

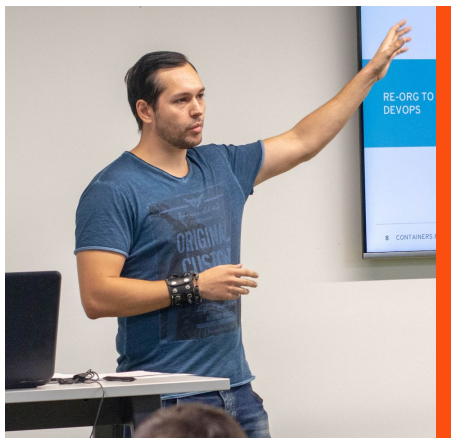
A photograph of an industrial facility, likely a refinery or chemical plant. In the foreground, numerous large, light-colored pipes run parallel to each other, receding into the distance. In the background, there are large white storage tanks and various industrial structures under a clear sky. A semi-transparent orange banner covers the bottom half of the image, containing the title and date.

{paradigma

Service mesh: abriendo en canal Envoy

Noviembre 2018

Sobre nosotros



Abraham Rodríguez

Arquitecto Cloud Native @Paradigma



Luis Mora Medina

Arquitecto software @Paradigma

@luismoramedina



1. Repaso a service-mesh
2. Funcionalidades
3. Ejemplo de aplicación
4. Configuración y uso
5. Necesidad de un plano de control
6. Conclusiones



AUTORES Y CONTACTO

Abraham Rodríguez

Arquitecto Cloud Native

Luis Mora Medina

Arquitecto de Software

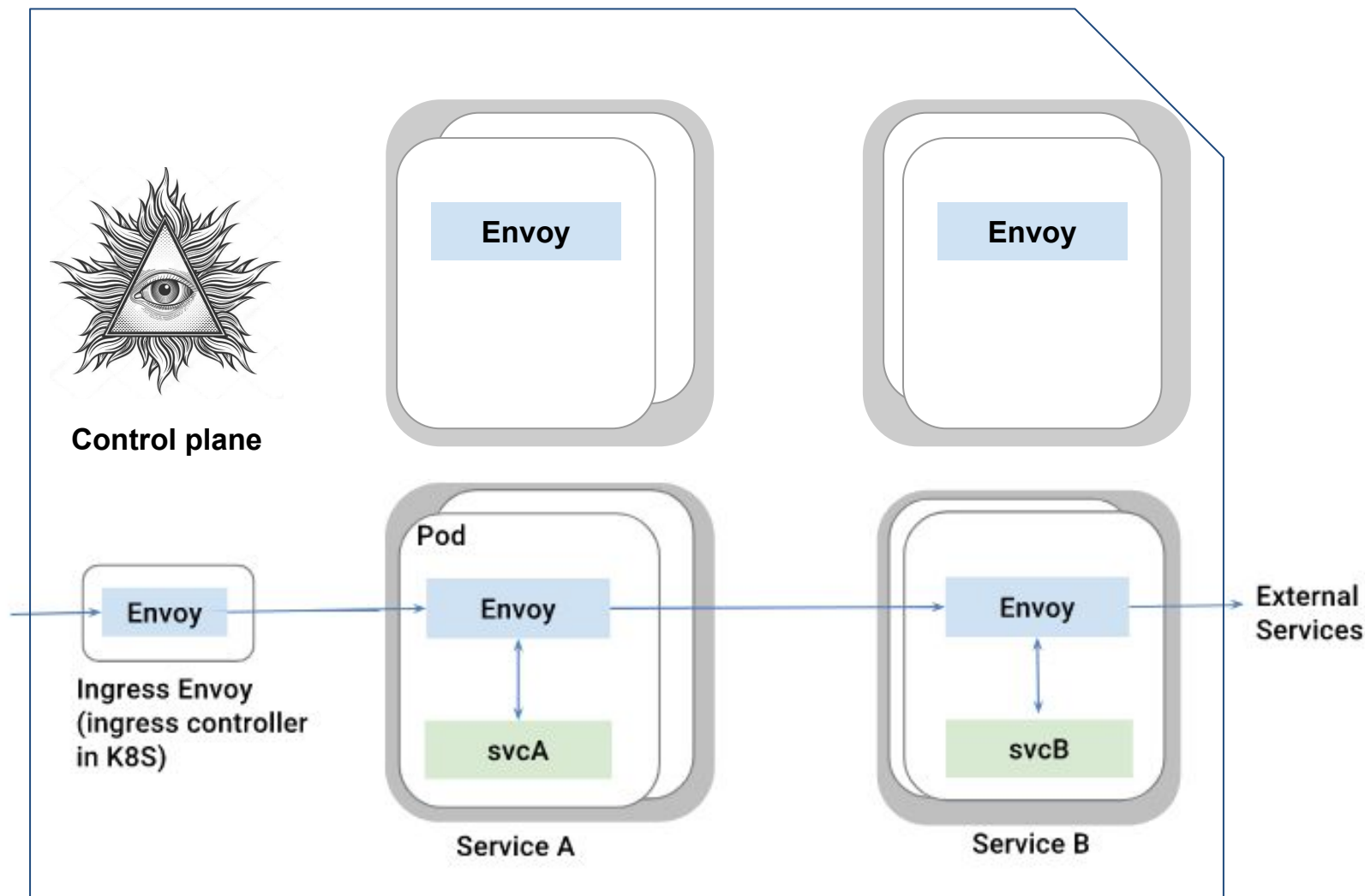
Repaso a service-mesh

*“A service mesh is a **dedicated infrastructure layer** for handling **service-to-service communication**. It’s responsible for the **reliable delivery of requests** through the complex topology of services that comprise a modern, cloud native application”*

William Morgan, Buoyant CEO
April 2017



Repaso a service mesh: patrón proxy-sidecar & plano de control



Patrón proxy-sidecar:

- Acompaña siempre a la aplicación
- Actuando como proxy
- Intercepta y maneja la complejidad de red
- Independiente del lenguaje de implementación

Plano de control:

- Coordina el plano de datos para que funcione como un único sistema integrado
- Monitoriza el sistema en su conjunto
- Securiza el sistema en su conjunto

Funcionalidades



Funcionalidades que cubre Envoy

*“The network should be **transparent to applications**. When network and application problems do occur it should be easy to determine the source of the problem.”*

- **Enrutado y balanceo**: mirroring de peticiones, diferentes algoritmos de balanceo, reintentos
- **Protocolos de comunicación**: HTTP1, HTTP2, gRPC
- **Service discovery**: DNS, consumo de APIs
- **Health check**: activo y pasivo
- **Edge proxy**: punto de entrada único al sistema.
- **Cambio de configuración en caliente**
- **Gestión del fallo**: circuit breaking, timeout, limitación de carga.
- **Monitorización**: recolección de métricas, trazabilidad distribuida
- **Integración de métricas con BBDD**: MongoDB, DynamoDB
- **TLS, JWT validation**

Principales recursos en Envoy

Listeners

Podemos tener varios dentro de una misma instancia de Envoy y pueden tener comportamientos diferentes en base a los filtros que definamos

HttpConnectionManager

Estadísticas, trazabilidad distribuida, reescritura de cabeceras, logado ...

Http Routing

Mapea dominios a clusters, enrutado por path, reintentos, gestión de timeout, priorización

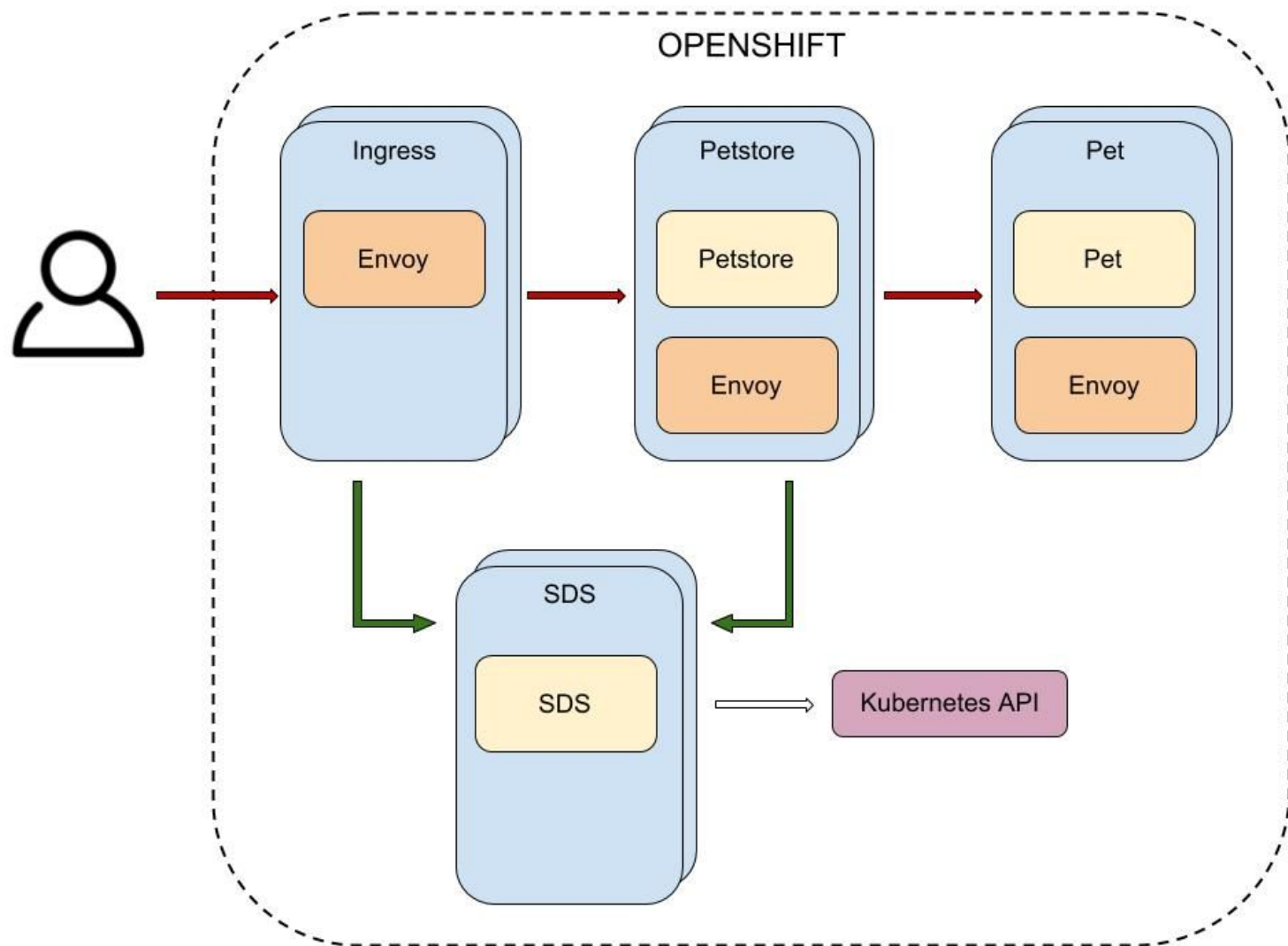
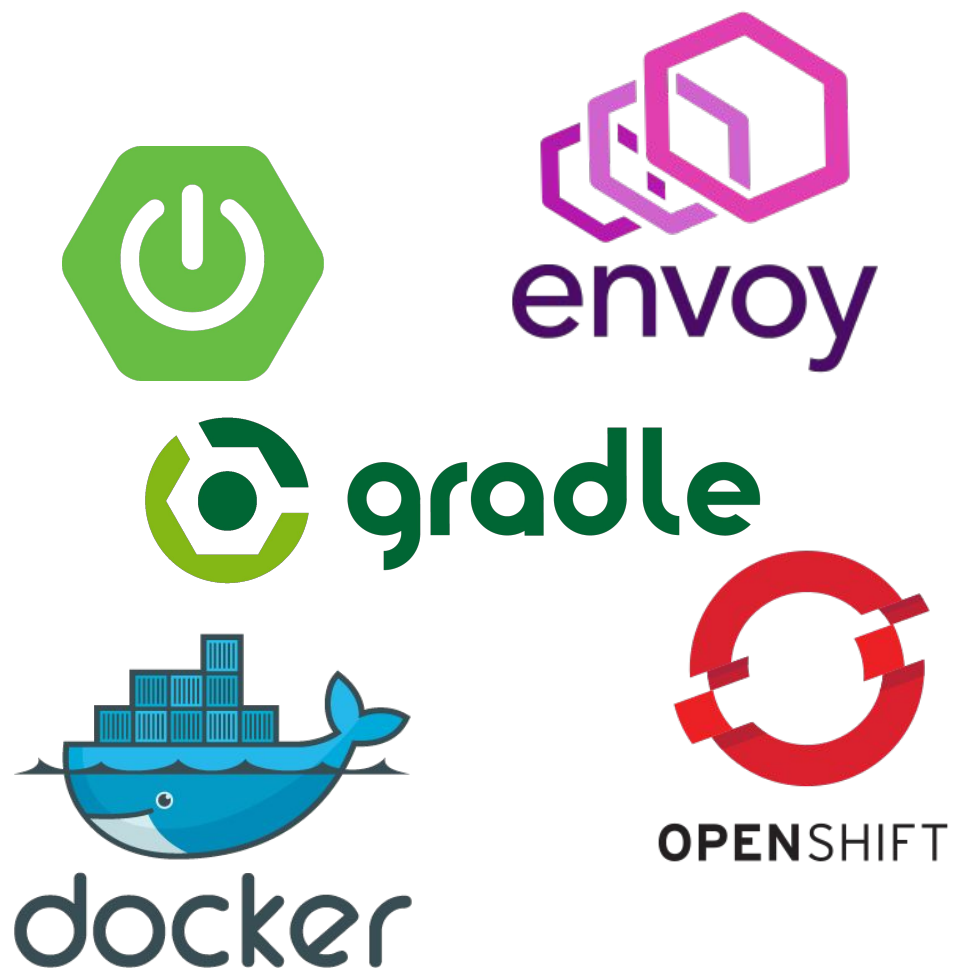
Clusters

Algoritmo de balanceo, timeouts, descubrimiento de instancias, gestión del fallo

Ejemplo de aplicación



Ejemplo de aplicación



Configuración y uso



1. Configuración y peticiones
2. Descubrimiento de instancias
3. Ingress
4. Gestión de error
 - a. Timeout
 - b. Circuit breaking
 - c. Outlier detection
5. Trazabilidad distribuida



Configuración y peticiones

Ejemplos de configuración

```
static_resources:
  listeners:
    - address:
        socket_address:
          address: 0.0.0.0
          port_value: 10000
        filter_chains:
          - filters:
              - name: envoy.http_connection_manager
                config:
                  codec_type: auto
                  stat_prefix: ingress_http
                  route_config:
                    name: ingress_route
                    virtual_hosts:
                      - name: service
                        domains:
                          - "*"
                        routes:
                          - match:
                              prefix: "/"
                            route:
                              cluster: local_service
                  http_filters:
                    - name: envoy.router
                      config: {}
```

Ejemplos de configuración

```
static_resources:
  listeners:
  - address:
    socket_address:
      address: 0.0.0.0
      port_value: 10000
    filter_chains:
    - filters:
      - name: envoy.http_connection_manager
        config:
          codec_type: auto
          stat_prefix: ingress_http
          route_config:
            name: ingress_route
            virtual_hosts:
            - name: service
              domains:
              - "*"
              routes:
              - match:
                  prefix: "/"
                route:
                  cluster: local_service
          http_filters:
          - name: envoy.router
            config: {}
```

En qué dirección y puerto va a escuchar dicho listener

Ejemplos de configuración

```
static_resources:
```

```
  listeners:
```

```
    - address:
```

```
      socket_address:
```

```
        address: 0.0.0.0
```

```
        port_value: 10000
```

```
    filter_chains:
```

```
      - filters:
```

```
        - name: envoy.http_connection_manager
```

```
          config:
```

```
            codec_type: auto
```

```
            stat_prefix: ingress_http
```

```
            route_config:
```

```
              name: ingress_route
```

```
              virtual_hosts:
```

```
                - name: service
```

```
                  domains:
```

```
                    - "*"
```

```
                  routes:
```

```
                    - match:
```

```
                      prefix: "/"
```

```
                      route:
```

```
                        cluster: local_service
```

```
            http_filters:
```

```
              - name: envoy.router
```

```
              config: {}
```

En qué dirección y puerto va a escuchar dicho listener

Configuración del filtro:

- nombrados
- dominios asignados
- mapeo rutas - clusters

Ejemplos de configuración

clusters:

- name: local_service
connect_timeout: 0.50s
type: strict_dns
lb_policy: round_robin
hosts:
 - socket_address:
address: 127.0.0.1
port_value: 8080

admin:

access_log_path: "/tmp/admin_access.log"

address:

- socket_address:
address: 0.0.0.0
port_value: 8081

Ejemplos de configuración

clusters:

```
- name: local_service
  connect_timeout: 0.50s
  type: strict_dns
  lb_policy: round_robin
  hosts:
    - socket_address:
        address: 127.0.0.1
        port_value: 8080
```

admin:

```
access_log_path: "/tmp/admin_access.log"
address:
  socket_address:
    address: 0.0.0.0
    port_value: 8081
```

Configuración del cluster:

- Timeout
- Descubrimiento
- Algoritmo de balanceo

Ejemplos de configuración

```
clusters:  
- name: local_service  
  connect_timeout: 0.50s  
  type: strict_dns  
  lb_policy: round_robin  
  hosts:  
  - socket_address:  
      address: 127.0.0.1  
      port_value: 8080
```

Configuración del cluster:

- Timeout
- Descubrimiento
- Algoritmo de balanceo

```
admin:  
  access_log_path: "/tmp/admin_access.log"  
  address:  
    socket_address:  
      address: 0.0.0.0  
      port_value: 8081
```

Administración de Envoy

- Logs
- Puerto de Escucha

[Interfaz de administración](#)

Llamadas entre microservicios

```
- address:
  socket_address:
    address: 0.0.0.0
    port_value: 9900
  filter_chains:
  - filters:
    - name: envoy.http_connection_manager
      config:
        codec_type: auto
        stat_prefix: egress_mesh_http
        route_config:
          name: mesh_route
          virtual_hosts:
            - name: service_mesh
              domains:
                - "*"
              routes:
                - match:
                    prefix: "/"
                  route:
                    cluster_header: ":authority"
        http_filters:
          - name: envoy.router
            config: {}
```

Puertos configurados:

- 10000: Entrada
- 9900: Salida
- 8081: Administración

Llamadas entre microservicios

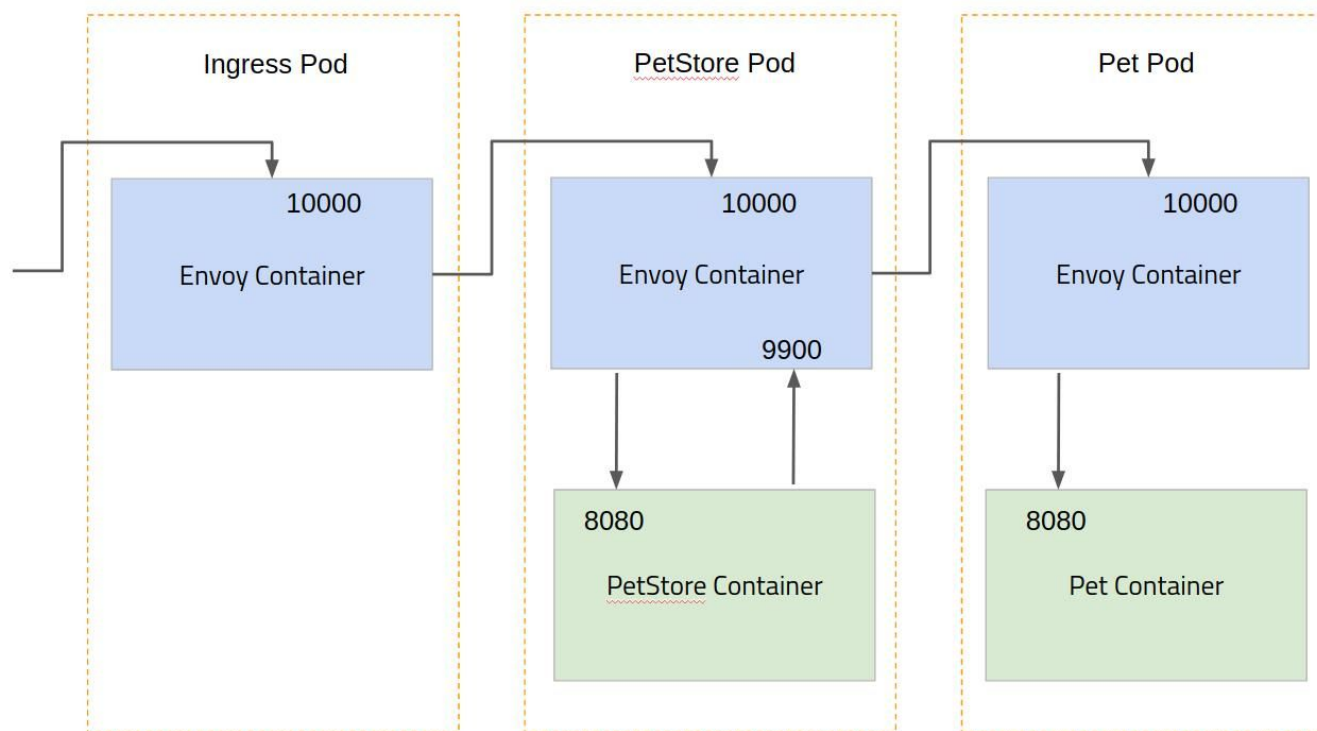
```

- address:
  socket_address:
    address: 0.0.0.0
    port_value: 9900
  filter_chains:
  - filters:
    - name: envoy.http_connection_manager
      config:
        codec_type: auto
        stat_prefix: egress_mesh_http
        route_config:
          name: mesh_route
          virtual_hosts:
            - name: service_mesh
              domains:
                - "*"
              routes:
                - match:
                    prefix: "/"
                  route:
                    cluster_header: ":authority"
          http_filters:
            - name: envoy.router
              config: {}

```

Puertos configurados:

- 10000: Entrada
- 9900: Salida
- 8081: Administración



Llamadas entre microservicios

```
- address:
  socket_address:
    address: 0.0.0.0
    port_value: 9900
  filter_chains:
  - filters:
    - name: envoy.http_connection_manager
      config:
        codec_type: auto
        stat_prefix: egress_mesh_http
        route_config:
          name: mesh_route
          virtual_hosts:
            - name: service_mesh
              domains:
                - "*"
              routes:
                - match:
                    prefix: "/"
                  route:
                    cluster_header: ":authority"
        http_filters:
          - name: envoy.router
            config: {}
```

Puertos configurados:

- 10000: Entrada
- 9900: Salida
- 8081: Administración

El cluster invocado se determina por la cabecera Host, reconvertida a :authority

[Código de ejemplo](#)



Descubrimiento de instancias

Descubrimiento de clusters

- **Static**: definir directamente el listado de instancias
- **Strict DNS**: cada IP devuelta por el DNS es considerada un host
- **Logical DNS**: cada vez que se necesita una nueva conexión se utiliza la primera IP devuelta por el DNS
- **SDS**: API REST invocada periodicamente para resolver las instancias que componen un cluster. **2 versiones**
- ...

“SDS is the preferred service discovery mechanism for a few reasons

- Envoy has **explicit knowledge** of each upstream host (vs. routing through a DNS resolved load balancer) and can make **more intelligent load balancing decisions**.
- **Extra attributes** carried in the discovery API response for each host inform Envoy of the host's load balancing weight, canary status, zone ...”

Descubrimiento de clusters

- **Static**: definir directamente el listado de instancias
- **Strict DNS**: cada IP devuelta por el DNS es considerada un host
- **Logical DNS**: cada vez que se necesita una nueva conexión se utiliza la primera IP devuelta por el DNS
- **SDS**: API REST invocada periódicamente para resolver las instancias que componen un cluster. 2 versiones
- ...

“SDS is the preferred service discovery mechanism for a few reasons

- Envoy has **explicit knowledge** of each upstream host (vs. routing through a DNS resolved load balancer) and can make **more intelligent load balancing decisions**.
- **Extra attributes** carried in the discovery API response for each host inform Envoy of the host's load balancing weight, canary status, zone ...”

Su implementación

- El **microservicio se registra** en el servicio de descubrimiento
- El registro de instancias **se almacena en DynamoDB**

Descubrimiento de clusters

- **Static**: definir directamente el listado de instancias
- **Strict DNS**: cada IP devuelta por el DNS es considerada un host
- **Logical DNS**: cada vez que se necesita una nueva conexión se utiliza la primera IP devuelta por el DNS
- **SDS**: API REST invocada periódicamente para resolver las instancias que componen un cluster. 2 versiones
- ...

“SDS is the preferred service discovery mechanism for a few reasons

- Envoy has **explicit knowledge** of each upstream host (vs. routing through a DNS resolved load balancer) and can make **more intelligent load balancing decisions**.
- **Extra attributes** carried in the discovery API response for each host inform Envoy of the host's load balancing weight, canary status, zone ...”

Su implementación

- El microservicio se registra en el servicio de descubrimiento
- El registro de instancias se almacena en DynamoDB

NOT VALID

Descubrimiento de clusters

- **Static**: definir directamente el listado de instancias
- **Strict DNS**: cada IP devuelta por el DNS es considerada un host
- **Logical DNS**: cada vez que se necesita una nueva conexión se utiliza la primera IP devuelta por el DNS
- **SDS**: API REST invocada periódicamente para resolver las instancias que componen un cluster. 2 versiones
- ...

“SDS is the preferred service discovery mechanism for a few reasons

- Envoy has **explicit knowledge** of each upstream host (vs. routing through a DNS resolved load balancer) and can make **more intelligent load balancing decisions**.
- **Extra attributes** carried in the discovery API response for each host inform Envoy of the host's load balancing weight, canary status, zone ...”

Su implementación

- El microservicio se registra en el servicio de descubrimiento
- El registro de instancias se almacena en DynamoDB

Nuestra visión

- No queremos delegar a la aplicación la responsabilidad del registro
- No queremos utilizar DynamoDB
- No queremos utilizar los servicios de descubrimiento de Kubernetes - perdemos funcionalidad de Envoy

Descubrimiento de clusters

- **Static**: definir directamente el listado de instancias
- **Strict DNS**: cada IP devuelta por el DNS es considerada un host
- **Logical DNS**: cada vez que se necesita una nueva conexión se utiliza la primera IP devuelta por el DNS
- **SDS**: API REST invocada periódicamente para resolver las instancias que componen un cluster. 2 versiones
- ...

“SDS is the preferred service discovery mechanism for a few reasons

- Envoy has **explicit knowledge** of each upstream host (vs. routing through a DNS resolved load balancer) and can make **more intelligent load balancing decisions**.
- **Extra attributes** carried in the discovery API response for each host inform Envoy of the host's load balancing weight, canary status, zone ...”

Su implementación

- El microservicio se registra en el servicio de descubrimiento
- El registro de instancias se almacena en DynamoDB

Nuestra visión

- No queremos delegar a la aplicación la responsabilidad del registro
- No queremos utilizar DynamoDB
- No queremos utilizar los servicios de descubrimiento de Kubernetes - perdemos funcionalidad de Envoy

Nuestra solución

- [Consultar el API de Kubernetes](#)

Descubrimiento de clusters: configuración

Cluster de SDS

- name: **sds**
 - connect_timeout: 0.50s
 - type: strict_dns
 - lb_policy: round_robin
 - hosts:
 - socket_address:
 - address: **sds**
 - port_value: 80

Descubrimiento de clusters: configuración





Cluster de SDS

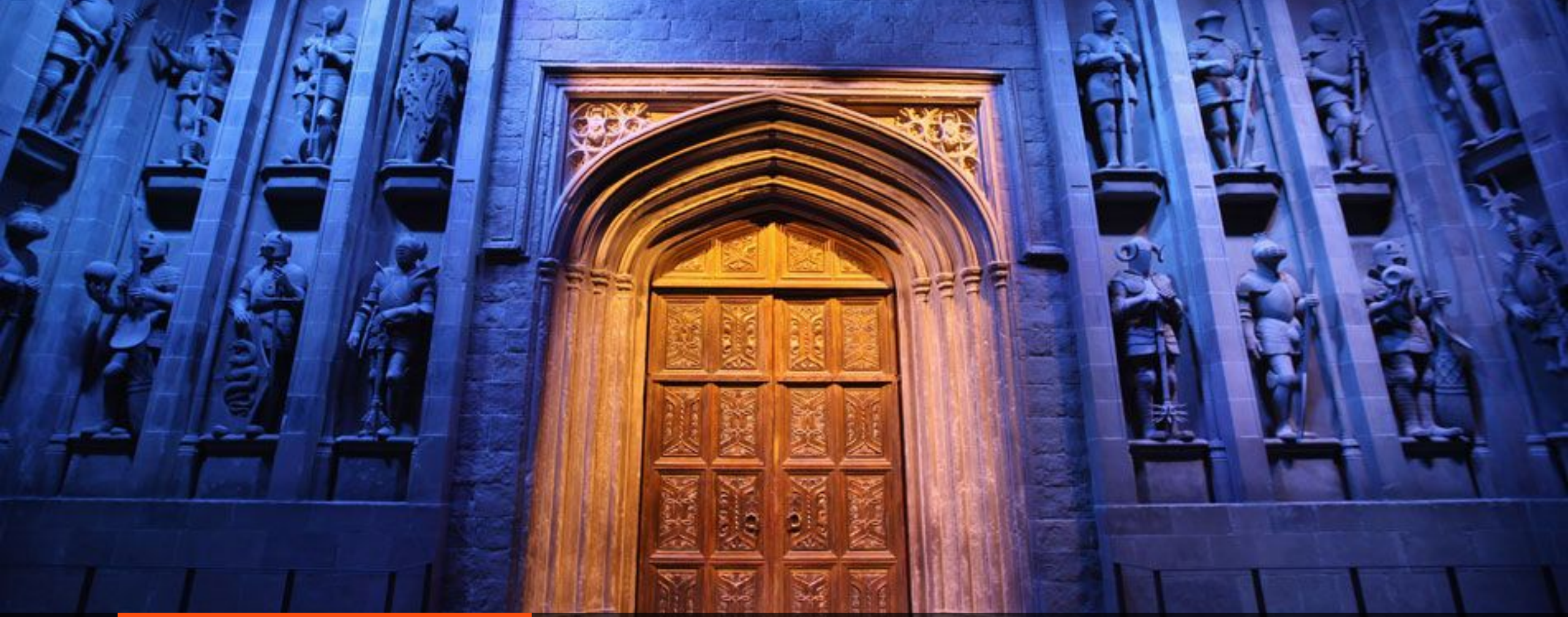
- name: **sds**
connect_timeout: 0.50s
type: strict_dns
lb_policy: round_robin
hosts:
 - socket_address:
address: **sds**
port_value: 80

Referenciar al SDS

- name: **pet**
connect_timeout: 0.50s
type: **eds**
lb_policy: round_robin
eds_cluster_config:
eds_config:
api_config_source:
api_type: REST_LEGACY
cluster_names: **sds**
refresh_delay: **60s**

Descubrimiento de clusters: API v2

-  Disponible para diferentes protocolos
-  Utilizando Protocol Buffers
-  Nos permite configurar cualquier recurso
-  Cambios de configuración en caliente



Ingress

Ingress

Envoy can act as an edge reverse proxy

An edge server is any server that resides on the “edge” between two networks, typically between a private network and the internet.

A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and **directs client requests to the appropriate backend server**. A reverse proxy provides an **additional level of abstraction and control** to ensure the smooth flow of network traffic between clients and servers.

Ingress

```
route_config:
  name: ingress_route
  virtual_hosts:
  - name: service
    domains:
    - "*"
    routes:
    - match:
        prefix: "/pet/"
        route:
          cluster: pet
          prefix_rewrite: "/"
    - match:
        prefix: "/petstore/"
        route:
          cluster: petstore
          prefix_rewrite: "/"
```

Mapeamos el cluster de cada servicio a un path con el mismo nombre

Con el prefix_rewrite evitamos el envío del mapeo al servidor



Gestión de error

Gestión de error

Circuit breaking

Control de rating, límite de carga, peticiones concurrentes

Gestión de timeout y reintentos

Timeouts a nivel general, a nivel de ruta

Outlier Detection

El circuit breaking de Martin Fowler

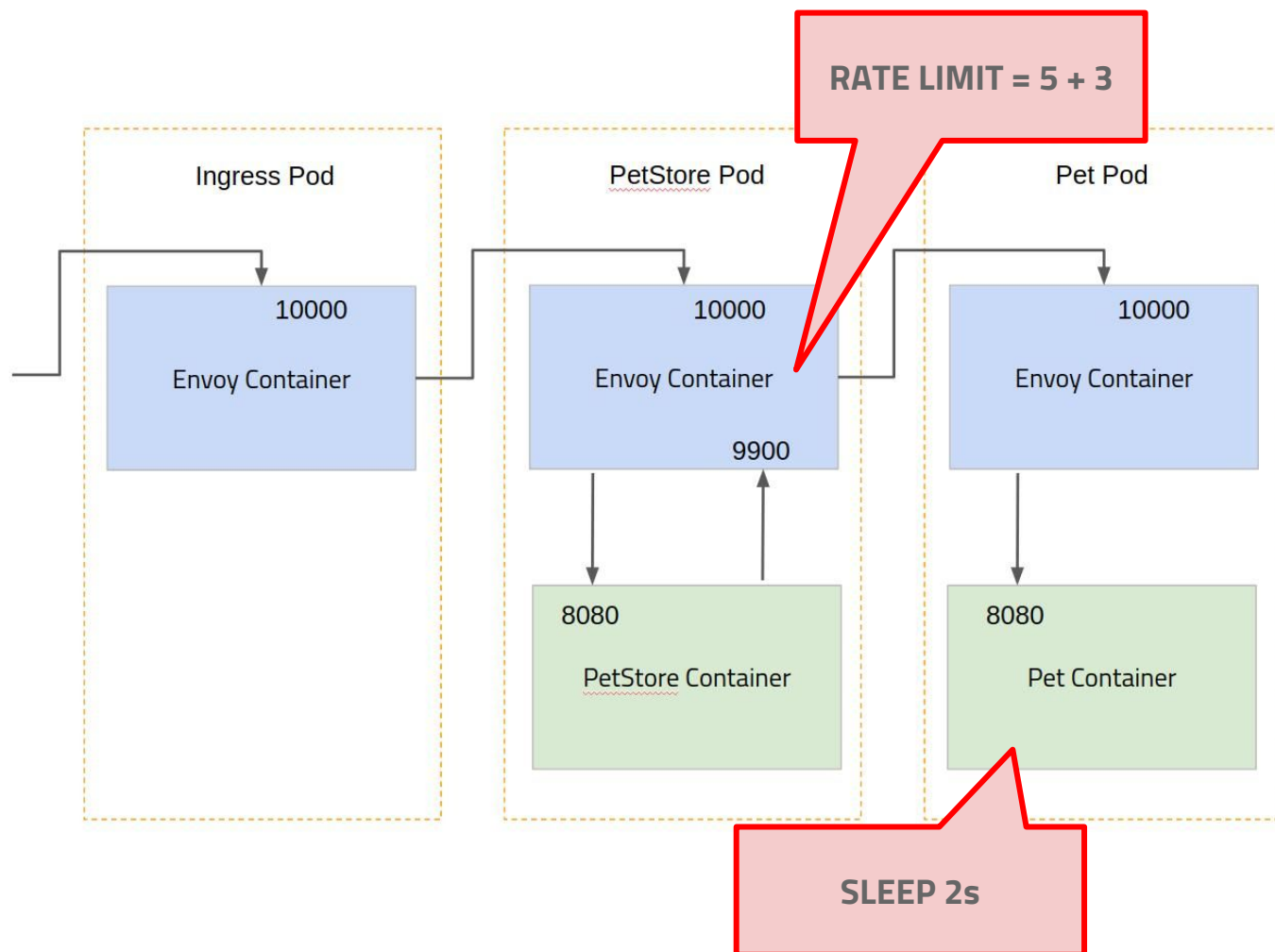
Retirada de nodos en función de errores 500, errores de gateway, latencias, errores consecutivos, porcentaje de errores, healthy_panic_threshold

Gestión de error

Fortio (Φορτίο)



Circuit breaking



in petstore configuration

- name: **pet**

connect_timeout: 0.50s

type: eds

lb_policy: round_robin

circuit_breakers:

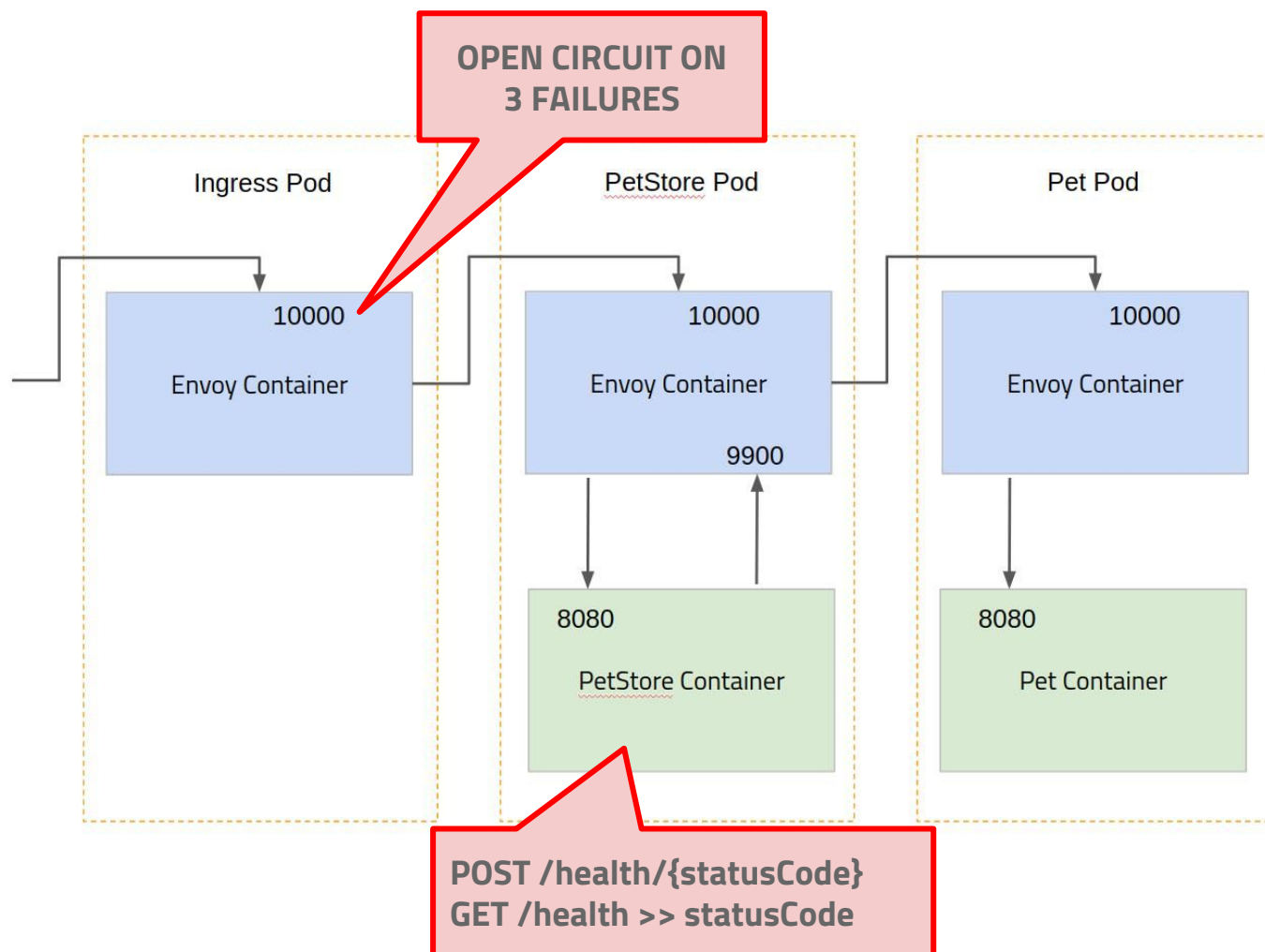
thresholds:

- priority: DEFAULT

max_pending_requests: 3

max_connections: 5

Outlier detection



in ingress configuration

outlier_detection:

consecutive_5xx: 3

interval: 10s

base_ejection_time: 10s

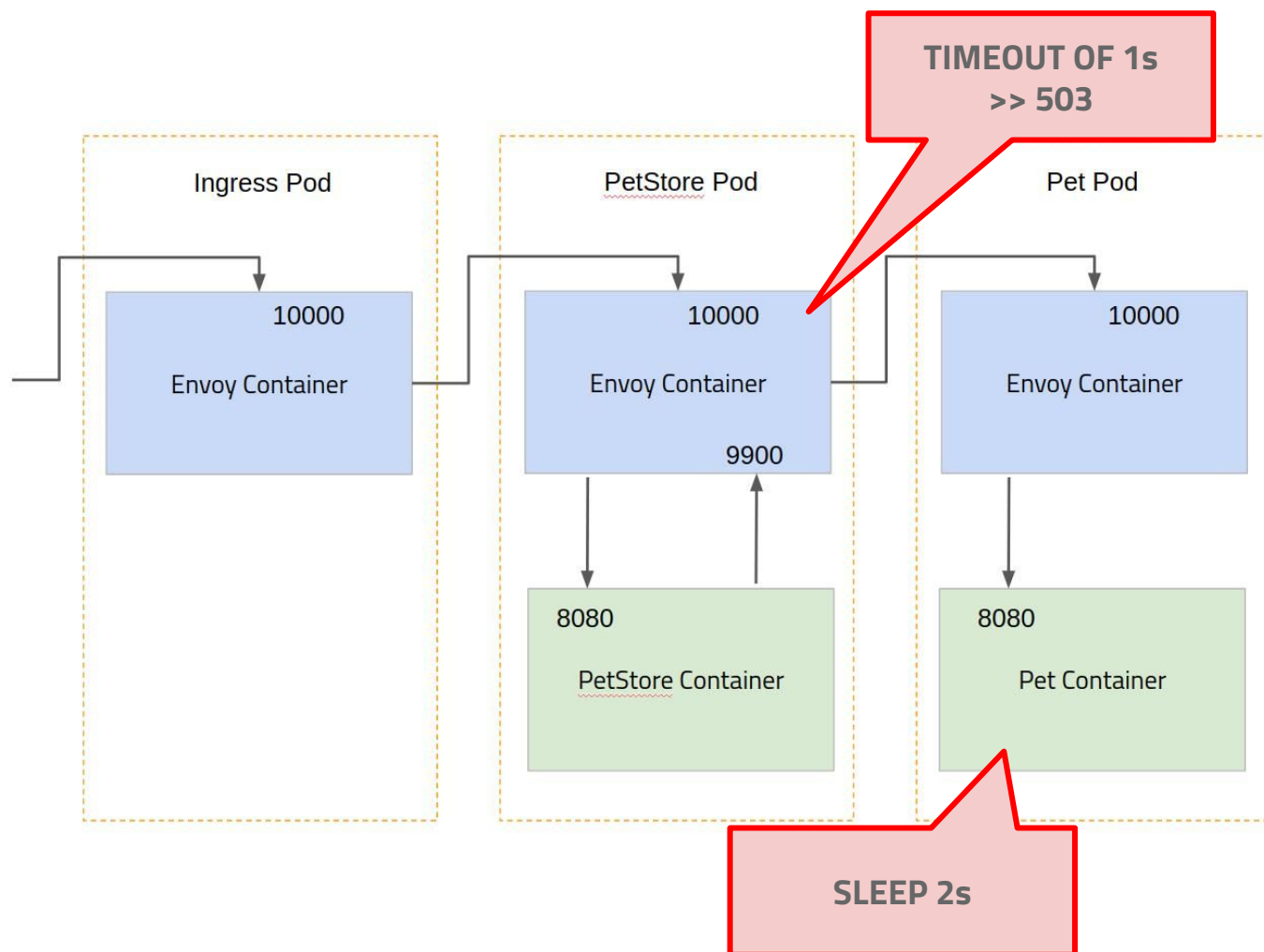
max_ejection_percent: 100

common_lb_config:

healthy_panic_threshold:

value: 0.0

Timeouts



in petstore configuration

routes:

- match:

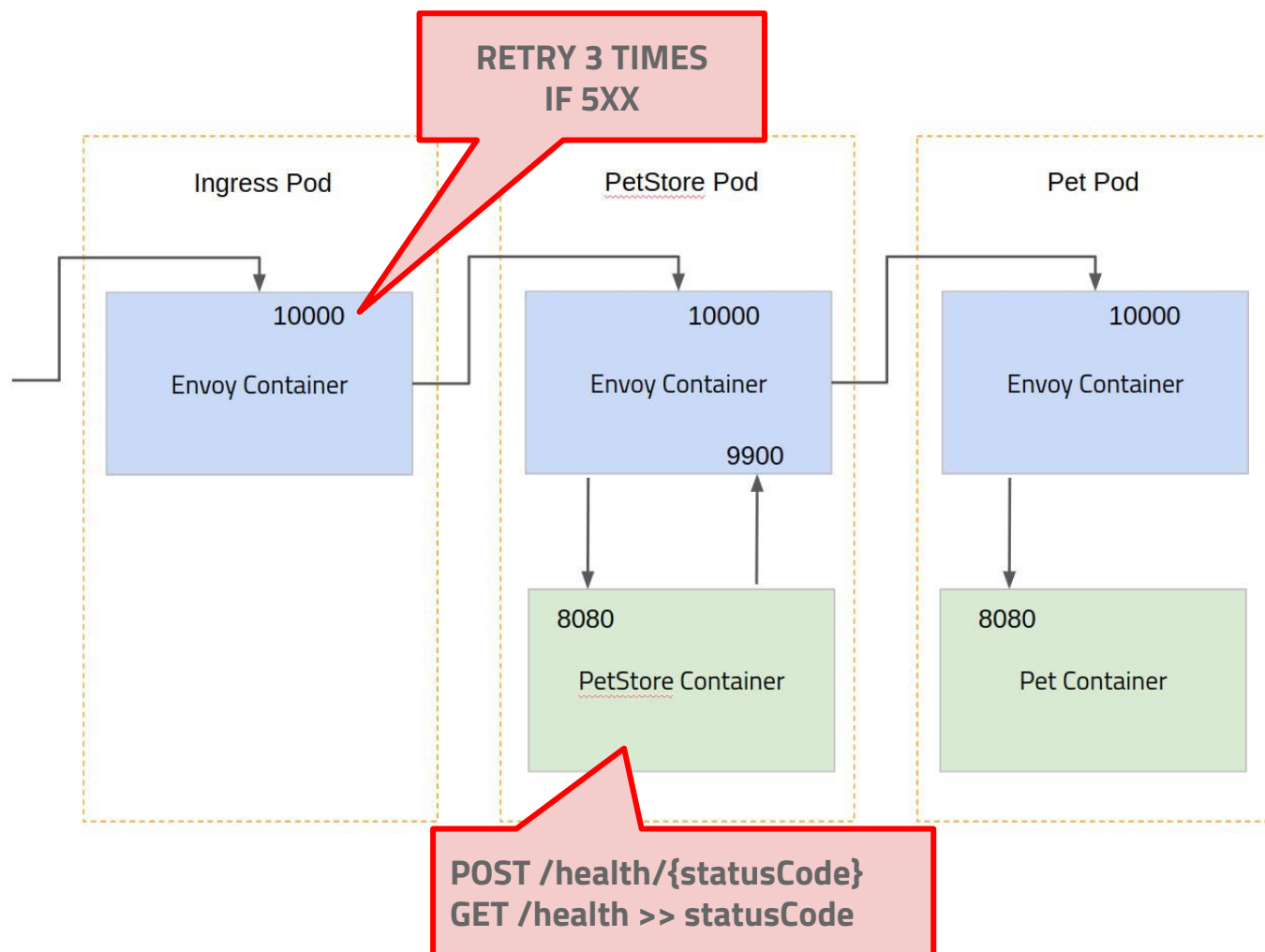
prefix: "/"

route:

cluster_header: ":authority"

timeout: 1s

Retries



in ingress configuration

routes:

- match:

prefix: "/"

route:

cluster_header: ":authority"

retry_policy:

retry_on: 5xx

num_retries: 3



Trazabilidad distribuida

Trazabilidad distribuida

```
filter_chains:  
  - filters:  
    - name: envoy.http_connection_manager  
      config:  
        tracing:  
          operation_name: egress
```

Configuración
del filtro

Trazabilidad distribuida con Zipkin

```
filter_chains:  
  - filters:  
    - name: envoy.http_connection_manager  
      config:  
        tracing:  
          operation_name: egress
```

Configuración
del filtro

```
- name: zipkin  
  connect_timeout: 1s  
  type: strict_dns  
  lb_policy: round_robin  
  hosts:  
    - socket_address:  
      address: zipkin  
      port_value: 80
```

Definición del
cluster de
zipkin

Configuración
de cliente
Zipkin

```
tracing:  
  http:  
    name: envoy.zipkin  
    config:  
      collector_cluster: zipkin  
      collector_endpoint: "/api/v1/spans"
```



Trazabilidad distribuida con Jaeger

```
filter_chains:
```

```
- filters:
```

```
- name: envoy.http_connection_manager
```

```
config:
```

```
tracing:
```

```
operation_name: egress
```

Configuración
del filtro

Configuración del
cliente nativo de
jaeger

```
tracing:
```

```
http:
```

```
name: envoy.dynamic.ot
```

```
config:
```

```
library: ../libjaegertracing_plugin.so
```

```
config:
```

```
service_name: petstore
```

```
sampler:
```

```
type: const
```

```
param: 1
```

```
reporter:
```

```
localAgentHostPort: jaeger:6831
```

```
wget -O
```

```
../libjaegertracing_plugin.so
```

```
https://github.com/jaegertracing
```

```
/../libjaegertracing\_plugin.li
```

```
nux_amd64.so
```

Instalación del
agente de
jaeger





Seguridad

Seguridad

TLS

Configuración de certificados local o remota
(mediante Secret Discovery Service SDS),
terminación y inicio de SSL, autenticación mutua,
filtros de ssl

RBAC

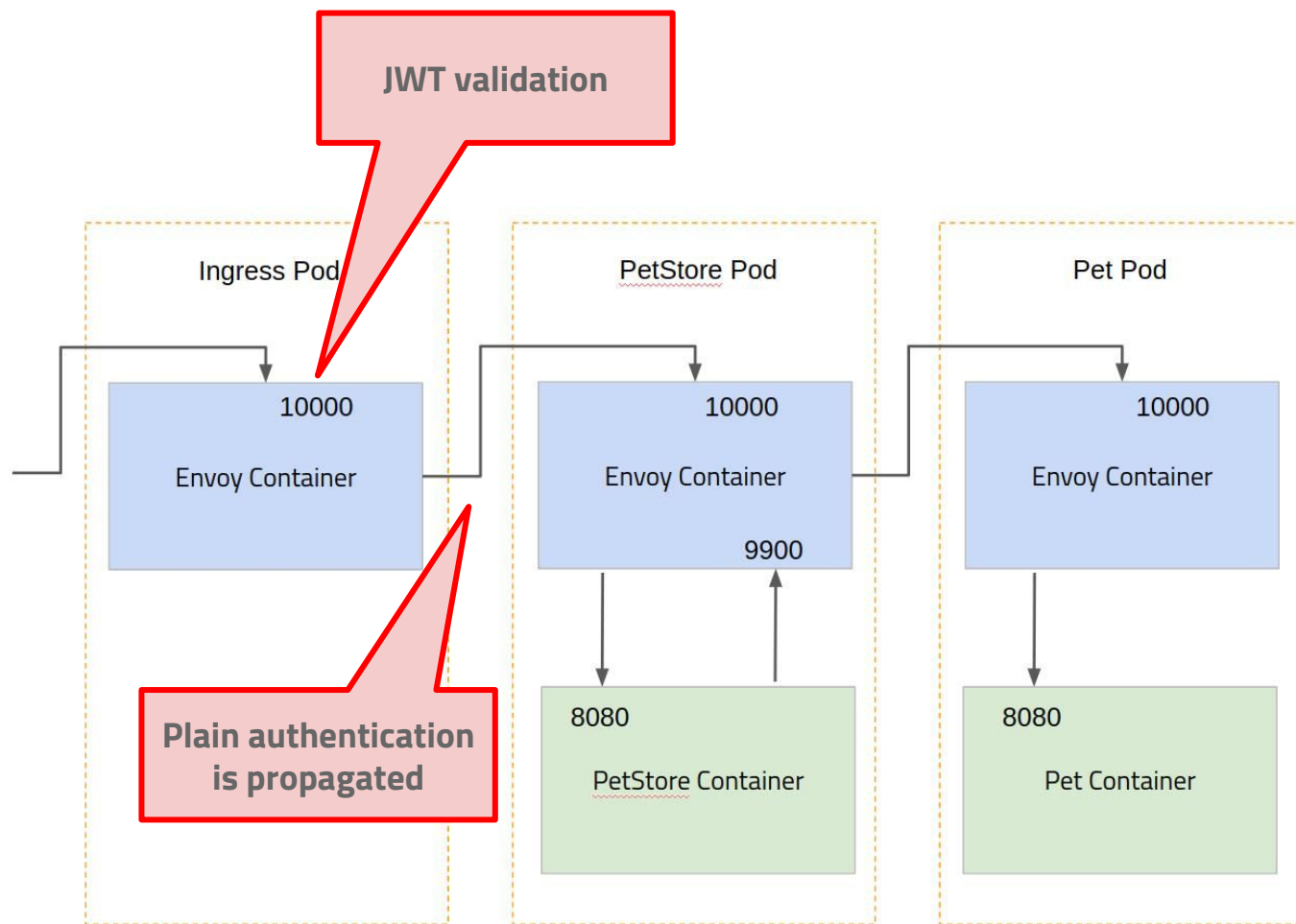
Filtro de autorización, Role Based Access Control
por ruta

End user authentication



Validación de tokens JWT, carga de fichero de
claves remoto, fichero local o inline

Validación de tokens JWT



```
- name: envoy.filters.http.jwt_authn
  config:
    providers:
      jwt_provider:
        issuer: test
        local_jwks:
          inline_string: ...public key...
        forward: true
        forward_payload_header: "plain-auth"
    rules:
      - match:
          prefix: /
        requires:
          provider_and_audiences:
            provider_name: jwt_provider
            audiences: pets.read
```


Necesidad de un plano de control



Necesidad de plano de control

- Hemos tenido que poner diversas configuraciones varias veces, una en cada pieza
- Ninguna pieza no sabe nada de las demás, de hecho podría ni saber de su existencia

Necesidad de plano de control

- Hemos tenido que poner diversas configuraciones varias veces, una en cada pieza
- Ninguna pieza no sabe nada de las demás, de hecho podría ni saber de su existencia

TRABAJO ENGORROSO

Necesidad de plano de control

- Hemos tenido que poner diversas configuraciones varias veces, una en cada pieza
- Ninguna pieza no sabe nada de las demás, de hecho podría ni saber de su existencia

TRABAJO ENGORROSO

"The control plane takes a set of **isolated stateless sidecar proxies** and turns them into **a distributed system.**"

"Ultimately, the **goal of a control plane is to set policy** that will eventually be enacted by the data plane. More advanced control planes **will abstract more of the system from the operator** and require less handholding"

Matt Klein, Engineer @lyft

Conclusiones



Conclusiones



PROS



Proxy muy **ligero**



Gestión avanzada de **circuit breaking**



Integración con **múltiples productos**



Soporte a validación de **JWT**



Carga en caliente



“Retroalimentación” funcionalidades istio-envoy

CONTRAS

Extensible, pero **difícil de extender***



Necesita de **componentes extra** para algunas de sus funcionalidades



Complejidad de configuración



No tiene **plano de control**



