OPEN
NETWORKING
& EDGE
SUMMIT

Service Mesh: To-do & What-not-to-do

Don Jayakody

Sr. Technical Marketing Engineer, Arista Networks

Hosted By

THE LINUX FOUNDATION | LF NETWORKING | LF EDGE

Find out more:

- Repo for slides: https://github.com/jayakody/ones-2020
- Packet Walks in K8S:
  - https://static.sched.com/hosted_files/onsna19/2f/Packet_Walks_In_Kubernetes.pdf
  - https://github.com/jayakody/ons-2019

OPEN
NETWORKING
& EDGE
SUMMIT

WHY Service Mesh?

Hosted By

THE LINUX FOUNDATION | LF NETWORKING | LF EDGE

#ONESummit

How we got here?

- Introduction of microservices architectures introduced greater flexibility with more loosely coupled services
- The issue with having many services (compared with limited number of services in monoliths) is now you need make sure the communication between these services are seamless. After all microservices can’t function alone
- To make matters worse, organizations are building out more microservices in heterogeneous ways: Different languages/Different deployment models etc. This will eventually results in more complexity & hinder innovation

Use Cases

Reliability

Service connectivity, discovery & traffic handling

Intelligently route traffic across microservices

Security

Zero-Trust Security

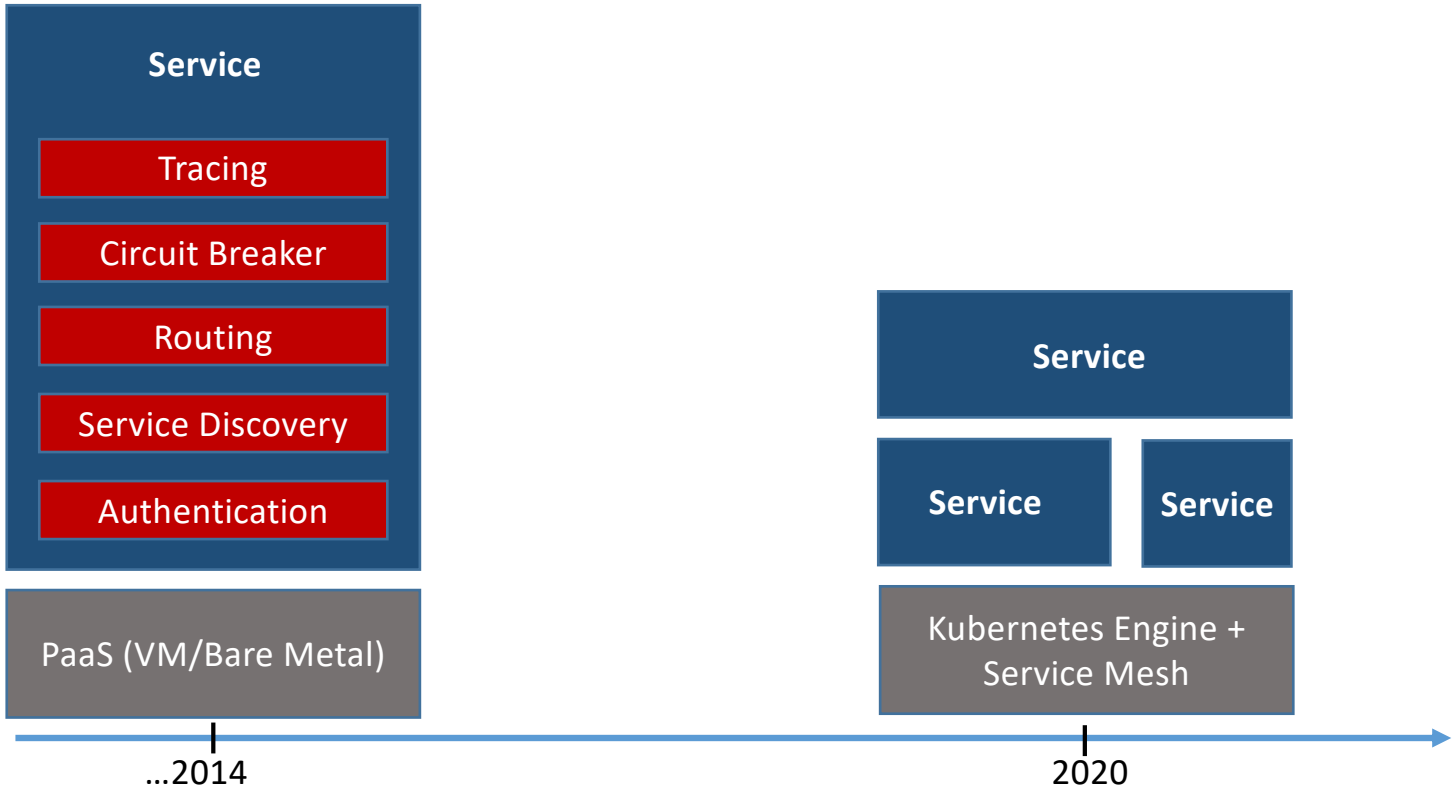
Encrypting traffic and enabling security policies across the microservices infrastructure

Observability

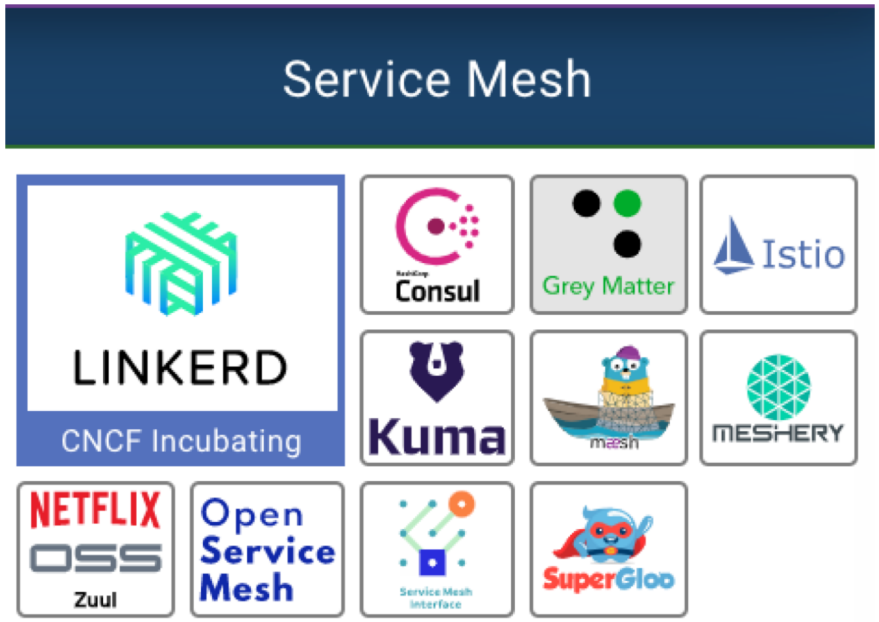
Increase Visibility & contextual awareness

Gain a detailed understanding of microservice behavior

Evolution


Service
Tracing
Circuit Breaker
Routing
Service Discovery
Authentication
PaaS (VM/Bare Metal)
…2014
Service
Service
Service
Kubernetes Engine +
Service Mesh
2020

Current State



landscape.cncf.io

Service mesh on CNCF

+

++

AWS App Mesh

Open
Service
Mesh

Anthos

VMware Tanzu

ASPEN MESH

backyards

tetrate

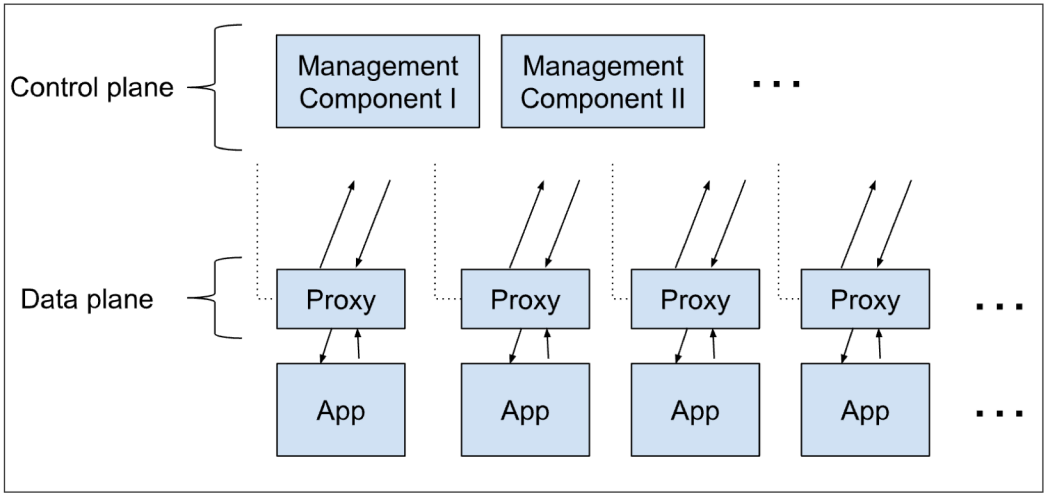
MuleSoft
Anypoint Service Mesh

More service meshs…..

WHAT is a Service Mesh?

Architecture

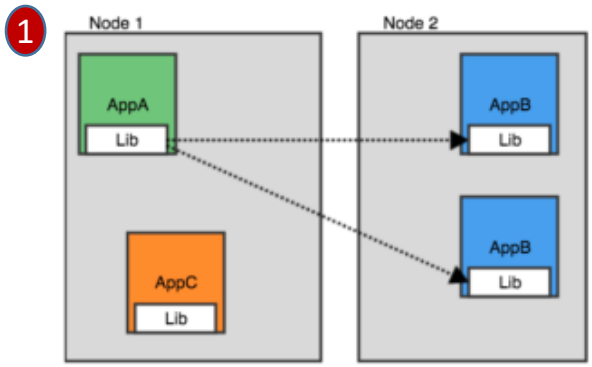
- Primary goal is to enhance the service-to-service communications by making it secure, fast, and reliable
- Abstracted architectural pattern
  - Primarily for microservices deployments, but rapidly expanding to other infrastructures such as VMs & Bare Metal environments

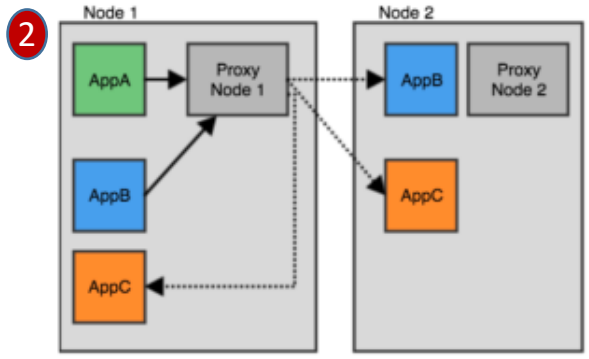


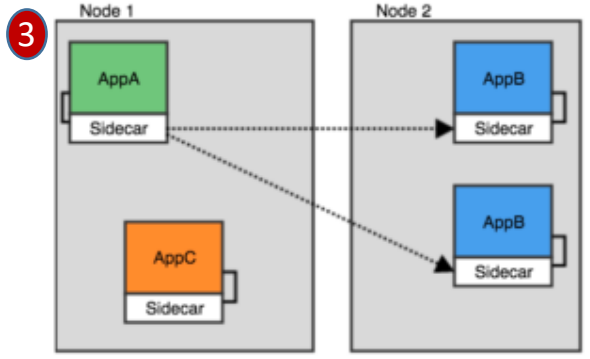
servicemesh.io/

Architecture

- Where should the “data-plane” (Proxy) live?
  1. In a **Library** that your microservices applications import and use
  2. In a **Node Agent** or daemon that services all of the containers on a particular node/machine
  3. In a **Sidecar** container that runs alongside your application container







aspenmesh.io/service-mesh-architectures/

Sidecar Proxy

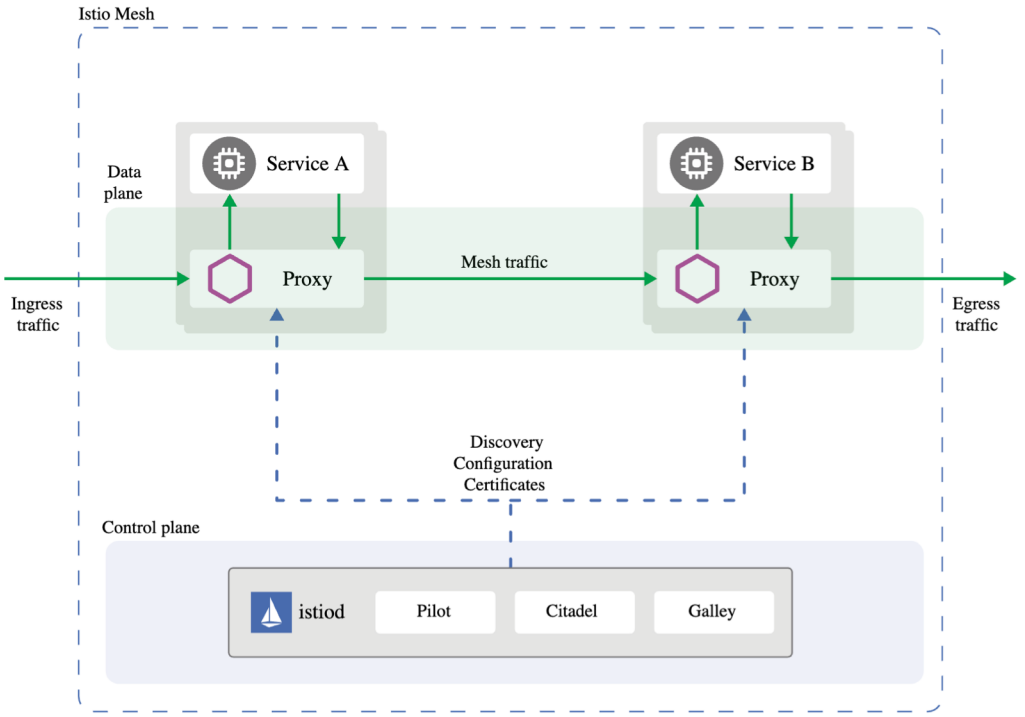
- Most common deployments:
  - Data Plane: Sidecar Proxies/ Microservices interact with each other through sidecar proxies
  - Control Plane: The developer can configure and add policies at the control plane level, to impose policies for both security and traffic handling

Data Plane: Envoy

- “Universal Data Plane” designed for microservices architectures
- High performance C++ distributed proxy
- Built-in class observability / load balancing
- Why Envoy got popular?
  - Envoy introduces K8S style declarative style APIs. Workflow is drastically different. Mainly around how it’s configured and usages around service meshes
  - xDS protocol: Configuration language of Envoy. Using xDS protocol we can now decouple control pane from the data plane

Control Plane

- **Example: Istio Control Plane:**
  - **Pilot**: Pushes the service discovery information to the envoys. Dedup the info and configures the Envoys. Hooks up to Kube for service discovery, Can support multiple service discovery systems. (Service discovery aggregator)
    - Pilot is where whole istio model translates down to the envoy model
    - Service Entry: What services/names exist?
    - Virtual Service: For a service how to I route to backends?
    - Destination Rules: How to I talk to those backends?
    - Gateway: Expose this Virtual Service to the Outside world
  - **Mixer:** has been replaced by telemetry v2



Istio Architecture

istio.io/latest/docs/concepts/what-is-istio/

Dos & Don’ts

Do I need a Service Mesh?

1 → How many people are in your engineering org? *

- A <5
- B 5-20
- C 21-100
- D 101+

2 → How many (micro)services are in your application? *

- A 1
- B 2-10
- C 11-100
- D 101+

3 → Are your services all written in the same language, with the same frameworks, etc? *

- A Yes
- B Most do, but a few are different
- C They're all different

4 → Have you successfully adopted open source **infrastructure software** in the past? *

- A We've never tried
- B We've tried, but failed
- C We've succeeded, but we needed help
- D We've adopted open source infrastructure without problems

6 → How confident are you in the end-to-end **security** of your application? *

1	2	3	4	5

Not confident — Very confident

observability of

4	5

Very confident

reliability of your

4	5

Very confident

buoyant.io

Considerations: Applications

- If your application split into lots of microservices it may makes sense to implement a Service Mesh
- Also if you have a good separation between applications/services and the organization structure, that's another plus point for implementing a Service Mesh
- One important factor to consider is microservices should not be dependent on each other
  - They should be as loosely coupled as they can be
  - There should not be any deployment dependencies! (For e.g. let's say your service-x is dependent on service-y to be deployed first; this might lead to complexities down the line when you put a Service Mesh on top of it)

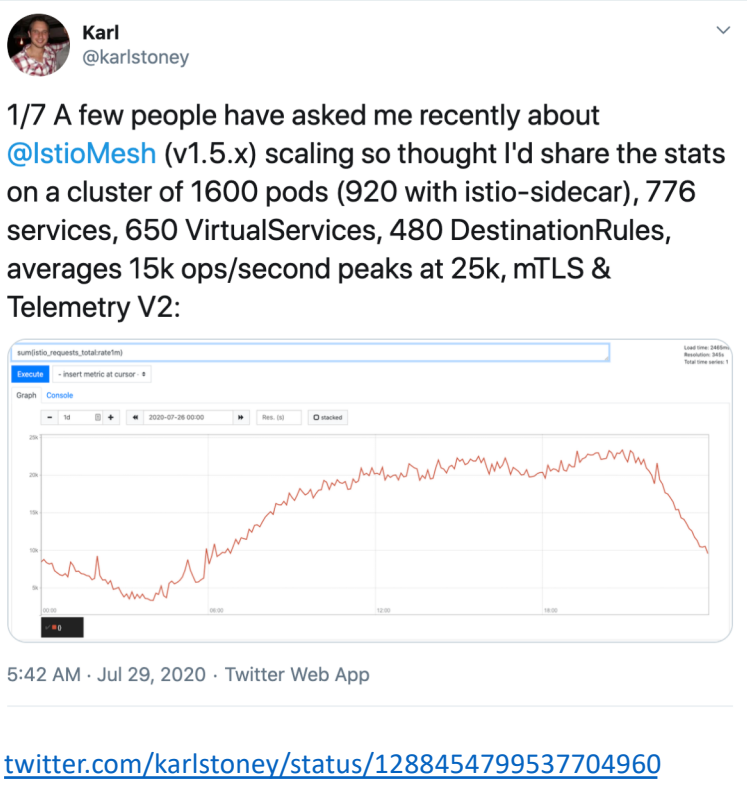
Considerations: Resources

- Control Plane Performance:
  - The rate of deployment changes
  - The rate of configuration changes
  - The number of proxies connecting to control plane
- Data plane performance:
  - Number of client connections
  - Target request rate/Request size and Response size
  - Number of proxy worker threads
  - Protocol
  - CPU cores

Considerations: Resources

- **1000** services and **2000** sidecars with **70,000** mesh-wide requests per second:
  - Envoy proxy uses **0.5 vCPU** and **50 MB memory** per 1000 requests per second going through the proxy
  - Istiod uses **1 vCPU** and **1.5 GB** of memory
  - Envoy proxy adds **2.76ms** to the **90th percentile** latency

https://istio.io/latest/docs/ops/deployment/performance-and-scalability/



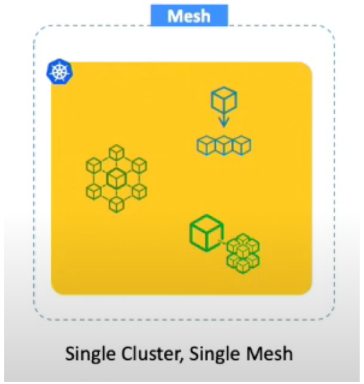
twitter.com/karlstoney/status/1288454799537704960

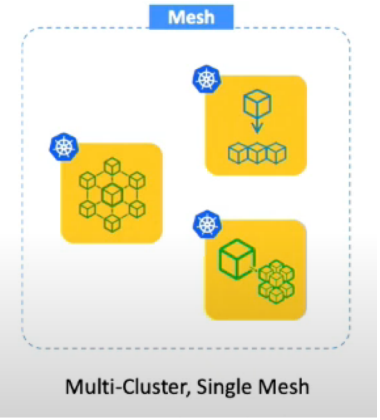
Considerations: Upgrades

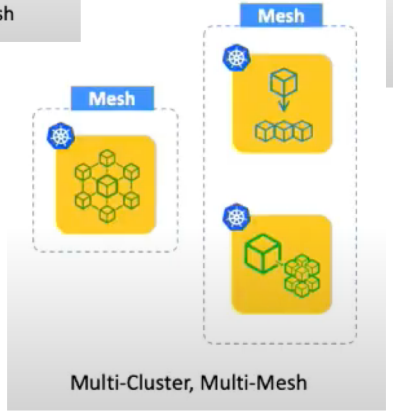
- Read the upgrade-notes/release-notes
- Careful pre-planning might be needed. Better to avoid in place upgrades:
  - Upgrade can be done to reduce disruptions to the active services
  - By first running a canary deployment of the new control plane
  - This will allow monitoring the effect of the upgrade with a small percentage of the workloads, before migrating all of the traffic to the new version

Considerations: DR

- Depending on the applications & services you may need to think through possible Disaster Recovery (DR) options
- Implementing a DR strategy that takes into all the necessary options can be bit tricky & your mileage may significantly vary depending on the service mesh type: Plan/POC/Implement & Test







ServiceMeshCon: youtube.com/watch?v=BsV5iHYIws8

Considerations: SMI

- Service Mesh Interface
  - A standardized interface for service meshes on Kubernetes
  - Collection of APIs registers with K8S
  - Providers that implement those APIs (Istio/Linkerd/Consul)
  - Standardize tooling & your own production APIs, and it'll allow abstraction from the implementation/provider level
- If you are building tooling on top of service mesh, SMI would be a great option. SMI would allow you to be transparent to the provider-Service-Mesh layer

Considerations: Ongoing Developments

- Proxy placement?
  - Sidecar vs Per-Node Proxy: (Is the debate really over?)
    - Sidecar approach gets operationally cumbersome during upgrades. Upgrading service mesh means rerouting the traffic
    - Per-Node proxy might be easier operationally but then it's a SPOF. Might need to deploy multiple node proxies

    youtube.com/watch?v=j7JKkbAiWuI
- Sidecar vs Racecar: Accelerate Envoy using eBPF: youtube.com/watch?v=ER9eIXL2_14
- Traffic director:
  - Google Cloud's fully managed traffic control plane for service mesh
  - Uses open source xDS APIs to communicate with the service proxies in the data plane https://cloud.google.com/traffic-director

OPEN NETWORKING & EDGE SUMMIT

Hosted By

THE LINUX FOUNDATION | LF NETWORKING | LF EDGE