KubeCon | CloudNativeCon

Europe 2019

Meshing Monolith to Microservices

Leo Liang

@leozc

$ whoami

@leozc

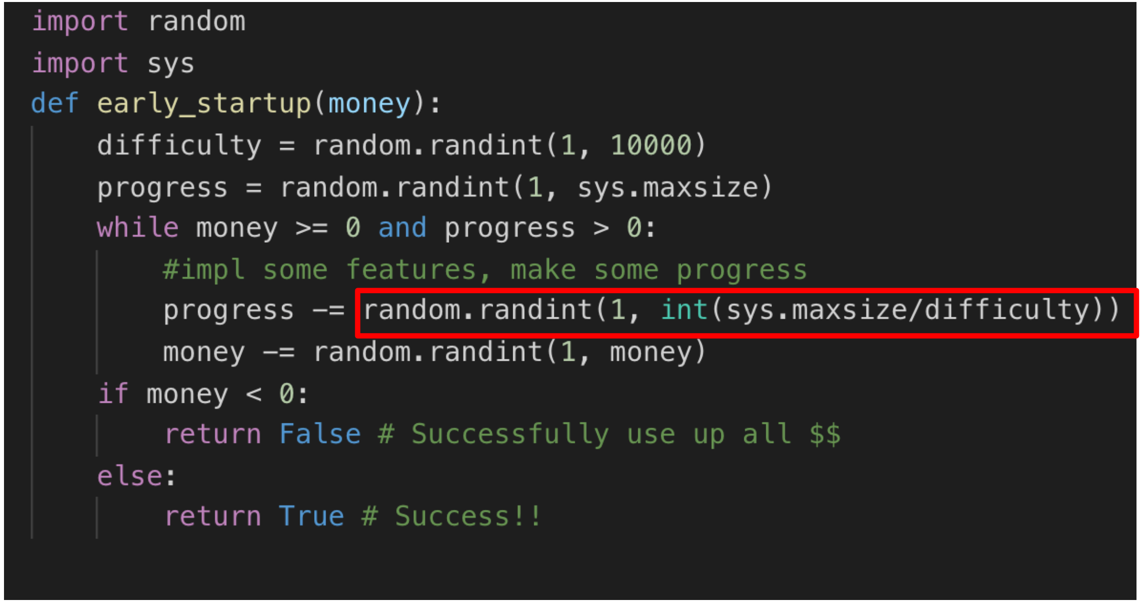
https://www.linkedin.com/in/leoliang

https://github.com/leozc

Machine Learning Platform
Ground Truth

How startup works

If a startup can be described in (python) code....



```python
import random
import sys
def early_startup(money):
    difficulty = random.randint(1, 10000)
    progress = random.randint(1, sys.maxsize)
    while money >= 0 and progress > 0:
        #impl some features, make some progress
        progress -= random.randint(1, int(sys.maxsize/difficulty))
        money -= random.randint(1, money)
    if money < 0:
        return False # Successfully use up all $$
    else:
        return True # Success!!
```

Scale enables faster and more meaningful iterations

> “ *Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization’s communication structure.*
>
> *— Conway’s law 1967*

To Maximize per iteration **`progress`**

=> Scale Teams (Quality + Quantity)

=> Scale Architecture

How the story began

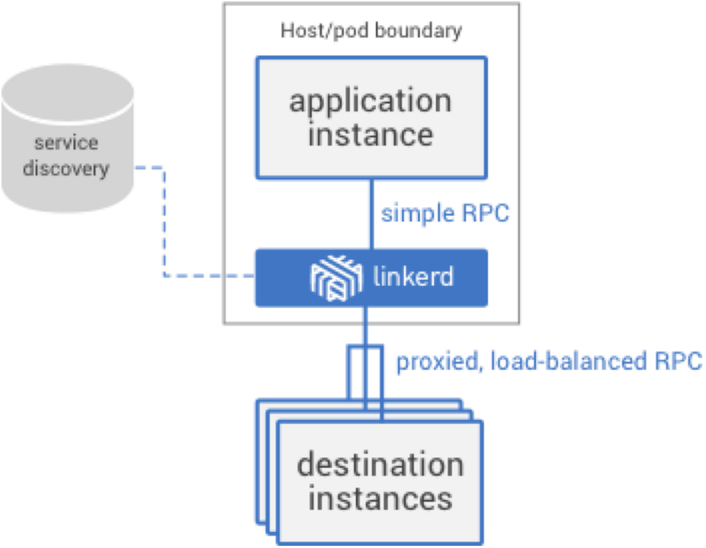
- Journey started in the beginning of 2016
- High growth Seattle unicorn (**Offerup**)
- 100% native on AWS => cluster neutral
- Magnitude of changes:

	2016	2017-2018
Number of Engineers	10	100+
Services	1 monolith	40+ services
Req/Daily	300M	2B+

- How?
  - Service Mesh driven Microservice architecture evolution

What is LinkerD (v1)

- A feature rich proxy
- Built on Twitter Stack
- Dtab (Delegation tables) is DSL for routing
- NamerD DNS for Service Mesh
- Powerful plugin support
  - JVM languages (Java, Scala)
  - For clarity - pseudo code in python



This talk is about...

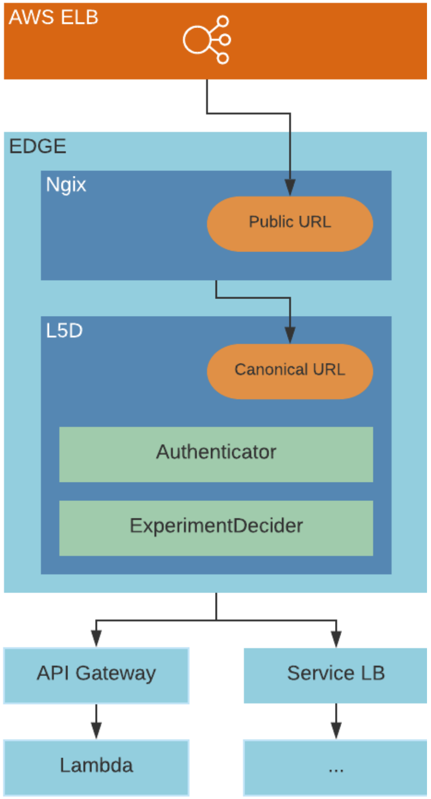
1. The architecture evolution from monolithic to microservice driven by service mesh
2. Pragmatic and systematic solution
3. Imperfect solution but respects to the legacy

Agenda

1. Edge - Split the world into **TWO**
2. Core of Mesh - Service to Service Communication
3. Observability
4. Conclusion

Split the world into TWO

- What is Edge Serrvice?
- Edge = Nginx + (Linkerd + Customized Plugins)



AWS ELB

EDGE

Ngix

Public URL

L5D

Canonical URL

Authenticator

ExperimentDecider

API Gateway

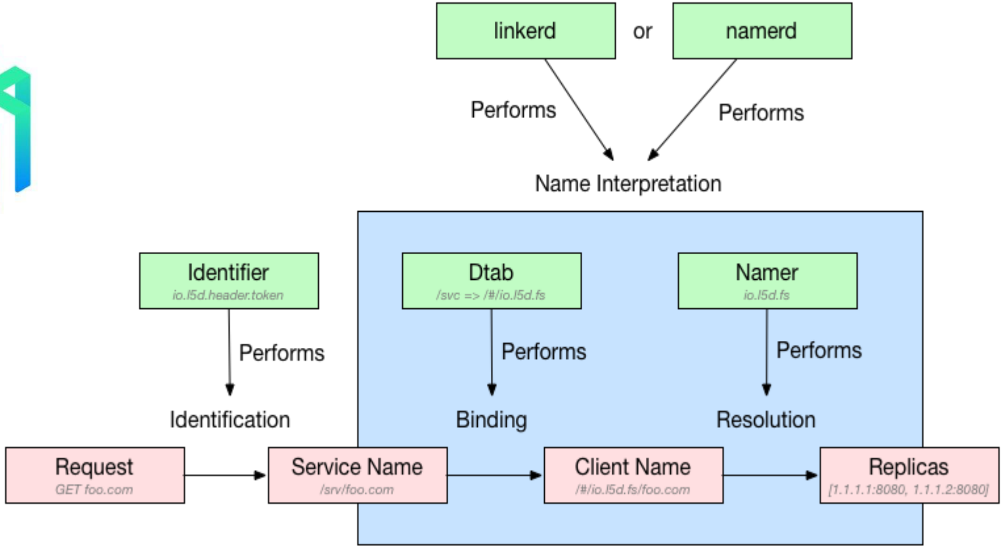
Service LB

Lambda

...

Split the world into TWO

Edge = Nginx + (Linkerd + Customized Plugins)



linkerd or namerd

Performs

Performs

Name Interpretation

Identifier
io.l5d.header.token

Performs

Identification

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

Performs

Binding

Namer
io.l5d.fs

Performs

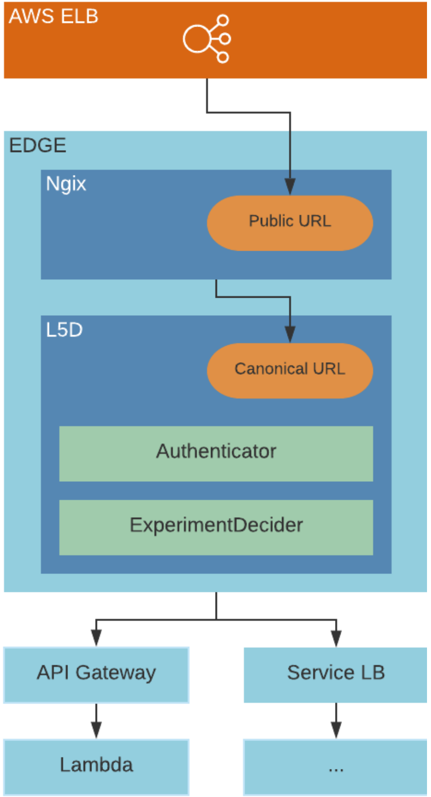
Resolution

Request
GET foo.com

Service Name
/srv/foo.com

Client Name
/#/io.l5d.fs/foo.com

Replicas
[1.1.1.1:8080, 1.1.1.2:8080]



AWS ELB

EDGE

Ngix

Public URL

L5D

Canonical URL

Authenticator

ExperimentDecider

API Gateway

Service LB

Lambda

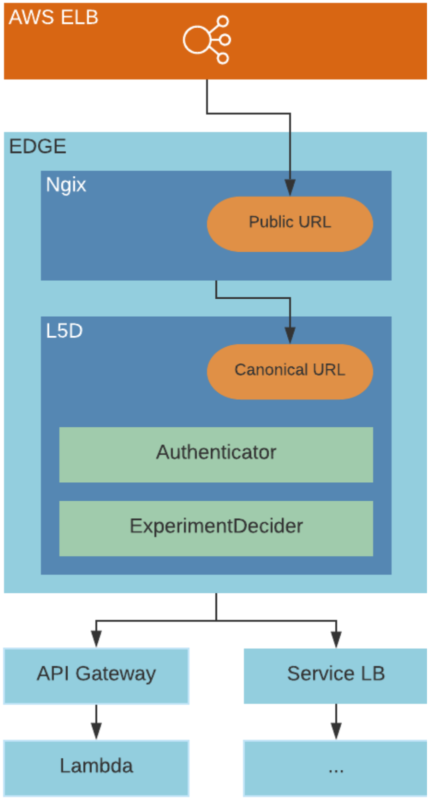
...

Split the world into TWO

Nginx Layer

- Police and security (CSRF validation)
- Header Normalization and injection (region)
- URL Normalization

```
/api/message/foo/bar?a=123
            ↓
/h1/us-east-1a/prod/foo/bar?a=123
```



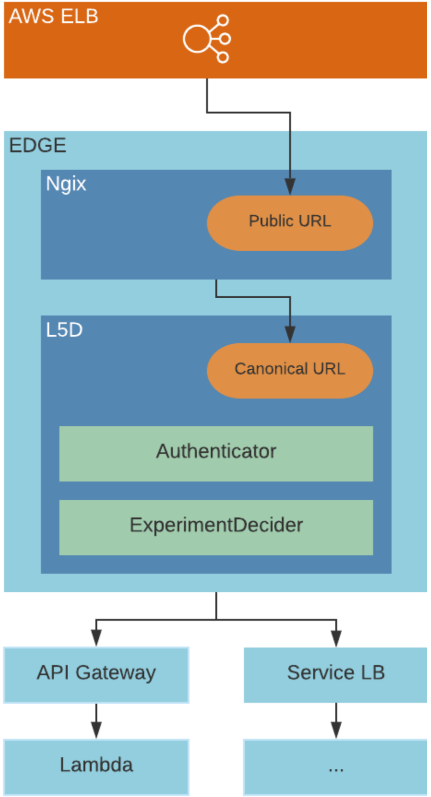
Split the world into TWO

LinkerD Layer

- URL Interpreted by Namerd Dtab
  - NamerD → Mesh DNS
- **`/h1/us-east-1a/prod/foo/bar?a=123`** =>

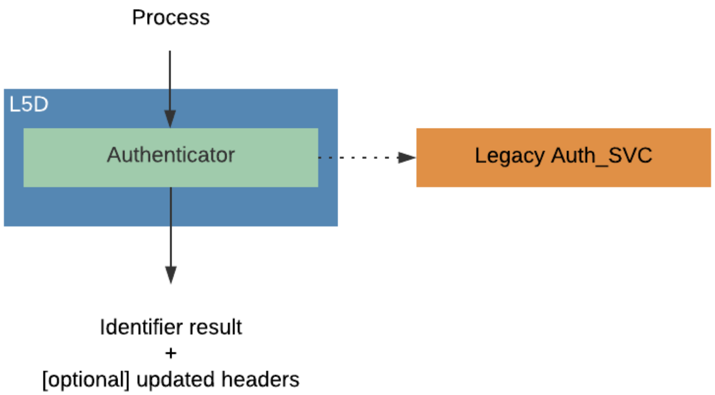
  `$host:$port/foo/bar?a=123`

  (Note: Routing/Discovery in later section)



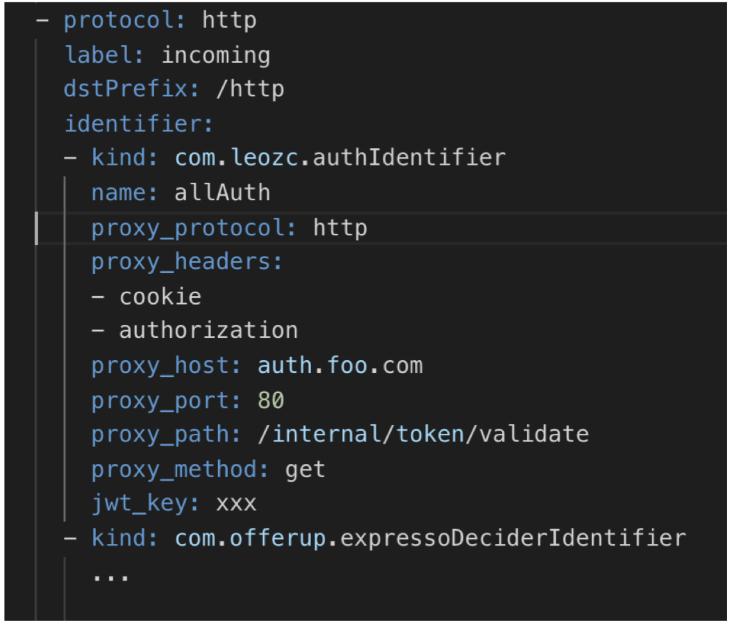
Edge LinkerD - Authentication

1. Transition
   - Legacy Client using cookie
   - Newer Clients using JWT
2. Ensure downstream services to have trusted user identify
3. Inject per user-specific context (e.g., user group)

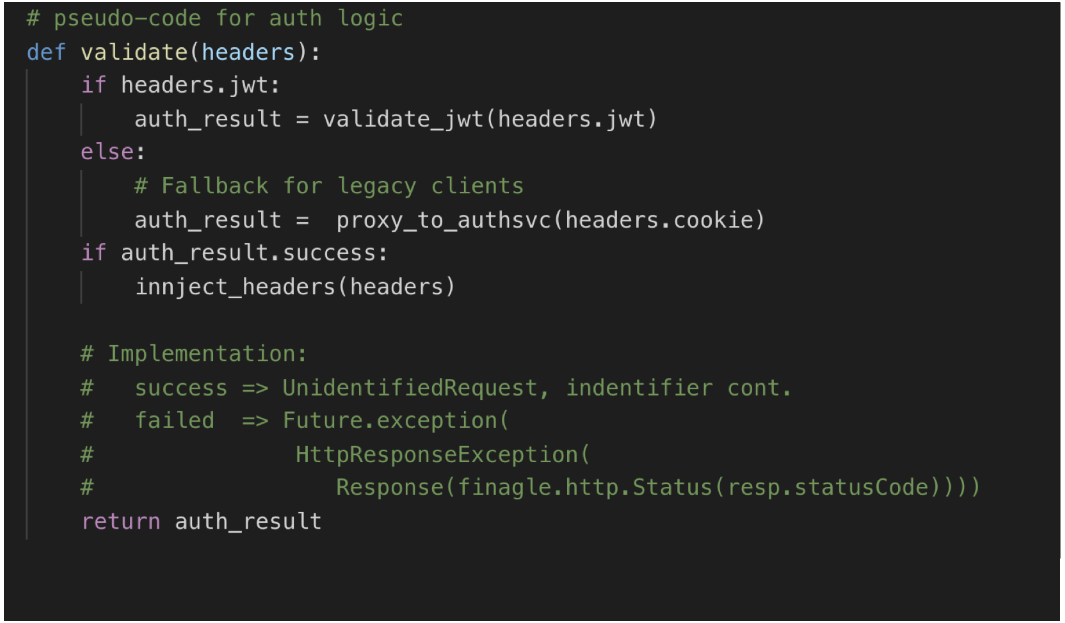


Edge LinkerD - Authentication

A LinkerD Identifier Plugin

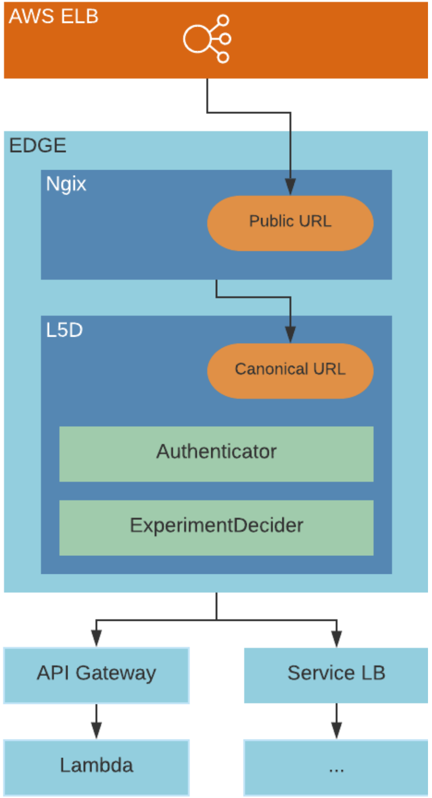


```yaml
- protocol: http
  label: incoming
  dstPrefix: /http
  identifier:
  - kind: com.leozc.authIdentifier
    name: allAuth
    proxy_protocol: http
    proxy_headers:
    - cookie
    - authorization
    proxy_host: auth.foo.com
    proxy_port: 80
    proxy_path: /internal/token/validate
    proxy_method: get
    jwt_key: xxx
  - kind: com.offerup.expressoDeciderIdentifier
    ...
```



```python
# pseudo-code for auth logic
def validate(headers):
    if headers.jwt:
        auth_result = validate_jwt(headers.jwt)
    else:
        # Fallback for legacy clients
        auth_result =  proxy_to_authsvc(headers.cookie)
    if auth_result.success:
        injject_headers(headers)

    # Implementation:
    #   success => UnidentifiedRequest, indentifier cont.
    #   failed  => Future.exception(
    #               HttpResponseException(
    #                  Response(finagle.http.Status(resp.statusCode))))
    return auth_result
```

Split the world into TWO

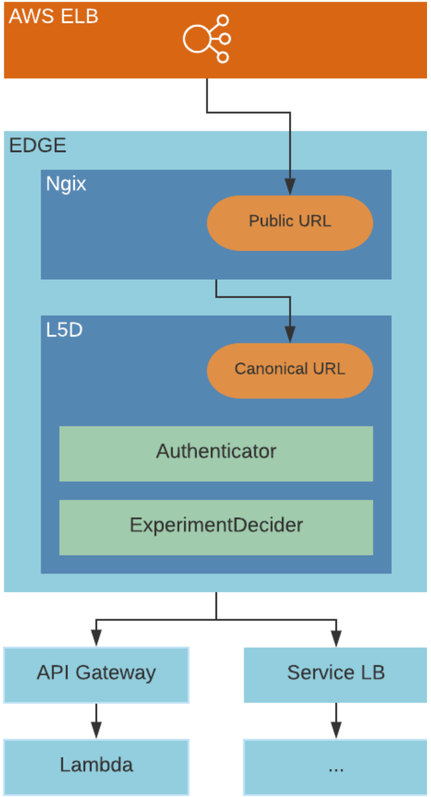


Edge LinkerD - Canary on Edge

ExperimentDecider

- Experiment, Service rollout
- Per service based
- Split traffic based on some criterial
  - User Id, group
- Controlled rolling out

```
/h1/dc1/prod/messaging
    => 0.5 * /h1/dc1/prod/messaging &
       0.5 * /h1/dc1/canary/messaging
```

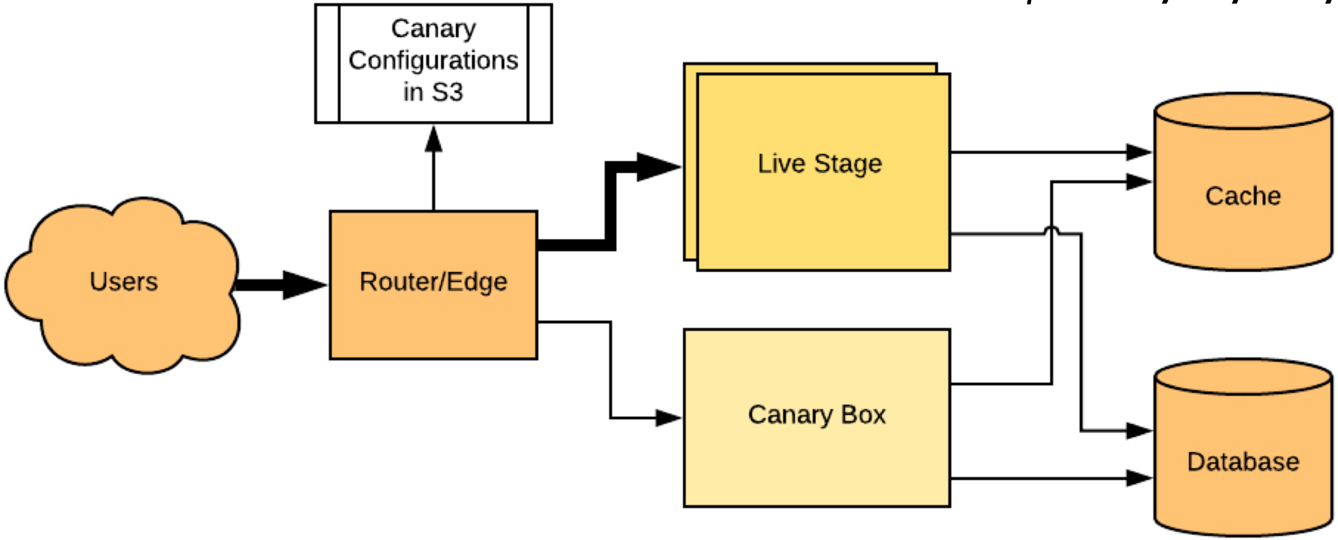


Edge LinkerD - Canary on Edge

Canary helps the roll out!

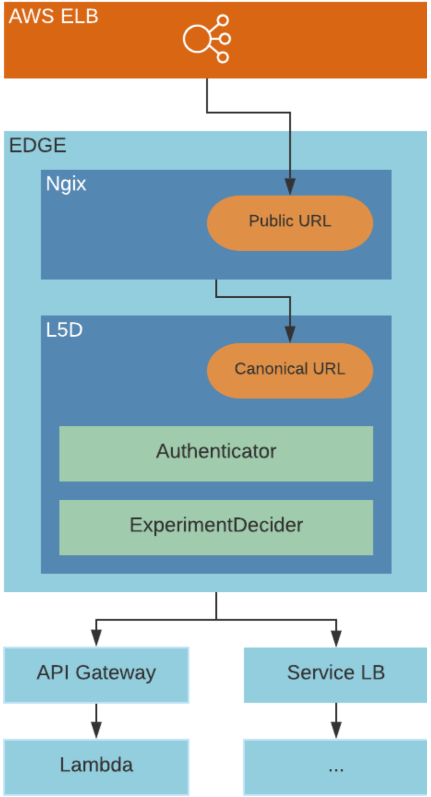
Controlled rolling out

```
/h1/dc1/prod/messaging
    => $r1 * /h1/dc1/prod/messaging &
       $r2 * /h1/dc1/canary/messaging
```



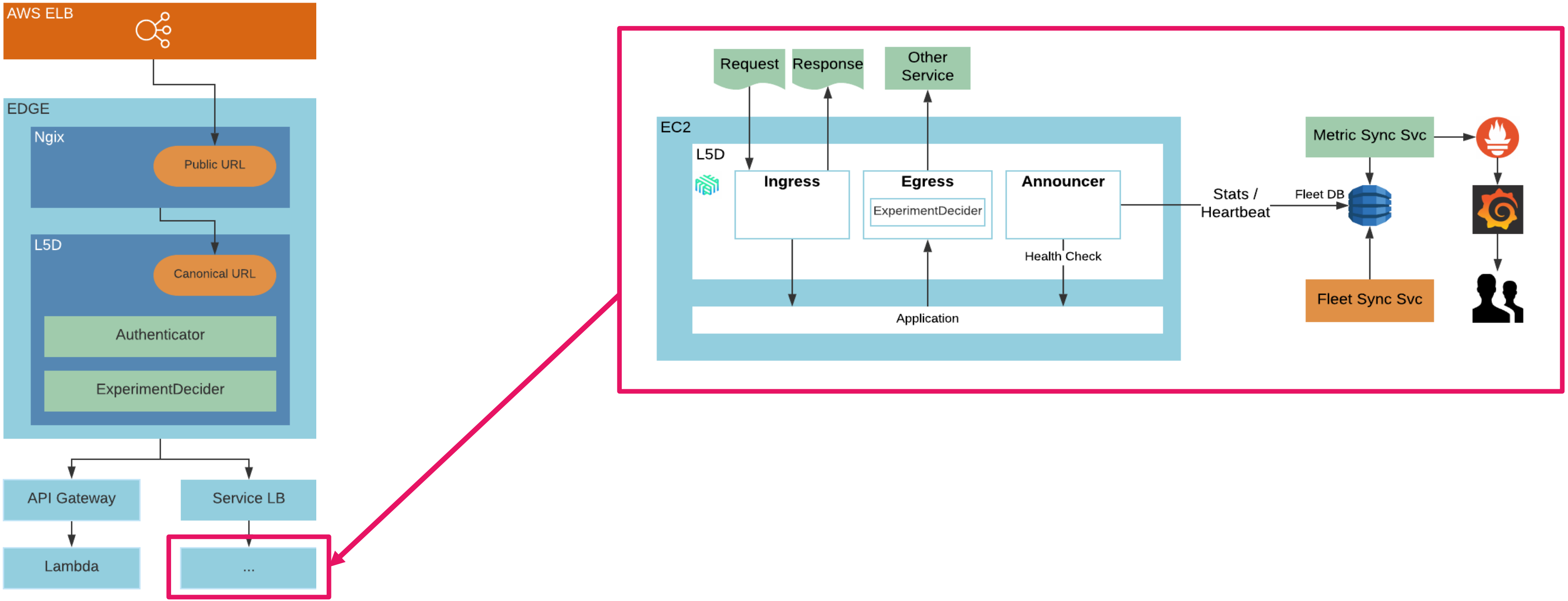
Summary: Split the world into TWO

- Pros
  - Horizontal scale ready
  - Flexibility Nginx + L5D
  - Full observability
- Cons
  - Double passing for all inbound traffic
  - (Free) Nginx lacks of control plate unlike L5D, but L5D provides some level of controlling features.



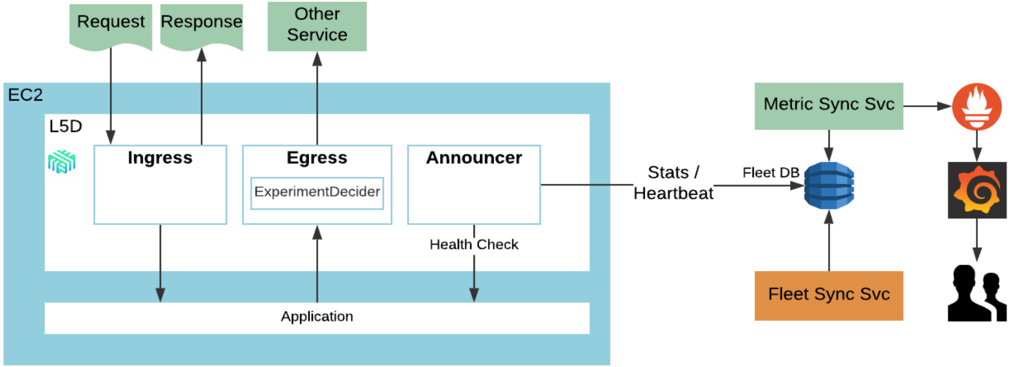
Service to Service Communication

Let’s get into the mesh!

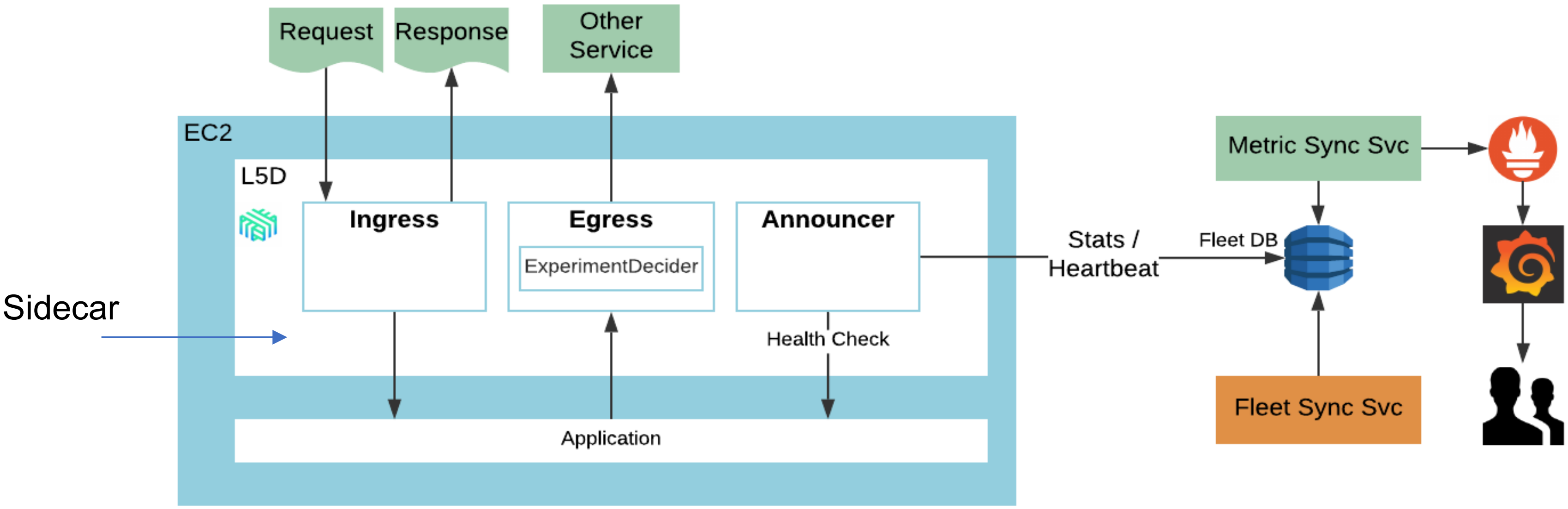


Service to Service Communication

- Supported protocols
  - HTTPv1
  - Thrift
- Why ingress/egress through LinkerD?
  - Connection policy control (retry/backout)
  - Connection pool
  - Circuit breaking
  - Etc...

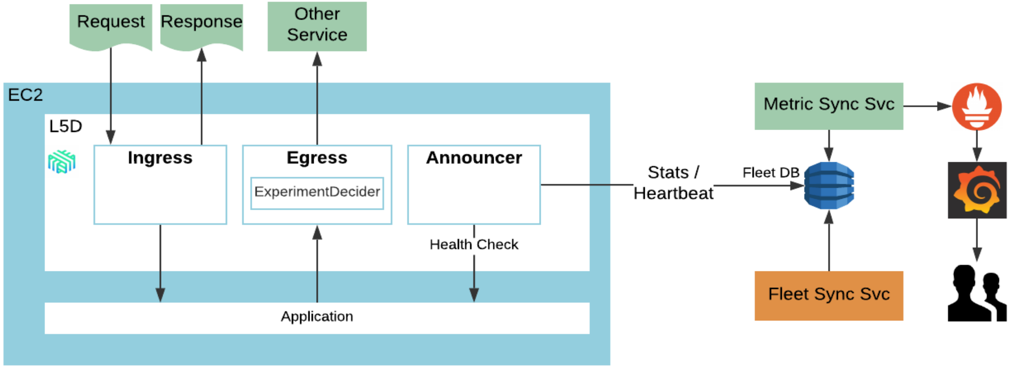


Service to Service Communication



Service to Service Communication

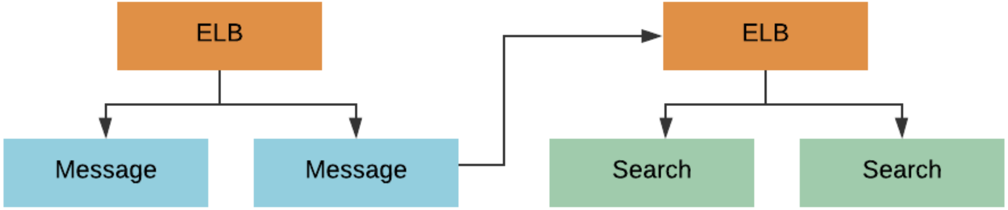
- How services talks to each other?
  - Discovery v1
    - Consul based registry
    - Linkerd Uses consul
    - Ingress point is ELB
  - Discover v2
    - Peer to Peer
    - Client based LB



Service to Service Communication - Discovery V1

- Discovery v1
  - Each service fronted by an ELB
  - DTab
    `#/io.l5d.consul/dc/$env/$service_name`
  - Namerd uses this information and query consul
    - **`$service_name maps to an ELB in consul`**

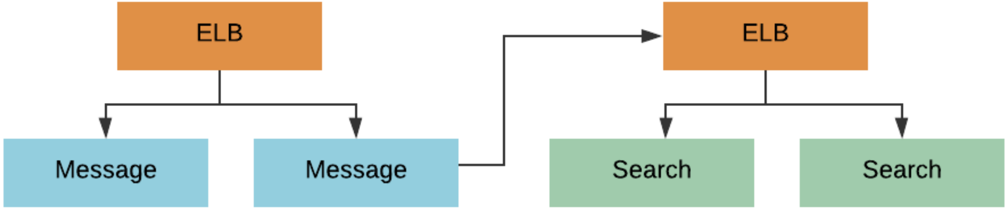
HashiCorp Consul



Service to Service Communication - Discovery V1

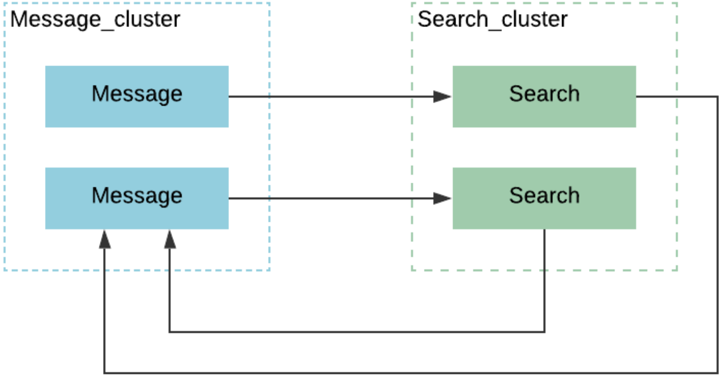
- All GOOD!
- But we can do better!
  - DNS
  - Features in L5D is per host
    - e.g. Retry budget
  - Single point of failure (LB level)
- Imbalance routing and load

HashiCorp Consul



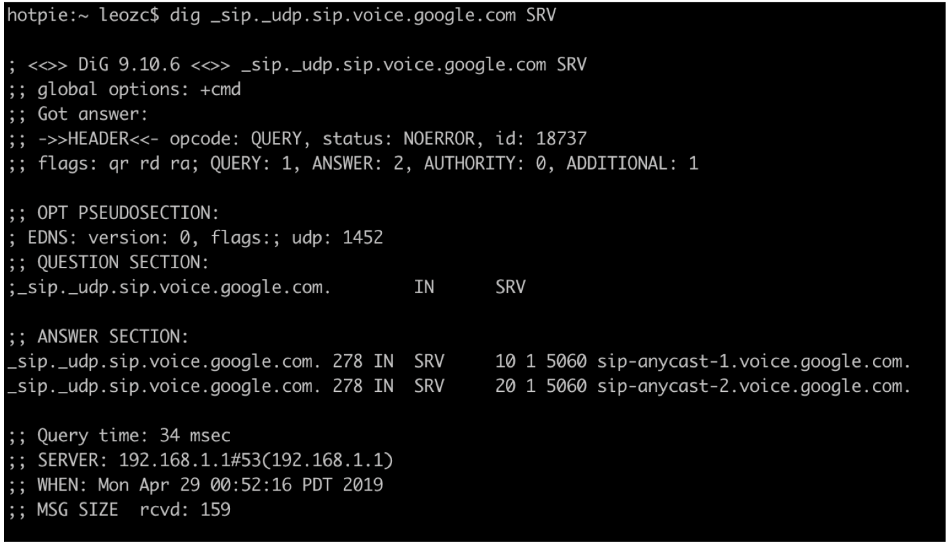
Service to Service Communication - Discovery V2

- Discovery v2 - SRV based routing
  - Per node
  - Peer to Peer
    - Powerful LB algorithms
      - Heap + Least Loaded, Power of Two Choices (P2C) + Least Loaded, Power of Two Choices (P2C) + Peak, EWMA Aperture + Least Loaded



Service to Service Communication - Discovery V2

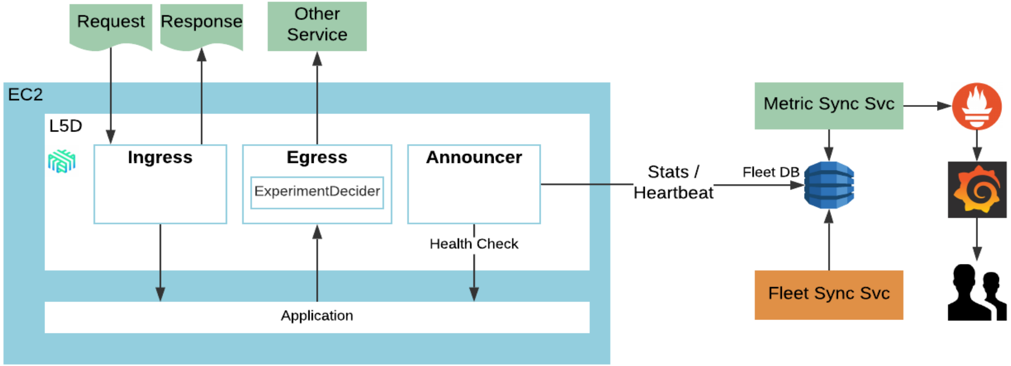
- What is SRV (RFC 2782)
  - CName for a GROUP of machines
  - Format: `Priority Weight Port Host/IP`
  - Example: **`10 5 80 172.0.0.4`**



```
hotpie:~ leozc$ dig _sip._udp.sip.voice.google.com SRV

; <<>> DiG 9.10.6 <<>> _sip._udp.sip.voice.google.com SRV
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18737
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1452
;; QUESTION SECTION:
;_sip._udp.sip.voice.google.com.        IN      SRV

;; ANSWER SECTION:
_sip._udp.sip.voice.google.com. 278 IN  SRV     10 1 5060 sip-anycast-1.voice.google.com.
_sip._udp.sip.voice.google.com. 278 IN  SRV     20 1 5060 sip-anycast-2.voice.google.com.

;; Query time: 34 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Mon Apr 29 00:52:16 PDT 2019
;; MSG SIZE  rcvd: 159
```

Service to Service Communication - Discovery V2

- Nodes automatically announce to FleetDB (Announcer)
- Fleet Sync Svc registers to Route53 as an SRV record
- NamerD returns all IPs of a SRV record.
- LinkerD LBs locally
- Fleet Sync Svc
  - Monitor FleetDB
  - hocks up signals
  - Rip off outdated info



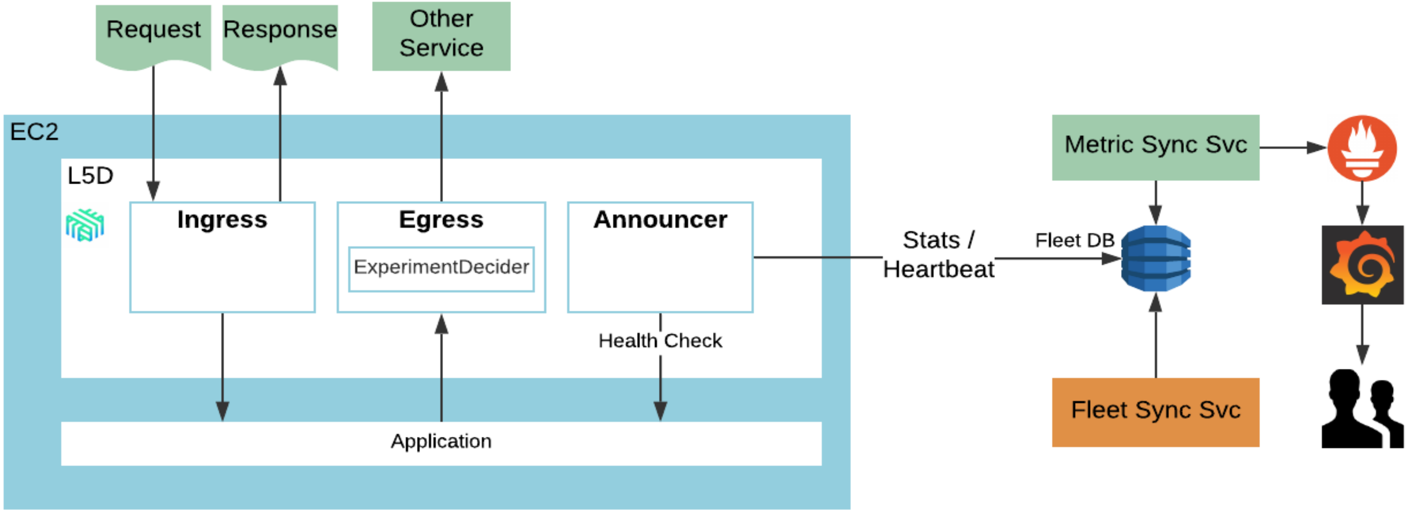
Service to Service Communication - Discovery V2

- Consistency - Eventual
- Availability - Strong
- Fault tolerance - Strong

DTab

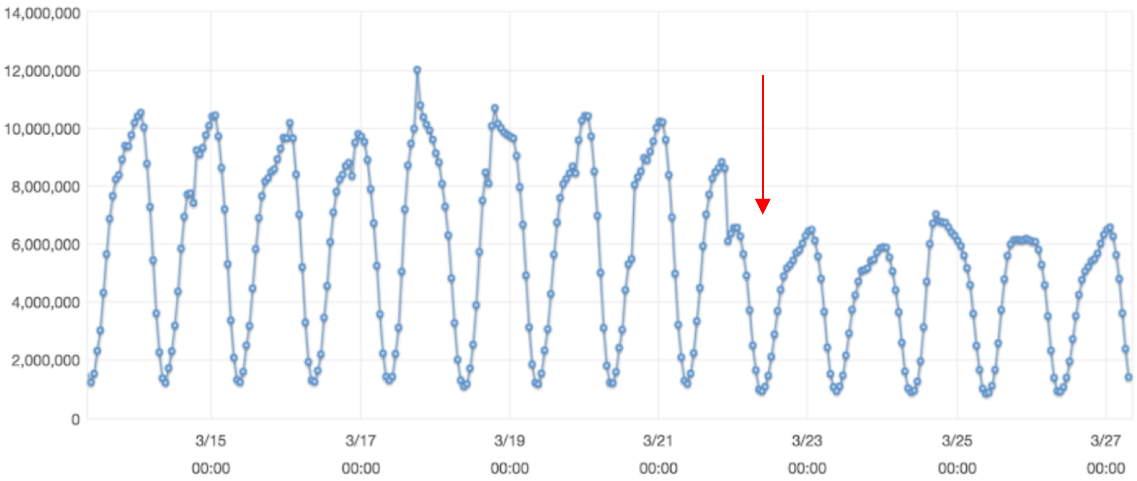
`/#/io.l5d.dnssrv/dc1/$env/$service_name.foo.com`

- `$service_name.foo.com` is SRV record



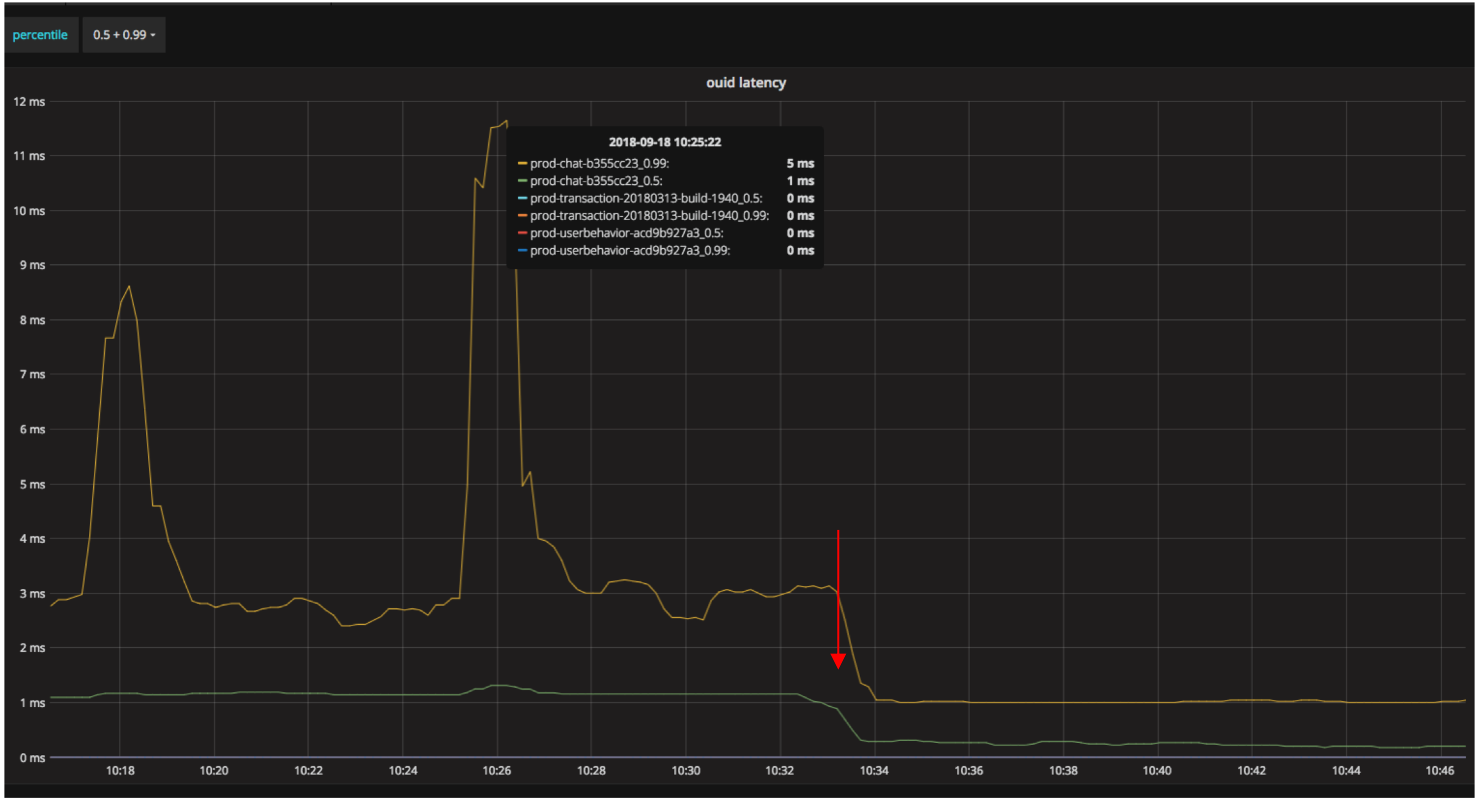
Service to Service Communication - Discovery V2

- Discovery v2 - SRV based routing
  - Peer to Peer
  - DTab `/#/io.l5d.dnssrv/dc1/$env/$service_name.foo.com`
  - `$service_name.foo.com` is SRV record



New Connection between two production services reduced significantly due to proper connection pooling via L5D

Discovery V1 vs Discovery V2

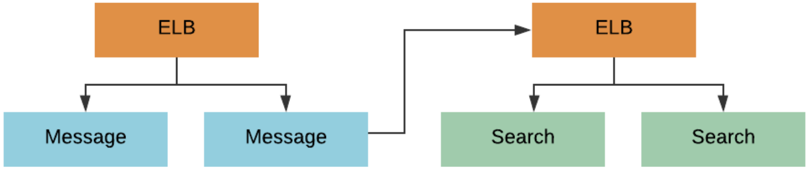


2018-09-18 10:25:22

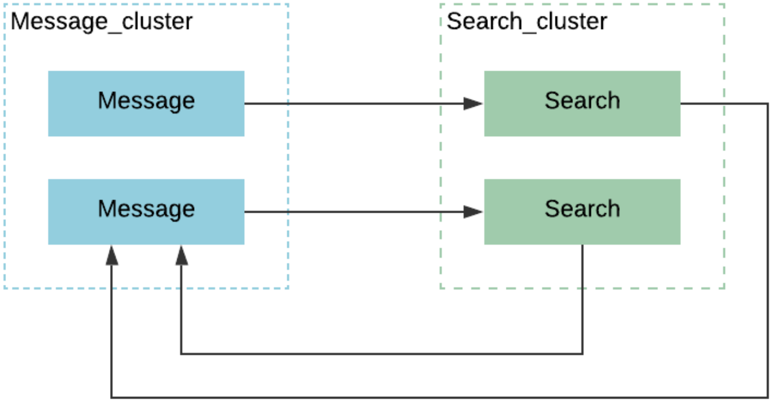
prod-chat-b355cc23_0.99:	5 ms
prod-chat-b355cc23_0.5:	1 ms
prod-transaction-20180313-build-1940_0.5:	0 ms
prod-transaction-20180313-build-1940_0.99:	0 ms
prod-userbehavior-acd9b927a3_0.5:	0 ms
prod-userbehavior-acd9b927a3_0.99:	0 ms

Discovery V1 vs Discovery V2

Discovery V1

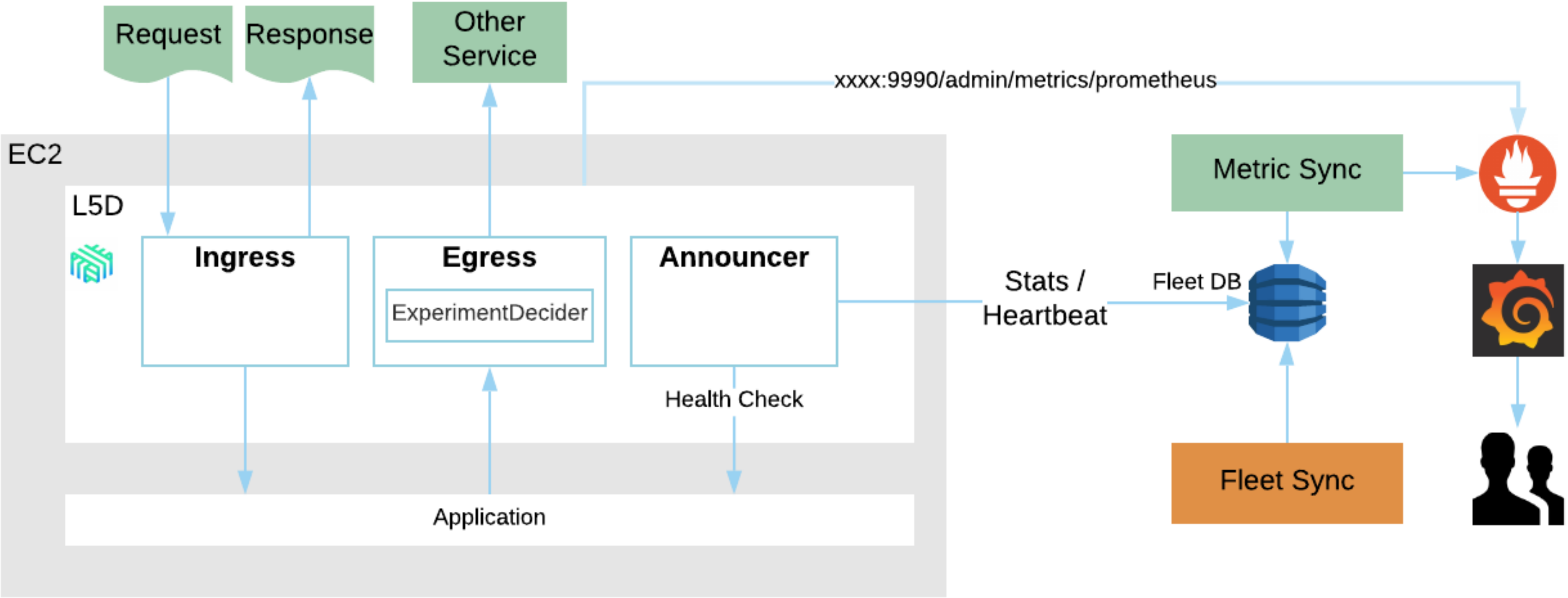


Discovery V2



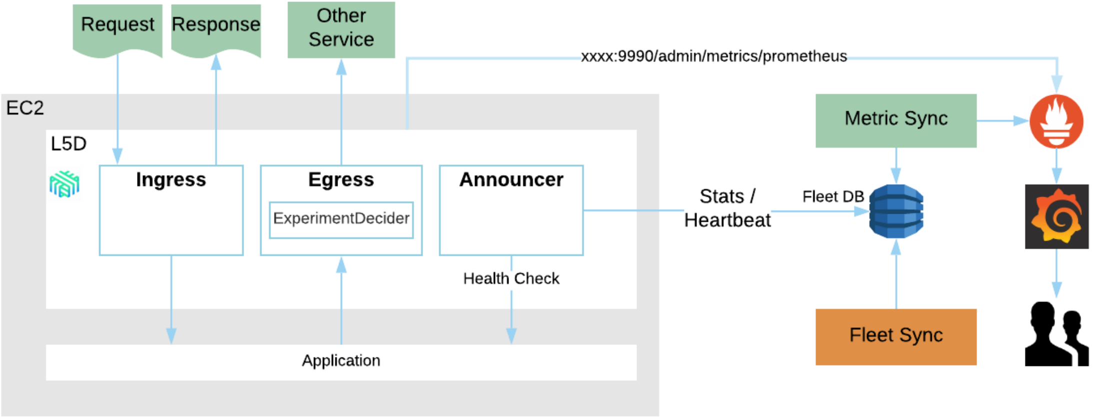
Observability

Observability



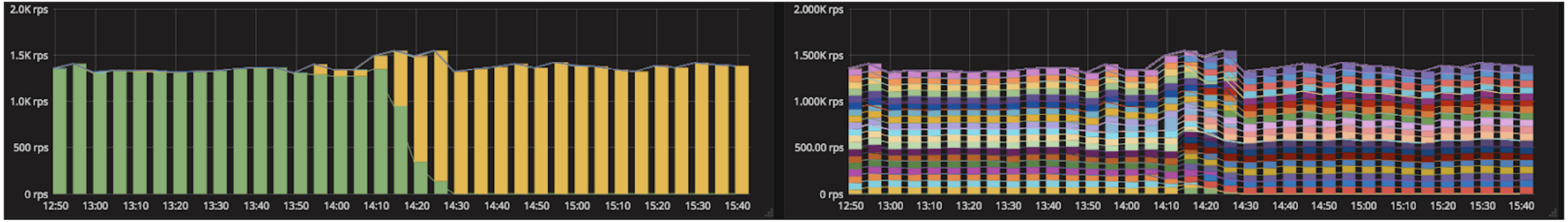
Observability

- 500K metrics
- 10 seconds intervals
- 40+ services
- Some metrics published to Cloudwatch which integrated with alarm flow.



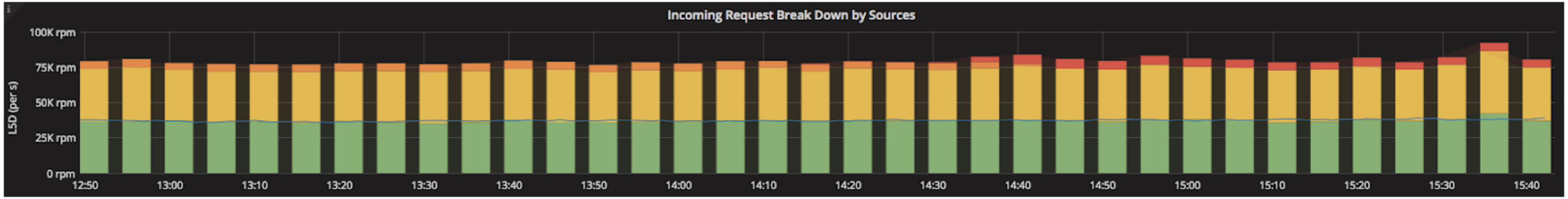
Observability - Gallery

Traffic shift from an old build to a new one during deployment



Observability - Gallery

A dashboard shows the incoming traffic breakdown by sources, with each color indicating a source of service



Conclusion

1. All inbound/outbound traffics are **observable** by L5D
2. All traffic between any two nodes are **controllable** by L5D using Dtab
3. All Service communication is **point-to-point**
4. Language agnostic traffic management
5. JVM 9+ - better GC & solved majority of long tail latency issues

KubeCon | CloudNativeCon

Europe 2019

Thank you!

@leozc