s
```

ciò consentirà all'*Envoy Proxy* di conoscere quale è la risorsa del cluster relativa al *Rate Limit Service.*

Occorrerà poi configurare il *Rate Limit Filter* sul *Listener dell' Envoy Proxy* (e.g. *Network Level Filter*):

```
listeners:
    - address:
        socket_address:
          address: 0.0.0.0
          port_value: <listener_port>
filter_chains:
    - filters:
        - name: envoy.filters.network.ratelimit
          config:
              stat_prefix: ingress_ratelimit
              domain: <filter_domain>
              failure_mode_deny: true
              descriptors:
                  - entries:
                      - key: <filter_key>
                        value: <filter_value>
```

Come da documentazione Envoy relativa al *Network Level Filter*:

https://www.envoyproxy.io/docs/envoy/latest/api-v3/extensions/filters/network/ratelimit/v3/rate_limit.proto#envoy-v3-api-msg-extensions-filters-network-ratelimit-v3-ratelimit

I valori all'interno del campo *descriptors*, saranno inviati dall' *Envoy Proxy* al *Limit Service* tramite una *request* del tipo:

```
RateLimitRequest:
    domain: <filter_domain>
    descriptors: (<filter_key>, <filter_value>)
```

in accordo con il protocollo RPC/IDL definito:

https://github.com/envoyproxy/data-plane-api/blob/main/envoy/service/ratelimit/v3/rls.proto

Ratelimit Service configuration

Così come l'*Envoy Proxy*, il *Rate Limit Service* accetta una configurazione YAML del tipo

```
domain: <filter_domain>
descriptors:
 - key: <filter_key>
   value: <filter_value>
   rate_limit:
     unit: <second, minute, hour, day>
     requests_per_unit: <request_number>
```

Il capo *domain* specifica a quale dominio di richieste (configurato all'interno del *ratelimit filter*) si applicheranno le regole di limitazione, mentre il campo *descriptors* conterrà le regole di limitazione specifiche per i descrittori presenti nelle *RateLimitRequest.*

Il *Rate Limit Service* è infatti un servizio globale, la cui configurazione potrà gestire diversi *Envoy Proxy* configurati per il *rate limit.*

Alcune considerazioni

Sia all'interno degli articoli citati precedentemente, sia all'interno del *README.md* del repository contenente l'implementazione di riferimento di *Envoy* per il *Rate Limit Service*, sono presenti esempi di configurazione e deploy, principalmente per un *rate limiting* di tipo HTTP.

L'implementazione di riferimento di *Envoy* per il *Rate Limit Service* prevede una configurazione in cui il *rate limit* va specificato in modo statico, non è dunque possibile agganciare il servizio ad una metrica esterna (e.g. un Api Server esterno), sulla quale si possa basare per decidere se limitare o meno le richieste.

In questa implementazione del *Rate Limit Service* non sono presenti meccanismi di accodamento delle *request "*limitate", in modo da poter essere riprocessarle, in modo trasparente dal servizio, in momenti di basso traffico. Semplicemente, alle richieste "limitate" non ci sarà risposta, quindi un eventuale meccanismo di "retry" dovrà essere gestito lato client.