

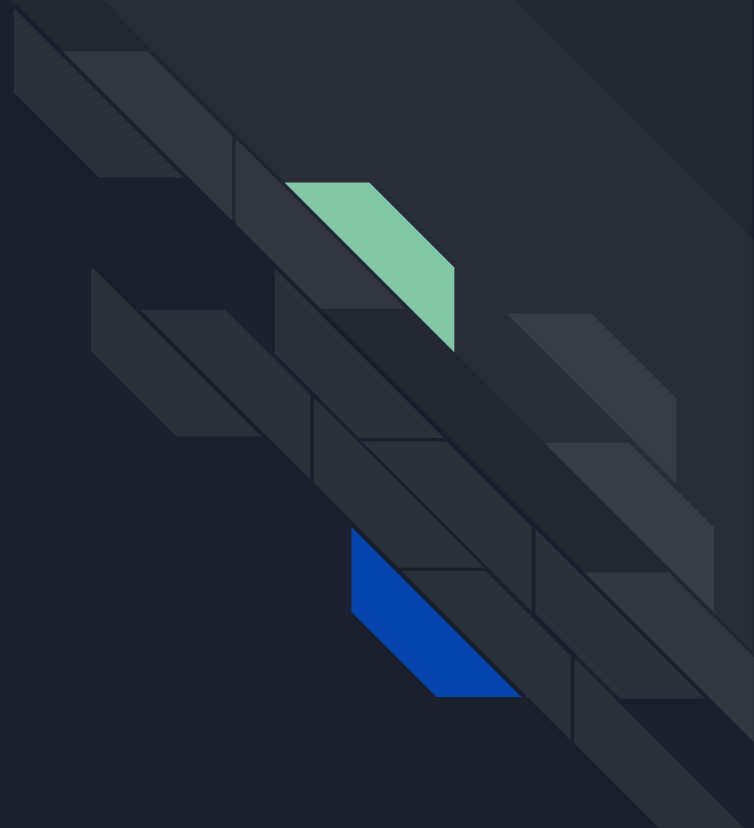
Way to solve Microservice problems with a Service Mesh

Andrii Minchekov
Software Architect

Linkerd 1.x implementation chosen for the purpose of a Demo

TOC

- Microservices challenges
- Service Mesh
 - Why
 - Overview
 - Features
 - Pros/cons
 - Frameworks comparison
 - Linkerd Demo
- Questions






Microservice challenges

1. Infrastructure is not reliable and sometimes slow, partial failures possible
2. Request routing and load balancing is hard under scaling
3. It's difficult to understand what's happening in real time.
4. Secure communication between services
5. Dev team requires experience with specific microservice libraries, technology diversity
6. Ops team need to manage and monitor lots of services, which are being redeployed regularly.

Why Service Mesh (Linkerd)?

Stakeholder	Concerns	Benefits
Management	<p>Why do we need it?</p> <p>What is the cost?</p> <p>What are the risks? Is it mature technology?</p> <p>What dependencies are we introducing to our business?</p>	<ul style="list-style-type: none">• We have fewer outages• Our services are resilient• We receive unified visibility across all services• Business services are not touched at all• We receive much faster pace of development• Cost: ~ 2 month to be integrated against 10 months by integrating dedicated libs into business apps 1 node needs up to 200mb of memory, can serve 10k req/sec• Only 2 years on the market but based on mature RPC framework from Twitter (Finagle). Member of CNCF• We introduce 1 more technical service (possibly multiple nodes) what is the good design principle to solve complex problems by introducing layer of indirection

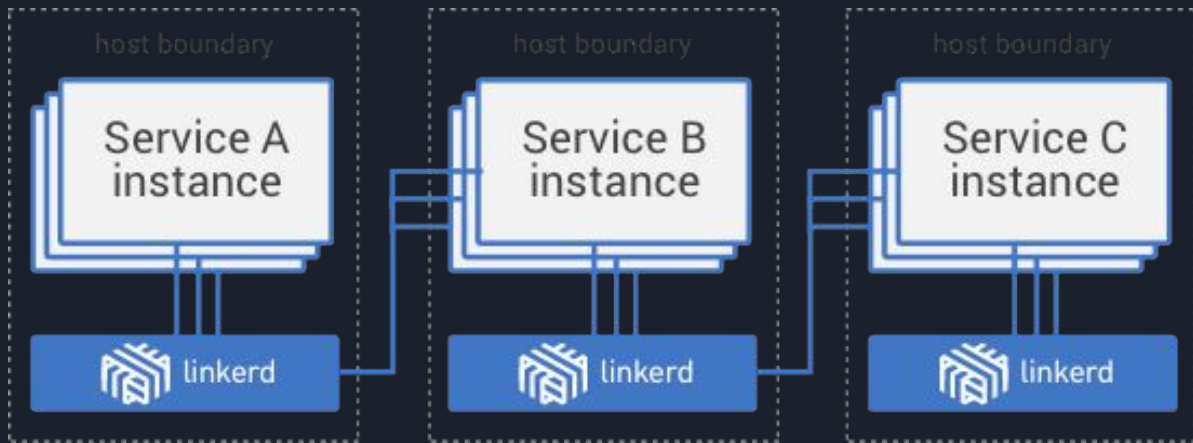


Stakeholder	Concerns	Benefits
Developers	What do I have to change? Do I have to learn a new technology? How to test it?	<ul style="list-style-type: none">• Get rid of the complex communication logic from your code• Yes, but it's only declarative YAML configuration mostly done by Senior Engineers or even Operations team• System tests should be in place. It's just 1 more technical service to dev infrastructure but allows easily run parallel versions of a service, do dynamic routing per-request
Operations	Is it reliable? Will it introduce complexity? Is it secure?	<ul style="list-style-type: none">• All service communication metrics will be at your hand with possible alerting in case of a problem No cascading failures, they will be isolated• Just 1 more service to deploy and configure but it allows to do easily staging, blue/green, canary deployment• Yes, consistent TLS across all services supported

Service Mesh Overview

It's a dedicated infrastructure layer to manage service to service communication in a fast and resilient manner for microservices. It abstracts the mechanics of reliably delivering requests between services.

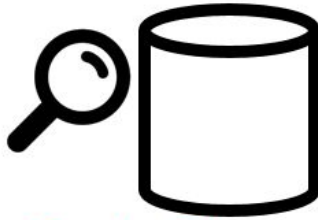
Service Mesh is typically implemented as an array of lightweight network proxies that are deployed alongside app code, without the app needing to be aware.



Features



Circuite Breaker



Service Discovery



Timeouts and Retry



Security



Loadbalancing



Dynamic Routing



Access Control



Observability



PROS

- It provides Service Metrics such as success rates, request volumes, latencies, etc without requiring changes to application code.
- Easy and quick to adopt Load Balancing, Circuit Breaker, Timeouts/Retries, Dynamic Routing, Distributed Tracing, Security for an existing/legacy applications. Good coordinating of retries and timeouts across multiple services. Some Estimates just to implement Discovery client:
Linkerd setup: ~ 10 days
Add lib per srv: 1 day * 60 services ~ 60 days
- Service developers can focus more on the business logic while most of service-to-service communication related functionality is offloaded to Service Mesh.
- Possibility to easily do Staging, Blue/Green, Canary, Continuous Deployment, A/B testing without needing to restart

CONS

- Immature
- Only Internal traffic can be handled. No single entry point is exposed to external clients.
- Resource intensive that increase the number of runtime instances that you have in a given microservice implementation.
Up to 200mb of memory required to operate for 1 node, 10k req/sec per node
- Extra hops will be added. Each service call has to go through an extra hops, xx ms (through Service Mesh sidecar proxy)

Frameworks Comparison

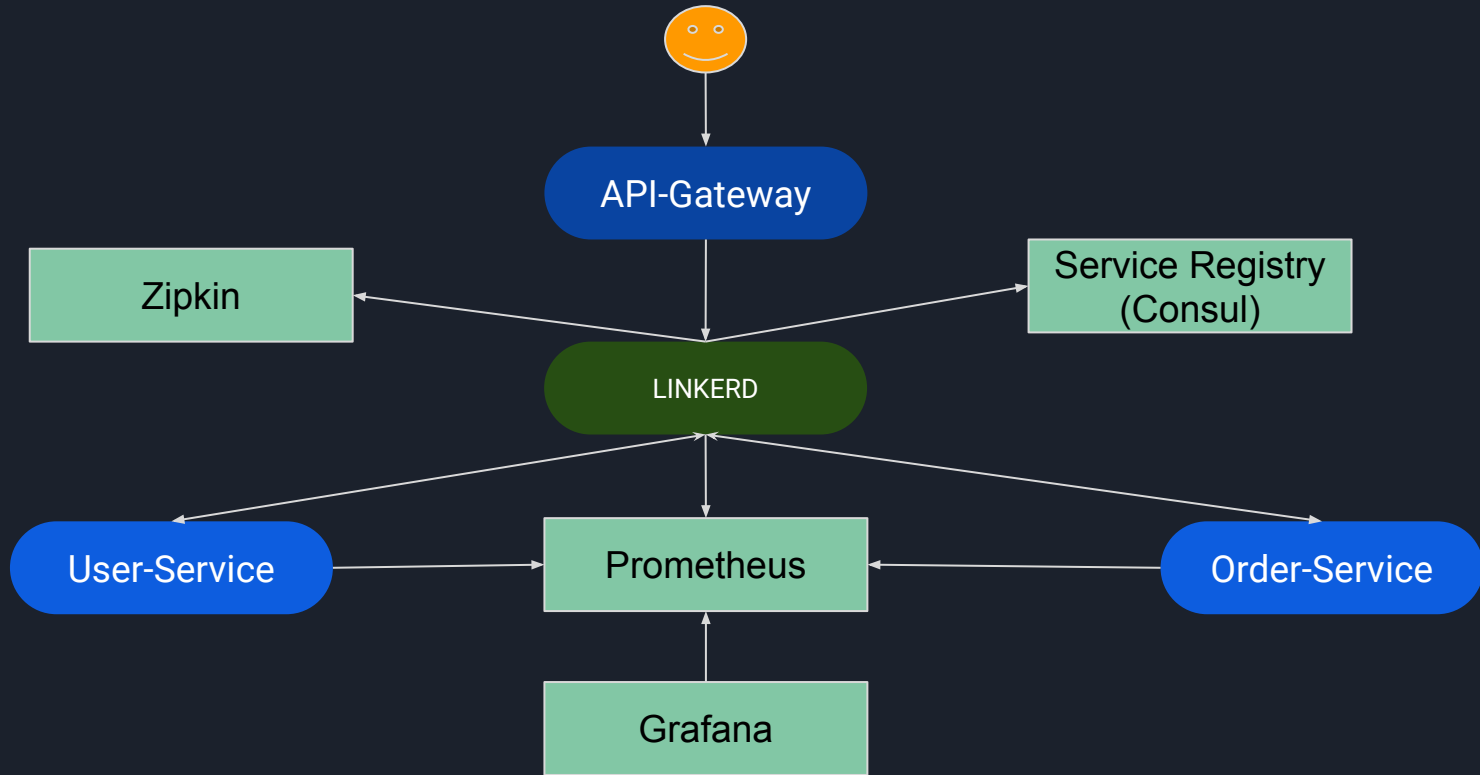
	Istio	Linkerd	Linkerd2	Consul Connect
Model	Sidecar	Node Agent	Sidecar	Sidecar
Platform	Kubernetes	Any	Kubernetes	Any
language	Go	JVM	Go / Rust	Go
Protocol	HTTP1.1 / HTTP2 / gRPC / TCP	HTTP1.1 / HTTP2 / gRPC	HTTP1.1 / HTTP2 / gRPC / TCP	TCP
Default Data Plane	Envoy (supports others)	Native	Native	Native (or Envoy)
Sidecar Injection	Yes	No	No	No
Encryption	Yes	Yes	Experimental	Yes
Traffic Control	label/content based routing, traffic shifting	Dynamic request routing, traffic shifting, per request routing	?	static upstream, prepared query, http api / dns with native integration
Resilience	timeouts, retries, connection pools, outlier detection	timeouts, retries, deadlines, circuit breaking	?	Pluggable
Prometheus Integration	Yes	Yes	Yes	No
Tracing Integration	Jaeger	Zipkin	None	Pluggable
Host to Host auth	Service Accounts	TLS Mutual Auth	No	Consul ACL
Agent Caching	Yes	No	No	Yes
Secure connection outside cluster	No	Yes	No	Yes
Complexity	High	High	Low	Low
Paid Support	No	Yes	Yes	Yes



Demo time

1. Demo project architecture.
Docker-compose configuration.
2. Linkerd Integration.
Business services are free of code changes.
Only Proxy address as a System property should be provided to service deployment.
3. Linkerd configuration.
Basic concepts.
Load balancing, Circuit Breaker, Timeouts/Retry, Observability, Dynamic Routing

1. Demo project Architecture





2. Linkerd Integration

- HTTP Proxy.
Use JVM flags
`-Dhttp.proxyHost=linkerd-host -Dhttp.proxyPort=4141`
- Send traffic directly to Linkerd
use Http Headers.

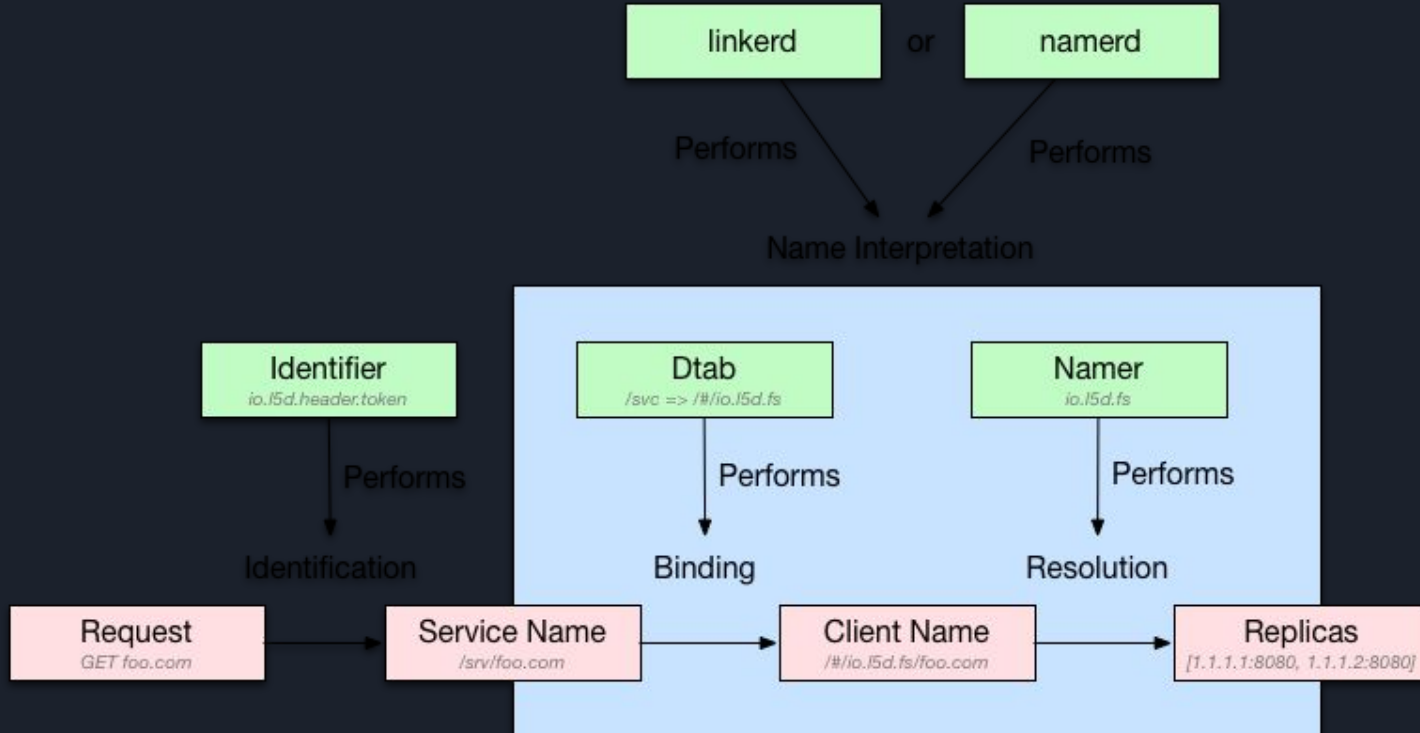


3. Linkerd Configuration

1. Routers
 - a. Servers
 - b. Protocols (HTTP/1.1, HTTP/2, TCP, gRPC, Thrift, Mux)
 - c. Delegation tables (dtab)
 - d. Identifiers
 - e. Service, prefixes
 - f. Client, prefixes
2. Namers
3. Telemetry

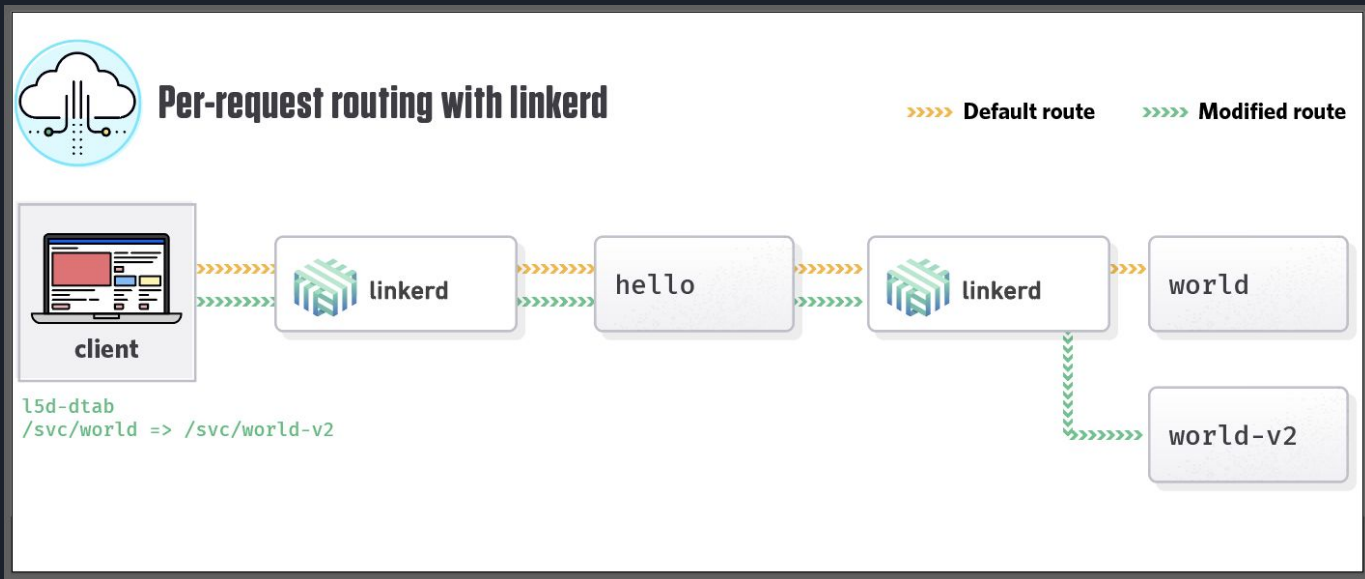
Routing

Consists of 4 steps:
identification, binding, resolution, load balancing.



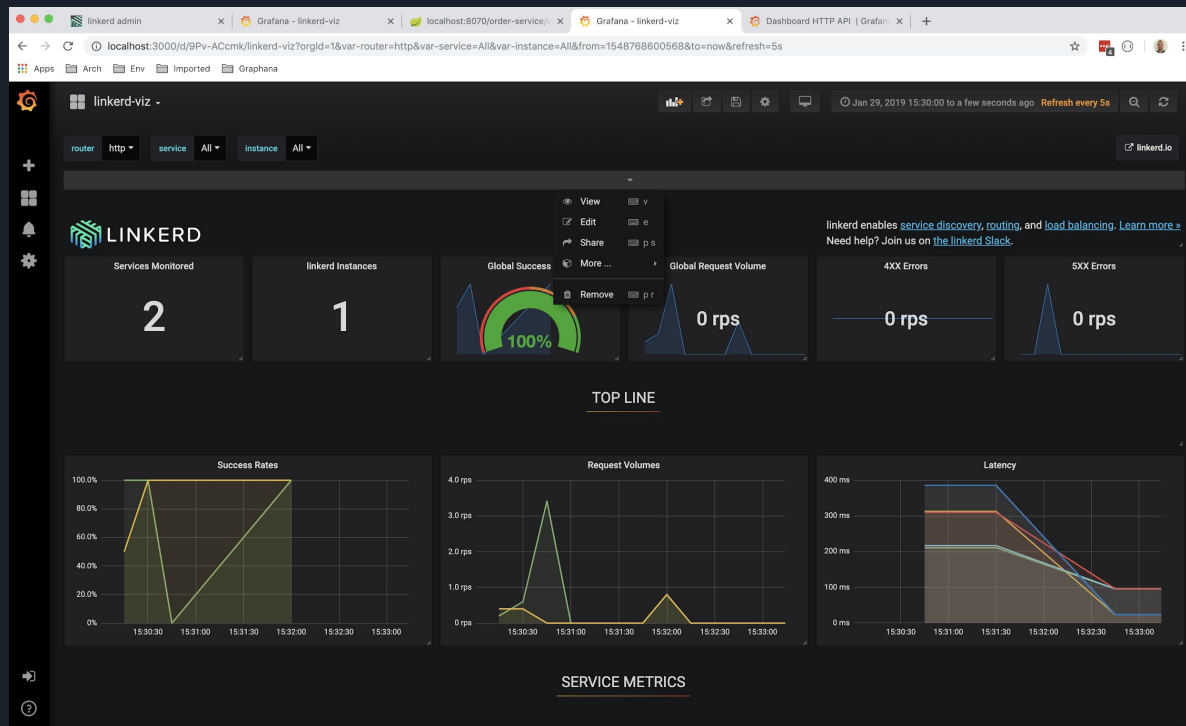
Dynamic request routing and traffic shifting

Use HTTP header "l5d-dtab" as a dtab rule to be used for routing a particular request
Use namerd service to enable the ability to make dtab changes at runtime, without needing to restart Linkerd.



Observability

Grafana Dashboards with success rate, latency, throughput, etc





Thank you!

Questions?

POC Source Code

<https://github.com/menya84/servicemesh>