GITTER CHANNEL
gitter.im/datawire/summit

TWEET ABOUT THE SUMMIT USING
#msvsummit

The Mechanics of Deploying Envoy at Lyft

Matt Klein
Senior Software Engineer at Lyft

Sponsored by DATAWIRE STACK POINT CLOUD cloudbees envoy

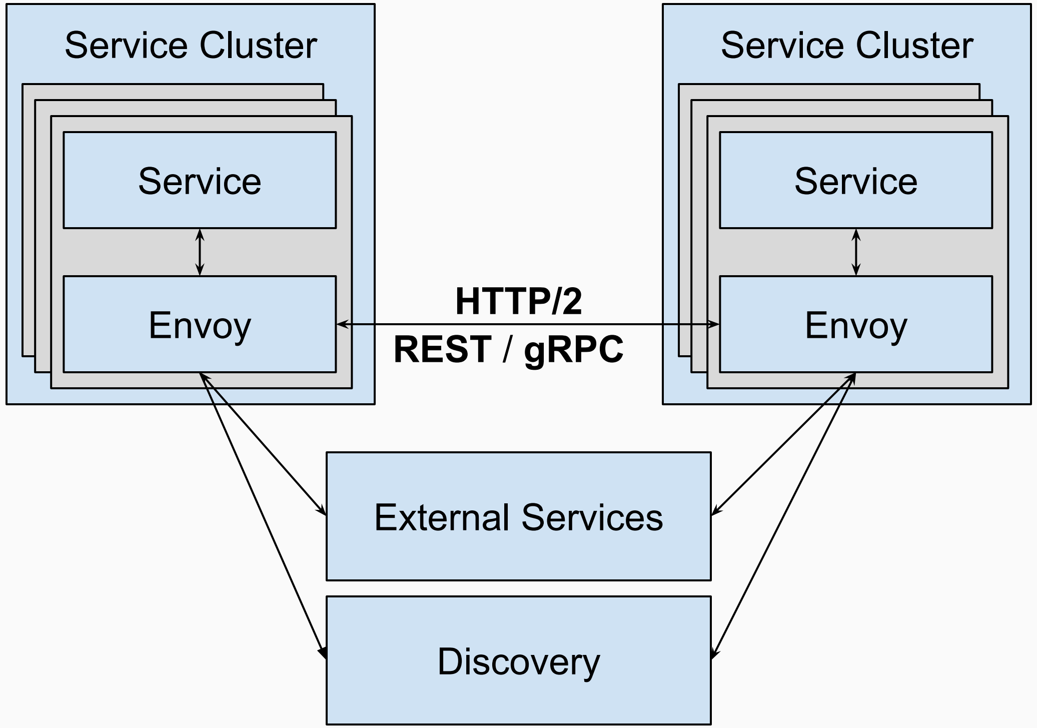
by lyft

The mechanics of deploying Envoy at Lyft

Microservices Practitioner Virtual Summit

Matt Klein / @mattklein123, Software Engineer @Lyft

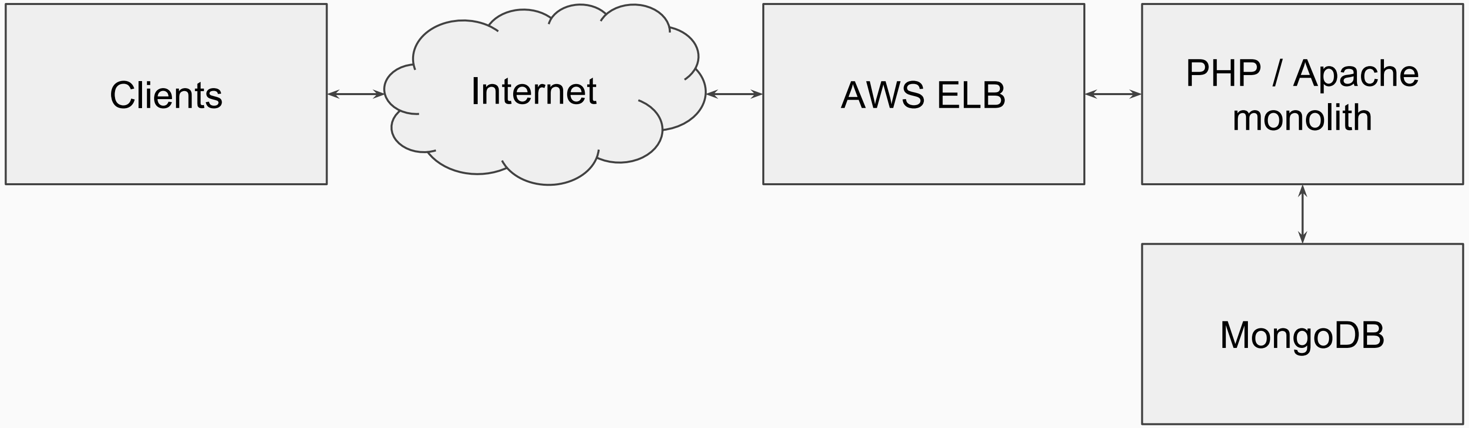
Envoy refresher



Envoy refresher

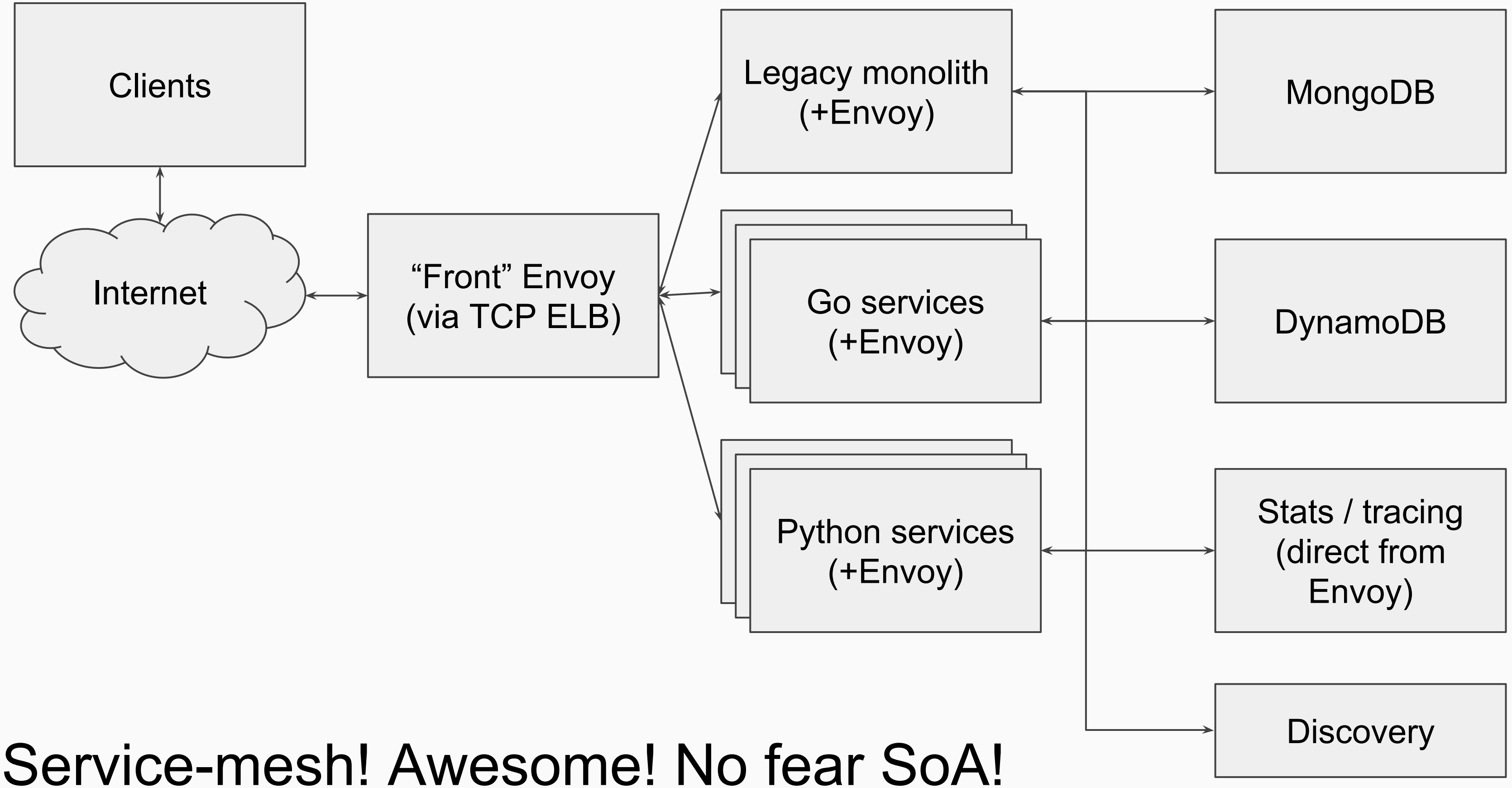
- **Out of process architecture**: Let's do a lot of really hard stuff in one place and allow application developers to focus on business logic.
- **Modern C++11 code base**: Fast and productive.
- **L3/L4 filter architecture**: A byte proxy at its core. Can be used for things other than HTTP (e.g., MongoDB, redis, stunnel replacement, TCP rate limiter, etc.).
- **HTTP L7 filter architecture**: Make it easy to plug in different functionality.
- **HTTP/2** first! (Including **gRPC** and a nifty gRPC HTTP/1.1 bridge).
- **Service discovery** and **active/passive health checking.**
- **Advanced load balancing**: Retry, timeouts, circuit breaking, rate limiting, shadowing, outlier detection, etc.
- Best in class **observability**: stats, logging, and tracing.
- **Edge proxy**: routing and TLS.

Lyft ~4 years ago



Simple! No SoA! (*but still not that simple*)

Lyft today



Service-mesh! Awesome! No fear SoA!

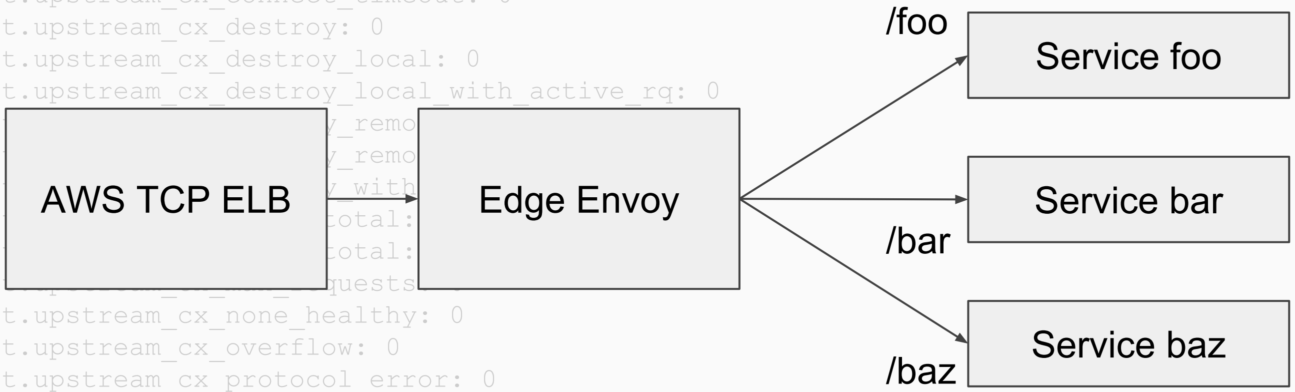
How did Lyft go from then to now

- Can’t go from before to after overnight.
- Must be incremental. Show value in steps.
- Perfect is the enemy of the good.
- This is how we did it…(teaser: it wasn’t easy)

Start with edge proxy

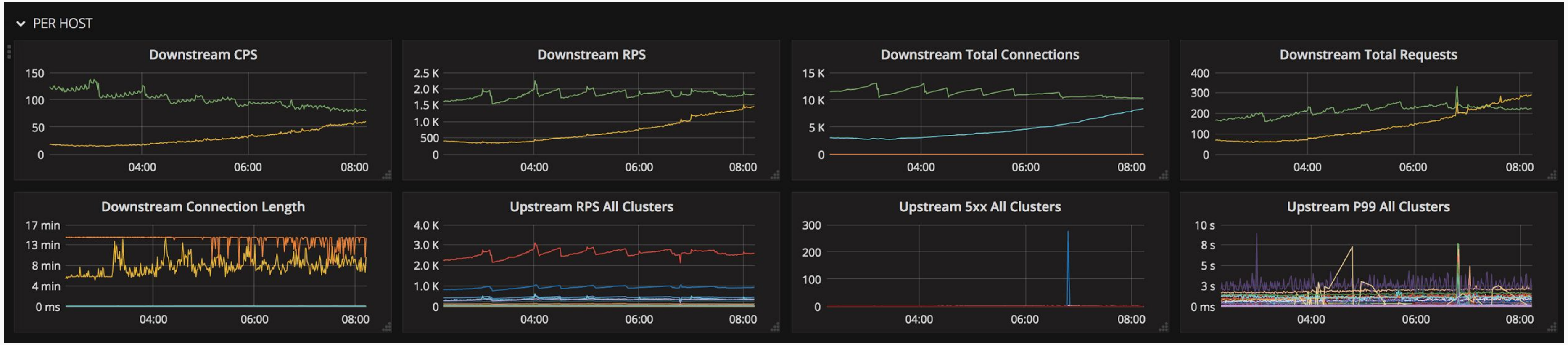
- Microservice web apps need edge reverse proxy.
- Existing cloud offerings are not so good (even still).
- Easy to show value very quickly with stats, enhanced load balancing and routing, and protocols (h2/TLS).

```
cluster.account.upstream_cx_active: 18
cluster.account.upstream_cx_close_notify: 725
cluster.account.upstream_cx_connect_fail: 0
cluster.account.upstream_cx_connect_timeout: 0
cluster.account.upstream_cx_destroy: 0
cluster.account.upstream_cx_destroy_local: 0
cluster.account.upstream_cx_destroy_local_with_active_rq: 0
cluster.accoun                   y_remo
cluster.accoun                   y_remo
cluster.accoun                   y_with
cluster.accoun                   total:
cluster.accoun                   total:
cluster.accoun                   quests
cluster.account.upstream_cx_none_healthy: 0
cluster.account.upstream_cx_overflow: 0
cluster.account.upstream_cx_protocol_error: 0
cluster.account.upstream_cx_rx_bytes_buffered: 9092
cluster.account.upstream_cx_rx_bytes_total: 1224096757
```



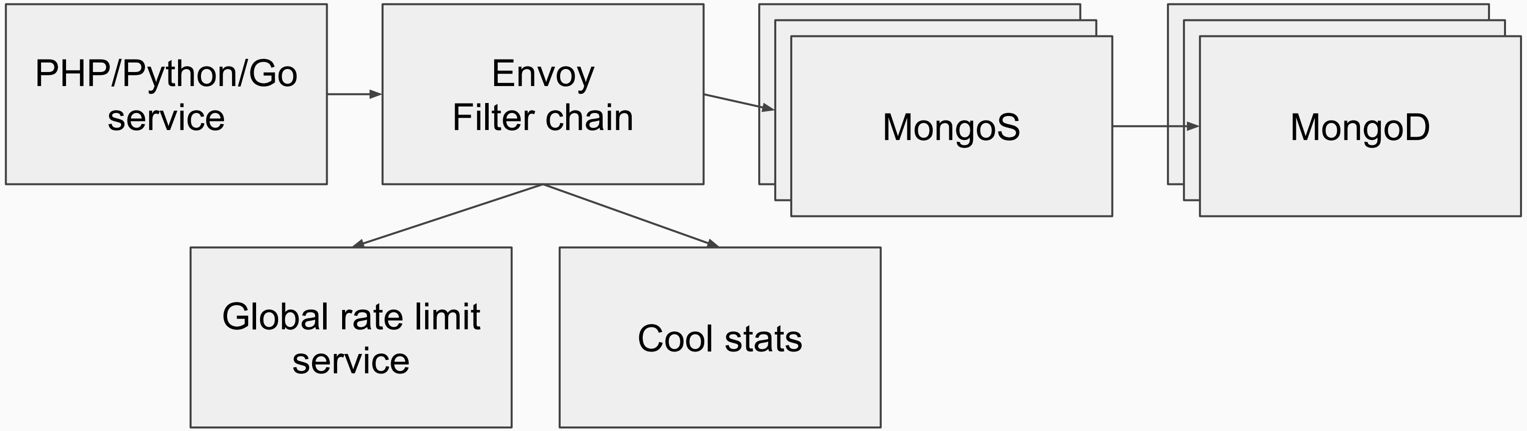
Start with edge proxy

Observability, observability, observability...



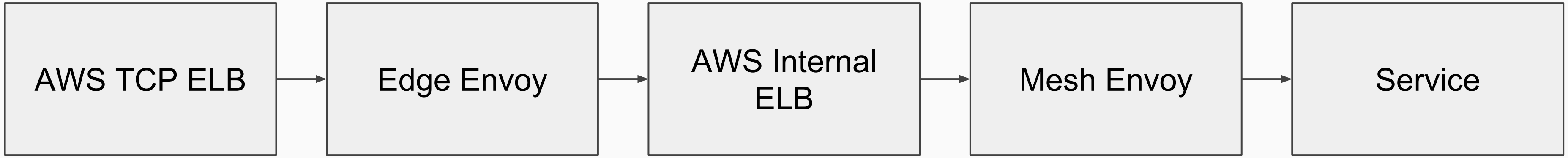
Next: TCP proxy and Mongo

- PHP to sharded Mongo not efficient with connection counts.
- Mongo bad at connection handling.
- Limit connections into Mongo.
- ... We can parse Mongo at L7 and generate cool stats!
- ... We can ratelimit Mongo to avoid death spirals!
- ... We can do this for all services, not just Python!



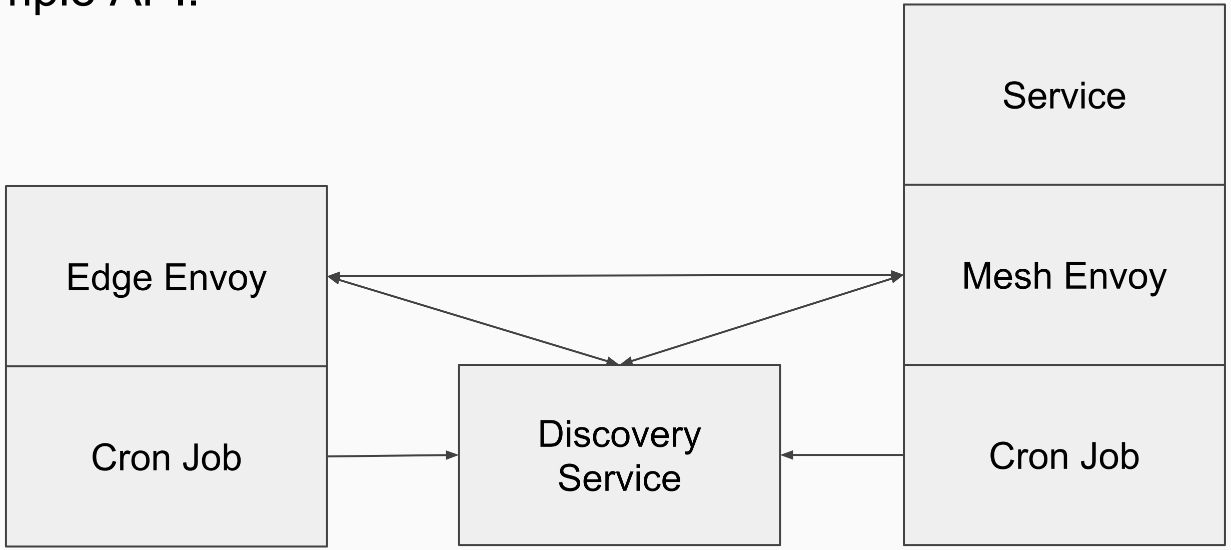
Next: service sidecar

- Because of Mongo, Envoy already running on services.
- Let’s use for ingress buffering, circuit breaking, and observability.
- Still using internal ELBs at this point for service to service traffic.
- Still get tons of value out of above without direct mesh connect.



Next: service discovery and real mesh

- ELBs are (and intermediate LBs in general) … not great. Let’s do direct connect.
- Need service discovery.
- No ZK, etcd, or Consul. Eventually consistent. Build dead simple system with dead simple API.



Envoy thin clients @Lyft

```python
from lyft.api_client import EnvoyClient
switchboard_client = EnvoyClient(
    service='switchboard'
)
msg = {'template': 'breaksignout'}
headers = {'x-lyft-user-id': 12345647363394}
switchboard_client.post("/v2/messages", data=msg, headers=headers)
```

- Abstract away egress port
- Request ID/tracing propagation
- Guide devs into good timeout, retry, etc. policies
- Similar thin clients for Go and PHP

Next … Envoy everywhere

- OK this Envoy thing is pretty neat.
- Need to run it *everywhere* for full value.
- And so begins the slog. Many month burndown to convert everything and remove ELBs.
- Why a slog? Because of how Lyft manages services and configurations…
- But after slog complete, payoff is tremendous. Keep adding *features that developers want* and deploy them widely very quickly...

Initial Envoy config management

- Envoy config is JSON (please don’t ask about JSON vs. YAML 😉).
- Initially we had full JSON committed.
- Deploys compile Envoy and bundle configs. Don’t need to do back compat.
- Config per deployment type.
  - front_envoy.json
  - service_to_service_envoy.json
- Envoy deployed via “pull deploy” and salt on each host.
- 1 service per host, 1 envoy per service, 1 envoy per host.
- Hot restart used for both config/binary deploys (no difference).

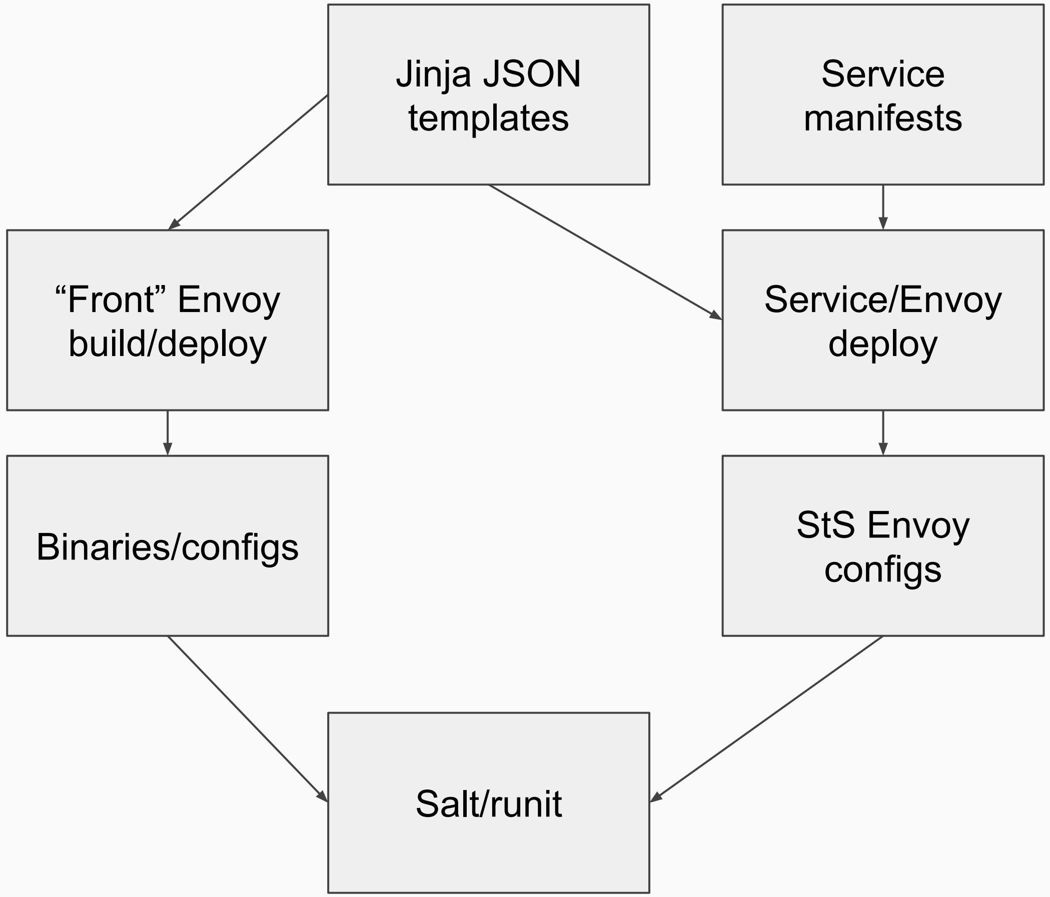
```
{
  “listeners”: {...},
  “clusters”: {...},
  ...
}
```

Next comes configgen.py and … jinja!

- These configs are starting to get really tedious to write!
- Would like some amount of static scripting.
- Develop a tool called configgen.py which takes in a bunch of templates and inputs, runs jinja on them, and produces final outputs.
- Already starting to see trend that ultimately no way around complex Envoy configs being **machine generated**.

```
{% import 'certs.json' as certs -%}
{% macro listener(port,ssl,proxy_proto) %}
  {
    "address": "tcp://0.0.0.0:{{ port }}",
    {% if ssl -%}
    "ssl_context": {
      "alpn_protocols": "h2,http/1.1",
      {{ certs.public(service_instance) }}
    },
    {% endif -%}
```

Envoy config/process management @Lyft now

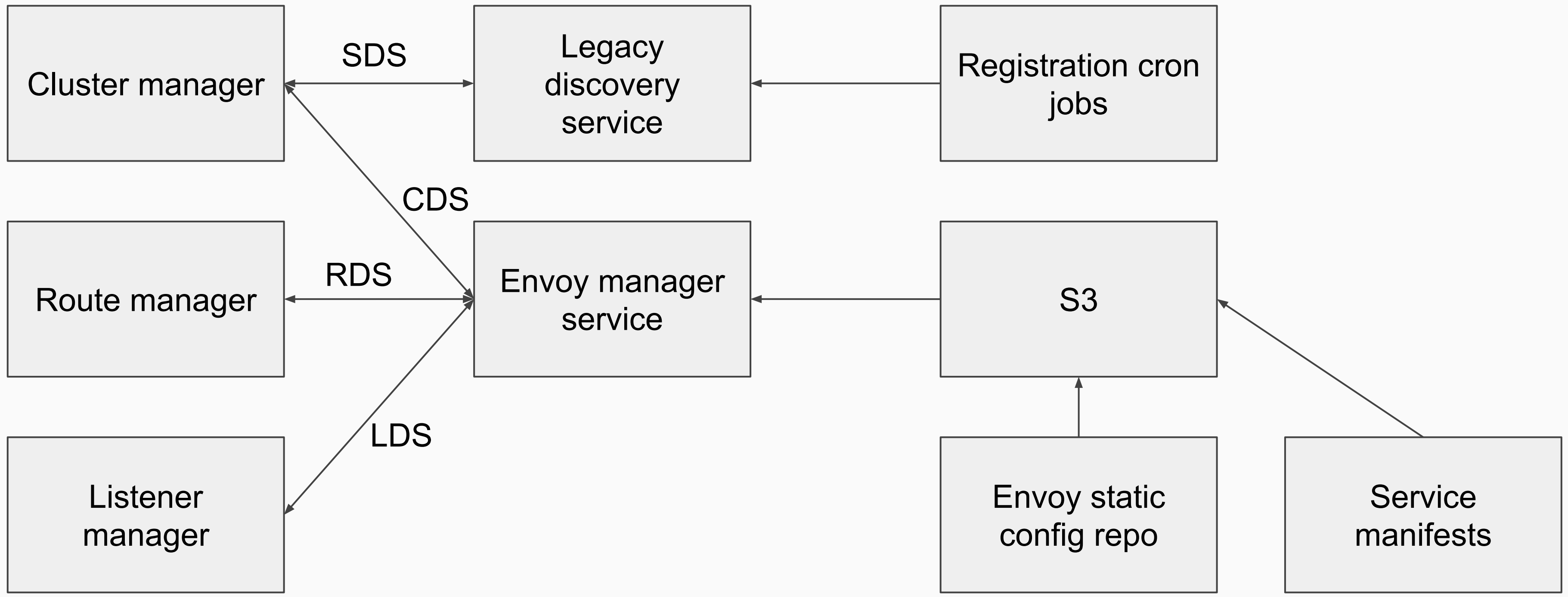


- Combination of static and dynamic configs.
- Service egress, circuit breaking, etc. configs specified in manifest.
- Service configs built on service host at service/envoy deploy time.

Envoy control plane APIs

- The goal of Envoy has always been as a *universal data plane*.
- Support APIs to enable control plane integration.
- V1 JSON/REST APIs (order of implementation):
    - **Service** Discovery Service (SDS) (note: poorly named)
    - **Cluster** Discovery Service (CDS)
    - **Route** Discovery Service (RDS)
    - **Listener** Discovery Service (LDS)
- @Lyft we **only currently use SDS**. Everything else driven by Jinja/JSON templating.

Lyft next-gen Envoy config management



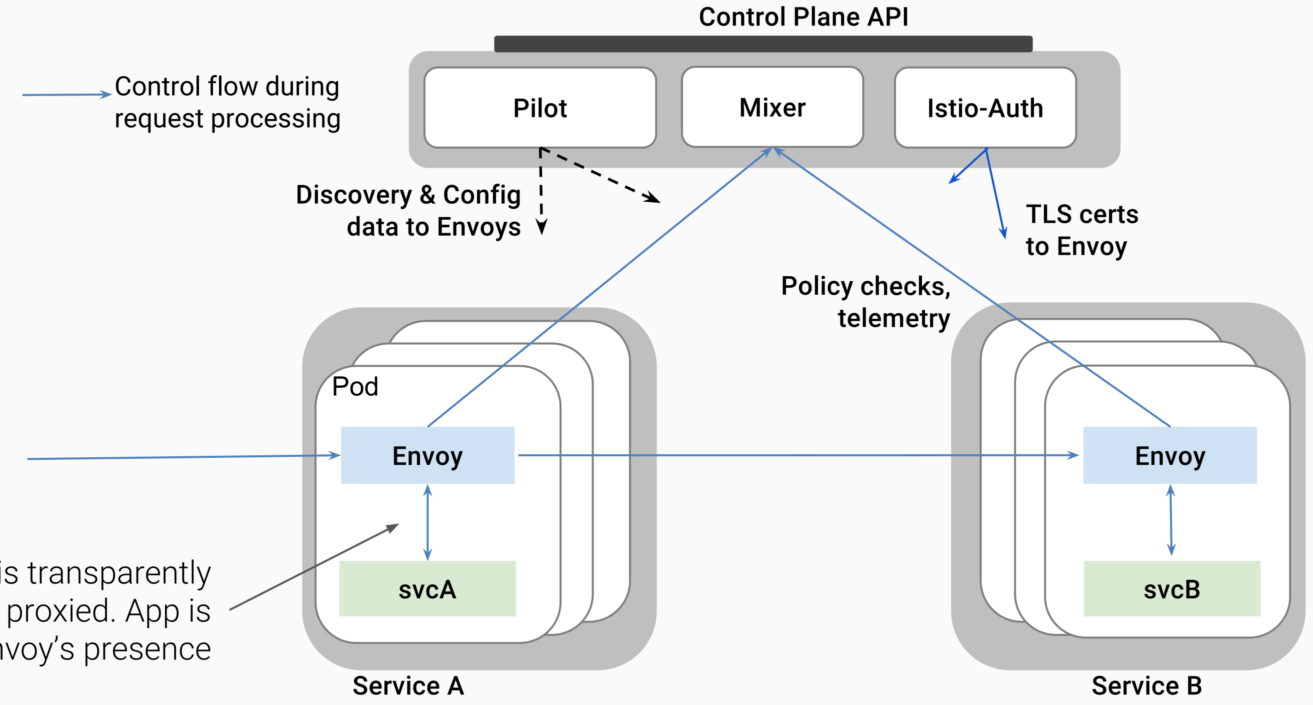
Only need a very tiny bootstrap config for each envoy...

Future gRPC/REST v2 APIs

- Canonical v2 APIs are **gRPC** (JSON also supported).
- Enhanced with **bidirectional streaming** and more features. Same spirit as v1.
- **Endpoint** Discovery Service (EDS, replaces SDS).
    - Fetch endpoints/hosts.
    - Report load info.
- **Cluster** Discovery Service (CDS).
- **Route** Discovery Service (RDS).
- **Listener** Discovery Service (LDS).
- **Health** Discovery Service (HDS).
- **Aggregated** Discovery Service (ADS).
    - Allowing all xDS APIs to be served by a single stream if desired. Useful for enforcing ordering of updates. E.g., CDS -> EDS -> RDS.

Istio and config APIs

- “Raw Envoy” is still too hard to configure for most folks
- Hard to decouple network from deploy orchestration
- Build higher level control systems (universal dataplane)
- Istio!
- K8s first. More later


Control Plane API
Control flow during request processing
Pilot
Mixer
Istio-Auth
Discovery & Config data to Envoys
TLS certs to Envoy
Policy checks, telemetry
Pod
Envoy
svcA
Envoy
svcB
Traffic is transparently intercepted and proxied. App is unaware of Envoy’s presence
Service A
Service B

Q&A

- Thanks for coming! Questions welcome on Twitter: @mattklein123
- We are super excited about building a **community** around Envoy. Talk to us if you need help getting started.
- https://lyft.github.io/envoy/ (@envoyproxy)
- https://istio.io/ (@istiomesh)

envoy

by lyft