LINKERD

Service mesh - Linkerd a fondo

Luis Mora

paradigma

Luis Mora Medina

Arquitecto software @Paradigma

@luismoramedina

1 introducción a service mesh

“The Hardest Part of Microservices: Calling Your Services”

Christian Posta

Aplicaciones deslocalizadas y **distribuidas** en numerosos componentes que hablan con numerosos protocolos de red, **arquitecturas complejas**, que crecen en número de **dependencias**

- Service Discovery
- Enrutamiento y balanceo de carga
- Gestión de errores: time-out, reintentos, circuit-breaker
- Securización y control de acceso (autenticación / autorización)
- Comprobación de disponibilidad (Health check)
- Métricas, monitorización, logging y trazabilidad distribuidos

*"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

¿Es nuevo este concepto?

NETFLIX OSS

- spring-cloud-netflix-hystrix
- spring-cloud-netflix-ribbon
- spring-cloud-netflix-eureka
- spring-cloud-netflix-zuul
- spring-cloud-sleuth
- spring-cloud-zipkin

VERT.X

- vertx-circuit-breaker
- vertx-service-discovery
- vertx-dropwizard-metrics
- vertx-zipkin

finagle
de Twitter

Stubby

- A alto nivel, la responsabilidad del service mesh es **asegurar que las solicitudes sean entregadas** en las conexiones entre los servicios de nuestra arquitectura distribuida
- Propicia **arquitecturas políglotas**
- Permite que los **equipos de desarrollo se centren en la funcionalidad** de sus aplicaciones y que la lógica de negocio no se difumine con el código de instrumentación de las conexiones

Sidecar

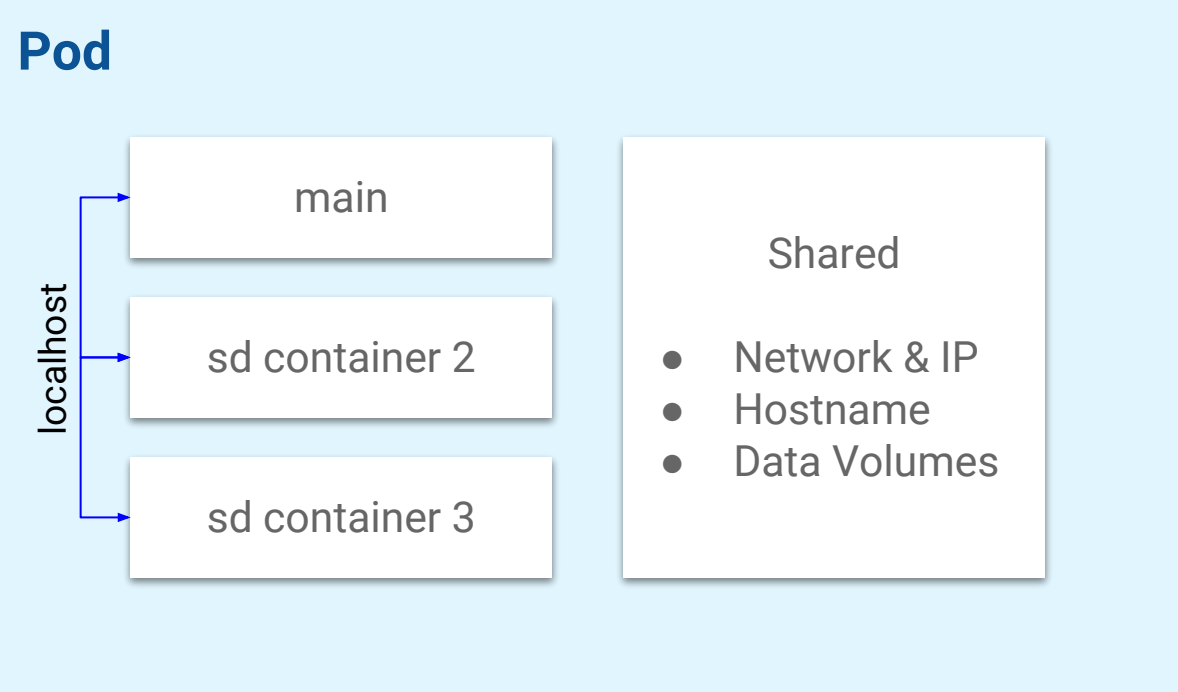
Ofrece ayuda a la aplicación principal

- Telemetría y monitorización
- Logging
- Proxy
- Protocol transformation
- Configuración
- Health
- Auditoría
- ...

Aplicación principal

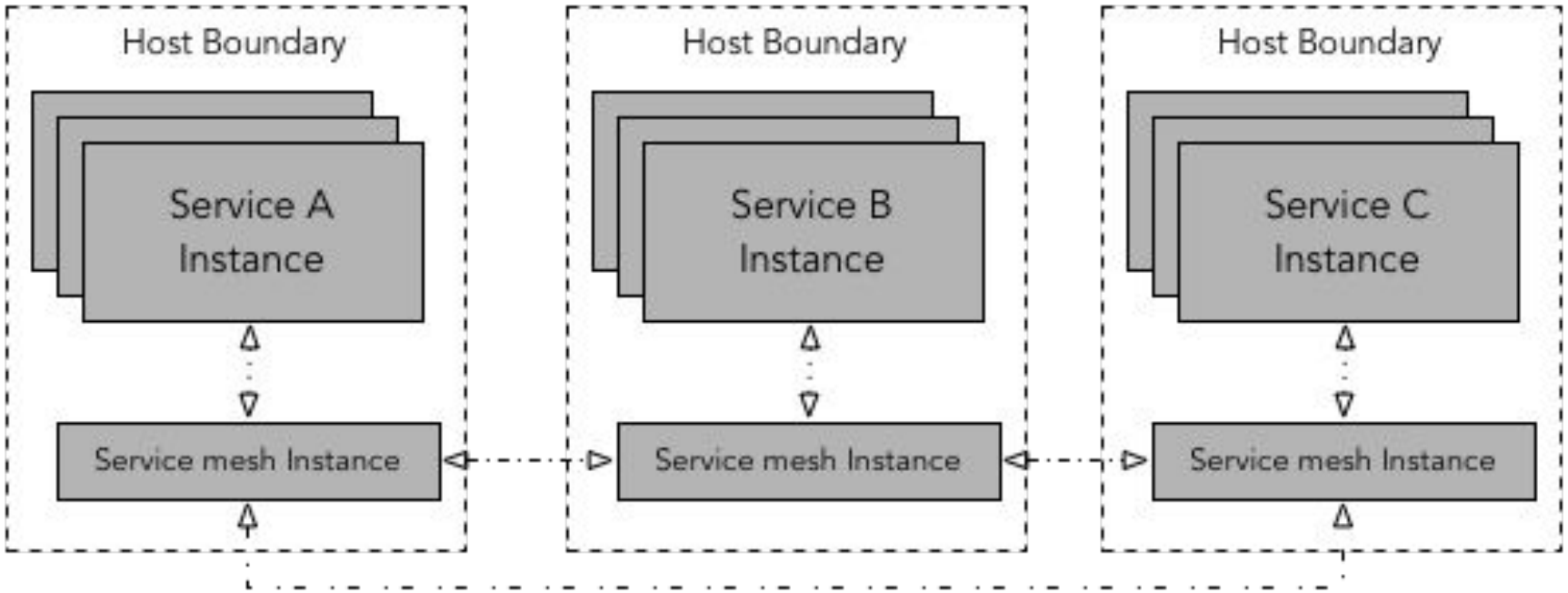
- Core de la aplicación
- Tiene la funcionalidad y la ofrece por sí sola
- Corre en un proceso o contenedor diferente

POD: Unidad de computación mínima que puede ser desplegada en Kubernetes



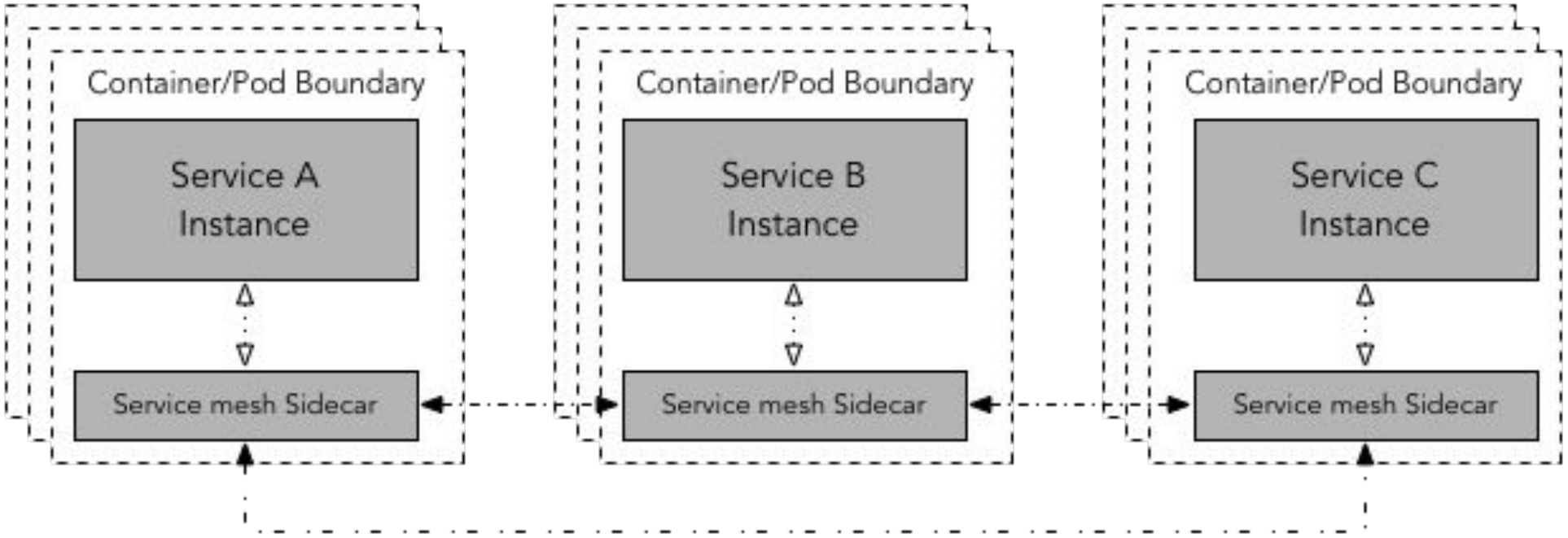
Per host

- 1 proxy en cada host
- n..1 servicios - proxy
- menor gasto de memoria
- puede llegar a ser un SPOF



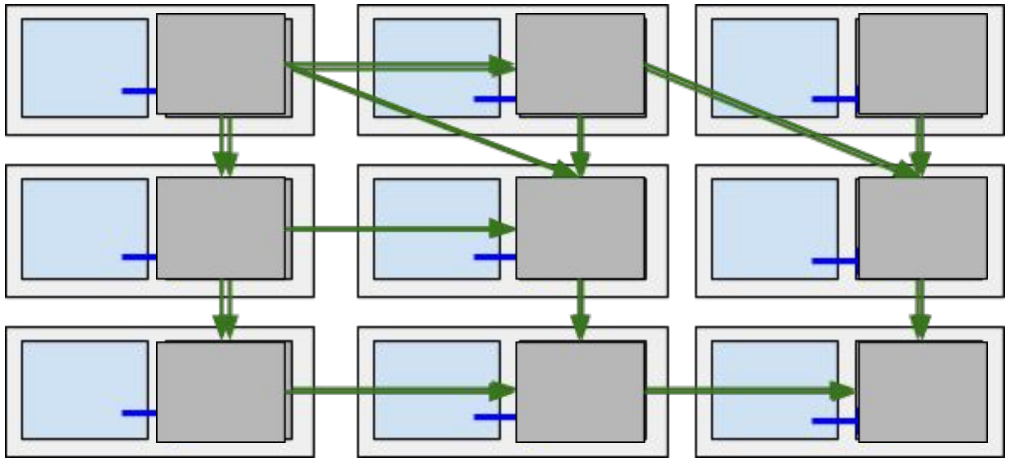
Sidecar

- 1 proxy por cada servicio
- 1..1 servicios - proxy
- más versátil
- mejor rendimiento



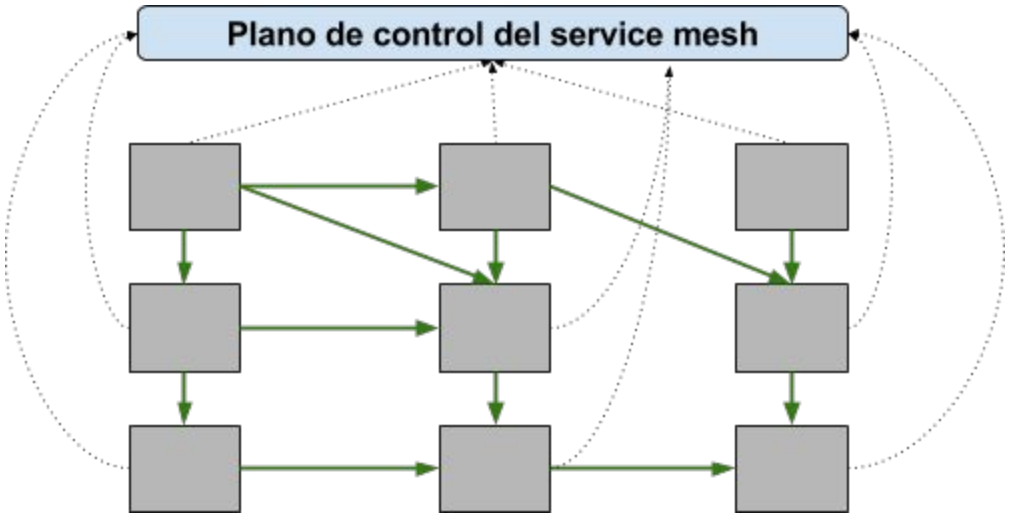
Los service mesh suelen tener dos partes (1/2):

- El plano de datos (maya de proxies encargados de gestionar conexiones entre componentes)



Los service mesh suelen tener dos partes (2/2):

- El plano de control (políticas, gestión, configuración, monitorización del data plane)



2 qué es Linkerd?

/linkerd-DEE/

What is Linkerd?

*"Linkerd is an open source **network proxy** designed to be deployed as a service mesh: a dedicated **layer for managing, controlling, and monitoring service-to-service communication** within an application."*

- Linkerd (l5d) aparece en **2016**
- Open-source con Soporte comercial (**Bouyant**)
- Desarrollado en scala y ejecutado en la **JVM**
- Basado en *Finagle* y *Netty*
- Soporte a múltiples clouds, herramientas y protocolos
- "Performante" p99 (99% peticiones) < 1 milisegundo
- Extensible
- Dentro de la iniciativa **CNCF**
- Nuevo producto **Linkerd2** (f.k.a. Conduit)

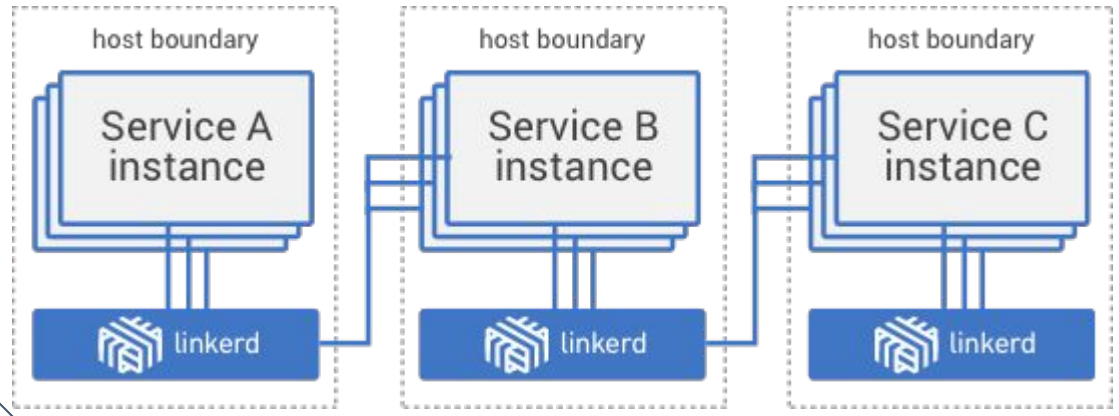
BUOYANT

CLOUD NATIVE COMPUTING FOUNDATION

LINKERD

CONDUIT

- **Enrutamiento** dinámico
  - Blue green
  - Canary
- **Balanceo** de carga avanzado
  - Least loaded
  - Round robin, etc.
- Gestión de errores
  - **Circuit breaking**
  - **Reintentos** para llamadas fallidas
  - Límite de carga
- **Autodescubrimiento**
- Seguridad de red service-to-service (TLS)
- **Trazabilidad** distribuida
- **Monitorización** y métricas
- **Sidecar** o proxy **por host**
- Multi protocolo
  - Http1
  - Http2 (Grpc)
  - Thrift



Per host

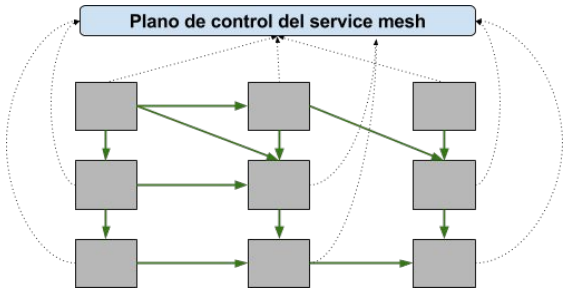
- 1 Linkerd en cada host
- n..1 servicios - linkerd
- menor gasto de memoria
- puede llegar a ser un SPOF

Sidecar

- 1 Linkerd por cada servicio
- 1..1 servicios - linkerd
- más versátil
- mejor rendimiento

https://linkerd.io/1/advanced/deployment/

¿Tiene Linkerd un plano de control?



- Los “**namers**” son los encargados de **resolver** nombres lógicos de servicios y transformarlos en una dirección “física”
- Delegation tabs (**dtabs**) son las reglas de **enrutado** para resolver los path lógicos
- Existen numerosos namers: k8s, marathon, consul, filesystem, zookeeper, etc.
- Toda esta configuración puede estar en cada proxy o puede estar en un servicio llamado “**namerd**”, que permite cambios en caliente

/meetup/goto

Logical path

```
/meetup     => /kschool;
/kschool    => /madrid/magallanes/1;
/madrid     => /spain/madrid;
```

Finagle delegation table

/meetup/goto

/kschool/goto

/madrid/magallanes/1/goto

/spain/madrid/magallanes/1/goto

Physical name

```
/meetup     => /kschool;
/kschool    => /madrid/magallanes/1 * 0.9 & /madrid/sol/5 * 0.1;
/madrid     => /spain/madrid;
```

Canary or AB deployments

```
/svc/local   => /$/inet/127.0.0.1/8080
```

/$ a classpath namer

/$/inet namer

```
/svc => /#/io.l5d.fs
```

A configurable namer

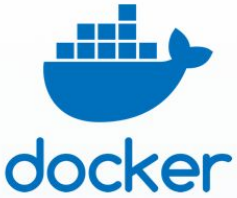
file system namer (not for production)

+

```
192.0.2.220 8080
192.0.2.210 8080 * 2.0
```

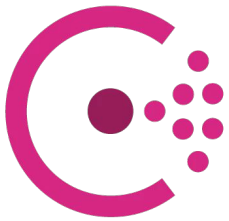
Kubernetes

DC/OS

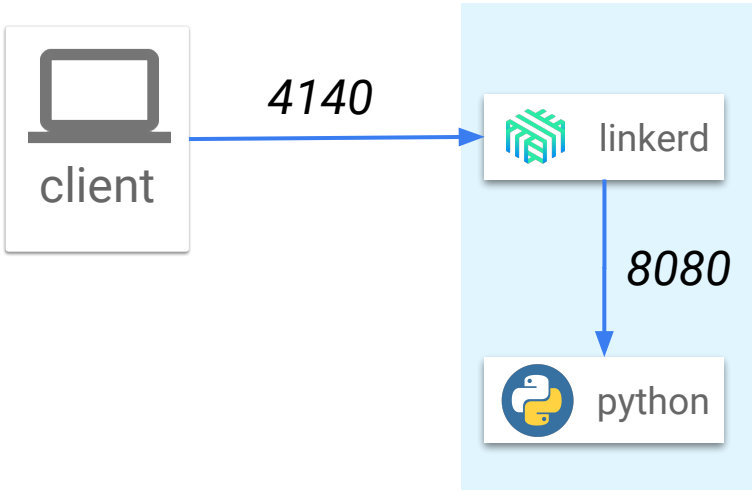


Docker

AWS ECS

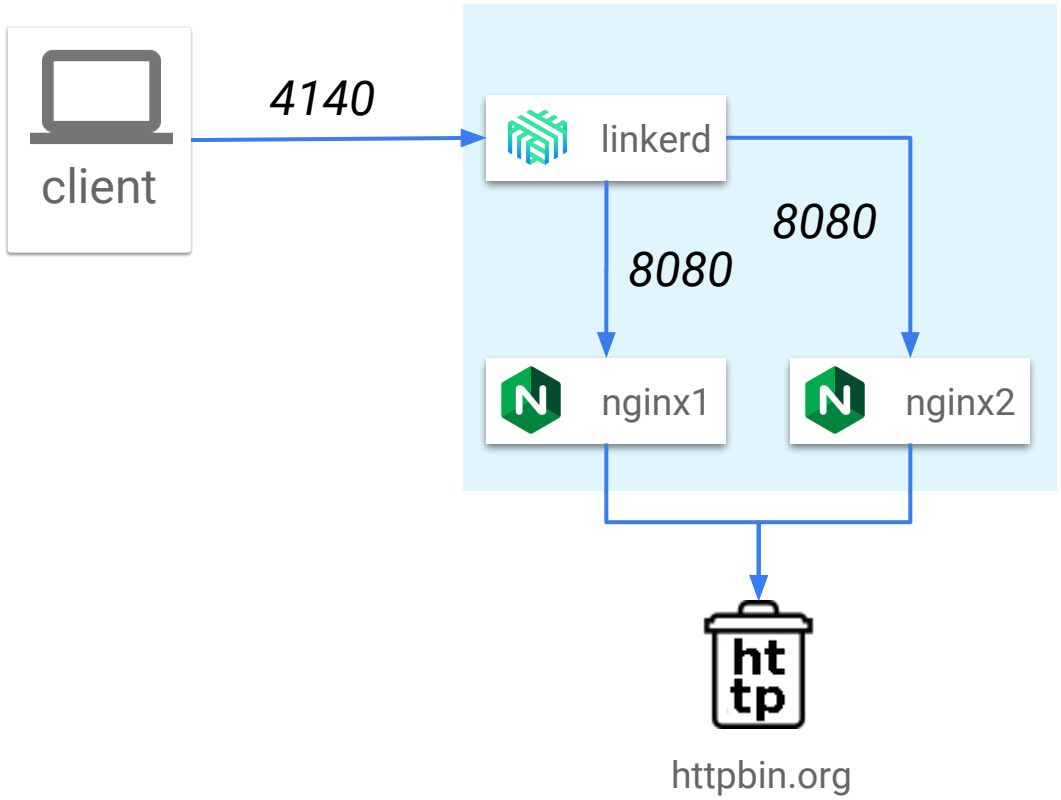


3 playground




```
admin:
  port: 9990
  ip: 0.0.0.0

routers:
- protocol: http
  dtab: /svc/* => /$/inet/python/8080;
  servers:
  - port: 4140
    ip: 0.0.0.0
```

- **Round Robin**
- P2C (Power of two choices) **Least loaded**: coge 2 nodos del pool de manera aleatoria y envía la petición al menos cargado de la tupla (el que menos peticiones pendientes tiene). **Por defecto.**
- P2C (Power of two choices) **Peak EWMA**: variación del anterior pero que tiene en cuenta los servidores con mayores latencias para dejarles tiempo para su recuperación
- Aperture Least Loaded
- Heap Least Loaded



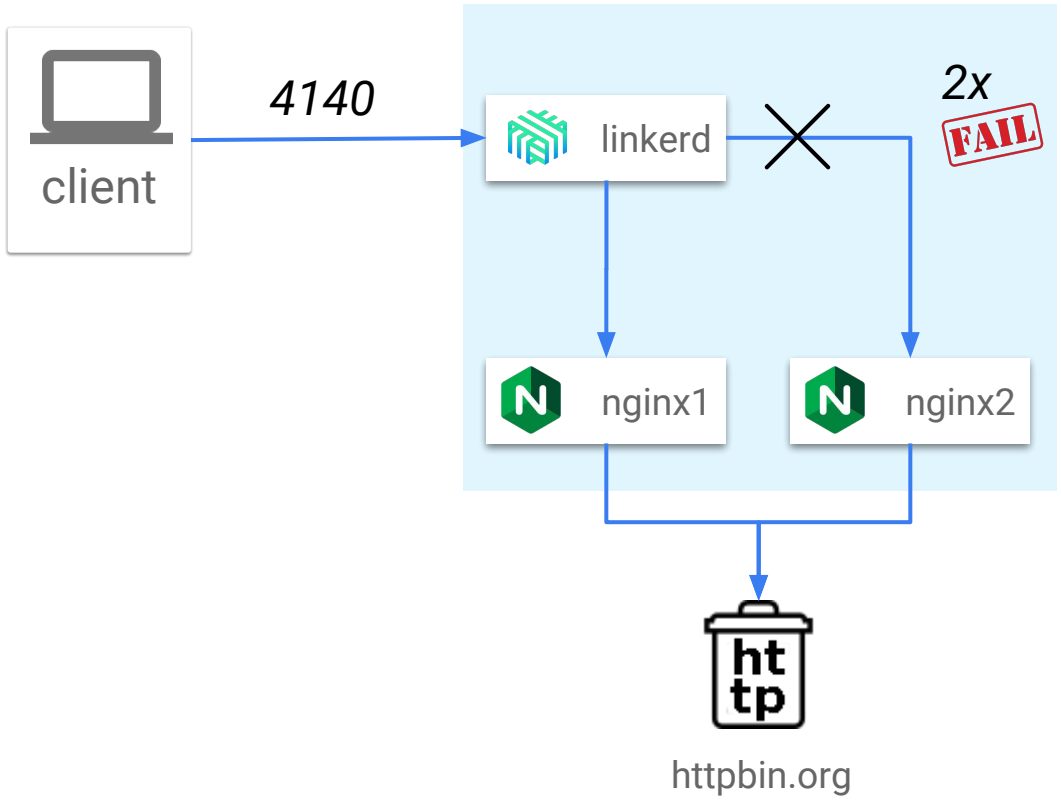

```
namers:
- kind: io.l5d.fs
  rootDir: disco

routers:
- protocol: http
  dtab: /svc => /#/io.l5d.fs/nginx;
  servers:
  - port: 4140
    ip: 0.0.0.0
```

- Linkerd **elimina nodos del pool** de balanceo cuando no están “sanos”
  - A nivel de conexión (**failFast**)
  - Basado en fallos (**failureAccrual**)
    - Número de **fallos consecutivos**
    - **Porcentaje** de peticiones erróneas (contabilizando las últimas peticiones o usando una ventana de tiempo)

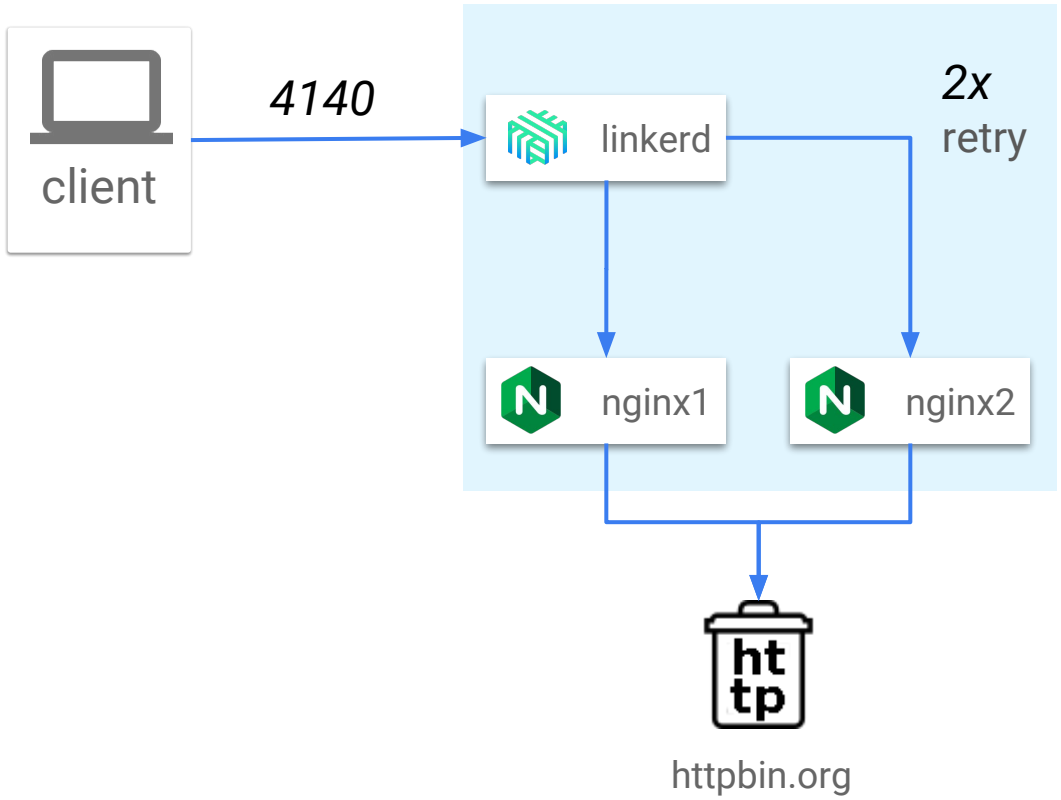
- Existe una **política de reinserción en el pool**
  - tiempo constante
  - jittered (tiempo incremental)

- El pool de nodos no se puede quedar vacío, se siguen enviando peticiones a alguno de los nodos aunque todos hayan sobrepasado el umbral de “retirada”

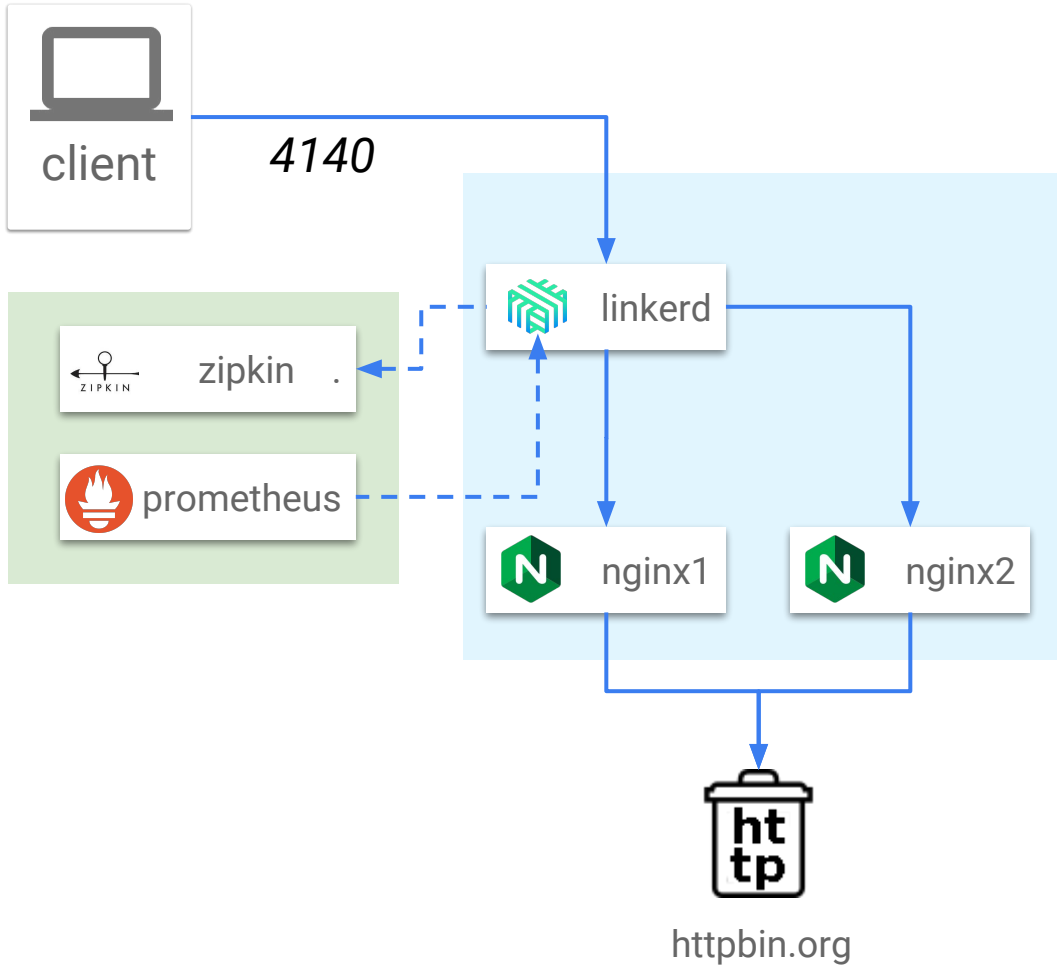
- Por defecto se eliminará del pool un nodo a los 5 fallos consecutivos




```
routers:
- protocol: http
  ...
  client:
    failureAccrual:
      kind: io.l5d.consecutiveFailures
      # remove from the pool on 2 FAILs
      failures: 2
      backoff:
        kind: constant
        # wait 15s after reinserting
        ms: 15000
```

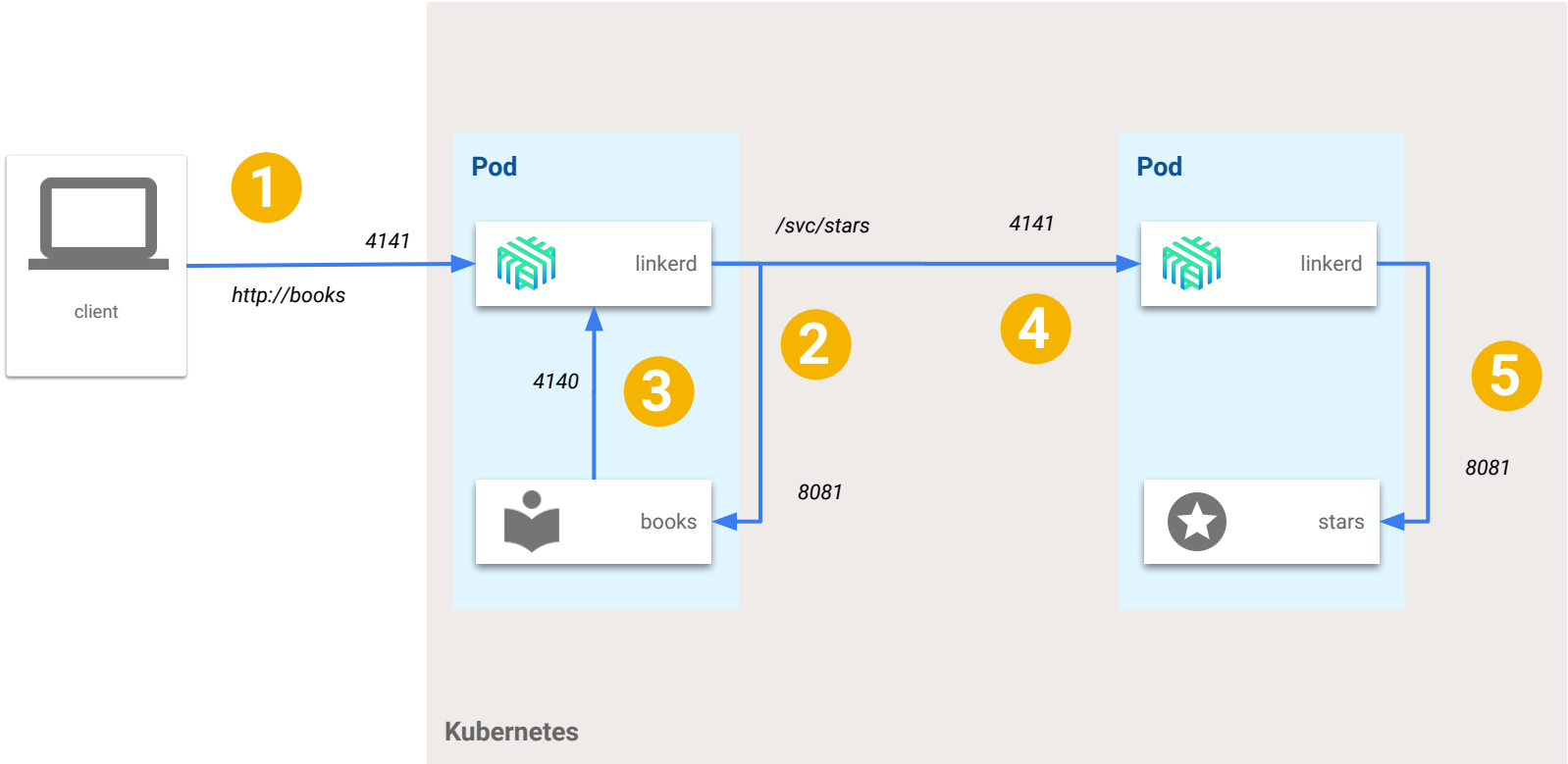



```
service:
 responseClassifier:
   kind: io.l5d.http.retryableRead5XX
 retries:
   budget:
     minRetriesPerSec: 0
     # every failure is retried twice
     percentCanRetry: 2
   backoff:
     kind: constant
     ms: 300
```

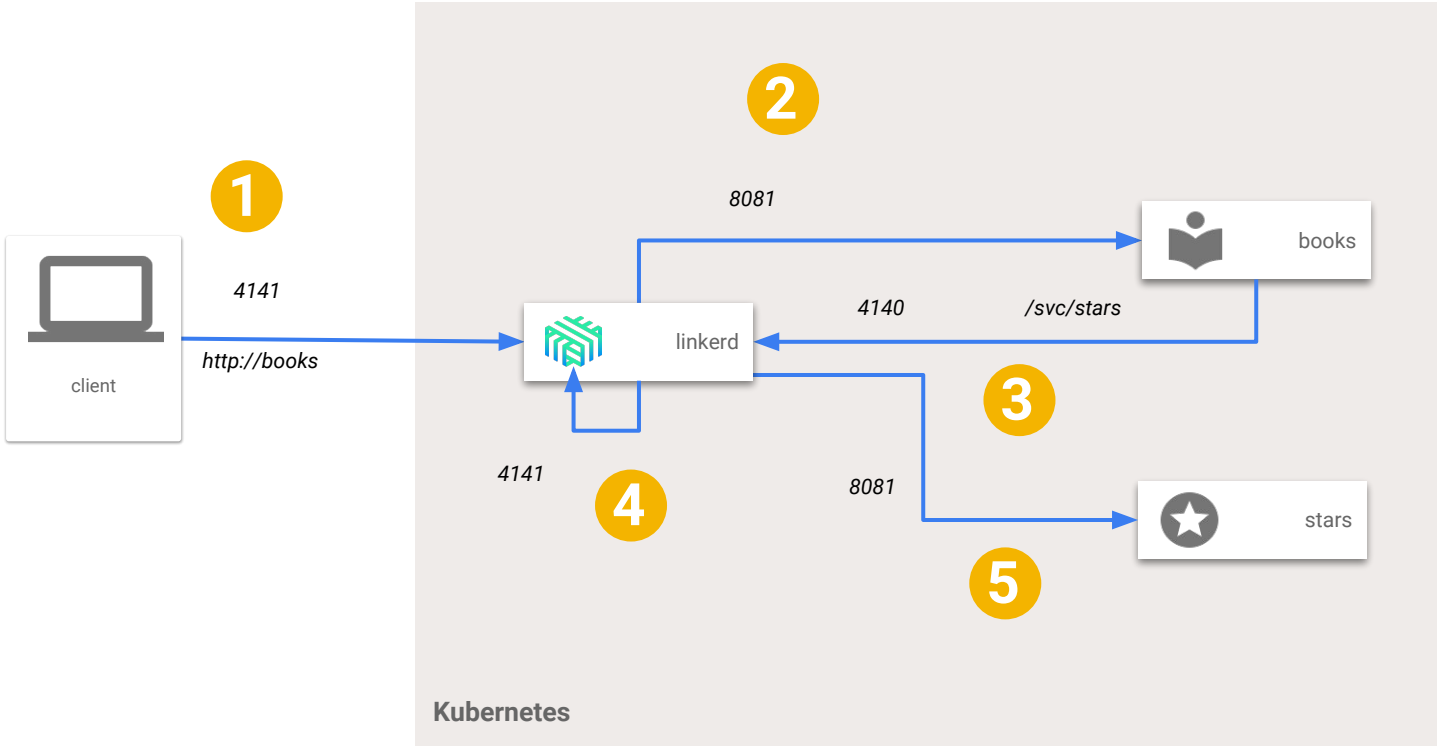



```
telemetry:
- kind: io.l5d.recentRequests
  sampleRate: 1.0
- kind: io.l5d.prometheus
- kind: io.l5d.zipkin
  host: zipkin
  port: 9410
  sampleRate: 1.0
```

Modo Sidecar



Modo Proxy por host



Y todo esto... ¿sin modificar nuestras aplicaciones?

- Nuestros clientes http deben usar Linkerd como proxy* (**HTTP_PROXY**, **http.proxyHost, http.proxyPort**)
- Se deben propagar, en las llamadas a otros servicios upstream, las cabeceras de Linkerd (**l5d-ctx-***)

* Linkerd puede actuar como proxy transparente en k8s, modificando las iptables en un init container

4 conclusiones

PROS

- Tiene **soporte comercial**
- Soporte a **múltiples entornos** y clouds
- Instalación en arquitectura **sidecar o nodo por host**
- Integración con **múltiples productos**
- Producto **robusto**, sólido y rápido
- Abierto a la implementación de plugins con los lenguajes de la JVM

CONTRAS

- **No** tiene un **plano de control** avanzado
- El **elevado footprint** del proxy dificulta su instalación en modo sidecar
- La funcionalidad de **circuit breaker** tiene algunas **deficiencias**
- Se echan de menos **funcionalidades** extra de **seguridad**

Linkerd 1	Linkerd 2
No tiene un plano de control diferenciado, se queda en el plano de datos	Tiene un **plano de control**
Diferentes modos de instalación (**Sidecar proxy/**Proxy **per host**)	Instalación en **Sidecar** dentro de pods
Proxy con footprint medio (JVM) ~150 megas	**Proxy ultraligero (Rust)**
Múltiples integraciones con productos de terceros	Es un producto relativamente nuevo, menos integraciones y menos funcionalidad (**no circuit breaker, no routing**, etc.)
Instalación y configuración es artesanal	Cuenta con un **cli** para realizar instalación y configuración del sistema
Soporte a múltiples plataformas (**AWS, DC/OS, Kubernetes, Docker, On-Premise**)	Diseñado inicialmente para funcionar en **Kubernetes** (>=1.9)


¿Preguntas?
https://github.com/luismoramedina/linkerd-playground
https://github.com/luismoramedina/linkerd-microservices
paradigma