MIRANTIS

Open Infrastructure Summit, Denver

Service Mesh Comparisons

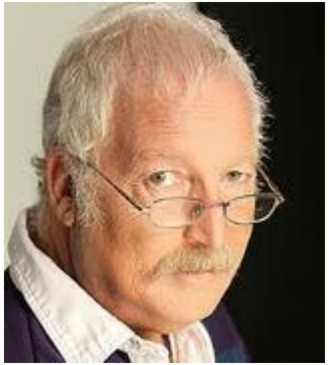
Or… The RETURN of the SMESH!

Introduction

Bruce Basil Mathews

Sr. Solutions Architect at Mirantis

Bruce has been a Senior Solutions Architect in the computer industry for 40+ years, working at multiple technology companies including Mirantis, HP, Oracle, Sun Microsystems and others. Email Bruce: bmathews@mirantis.com



What is a Service Mesh?

A service mesh is:

- A configurable infrastructure layer for a microservices application
- Makes communication between service instances flexible, reliable, and fast
- Provides a whole bunch of capabilities to relieve the developer from having to provide them each time uniquely.

What Capabilities Does a Service Mesh Provide?

- **Service Discovery** - This capability allows the different services to “discover” each other when needed. The Kubernetes framework keeps a list of instances that are healthy and ready to receive requests.
- **Load Balancing** - In a service mesh, load balancing capabilities place the least busy instances at the top of the stack, so that more busy instances can get the greatest amount of service without starving the least busy instances needs.
- **Encryption** - Instead of having each of the services provide their own encryption/decryption, the service mesh can encrypt and decrypt requests and responses instead.
- **Authentication and authorization.** The service mesh can validate requests BEFORE they are sent to the service instances.
- **Support for the circuit breaker pattern.** The service mesh can support the circuit breaker pattern, which can stop requests from ever being sent to an unhealthy instance. We will discuss this specific feature later.

Key Benefits of a Service Mesh

The combined use of the features and capabilities listed on the previous slide provide the means for *traffic shaping* or *QoS*:

- **Traffic shaping,** also known as packet shaping, is a type of network bandwidth management for the manipulation and prioritization of network traffic to reduce the impact of heavy use cases from affecting other users.
- **QoS** provides a uniform way to connect, secure, manage and monitor microservices and provides traffic shaping between microservices while capturing the telemetry of the traffic flow for prioritizing network traffic.

Circuit-breaking helps to guard against partial or total cascading network communication failures of application service instances by maintaining a status of the health and viability of each service instance:

- The Service Mesh’s **circuit-breaker feature** determines whether traffic should continue to be routed to a given service instance.

  NOTE: The **application developer** must determine **what to do** as a design consideration **when** the **service instance** has been marked as **not accepting requests**.

Enterprises can create **application features** using **modular software development** across **disparate teams** of developers.

- Faster development, testing and deployment of applications.

Service Mesh Architectures

Library

- Needed services are sitting in a Library that your microservices applications import and use.

Node Agent

- The services are provided by a Node Agent or daemon. The daemon services all of the containers on a particular node/machine.

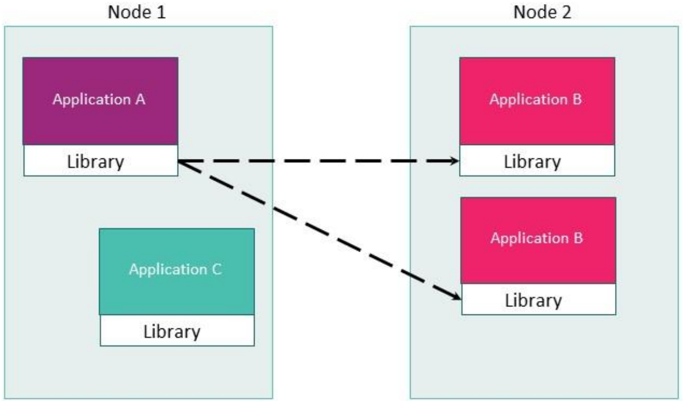
Sidecar

- The services are provided in a Sidecar container that runs alongside your application container.

The Types of Service Mesh Architectures

SMESH-ARCHS!

The Library Architecture



Examples:

Twitter Finagle
Netflix Hystrix
Netflix Ribbon

Each of the microservices carries a copy of the library that contains all of the desired Service Mesh functions.

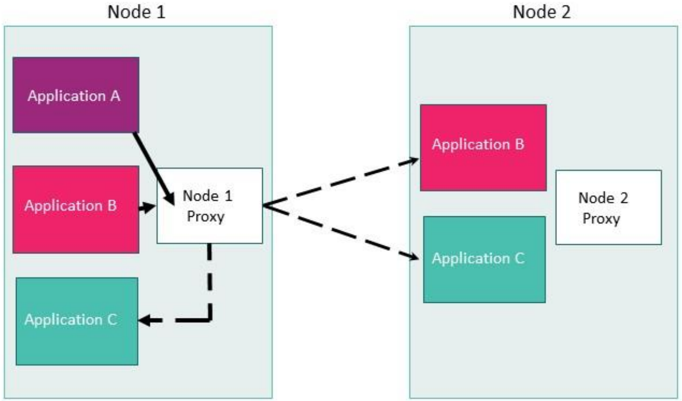
The Library architectural approach was:

- The first architecture to be adopted
- Simplest method to implement
- Has some drawbacks in terms of performance
- More difficult to maintain.

Separate copies are distributed with every microservice, which results in:

- Potential for version control issues in multi-cluster implementations
- Conflicting demand and performance is harder to determine and resolve quickly.

The Node Agent Architecture



Examples:

Linkerd
Consul

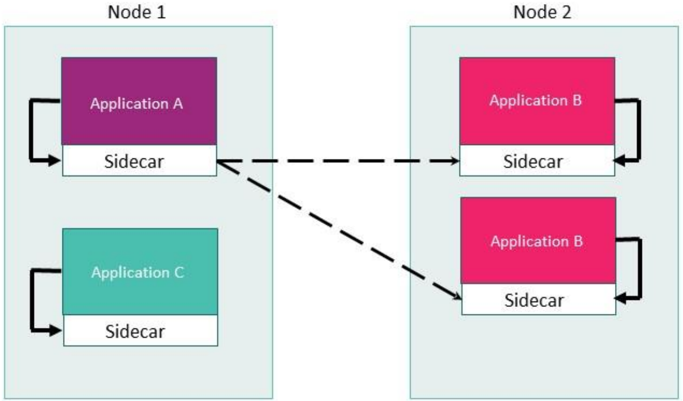
The Node Agent architecture is easier to manage and maintain than the Library architecture:

- It distributes one copy of the configuration to each node, rather than one copy of the configuration to each pod on each node.

In the Node Agent service mesh architecture, a separate agent is running on every worker node of a cluster:

- Node Agent, usually running in user process space, services the heterogeneous mix of workloads hosted on that worker node.
- The Node Agent architectural model emphasizes work resource sharing. This method can provide a great deal more efficiency than the Library model, but leaves a door open for resource abuse.
- Resource requests for memory and other components can be demanded and fulfilled immediately, starving other microservices. It is left up to the application developer to play nicely with others.

The Sidecar Architecture



Examples:

Istio
Aspen Mesh

Sidecar is the latest method developed for service meshing:

- The Sidecar service mesh deploys one adjacent container for every application container.
- The sidecar container handles all the network traffic in and out of the application container.
- To eliminate the potential for a network based attack, the sidecar has the same privileges as the application to which it is attached.
- Most Sidecar implementations, founded on security best-practices, limit the scope of the authorities necessary to complete the required intercommunication and then end their own process.
- The Sidecar acts in closely secured proximity to the application, almost like a function call from a Library rather than having to traverse the network to an external Node Agent for each intercommunication.

Service Meshes to Compare

SMESH-COMPS!

Service Mesh Offerings for Kubernetes and Mesos

Open Source Offerings

- Envoy
- Istio
- Linkerd
- Linkerd2

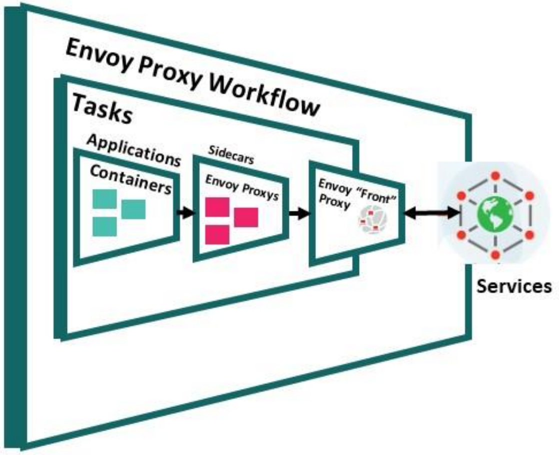
Commercial Offerings

- Consul
- Aspen Mesh
- Kong Enterprise Mesh
- AWS App Mesh

Open Source Service Meshes

OPS-SMESH!

Envoy’s Sidecar Architecture



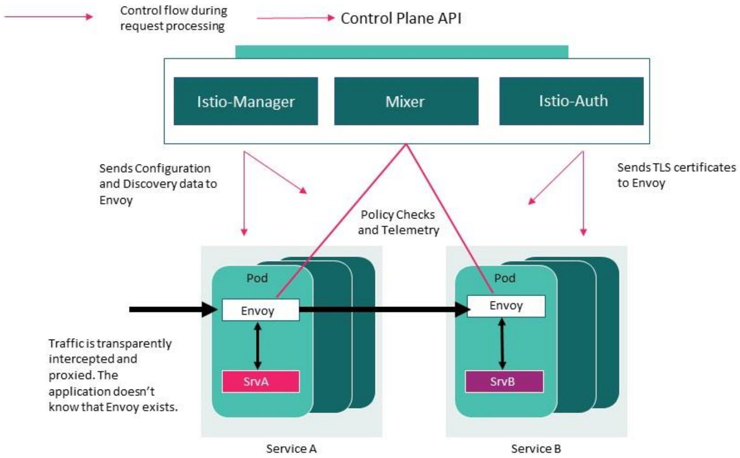
Envoy is a high performance C++ distributed proxy designed for single services and applications:

- Originally designed by Lyft
- Proxy architecture provides two key pieces missing in most stacks transitioning from legacy systems to a more Software Oriented Architecture (SOA.)
- Envoy’s two features:
  - Robust observability
  - Easy debugging
- Envoy uses gRPC bridge to unlock Python gevent clients

But the primary focus of Envoy is raw PERFORMANCE! To do this, Envoy broke its threading model down into three categories:

- **MAIN**: The main thread coordinates all the most critical process functionality
- **WORKER**: Each worker thread spawned processes all IO for the connection’s it accepts for the lifetime of the connection
- **FILE FLUSHER**: Worker writes to files are buffered to memory and then physically written by the file flush process thread.

Istio’s Sidecar Architecture and Integration with Envoy



Control flow during request processing
Control Plane API
Istio-Manager
Mixer
Istio-Auth
Sends Configuration and Discovery data to Envoy
Sends TLS certificates to Envoy
Policy Checks and Telemetry
Pod
Envoy
SrvA
Pod
Envoy
SrvB
Traffic is transparently intercepted and proxied. The application doesn’t know that Envoy exists.
Service A
Service B

Istio provides a uniform way to connect, secure, manage and monitor microservices and provides traffic shaping between microservices in a multi-cluster scenario:

- Originally developed by Netflix,
- includes the capability of *circuit-breaking* to the application development process.
- Guards against partial or total cascading network communication failures by maintaining a status of the health and viability of all service instances
- Envoy is integrated as the backend proxy for Istio

The Istio components and their functions are listed below:

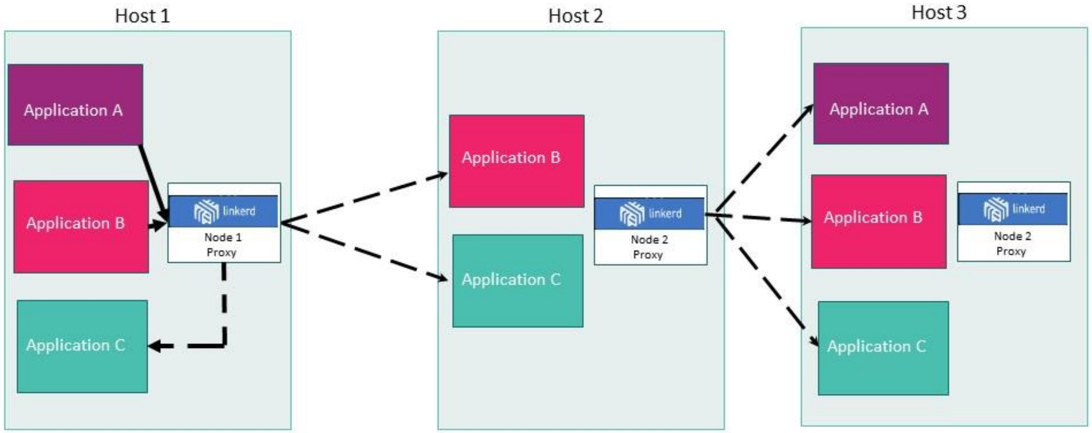
Control plane:

- Istio-Manager: provides routing rules and service discovery information to the Envoy proxies.
- Mixer: collects telemetry from each Envoy proxy and enforces access control policies.
- Istio-Auth: provides “service to service” and “user to service” authentication. This component also converts unencrypted traffic to TLS based traffic between services, as needed.

Data plane:

- Envoy: a feature rich proxy managed by control plane components. Envoy intercepts traffic to and from the service, applying routing and access policies following the rules set in the control plane.

Linkerd 1.0 Node Agent/Sidecar Architecture

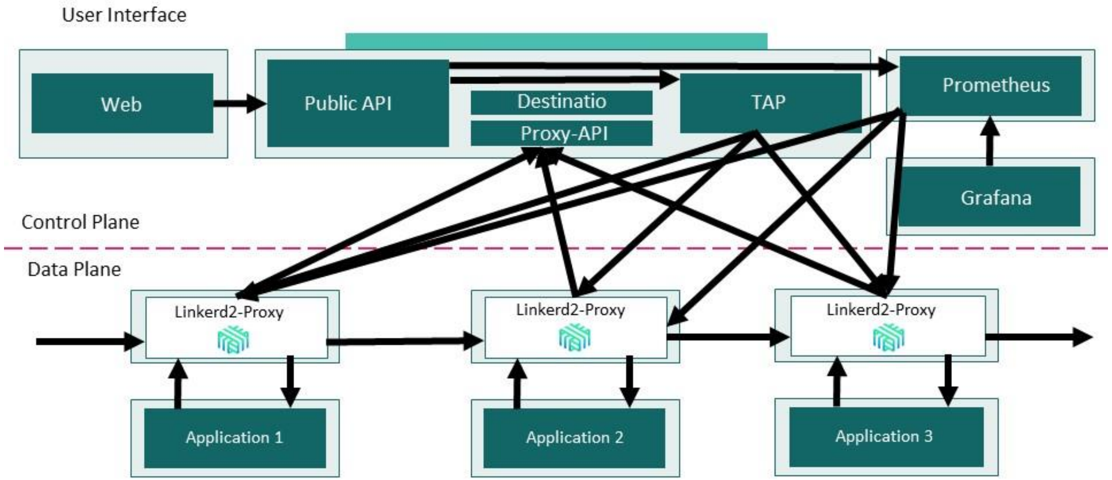


Note: Could be implemented as either, but the weight of the Java engine was so heavy it made it more resource effective as a node agent deployment.

In the per-host deployment model for Linkerd, one Linkerd instance is deployed per host (whether physical or virtual):

- Originally developed by Boyant.
- All application service instances on that host route traffic through the single instance.
- This model is useful for deployments that are primarily host-based.
- Each service instance on the host can address its corresponding Linkerd instance at a fixed location (typically, localhost:4140)
- Eliminates the need for any significant client-side logic.
- Since this model requires high concurrency of Linkerd instances, a larger resource profile is usually appropriate.
- In this model, the loss of an individual Linkerd instance is equivalent to losing the host itself.

Linkerd2 Sidecar Architecture



Linkerd 2 transitioned to deploying a sidecar methodology that eliminates the single point of failure per host.

Linkerd 2 has three basic components:

- User Interface
- Control Plane
- Data Plane

The control plane is made up of four components:

- Controller - In multiple containers
- Web - Dashboard for telemetry data display
- Prometheus - Telemetry data storage
- Grafana - Used to display the Dashboard

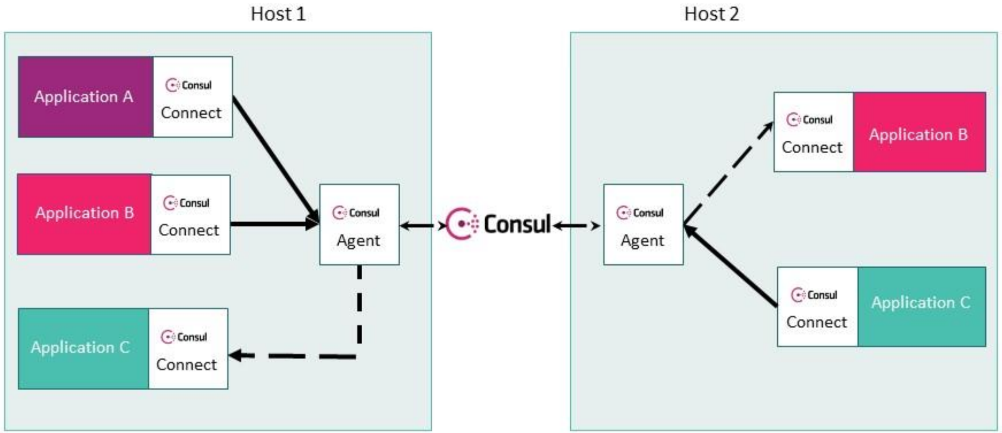
The data plane consists of the sidecar Proxy processes injected into each Pod

The Dashboard UI provides telemetry and Qos data for each service

Commercial Service Meshes

COMMS-SMESH!

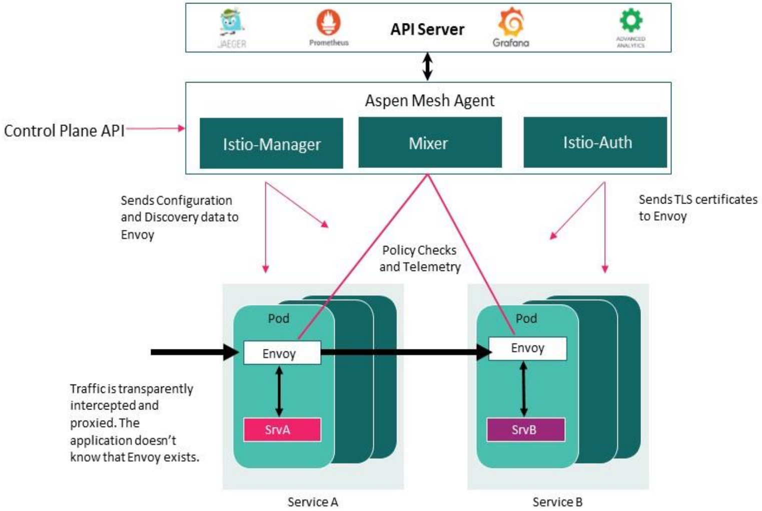
Consul Node Agent Architecture



Consul, is an ideal service to support control plane service mesh activities whether you employ a “dumb pipe” or a “smart network”:

- Originally developed by Hashicorp
- The Consul architecture ensures it is highly available, and extends beyond a single data center.
- Consul provides service discovery for both Dumb Pipe and Smart Network scenarios.
- Applications can use Consul’s key-value store to store retries, timeouts and circuit breaking settings and request them when needed.
- The Consul K/V store can be used for persistent state storage such as network policies as well.

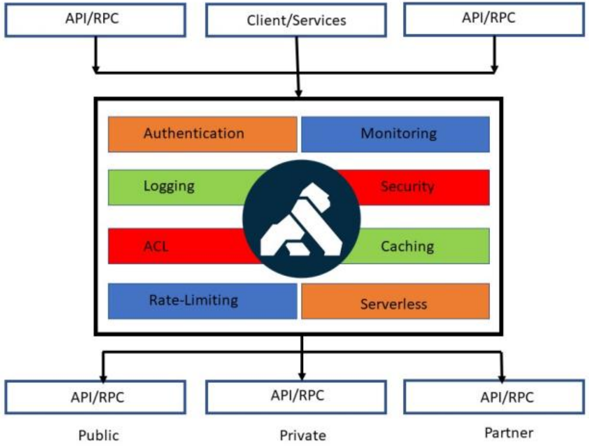
Aspen Mesh Sidecar Architecture and Integration with Istio


JAEGER
Prometheus
API Server
Grafana
ADVANCED ANALYTICS
Aspen Mesh Agent
Control Plane API
Istio-Manager
Mixer
Istio-Auth
Sends Configuration and Discovery data to Envoy
Sends TLS certificates to Envoy
Policy Checks and Telemetry
Pod
Envoy
SrvA
Pod
Envoy
SrvB
Traffic is transparently intercepted and proxied. The application doesn't know that Envoy exists.
Service A
Service B

Aspen Mesh is built on Istio and includes all the features you get with Istio plus:

- Intuitive UI
- Multi Cluster and Hybrid Cloud
- Analytics and Alerting
- Pre-configured Canary Testing
- Advanced Policy Management
- Isolated Management Platform with SLA
- A fully tested and hardened version
- Enterprise-level Support

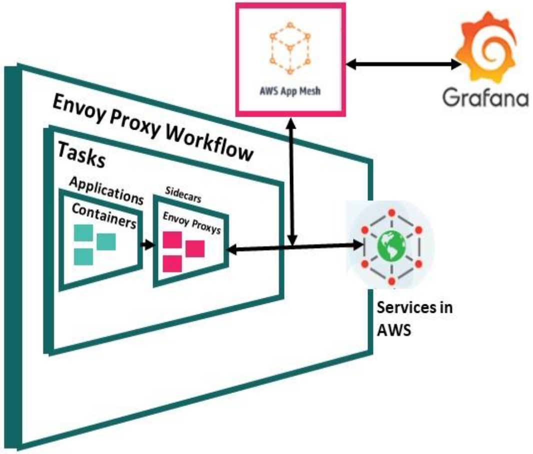
Kong Enterprise Monolith, Microservice, Service Mesh, & Serverless Architecture



Kong Ingress Controller is a dynamic and highly available Ingress Controller which configures Kong using Ingress resources created in your Kubernetes cluster:

- Kong can configure plugins, load balancing, health checking on your services running in Kubernetes.
- Kong Ingress Controller is a Go application that listens to events from the API-server of your Kubernetes cluster
- The Ingress Controller sets up Kong to handle your configuration accordingly. You never have to configure Kong manually.
- The controller can configure any Kong cluster via a Kong node running either in a control-plane mode or running both, control and data planes.
- Kong's state is stored in Postgres or Cassandra.
  - The database should be deployed as a StatefulSet.
  - All Kong nodes in your Kubernetes cluster should be able to connect to it.
- If Kong is deployed in Control-plane and Data-plane mode, then Kong proxy can be scaled

AWS APP MESH Sidecar Architecture and Integration with Envoy



AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control microservices:

- App Mesh standardizes how your microservices communicate: End-to-end visibility
- Helps to ensure high-availability
- App Mesh is made up of the following components:
  - **Service mesh**
  - **Virtual services**
  - **Virtual nodes**
  - **Envoy proxy**
  - **Virtual routers**
  - **Routes**

To use App Mesh, you must have an existing application running on AWS Fargate, Amazon ECS, Amazon EKS, Kubernetes on AWS, or Amazon EC2.

Comparing Service Meshes

COMPS-SMESH!

Some Service Mesh Feature Comparative Data

- Envoy
  - Low p99 tail latencies at scale when running under load,
  - Acts as a L3/L4 filter at its core with many L7 filters provided out of the box,
  - Support for gRPC, and HTTP/2 (upstream/downstream),
  - API-driven, dynamic configuration, hot reloads,
  - Strong focus on metric collection, tracing, and overall observability.
- Istio
  - Security features including identity, key management, and RBAC,
  - Fault injection,
  - Support for gRPC, HTTP/2, HTTP/1.x, WebSockets, and all TCP traffic,
  - Sophisticated policy, quota, and rate limiting,
  - Multi-platform, hybrid deployment.

Some Service Mesh Feature Comparative Data (Continued)

Linkerd was rewritten from Java to Rust - Data Plane/Go - Control plane about 1 ½ years ago. They are striving to reach feature parity between the two versions and are currently very close:

Linkerd and Linkerd2 Features

- ○ Support for multiple platforms (Docker, Kubernetes, DC/OS, Amazon ECS, or any stand-alone machine),
- ○ Built-in service discovery abstractions to unite multiple systems,
- ○ Support for gRPC, HTTP/2, and HTTP/1.x requests + all TCP traffic.

Note: Linkerd 2 is much faster than Linkerd as the Data Plane Proxy was rewritten in Rust and designed to be purpose built to serve as a Service Mesh Proxy for the Linkerd Control Plane. As the developer put it, where the others are a Swiss Army Knife, the Linkerd-Proxy is like a very fine needle to fit the Proxy role specifically.

- Linkerd is fully Open Source, but they provide Buoyant for commercial support of Linkerd

Linkerd 1 and Linkerd 2 Feature Comparative Data

	Linkerd 1.x (latest: 1.6.2.1)	**Linkerd 2.x** (latest: 2.3)
Theme	Powerful, highly configurable, “industrial strength”	Lightweight, minimalist, zero config
Observability	Automatic, non-aggregated	Automatic, aggregated, live, per-path
Reliability	Load balancing, retries, circuit breaking	Load balancing, retries, circuit breaking*, rate limiting*
Security	TLS, cert validation	TLS, cert validation, cert management, policy enforcement*
Protocol support	HTTP/1.x, HTTP/2, gRPC, Thrift	HTTP/1.x, HTTP/2, gRPC
Installation	Cluster-wide	Per service, per namespace or cluster-wide (incremental)
Resource footprint	100--150mb per proxy	<10mb per proxy
Latency introduced	<5ms p99	<1ms p99
License, building blocks	CNCF, Apache v2, Finagle, Netty, Scala	CNCF, Apache v2, Rust, Go
Supported platforms	Kubernetes, DC/OS, Mesos, bare metal, ECS, ...	Just Kubernetes… for now! (*roadmap)

Some Service Mesh Feature Comparative Data (Continued)

- Consul
  - Consul is a single binary providing both server and client capabilities
  - includes all functionality for service catalog
  - Includes configuration capabilities, TLS certificates, Authorization
  - Consul optionally supports external systems such as Vault to augment behavior
  - No additional systems are required to deploy Consul
- Aspen Mesh
  - Provides an intuitive UI that brings the most important information to the forefront
  - Easy-to-understand real-time status information
  - Allows for the display of multiple clusters in a single pane of glass
  - Helps the user to monitor their entire architecture in one place
  - Allows the user to sort, search and drill into individual services details

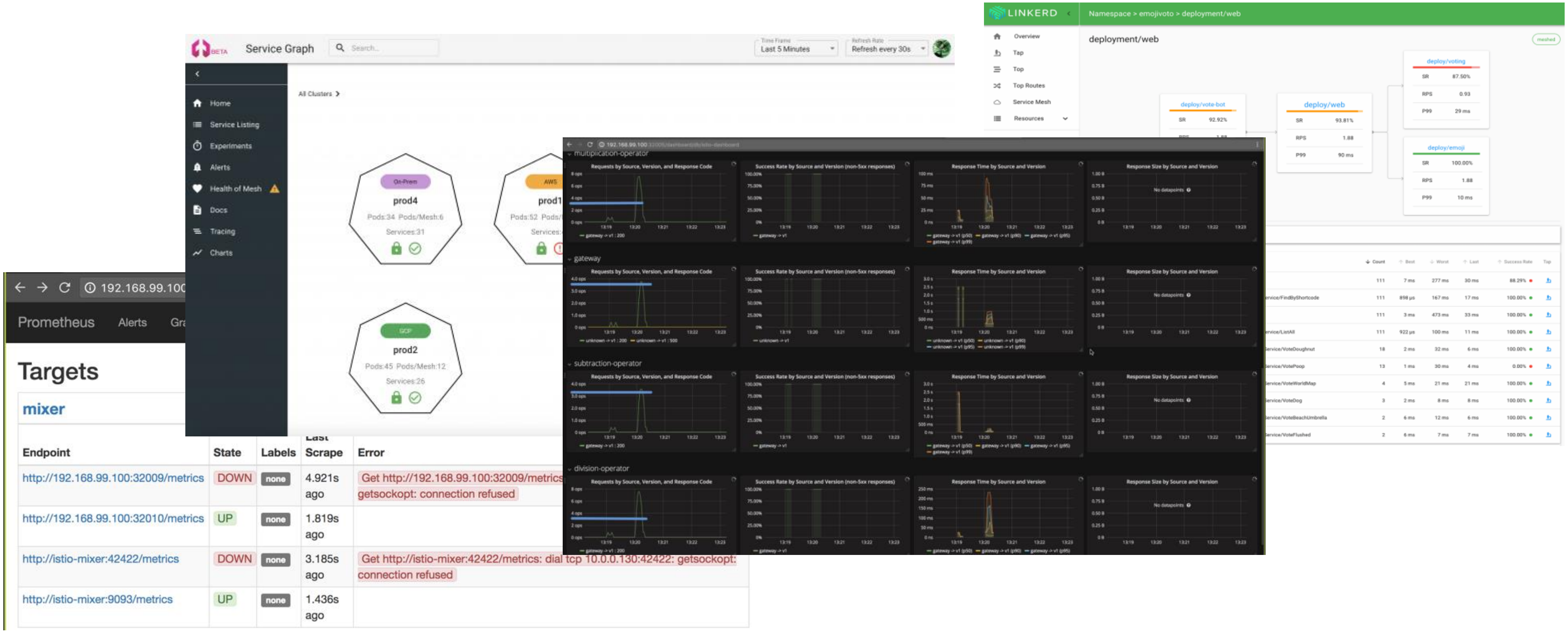
Some Service Mesh Feature Comparative Data (Continued)

- ## Kong Enterprise 1.0
  - Engineers can now make a change in a centralised location that will be reflected across multiple Kong clusters
  - Includes an implementation of the Plugin Development Kit (PDK)
  - All Kong plugins require a standard set of functionality, which the PDK provides out of the box
  - PDK based on the ngx_lua API
  - A Kong cluster deployment requires the installation of a Cassandra of PostgreSQL to act as the datastore
- ## AWS App Mesh
  - App Mesh separates the logic for monitoring and controlling communications into a proxy that manages all network traffic for each service
  - App Mesh uses the open source Envoy proxy to manage all traffic into and out of a service’s containers
  - Integrated with Amazon CloudWatch – monitoring and logging service for complete visibility of resources and applications.
  - Developers configure services to connect directly to each other instead of requiring code within the application or using a load balancer
  - App Mesh works with services managed by Amazon ECS, Amazon EKS, AWS Fargate, Kubernetes running on EC2, and services running directly on EC2

Service Mesh Visualization

SMESH-VIS!

Most Service Meshes use Prometheus with Grafana



Targets

mixer

Endpoint	State	Labels	Last Scrape	Error
http://192.168.99.100:32009/metrics	DOWN	none	4.921s ago	Get http://192.168.99.100:32009/metrics getsockopt: connection refused
http://192.168.99.100:32010/metrics	UP	none	1.819s ago	
http://istio-mixer:42422/metrics	DOWN	none	3.185s ago	Get http://istio-mixer:42422/metrics: dial tcp 10.0.0.130:42422: getsockopt: connection refused
http://istio-mixer:9093/metrics	UP	none	1.436s ago	

NOTE: Commercial offerings provide a greater variety of templates and statistics to choose from for traffic shaping analysis.

Comparing Service Mesh Performance

PERF-SMESH!

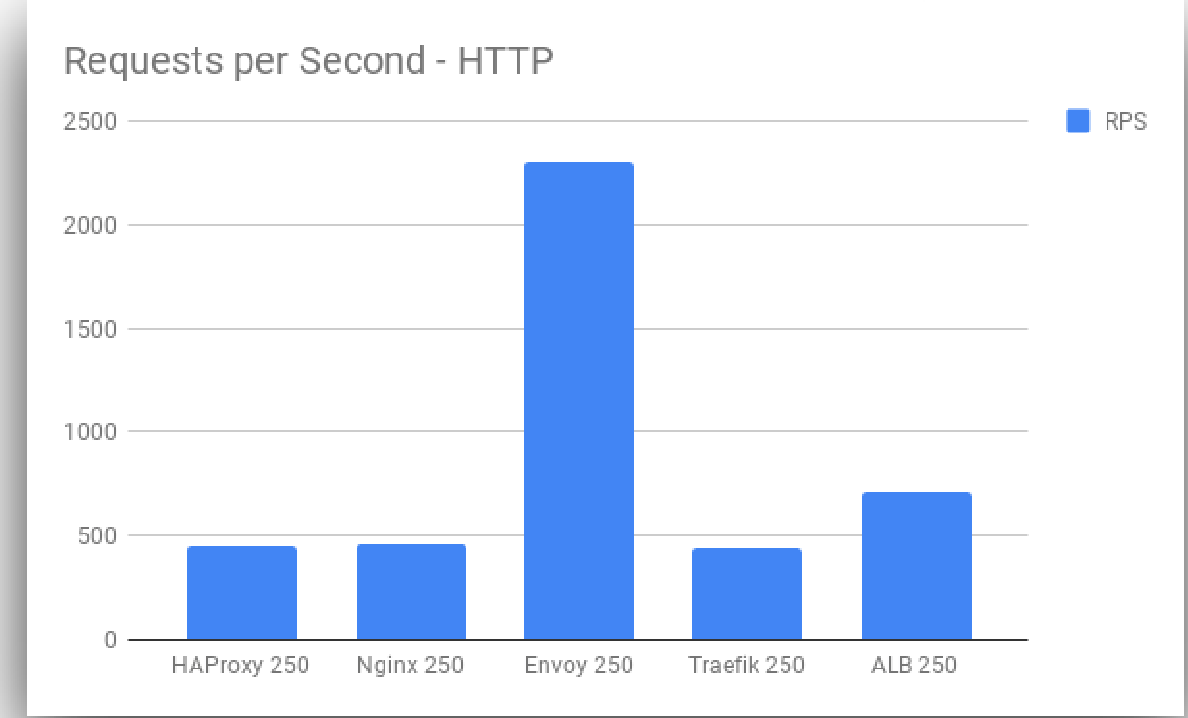
Some Service Mesh Performance Factors

- Control Plane performance factors include:
  - The rate of deployment changes.
  - The rate of configuration changes.
  - The number of proxies connecting to the service gateway.
- Data Plane performance factors include:
  - Number of client connections
  - Target request rate
  - Request size and Response size
  - Number of proxy worker threads
  - Protocol
  - CPU cores
  - Number and types of proxy filters
- Benchmarking Tools:
  - `fortio.org` - a constant throughput load testing tool.
  - `blueperf` - a realistic cloud native application.

Some Service Mesh Comparative Performance Data

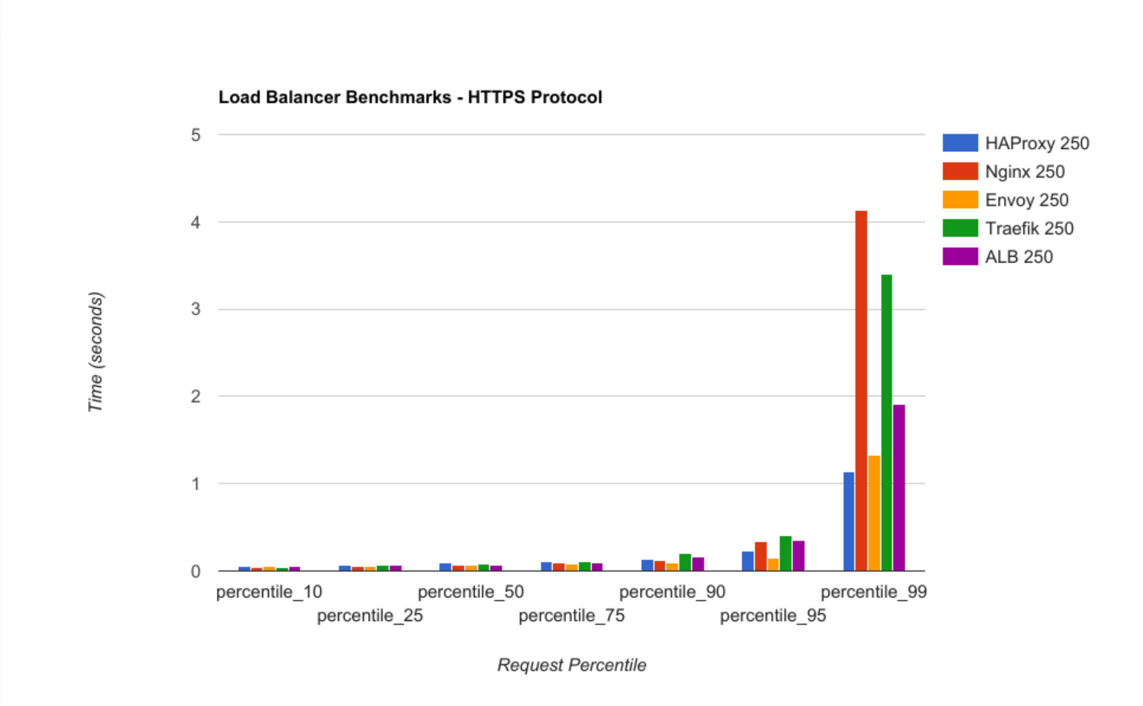
- Load Balancing Throughput and Latency
  - One of the key features of a service mesh is to provide a load balancing capability.
  - Some of the more popular load balancers that can be integrated with the Service Meshes compared in this presentation are:
    - NGINX
    - HAProxy
    - Envoy
    - Traefik
    - Amazon Application Load Balancer (ALB)

Load Balancer Requests-per-Second Performance Data



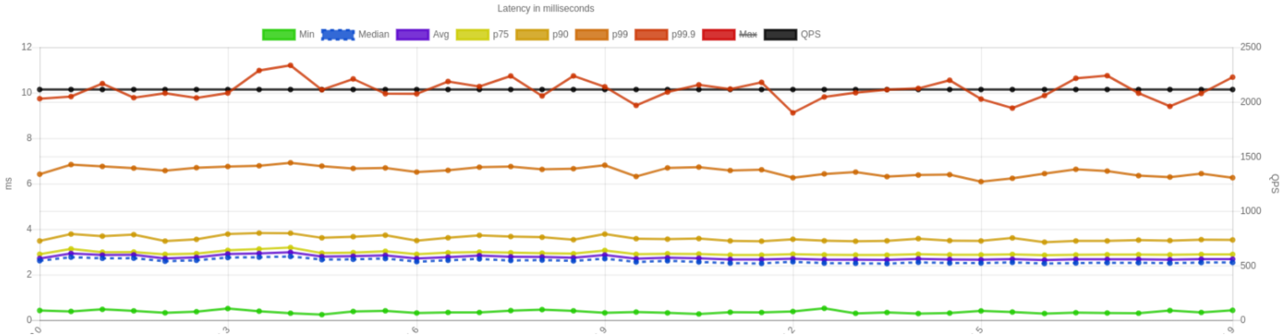
Data found here: https://www.loggly.com/blog/benchmarking-5-popular-load-balancers-nginx-haproxy-envoy-traefik-and-alb/

Load Balancer Latency and Concurrency Performance Data

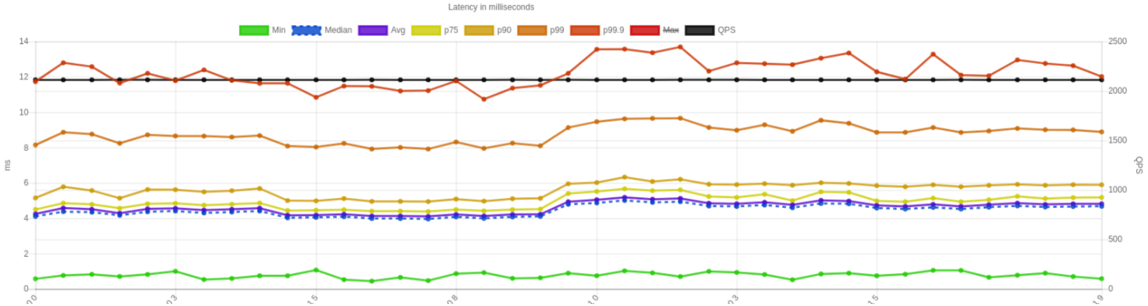


Data found here: https://www.loggly.com/blog/benchmarking-5-popular-load-balancers-nginx-haproxy-envoy-traefik-and-alb/

Load Balancer CPU and Latency and Concurrency Performance Data - Disagreement



The latency observed in the Linkerd2-meshed setup ranged from 11.0 ms to almost 14.0 ms.



The latency observed in Istio-meshed setup ranged from 36.0 ms to 45.0 ms.

Data found here: https://medium.com/@ihcsim/linkerd-2-0-and-istio-performance-benchmark-df290101c2bb

Service Mesh Pros-Cons

Pro-SMESH-Cons!

Service Meshes Architecture Pros and Cons

Library Architecture

Advantages:

- Resources are locally accounted for each and every service
- Self-service adoption for developers

Disadvantages:

- Strong coupling is a significant drawback
- Non-uniform; upgrades are challenging in large environments

Node Agent Architecture

Advantages:

- Less overhead (especially memory) for things that could be shared across a node
- Easier to scale distribution of configuration information than it is with sidecar proxies (if you're not using a control plane)
- This model is useful for deployments that are primarily physical or virtual server based. Good for large monolithic applications

Disadvantages:

- Coarse support for encryption of service-to-service communication, instead host-to-host encryption and authentication policies
- Blast radius of a proxy failure includes all applications on the node, which is essentially equivalent to losing the node itself
- Not a transparent entity, services must be aware of its existence

Sidecar Architecture

Advantages:

- Granular encryption of service-to-service communication
- Can be gradually added to an existing cluster without central coordination
- App-to-sidecar communication is easier to secure than app-tonode proxy
- Resources consumed for a service are attributed to that service
- Blast radius of a proxy failure is limited to the sidecar app

Disadvantages:

- Lack of central coordination. Difficult to scale operationally
- Sidecar footprint—per service overhead of running a service proxy sidecar

Service Mesh Comparison Final Thoughts

- Istio was presented to the Cloud Native Computing Foundation Technical Oversight Committee in November 2017. This represented an early step for the project to join CNCF. We suspect they will become a full project member in 2019. Linkerd is already a member of CNCF.
- There are both fully Open Source and Commercial implementations of Istio available
- This will be a key to widespread adoption which will be key to the long-term success of Istio
- The following question will be answered in 2019: “What are the real world benefits I will receive by adopting service mesh as a standard?”
- The answer will come from use cases stimulating adoption.

Bottom Line:

- A service mesh will be mandatory by 2020 for any organization running microservices in production.
- If you are currently using a service mesh, you already know the value it brings.
- If you’re considering using a service mesh, make sure you stay in tune with what is going on with this technology!
- If your company has not yet decided on whether you need a service mesh or not, read the recent industry reports on microservices!

Thank You

Bruce Basil Mathews

bmathews@mirantis.com