Upgrading your service mesh to Linkerd 2

Lessons learned
Tilen Faganel
Senior Consultant
@ OpenCredo

You can find me on WeChat via @tfaganel and twitter via @TilenFaganel
Agenda

A brief intro

Evolution of service meshes and Linkerd

Redesign and upgrade

Lessons learned
“Service mesh is an approach and a dedicated infrastructure layer for operating a secure, fast and reliable microservices ecosystem.”
Low-latency infrastructure layer

A new paradigm

Layer 7 network exclusively for applications

A way to increase the observability, resilience and security

Extension to Kubernetes

Software defined networking

Magic?
An example
An example

Payments \[\Rightarrow\] Ledger \[\Rightarrow\] Fraud System
An example
An example

- Service discovery
- Load balancing
- Circuit Breaking
- Retries
- Authentication & Authorization
- Automatic
An example

Payments → Ledger → Fraud System

Resiliency
An example
An example

Payments ➔ Ledger ➔ Fraud System
An example

Payments  Core Banking  Fraud System
An example

Payments ────→ Core Banking ────→ Fraud System
An example
An example

Payments → Core Banking

Core Banking → Fraud System

@TilenFaganel
Service mesh

- Service discovery
- Load balancing
- Circuit Breaking
- Retries
- Authentication & Authorization
- Automatic
A collective of smart configurable autonomous proxies
Which proxy to use?
Linkerd

- Single all-in-one network proxy - Finagle
- Runs in the JVM
- Routing policies
- Supports most resiliency requirements
- Generic pluggable design
- Single config
service:
responseClassifier:
  kind: io.l5d.http.retryable5XX
retries:
budget:
  minRetriesPerSec: 10
  percentCanRetry: 0.5
  ttlSecs: 15
backoff:
  kind: jittered
  minMs: 10
  maxMs: 2000
client:
  kind: io.l5d.static
  configs:
  - # Use HTTPS if sending to port 443
    prefix: "$/io.buoyant.rinet/443/{service}"
    tls:
      commonName: "{service}"
      disableValidation: true
  - # Use HTTPS if sending to remote cluster linkerd
    prefix: "%/io.l5d.k8s.daemonset"
    loadBalancer:
      kind: ewma
      maxEffort: 10
      decayTimeMs: 30000

dtab: |
  /ph  => /$/io.buoyant.rinet ;
  /svc => /ph/443 ;
  /svc => /$/io.buoyant.porthostPfx/ph ;
  /k8s => /#/k8sIncoming ;
  /portNsSvc => /#/portNsSvcToK8s ;

  /host => /$/io.buoyant.porthostPfx/portNsSvc ;
  /host => /portNsSvc/http/default ;
  /host => /portNsSvc/http ;

  /svc => /$/io.buoyant.hostportPfx/legacy ;
  /svc => /$/io.buoyant.http.domainToPathPfx/host ;
Architecture - Take 1

On Premise - Virtual Machines

Cloud - Instances

@tfaganel  @TilenFaganel
Architecture - Take 1

On Premise - Virtual Machines

S1

S2

V1

Cloud - Instances

S3

S4

AWS

Kubernetes - Nodes

?
Architecture - Take 1

Linkerd as a DaemonSet
Problems

• Siloed instances on every node
• Large resource consumption
• Proxy per node instead of instance
• Complex configuration and its updates
• Disjointed monitoring
• No proper mTLS support
• High developer friction
What’s next?

Proxy
What’s next?

Proxy  Proxy  Proxy  Proxy

Control plane
Control plane

- Manages and configures the proxies
- Standard stateless deployment
- Public API
- Collects & exposes telemetry
- Enforces policies
- Issues certificates
- Fully cloud-native
Data plane

1. Proxy requests
Let’s plan a migration
Architecture - Take 3

Kubernetes - Nodes

Kubernetes - Pods

Control

@tfaganel  @TilenFaganel
apiVersion: linkerd.io/v1alpha1
kind: ServiceProfile
metadata:
  name: ledger.default.svc.cluster.local
  namespace: default
spec:
  routes:
  - condition:
      method: GET
      pathRegex: /transactions
      name: GET /transactions
      isRetryable: true
  - condition:
      method: POST
      pathRegex: /transactions
      name: POST /transactions
      timeout: 100ms
  - condition:
      method: GET
      pathRegex: /transactions/[\^/]*/
      name: GET /transactions/{id}
Goals

• No required developer involvement or friction
• No disruption
• Minimal changes
```yaml
env:
  - name: NODE_NAME
    valueFrom:
      fieldRef:
        fieldPath: spec.nodeName
  - name: http_proxy
    value: $(NODE_NAME):4140
```
Admission Webhooks

- Alter pod definitions before scheduling
- Used by Linkerd 2
- Run in Kubernetes as any deployment
- Completely transparent
The plan

1. Begin
2. Trigger
3. Deploy control plane
4. Stop old AW
5. Start new AW
6. Rolling redeploy
7. Cleanup

k8s-infrastructure

@tfaganel  @TilenFaganel
The plan

1. **Deploy Control Plane**
2. **Stop old AW**
3. **Start new AW**
4. **Rolling redeploy**
5. **Cleanup**

Rollback if needed
Fully automate your infrastructure from the start
Let the platform do the heavy lifting
“Service mesh provides a transparent, reliable and autonomous network hub for any service.”
Demo