ActualTech Media

PLATFORM9

Best Practices for Selecting and Implementing Your Service Mesh

Joep Piscaer

CONTENTS

IN THIS PAPER

A service mesh can standardize and automate inter-service communication. It helps you control traffic, security, permissions, and observability in complex microservices landscapes.

In this tech brief, we'll talk about the key to being successful with a service mesh:

- Start your service mesh journey early to allow your service mesh knowledge to grow organically as your microservices landscape evolves, grows, and matures
- Avoid common design and implementation pitfalls due to lack of knowledge
- Leverage your service mesh as the mission control of your multi-cloud microservices landscape

As applications are being broken down from monoliths into microservices, the number of services making up an application increases exponentially. And as anyone in IT knows, managing a very large number of entities is no trivial task.

Service meshes solve challenges caused by container and service sprawl in a microservices architecture by standardizing and automating communication between services. A service mesh standardizes and automates security (authentication, authorization, and end-to-end encryption), service discovery and traffic routing, load balancing, service failure recovery, and observability. Just as virtualization abstracted the hardware layer of computer systems and containers abstracted the operating system, a service mesh abstracts away communication within the network.

Why a Service Mesh?

As monoliths are pulled apart into their smallest constituents, the resulting microservices are usually distributed across multiple systems and communicate over HTTPS, so they become heavily dependent on network communications.

> Although a service mesh is very useful to development teams, implementing the service mesh itself still takes some work.

A service mesh manages the network communications by setting up standards and automating their implementation. It frees developers from defining and implementing the communications for every service, over and over again (see **Figure 1**).

This is much more scalable, more automated, and less error-prone. The service mesh also improves security and reliability by standardizing the interface between services. The service mesh acts like an automatic walled garden for each service on the network.

This is done by making sure other services know the service exists (called "service discovery"), managing authorization and authentication between services, taking care

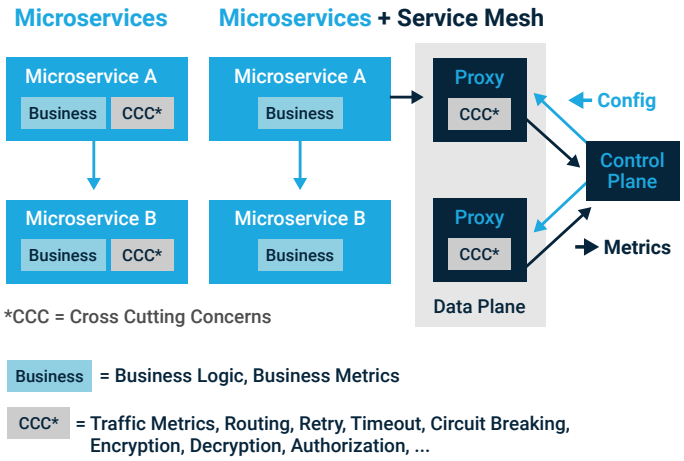

Figure 1: Service mesh architecture

of load balancing, and adding security policies for communication to and from the service by the outside world.

Because a service mesh has control over the network communication between all services in the mesh, it unlocks some advanced deployment and release strategies, such as canary releases, blue/green releases, and rolling upgrades.

This improves the reliability of the services in production. In some cases, the service mesh can react to changes in the traffic patterns, adding circuit breakers and rate limiters between services to prevent cascading failures. In order for teams to gauge the performance and quality of each release, a service mesh often has observability tooling (for collecting telemetry and metrics, as well as building in distributed tracing capabilities).

In short, a service mesh acts like an operational mission control to determine the behavior of microservices at scale, making sure the landscape of microservices is communicating securely, and monitoring performance and service quality. It removes much of the manual work from the developer's plate, so they need to focus only on the business logic, not the network, security, and communication plumbing.

The result is not only higher quality in business logic code, but also a reduction in variations and human errors in the plumbing, by standardizing and automating much of that work.

Reducing Service Mesh Complexity

Although a service mesh is very useful to development teams, implementing the service mesh itself still takes some work. Because there are many moving parts, a service mesh leaves a lot of flexibility and room to customize it to your specific needs. As always, flexibility comes at the cost of complexity.

Balancing the features, functionality, and value of a service mesh with its inherent complexity it is highly challenging, and requires expertise, but is well worth the effort. With an experienced team in place, organizations can overcome the complexity associated with running a service mesh at scale.

The best way to start developing the necessary skills and experience is no different from any other technology: start early, and start simple. You don't need to acclerate from 0 to 60 miles per hour instantly. Instead, start small, and incrementally add more features and functionality as you build trust in the service mesh.

It's recommended to start developing service mesh skills in tandem with your microservices architecture, because adding service mesh features to a relatively simple microservices architecture is much easier than when it's already complex and large. Let the service mesh grow organically alongside your ever-evolving microservices architecture. This keeps services secure and compliant, and helps maintain visibility.

The Service Mesh Team

As your organization grows and your use of the service mesh increases, it makes sense to create a dedicated team focused on the continual improvement of the service mesh, as well as helping application development teams make the most of the features and functionality it offers.

The dedicated team owns the service mesh platform and is responsible for the adoption of the service mesh across application teams and the entire microservices landscape. With this team structure, application development teams can focus on building business logic and microservices.

As you'll see in the following sections, having a dedicated team keep tabs on service mesh use cases (like multi-cloud and heterogenous workloads) may save you from an expensive, intrusive, and complex migration project because reality got in the way.

The Service Mesh Catch-22

Choosing the right service mesh technology, and nailing the implementation details, are crucial factors in your service mesh success. But how do you make the right decisions and do the right things when you don't have the right knowledge and experience yet? This is the catch-22 for the initial deployment and configuration of every new technology, including a service mesh.

> Having a dedicated team keep tabs on service mesh use cases (like multi-cloud and heterogenous workloads) may save you from an expensive, intrusive, and complex migration project because reality got in the way.

This is a common pitfall for organizations, as engineers start designing and implementing a new technology enthusiastically. The inefficiencies and sub-optimal decisions due to lack of experience don't immediately come to light, but often surface only weeks, months, or even years later, when it's too late to drastically change anything.

How do you prevent these mistakes? And how do you kickstart the learning process without the associated risk and possibly massive impact down the road? Turning to a simpler, less feature-rich alternative carries its own risk, as you introduce a future point in time where your own maturity outpaces the limited feature set, forcing you to do a forklift upgrade of the mesh, introducing a migration not only of the mesh itself, but a migration of all the microservices in the mesh, too.

Instead, choosing the right mesh technology with the end-goal in sight makes more sense. Currently, there are three leading, mature options available in the Kubernetes ecosystem: Consul Connect, Istio, and Linkerd.

While there are differences, all three are battle-tested, production-ready, and enterprise-grade solutions. It's a matter of finding the right one given your unique context, requirements, and goals.

> The paradox here is knowing which level of flexibility you need a few years down the line when you have zero experience and expertise to make that decision now.

Istio has the most functionality and flexibility, but is also the most complex, making the first steps harder. Linkerd is Kubernetes-only, making it easier to implement and use. If you need to support virtual machines (VMs) alongside Kubernetes, Consul is a good choice.

The paradox here is knowing which level of flexibility you need a few years down the line when you have zero experience and expertise to make that decision now. Let's dive into an overview of these three options to start building a picture of which one is right for your organization. This will help you make the right decision and prevent obvious pitfalls as you build trust and increase your service mesh proficiency.

HASHICORP CONSUL CONNECT

Connect is Consul's service mesh feature. It provides service-to-service networking and security (authorization, encryption). As seen in **Figure 2**, applications can use a *sidecar proxy* deployment model.

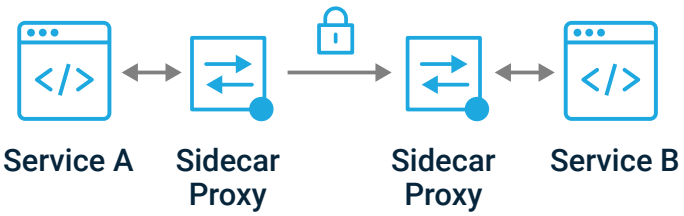

Figure 2: Consul Connect in a sidecar proxy model

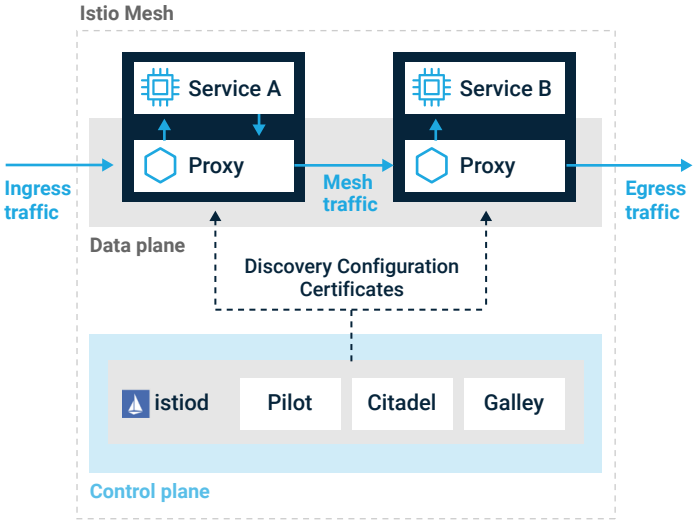

Figure 3: The Istio architecture

These sidecars handle the inbound and outbound TLS connections, with the application completely agnostic of Consul. Consul also has a native integration deployment model. In Kubernetes environments, Consul uses a per-host DaemonSet agent and Envoy sidecar proxies per application that handles application traffic. Consul applies a zero-trust security model, is platform agnostic, and supports multi-cluster deployments.

As with other HashiCorp tools, Consul Connect is easy to get started with. Its deployment and initial configuration are a little less daunting than other options, making it a good solution for those very new to the service mesh space.

ISTIO

Istio is the darling of the cloud-native space. Like many projects before it, it was open sourced by an end-user company (Lyft, in Istio's case), as they built a solution to handle complexity and scale.

Istio has seen massive adoption, especially as the basis of various public cloud offerings.

Istio's complexity is its downside for newcomers to the field, but also what makes it so powerful; one example is the addition of telemetry and analytics. As **Figure 3** shows, its architecture is much like Consul Connect.

A notable fact about Istio is that it is not part of the Cloud Native Computing Foundation (CNCF) landscape map,

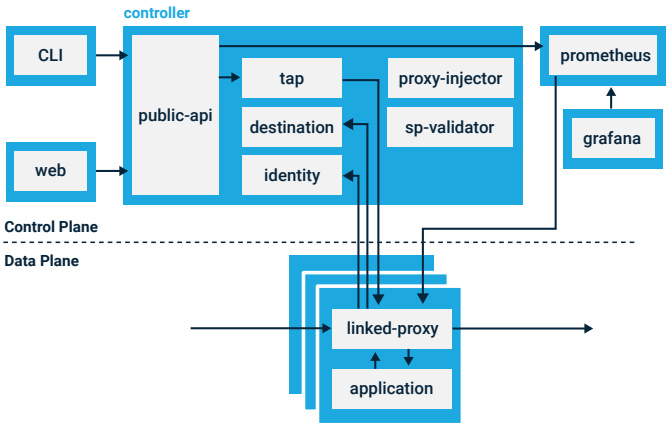

Figure 4: The Linkerd architecture

even though it's the most popular service mesh option for the CNCF's Kubernetes ecosystem.

LINKERD

Linkerd is the CNCF answer to a service mesh. Its v2 architecture mimics Istio, but favors simplicity over features and flexibility.

Where Consul and Istio work with Kubernetes and VMs, Linkerd exclusively works on Kubernetes. This means its architecture (**Figure 4**) has fewer moving parts and fits into the Kubernetes architecture more seamlessly, with deeper integration into many other CNCF projects like Prometheus.

To get the full details, the Platform9 blog has a post called "Kubernetes Service Mesh: A Comparison of Istio, Linkerd and Consul." It compares these three feature-by-feature.

> SMI offers a set of common, portable APIs that provide developers with interoperability across different service mesh technologies including Istio, Linkerd, and Consul Connect.

A SERVICE MESH CHOICE IS NOT FOREVER

Even though you should now have the knowledge to make an initial choice, remember that requirements and circumstances change, so your service mesh will need to evolve, catering to those changes.

In some cases, a different technology is needed. If you're using the sidecar deployment model, applications and microservices running as part of the mesh are not aware of the mesh, nor do they have any special customization or integration with any specific mesh. The sidecar model makes it easier to migrate between technologies.

For more deeply integrated service mesh approaches, the Service Mesh Interface, or SMI for short, may prove useful. SMI offers a set of common, portable APIs that provide developers with interoperability across different service mesh technologies including Istio, Linkerd, and Consul Connect.

Conquering Multi-Cloud

Reality is messy, and IT is no different. Migration from old technologies to new ones is always happening, whether from VMs to containers, from on-premises to public cloud, or from one public cloud to another. What use is a service mesh that helps you control traffic, security, permissions, and observability when it works for only a sub-set of workloads in just one environment?

Multi-cloud in a service mesh context means more than just *multiple public clouds*. It also needs to support on-premises deployments and support VMs. Last, the service mesh should span all these environments and have multi-cluster support.

This multi-cloud reality is often not explicitly designed by the organization, but "just happens." For instance, a group of developers starts using yet another public cloud, because it has the specific functionality they need to do their work. Whatever the cause, making sure your service mesh can handle this guarantees you can take a proactive approach to supporting the endless variety of multi-cloud scenarios in production. It gives you the piece of mind that you're in control of security in the untrusted world of public cloud, and have visibility into the entire microservices landscape.

In other words, if chosen correctly, a service mesh can serve as an abstraction layer on top of the public cloud, abstracting away the cloud and giving back control over traffic, security, permissions, and observability in a multi-cloud reality.

> Multi-cloud in a service mesh context means more than just multiple public clouds. It also needs to support on-premises deployments and support VMs.

Looking at the three options shows that while Linkerd's simplicity sounds great on paper, reality may get in the way, requiring you to use a service mesh technology that works across containers and VMs. And again, SMI may help you migrate service mesh technologies if you need to—accepting and acknowledging that reality is messy may save you from a painful service mesh migration project.

Mission Control

That's why it makes sense to select a service mesh that doesn't lock you into a single public cloud. Instead, choose a cloud-agnostic service such as Platform9's Managed Kubernetes service, so that your service mesh can become the *mission control* of your multi-cloud microservices landscape—the place for troubleshooting issues, enforcing traffic policies, controlling emergent behavior, and releasing new code safely to limit the blast radius.