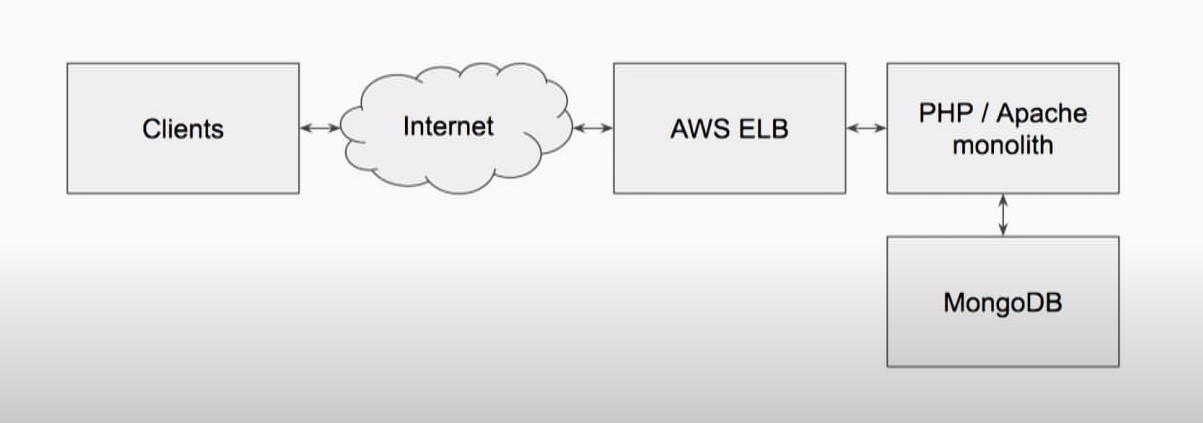
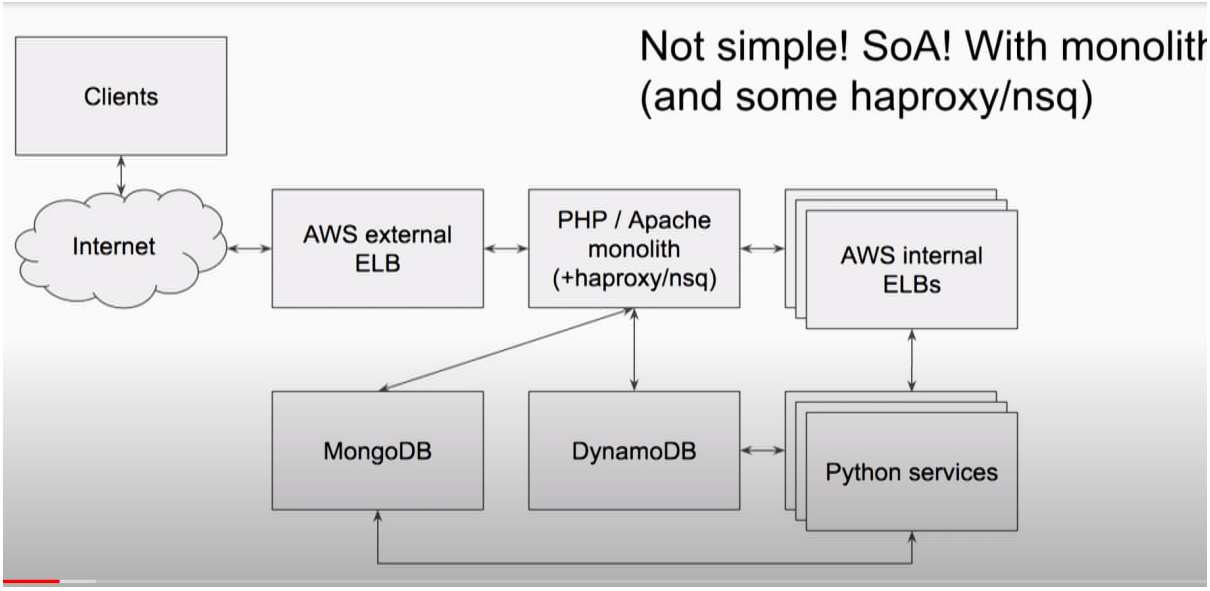
Envoy intro

Created By	
Stakeholders	
Status	
Type	
Created	@May 14, 2022 1:01 PM
Last Edited Time	@May 14, 2022 5:46 PM
Last Edited By	



- in the older days, clients would talk aover the internet to the loadbalancer, loadbalancer to php and php to MongoDB and even at these simple times, we start seeing some problems. Ex. PHP and apache are processor per connection model, can only handle so many connections at the same time. the loadbalancer doesn't know how to do that so it is connecting with many preconnections then we have to tell the aws to start turning off the features such as preconnect to make this work.



- Now, it is not simple anymore, we have clients talking over the internet to the external ELB, we still got this monolith but this monolith is trying to make a service calls to new backend services but the service architecture of the PHP doesn't allow us to make concurrent connections so we dropped in haproxy to fix that problem. We've got some queuing . We have got all of this operational tooling and scripts to actually make that work. So, making calls out to aws internal ELBs, we are also making calls to a bunch of python services. We've got two different data stores now- mangoDB and DynamoDB. and we are to keep them together. So from a deployement point of view it becomes very complicated. We don't have any tracking and logging is very difficult and it becomes very difficult to understand when things start to fail.

State of Service Oriented Architecture in Industry

- we have tons of languages and frameworks and it becomes very common for companies to use a polygot system.
- Per language libraries for making these service calls.
- Protocols(HTTP/1,HTTP/2,gRPC,databases, caching etc.)
- Infrastructure(IaaS,CaaS,on premise etc.)
- Immediate load balancers

- Observability output (stats, tracing and logging)
- implementations of retry, circuit breaking, rate limiting,timeouts and other distributed systems best practices.
- Authentication and authorizations

All of these things leads to a confusing mess. A lot of people don't want to build complex service oriented architecture due to lack of features and visibility in the traditional way of doing it which leads to helplessness in case the things go wrong.

People don't usually understand how all these components come together to build a system. So, debugging is difficult or impossible(each application exposes different stats and logs with no tracing), Also, there is limited visibility into infrastructure such as load balancers, databases, caches etc). Even if you do have a solution, you are likely using a library and are locked into a particular technology stack essentially forever. Also, libraries are really painful to upgrade.

Why Envoy?

- Ultimately, robust observability and easy debugging are everything.
- As SoAs become more complicated, it is critical that we provide a common solution to all of these problems or developer productivity grinds to a halt.

ok, What's Envoy?

The network should be transparent to applications.when network and application problems do occur it should be easy to determine the source of the problem.

- Out of the process architecture: Let's do a lot of really hard stuff in one place and allow application developers to focus on business logic.

It is a completely self-contained proxy (not a library). It is a server that you install and you run.

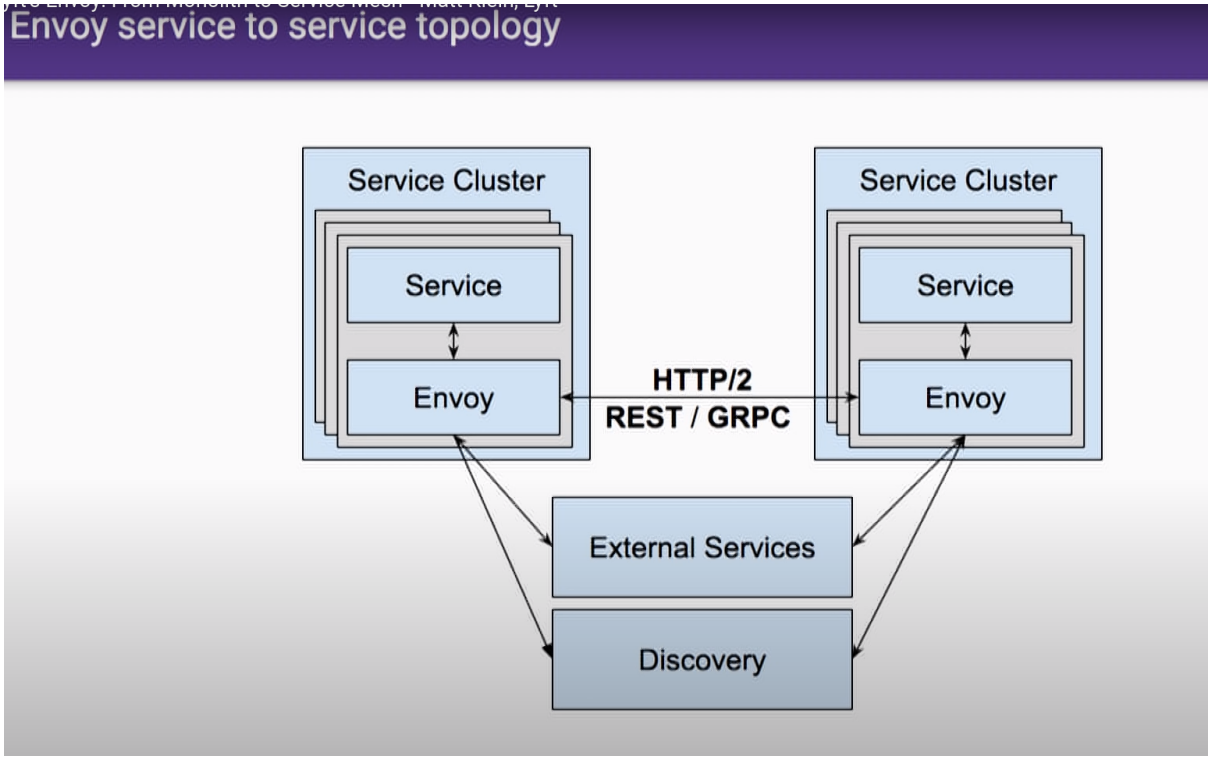
No matter what is your programming language, your application talks to envoy locally, envoy then does a lot of stuff and returns the response.

- 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.). So, connection come in, you read some bytes, you operate on those bytes and then you push those bytes back. That can be used for some simple things as

well as complex things such as MongoDB,redis etc. Hence, envoy is a generic holder for different filter mechanism that can operate on bytes.

- HTTP L7 filter architecture: Another layer on filter stack that allows us to filter messages on that level such as HTTP Header, body data etc. Also make it easy to plug in different functionality.
- HTTP/2 first : this is something that makes envoy unique among other proxies. other only support HTTP/2 on the front side and on the back side, they are using HTTP/1. But envoy is written to be HTTP/2 first, including gRPC and a nifty gRPC HTTP/1.1 bridge.
- Service Discovery(to know who your hosts are) and active/ passive health-check(to check whether they are healthy).
- Advanced load balancing.
- Best in class observability: stats,logging,tracing;
- Edge proxy: routing and TLS.

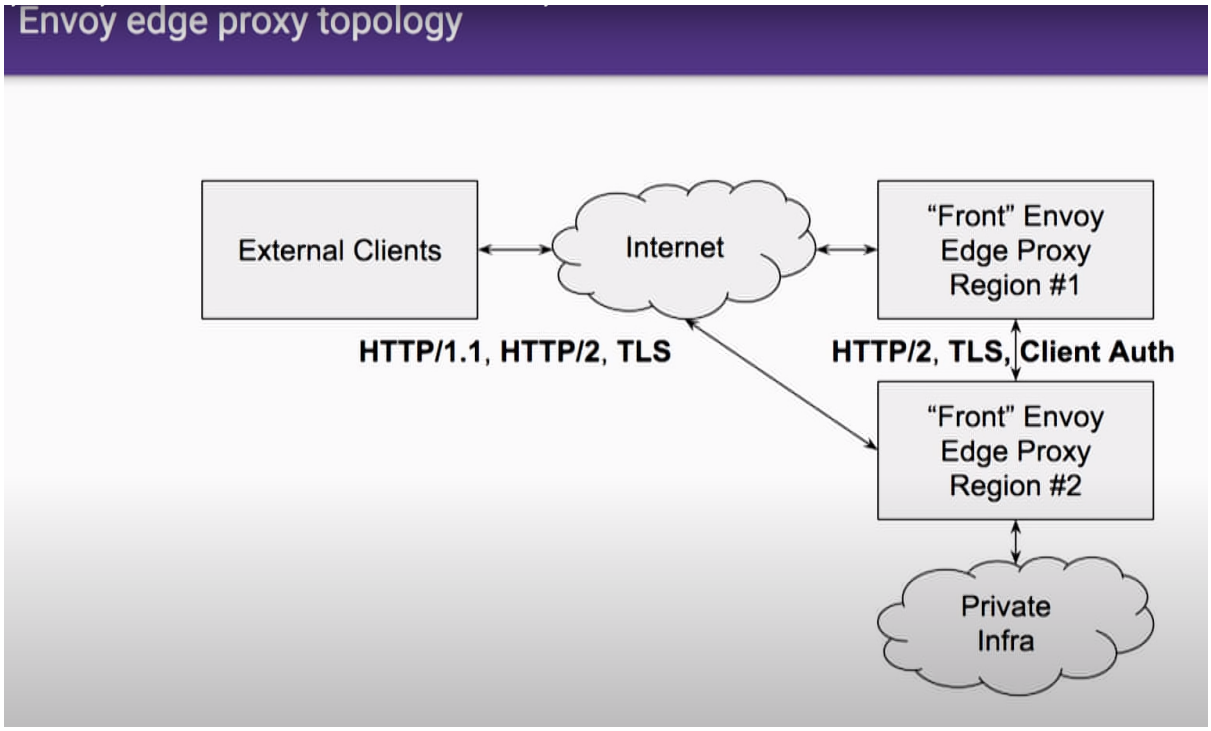
Envoy service to service topology



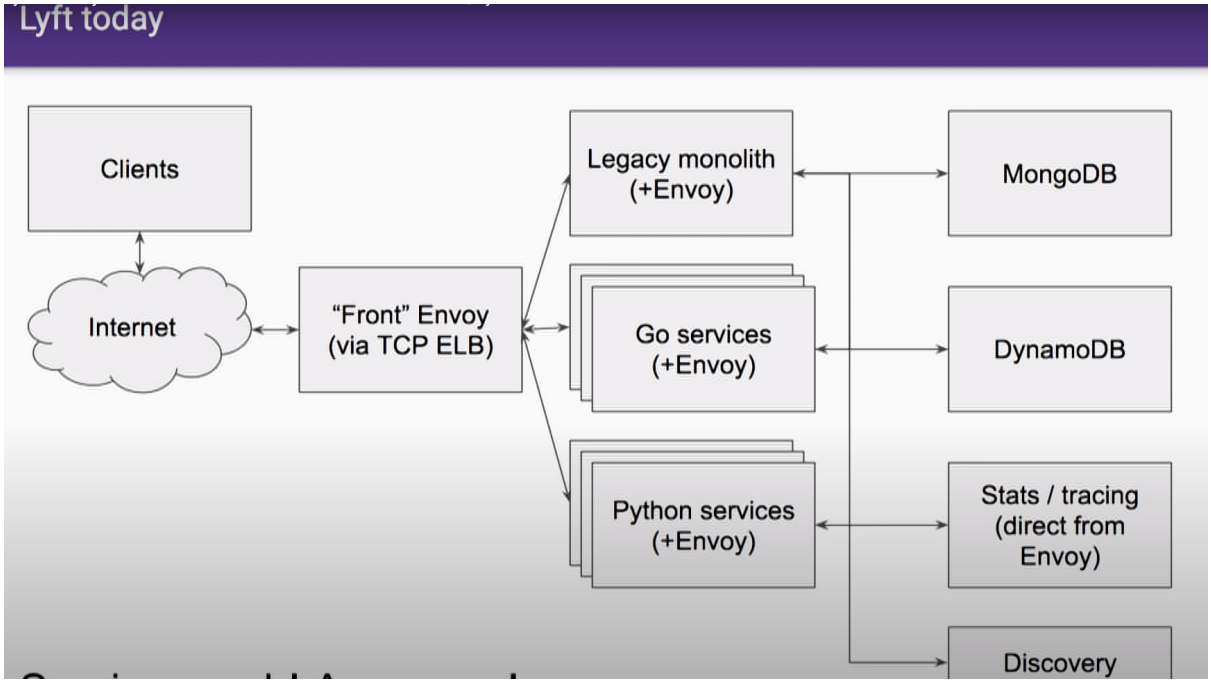
We have got our service cluster (could be pods/nodes etc). Colocated alognside a service (pod/nodes) is an envoy instance. The envoys communicate with each other typpically over HTTP2 (REST/GRPC). Envoys might also be proxing out to different services such as DynamoDB or other third party vendors . At last, they might also be talking to some type of discovery servers.

From a service perspective, the service is only aware of a local envoy and from a developer productivity perspective, it doesn't matter whether you are coding in your dev env or production, you write the code in the same way. The Dev instantiate an envoy client and they just give the name of the service that they wanna talk to . No matter what environment they are in, the system/ mesh is set up in a way that it works.

Envoy Edge Proxy Topology



Lyft Today



- many companies use fully consistent service discovery systems. Now, service discovery is not really a fully consistent problem. Nodes come, nodes go and there's failure and as long as you have an eventually consistent view of the host in your mesh and as long as that view converges, you are going to get to your end state. Due to the fully consistent nature of these consistent discovery services like zoo-keeper, these systems tend to be hard to run at scale.
- Envoy is designed from the get go to treat service discovery as lossy. Now, because of the lousy nature, we assume that we can't trust a particular service as it can go at any time (and come back) . So, we have another layer of active and passive health check which we use in combination with the service discovery and we get the following result

Discovery Status	HC OK	HC Failed
Discovered	Route	Don't Route
Absent	Route	Don't Route / Delete

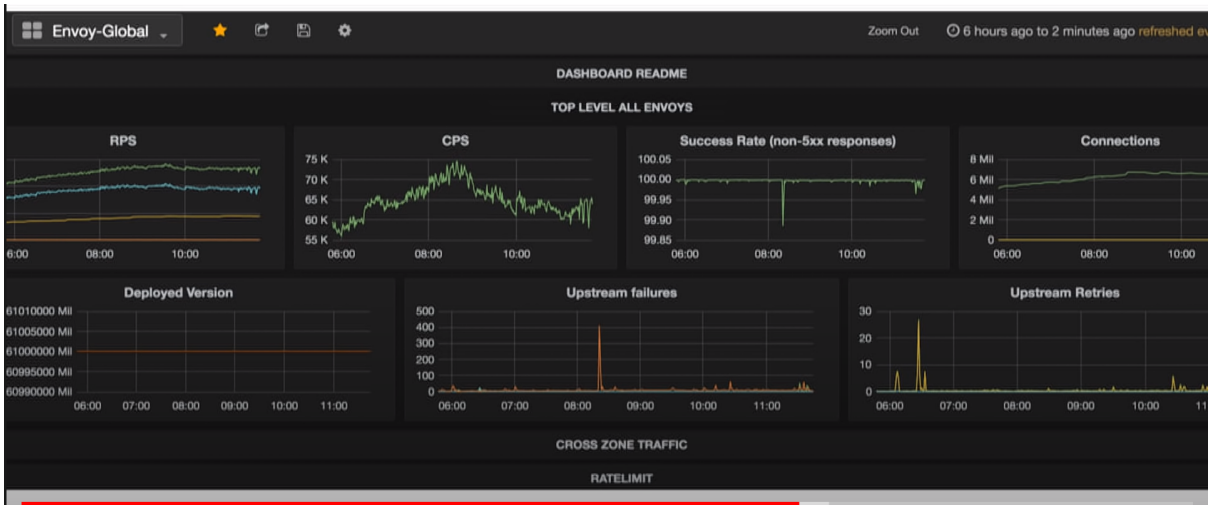
Over a period of time, we end up trusting the health check data more than the service discovery data. If the Health check data is OK then we route , no matter whether the discovery status show 'discovered' or ' absent'. And if the HC data fails then we don't route.

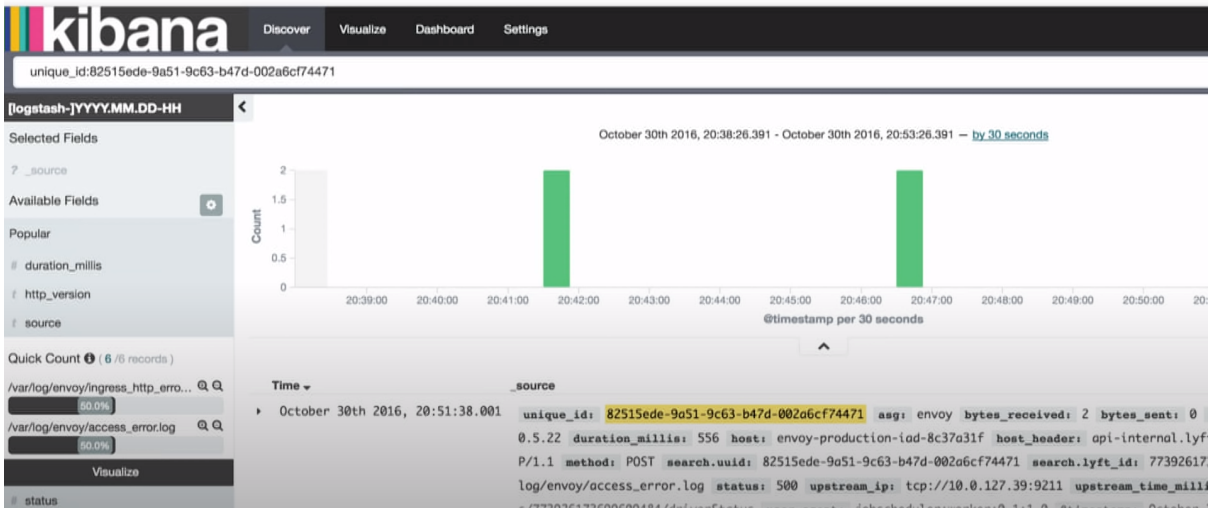
Advanced Load Balancing

- Zone aware least request load balancing: in services like AWS, cros zone requests may charge you money, but envoy optimizes the request to the same zone and then when it fails, it will properly fall over and send to multiple zones .(how does it save money?)
- Dynamic stats: Per zone stats, canary specific request etc.
- Circuit breaking: Max connections, requests and retries.
- Rate limiting : integration with global rate limiting services.
- Shadowing: Fork traffic to a test clusters.
- Timeoutes: Both outer and inner timeouts

Observability

- Observability is by far the most imp feature of the envoy.
- Consistent stats for every hop
- create and propagate a stable request ID / tracing context
- consistent logging




kibana
Discover
Visualize
Dashboard
Settings
unique_id:82515ede-9a51-9c63-b47d-002a6cf74471
[logstash-]YYYY.MM.DD-HH
Selected Fields
_source
Available Fields
Popular
duration_millis
http_version
source
Quick Count (6 /6 records)
/var/log/envoy/ingress_http_erro...
50.0%
/var/log/envoy/access_error.log
50.0%
Visualize
status
October 30th 2016, 20:38:26.391 - October 30th 2016, 20:53:26.391 — by 30 seconds
Count
@timestamp per 30 seconds
Time
_source
October 30th 2016, 20:51:38.001
unique_id: 82515ede-9a51-9c63-b47d-002a6cf74471 asg: envoy bytes_received: 2 bytes_sent: 0
0.5.22 duration_millis: 556 host: envoy-production-iad-8c37a31f host_header: api-internal.lyf
P/1.1 method: POST search.uuid: 82515ede-9a51-9c63-b47d-002a6cf74471 search.lyft_id: 77392617
log/envoy/access_error.log status: 500 upstream_ip: tcp://10.0.127.39:9211 upstream_time_mill