Linkerd: Why!?

Oliver Gould

Linkerd lead; Buoyant CTO

@olix0r

@olix0r

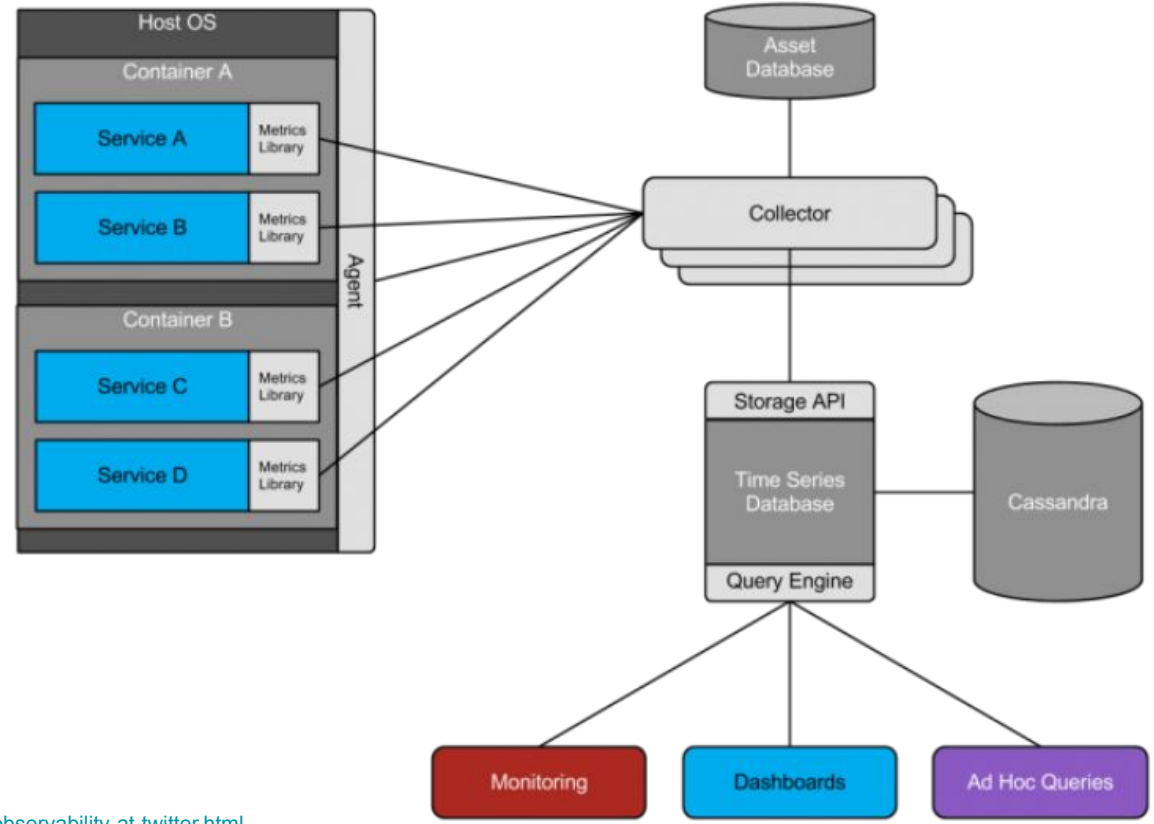
@olix0r

Why does Linkerd exist?

Twitter Observability (2010-2013)

- Store a timeseries of system metrics for every host;
- … and app metrics for every service;
- … and provide alerting;
- … and customizable dashboards;
- … and distributed tracing looks pretty cool;
- … and be more reliable than other services;
- … and scale with the company!

Twitter Observability



https://blog.twitter.com/engineering/en_us/a/2013/observability-at-twitter.html

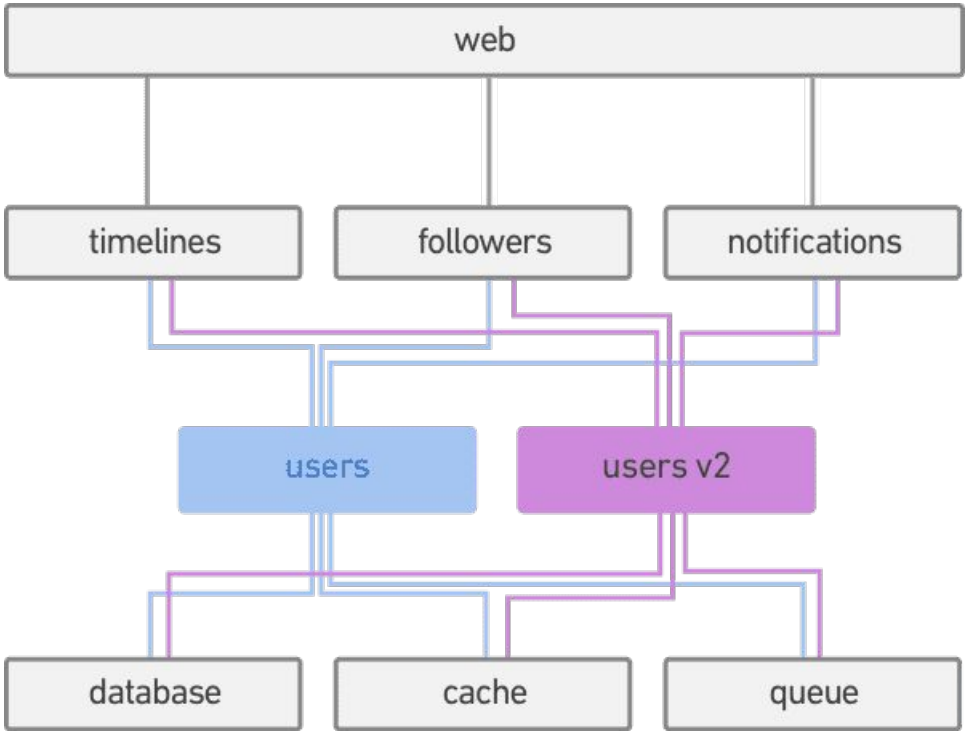
Twitter Observability Lessons

- Would have been nice to have Prometheus + Grafana
- Configuration is the root of all evil
- Operational data model is **critical**
- Over time, technical/scaling challenges are more tractable than organizational ones

Twitter Traffic (2013-2015)

- Aurora+ZooKeeper service discovery infra;
- Flexible request routing
    - Canary
    - Blue/green
    - Regional Failover

Twitter Traffic



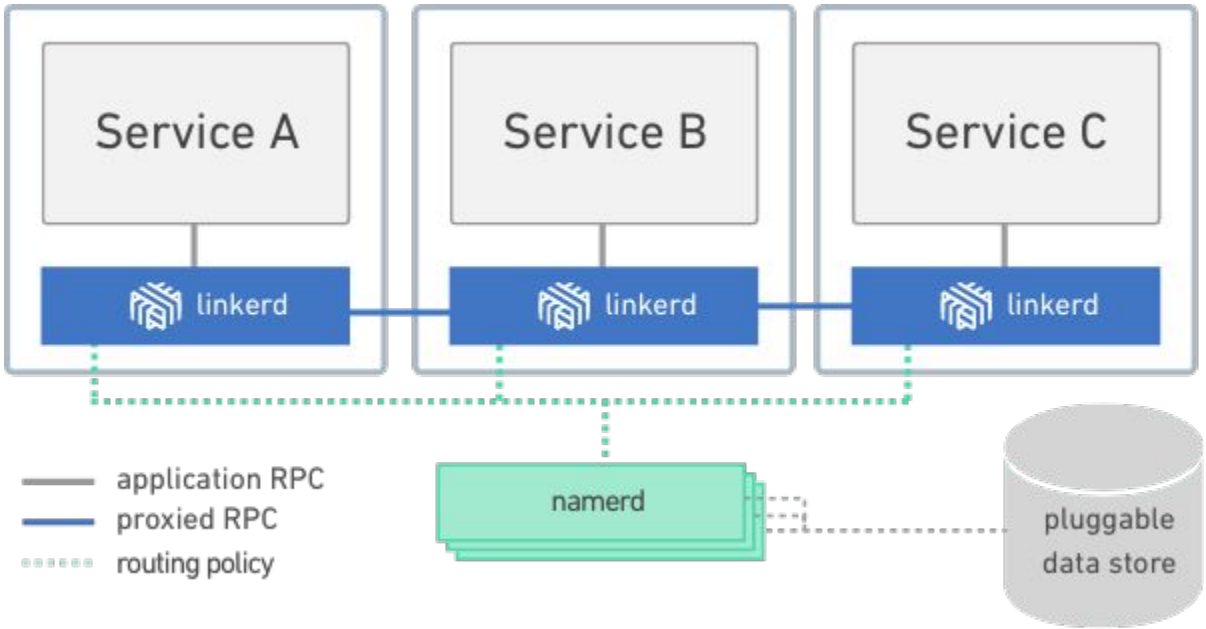
Twitter Traffic Lessons

- Microservices are all about communication
- Diagnostics, diagnostics, diagnostics
- Uniform instrumentation is essential to overcoming organizational inertia

Linkerd 1.x (2015-2020)

- Built on Twitter Finagle (JVM)
- Highly configurable/pluggable
    - ZooKeeper
    - Consul
    - Kubernetes, ...
- Flexible request routing
    - Canary
    - Blue/green
    - Regional Failover

Linkerd 1.x


Service A
Service B
Service C
linkerd
linkerd
linkerd
application RPC
proxied RPC
routing policy
namerd
pluggable
data store

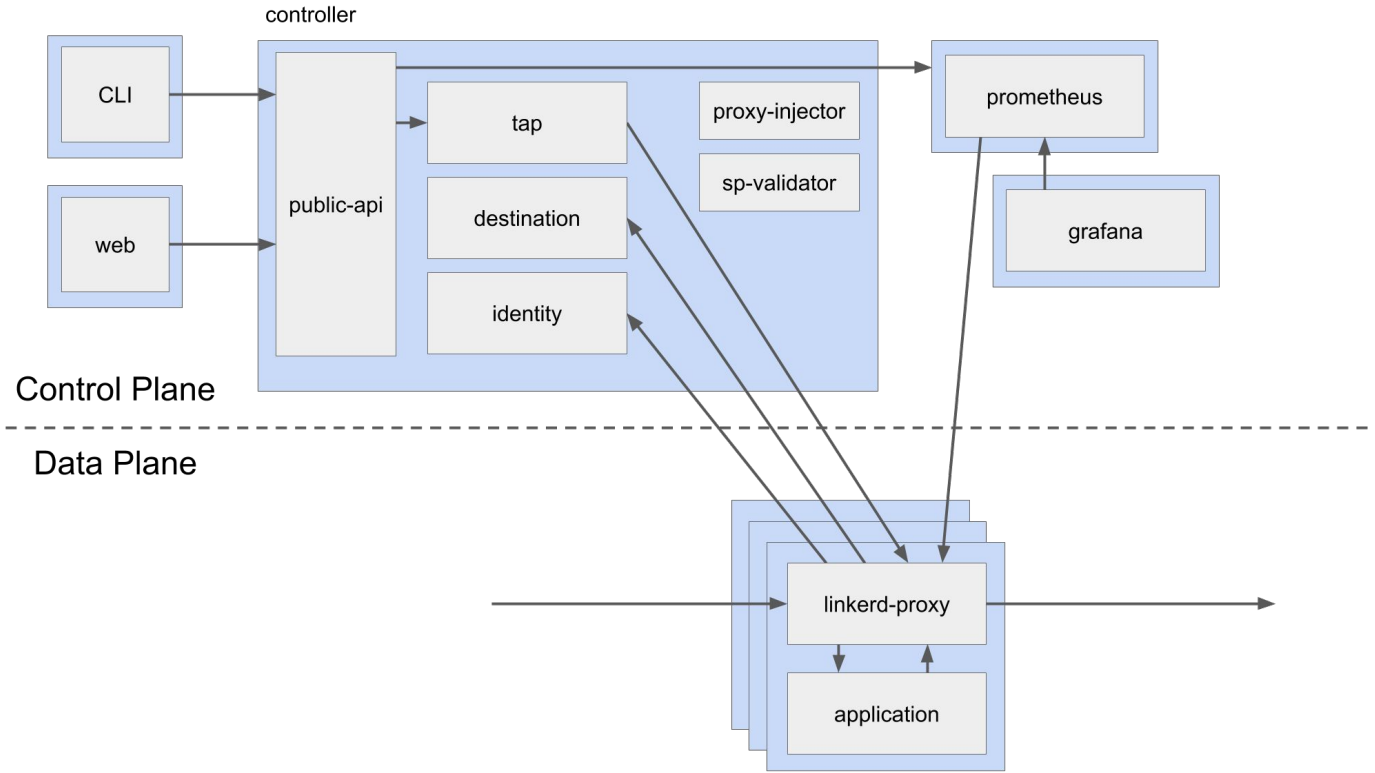
Linkerd 1.x Lessons

- Configuration is the root of all evil
- The JVM is also the root of at least some evil
- The Future is Microservices
- Kubernetes is King

Linkerd 2.x (2017-)

- Kubernetes Native Service Mesh
- Out-of-the-box traffic observability
- Out-of-the-box mTLS identity
- Out-of-the-box latency-based load balancing
- Go control plane (client-go, for better or for worse)
- Rust data plane (safe, small, fast sidecar)
- Prometheus + Grafana

Linkerd 2.x


controller
CLI
web
public-api
tap
destination
identity
proxy-injector
sp-validator
prometheus
grafana
Control Plane
Data Plane
linkerd-proxy
application

Linkerd 2.x Lessons*

- Kubernetes is The Database!
  - Pods and ServiceAccounts are the atomic building blocks.
- Infrastructure projects succeed by building trust over time.
- The world needs more reference architectures.

Why another proxy?

Proxy Goals

- Small (memory)
- Fast (low latency)
- Cheap (low CPU)
- Safe (no heartbleed)
- Malleable

Proxy Requirements

- No Garbage Collection
- Native Language
- Strong Type System

Proxy sidecar specialization

- No configuration*
  - Transparent protocol detection
- Automatic, transparent mTLS within mesh
- Automatic, transparent HTTP/2 multiplexing within mesh
- Rich prometheus labeling, raw histograms, …
- `linkerd tap`
- …?


Rust
EVANGELISM
STRIKE FORCE

```rust
// A per-`outbound::Endpoint` stack that:
//
// 1. Records http metrics  with per-endpoint labels.
// 2. Instruments `tap` inspection.
// 3. Changes request/response versions when the endpoint
//    supports protocol upgrade (and the request may be upgraded).
// 4. Appends `l5d-server-id` to responses coming back iff meshed
//    TLS was used on the connection.
// 5. Routes requests to the correct client (based on the
//    request version and headers).
// 6. Strips any `l5d-server-id` that may have been received from
//    the server, before we apply our own.
let endpoint_stack = client_stack
    .serves::<Endpoint>()
    .push(http::strip_header::response::layer(L5D_REMOTE_IP))
    .push(http::strip_header::response::layer(L5D_SERVER_ID))
    .push(http::strip_header::request::layer(L5D_REQUIRE_ID))
    // disabled due to information leakage
    //.push(add_remote_ip_on_rsp::layer())
    //.push(add_server_id_on_rsp::layer())
    .push(orig_proto_upgrade::layer())
    .push(tap_layer.clone())
    .push(http::metrics::layer::<_, classify::Response>(
        metrics.http_endpoint,
    ))
    .push(require_identity_on_endpoint::layer())
    .push(trace::layer(|endpoint: &Endpoint| {
        info_span!("endpoint", peer.addr = %endpoint.addr, peer.id = ?endpoint.identity)
    }))
    .serves::<Endpoint>();
```

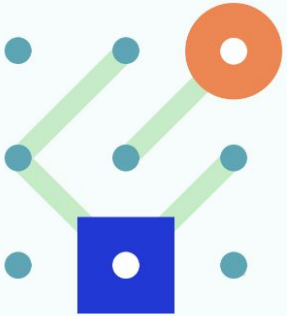
Cure53 Security Audit

The general indicators of security found on the Linkerd project during this June 2019 assessment are all very good. Cure53 needs to mention the atypically excellent code readability, careful choice of implementation languages, as well as the clearly written and well-maintained documentation for all attributes. These aspects contribute to the body of evidence about the overall exceptional quality of the project in terms of security.

Big Bets for 2020

1. Mandatory TLS by default
2. Inter-cluster identity, policy
3. Reduce total LOC by >10%

The Service Mesh Interface



What SMI covers

Service Mesh Interface is a specification that covers the most common service mesh capabilities:

- Traffic policy – apply policies like identity and transport encryption across services
- Traffic telemetry – capture key metrics like error rate and latency between services
- Traffic management – shift traffic between different services

LINKERD

Open source CNCF **service mesh**

- 🔥 **36+** months in production
- 🔥 **3,500+** Slack channel members
- 🔥 **10,000+** GitHub stars
- 🔥 **100+** contributors
- 🔥 **Weekly** edge releases
- 🔥 **6-8 week** stable releases
- 🔥 **Open governance** commitment

CLOUD NATIVE COMPUTING FOUNDATION

Shipped since Kubecon EU:

- 🛳️ **2.4**: Traffic splitting and SMI support
- 🛳️ **2.5**: Helm charts, **security audit**
- 🛳️ **2.6**: Distributed tracing

Up next:

- 🔮 **2.7 (2019)**: mTLS for all TCP traffic
- 🔮 **2.8+ (2020)**: Mandatory mTLS, better multi-cluster, policy, mesh expansion, ...

Linkerd talks this week from:

- 😎 Nordstrom
- 😎 Microsoft
- 😎 Paybase
- 😎 Buoyant

Join our community!

github.com/linkerd

slack.linkerd.io

@linkerd