AVI Networks®

An Essential Guide to Container Networking and Service Mesh

An Architectural Pattern for Modern Load Balancing and Application Services

EBOOK

c05.19

Executive Summary

The popularity of modern applications built on container-based microservices architecture has led to new requirements for application networking - unmet by existing solutions. Applications are becoming more distributed and application services, including load balancing, analytics, security are moving closer to (and in some cases even becoming part of) the applications (or microservices).

The emergence of the service mesh concept directly addresses the need for disaggregated service-to-service communications in container environments. It makes it possible to deliver application services such as traffic management, security, and observability matching the granularity required by container-based applications. The open source Istio project for Kubernetes is an example of a service mesh that embraces the notion of separate control and data planes.

We assert in this paper that while service mesh as a concept started in Kubernetes and microservices, its benefits should not be limited to just container-based applications. The service mesh can be applied as an architectural pattern so that application services can and should be extended to traditional and cloud-native applications in today's multi-cloud and multi-infrastructure world.

This whitepaper explores service mesh as an architectural pattern, and how both modern applications in container clusters as well as traditional applications in on-prem data centers and clouds can benefit from the granular application services made possible by a service mesh. It is an essential guide to understanding what a service mesh is and why it matters. It includes best practices used by enterprises who have deployed large container clusters in production.

MARKET DRIVERS

Service mesh has attracted increasing attention in the market as it's better aligned to support the agility, availability, elasticity, and security of cloud-native applications. In IDC's Market Perspective report "Vendors Stake Out Positions in Emerging Istio Service Mesh Landscape", Brad Casemore notes that "The service mesh represents the evolution of application delivery infrastructure into software-defined application delivery services that are modular and composable." Gartner also points out similar viewpoints in its "Innovation Insights for Service Mesh" paper that covers leading vendors.

> **The service mesh represents the evolution of application delivery infrastructure into software-defined application delivery services that are modular and composable.**
>
> **BRAD CASEMORE, IDC**

Google, IBM and Lyft have developed and launched the Istio service mesh, joining forces are vendors like Pivotal, Cisco, Red Hat, VMware and Avi Networks. This paper explains the motivation, overview, and challenges of a service mesh, as well as how to apply it as an architectural pattern beyond cloud-native applications.

APPLICATION EVOLUTION

Applications have undergone several generational shifts. Architecturally, monolithic applications have been broken down and disaggregated into loosely coupled, distributed microservices. Deployment models have evolved from bare metal servers to virtual machines, and now to container clusters. The operating environments span on-prem data centers and public clouds. All of these changes are creating increasingly complex and dynamic application ecosystems. However, the common goal of the changes is to make applications more available, responsive, and secure.

What does this mean for application services, or application delivery controller (ADC) solutions? See Figure 1. For "Monolithic, on-prem" applications, since the days of mainframe computing, there is a 1-to-1 mapping between the application and location i.e. you run this database app on this pool of servers in this data center. To ensure the application is available under high traffic load, you simply need to *physically* deploy a hardware load balancer between the user and the servers. The data and control points for load balancing are tightly coupled in one appliance but mirror the rigid infrastructure and application architecture.

For "Virtualized, software-defined principles" applications which have run in virtualized environments from the late 90s, *Virtual* load balancers deployed in each location are literally virtual copies of the hardware. This approach inherits the architectural challenges of appliances and results in multiple control points, making management and scale a nightmare. Software-defined application services separate the control plane from the data plane, allowing lightweight proxies to be distributed alongside VMs across environments, and a central repository (named controller) of policies and control point to manage the entire lifecycle of load balancing services (including VIP placement, autoscaling, recovery, and troubleshooting) with automation and self-service. This innovation enables a transition from load balanc*er* to load balanc*ing* as a service.

For "Gen 3" applications that emerged just in the past few years, the main driving force comes from developers who build, test, deploy and run container applications in CI/CD pipelines. Supporting this agile development model entails a whole suite of DevOps technologies. Kubernetes was created to orchestrate Docker containers, thousands of which spin up and down in seconds. Transient containers often leave no trace and make monitoring difficult in enterprise production environments. New requirements for monitoring, security, and analytics - often called tracing, microsegmentation, and observability have emerged to match the granularity and transient nature of cloud-native workloads. For these use cases, application services need to be even more distributed and are often built into the microservices applications themselves. For more details, see the Istio service mesh section (later in this document) where load balancers are deployed as sidecar proxies inside containers.

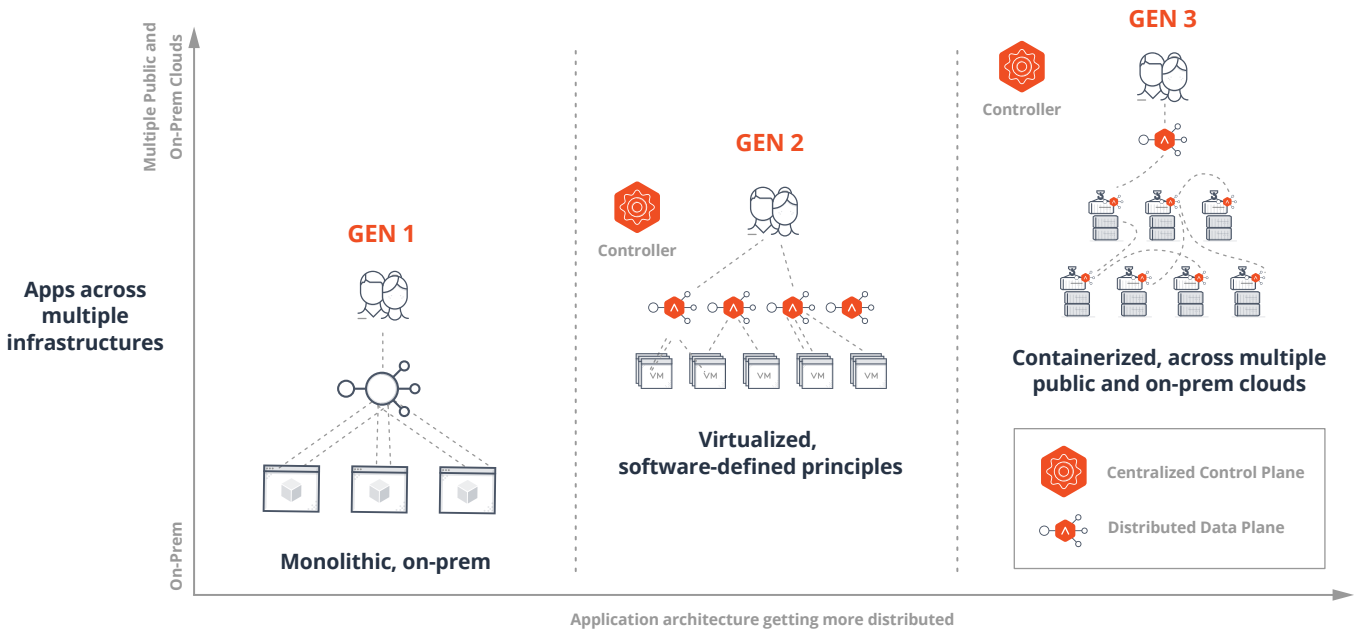

Figure 1: Evolution of Application Architectures and Implications on Application Services

All in all, the recurring theme is that application services need to align with application architectures tightly.

> **AN IMPORTANT BUT OFTEN GLANCED OVER FACT:**
> Although the shift to microservices architecture has started, a vast majority of workloads still remain in monolithic applications on virtual machines and bare metal servers. Since traditionally architected applications will continue to be used alongside microservices applications for the foreseeable future, application services need to bridge across and provide consistent capabilities across multiple infrastructures, be it containers, VMs or bare-metal servers, and across heterogeneous environments, such as or data centers and clouds.

SERVICE MESH OVERVIEW

CHALLENGE: The networking implications of microservices architecture

Service mesh emerged to address the networking challenges of the microservices architecture, which offers flexibility but comes with complexity in terms of network services. The promise of modularity, agility, and ease of development is complicated by the ephemeral nature of container workloads, the explosion in the number of application endpoints, and the frequency of changes. To effectively deliver applications with such an architecture, application and network services need to be disaggregated as the applications themselves.

However, the reality is that it results in a fragmented infrastructure ecosystem (see Figure 2). Solutions are needed at every infrastructure level – from distributed load balancing to firewalling, and visibility, and more. There are often interdependencies between technologies integrated into different platforms, placing additional burden on teams to keep up with the pace of change, and prepare container clusters for production.

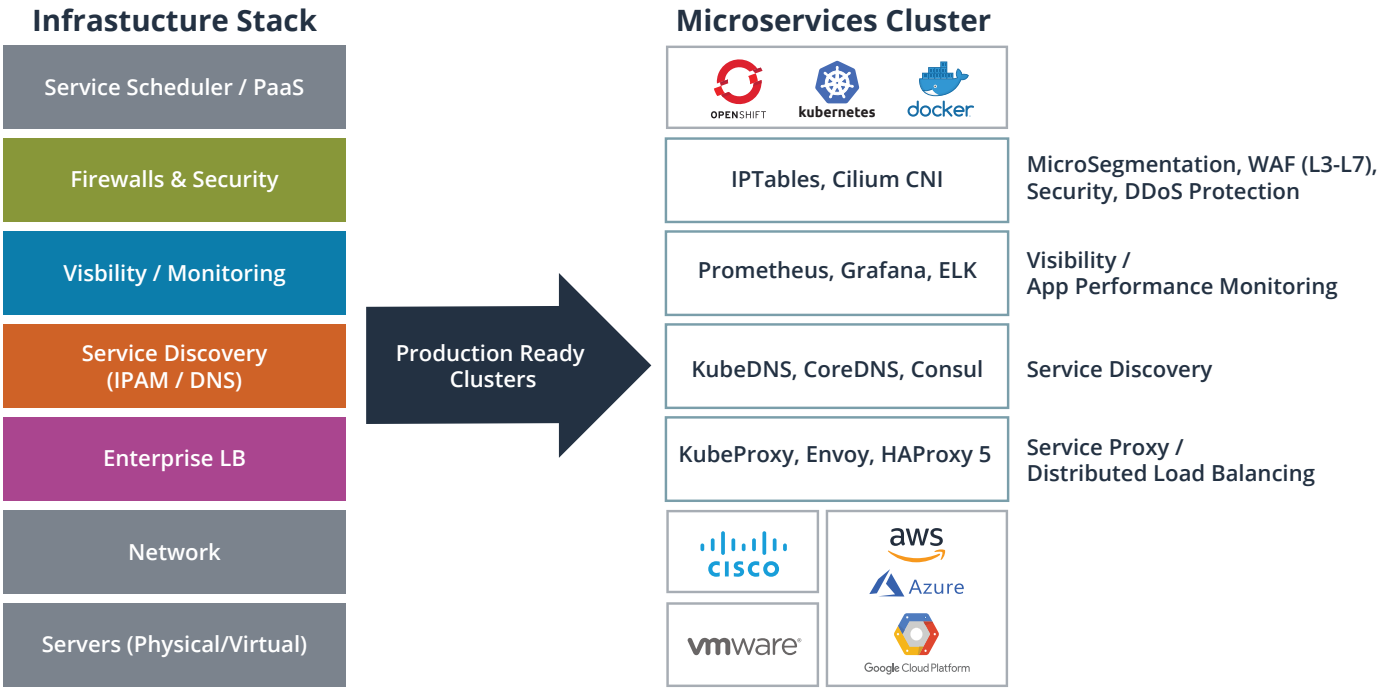

Figure 2: Before Service Mesh: Need to Use Multiple Point Solutions

Even with proven DevOps practices, the lack of container security and analytics that provide deeper insights into service-to-service communication can make or break large-scale cluster deployments. In these environments, thousands of containers are getting deleted, spawned, and updated often.

SOLUTION: Service Mesh

A service mesh is a *single* configurable infrastructure layer that provides a mesh of application services needed for traffic management between microservices, observability and dynamic discovery of services, and security of service-to-service communication. It provides a centrally managed, client-side load balancing, monitoring, and security solution. These capabilities offer different benefits to different teams depending on their roles:

- *Developers* – consume policy-based traffic management services such as circuit breaking, error injection, rate limiting, mirroring, and CI/CD integration with APIs and self-service
- *Operators* – require observability features like route tracing, logging, application maps, and monitoring to operationalize the clusters
- *Security teams* – ensure regulatory compliance if any, and that all communications are authenticated, authorized, encrypted, and protected for ingress and egress

EXAMPLE: Istio Service Mesh

The Istio service mesh is logically split into a data plane and control plane (see Figure 3).

The data plane is composed of a set of intelligent proxies (Envoy) deployed as sidecars (alongside each application container). These proxies mediate and control all network communication between microservices along with Mixer, a general-purpose policy and telemetry hub.

The control plane manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.

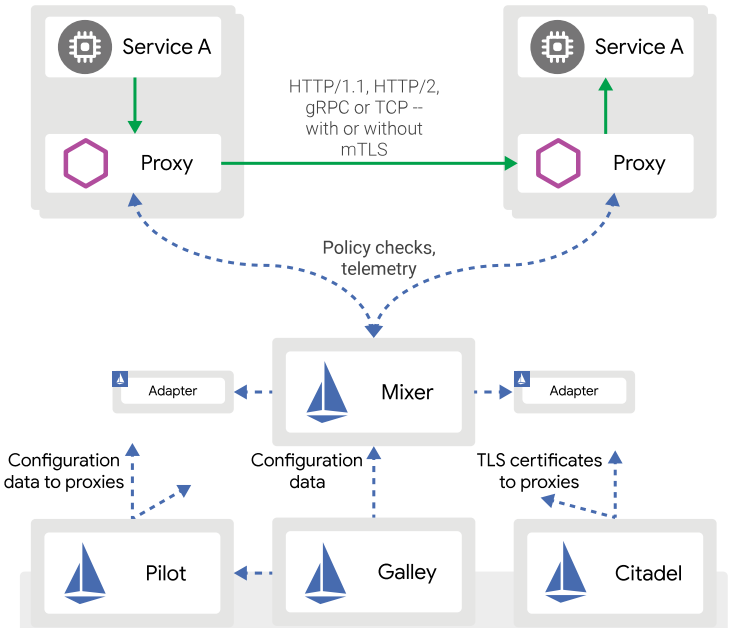

Figure 3: Istio Service Mesh High-Level Architecture

Source: https://istio.io/docs/concepts/what-is-istio/

This fully leverages software-defined principles (a breakthrough to solve the application services challenges of "Virtualized, software-defined principles" applications described in the previous section), with a focus on Kubernetes clusters and the associated ecosystems. For detailed information, including examples on how to inject sidecar, istio.io has extensive documentation into each component how they help to connect, secure, control, and observe services. This rest of the paper focuses on the implications of service mesh for both container and non-container based applications, and how the architectural pattern can be applied universally.

OPPORTUNITY: What is Still Needed

Service mesh allows for simplification and consolidation of the various capabilities needed from the infrastructure stack to support microservices architecture. Istio service mesh provides an open source community to further integrate application services into containers themselves and Kubernetes environments. However, for enterprises to fully operationalize service mesh, a few key considerations (see Figure 4) are required.

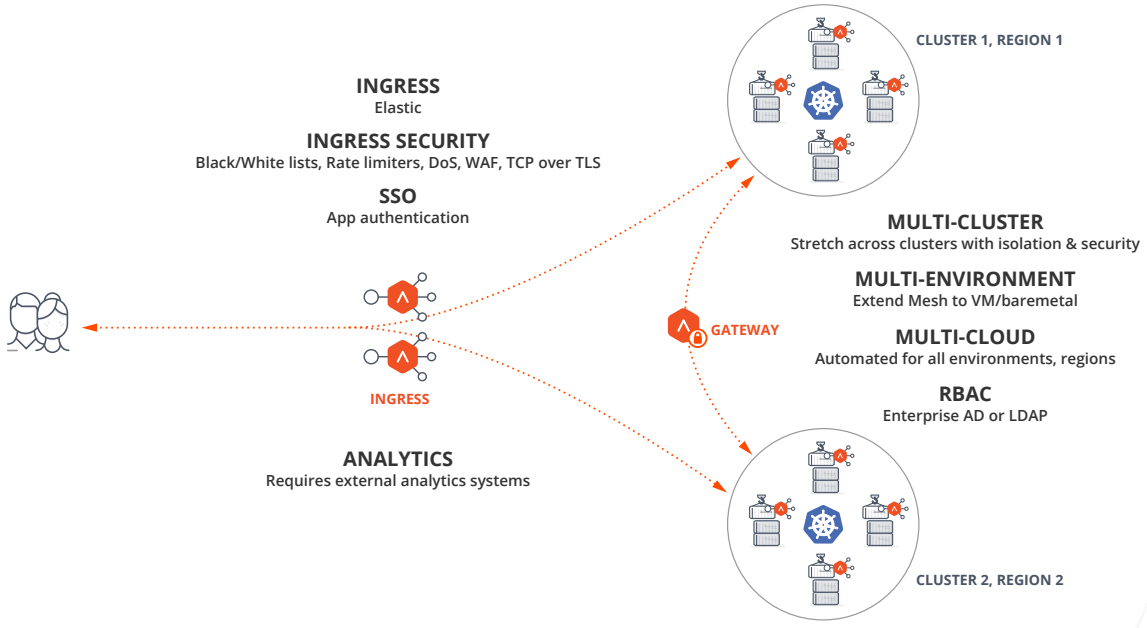

Figure 4: Service Mesh Extended Beyond Containers and Kubernetes

A highly scalable and secure ingress

As applications scale exponentially, they need infrastructure to autoscale on-demand based on traffic patterns. To get this capability, you must have detailed application performance monitoring built into the solution, instead of requiring external solutions. The first step to ensure elasticity is at the ingress gateway which is the entry point into the service mesh from the outside world. Then application scaling within the cluster need to be handled.

The second step is to ensure full isolation and enterprise-grade security, including black/white (B/W) lists, rate limiters, denial of service (DoS) protection, web application firewall (WAF), TCP over TLS, zero trust security, and more. Enterprises need single sign-on (SSO) for authentication and authorization, and role-based access control (RBAC) that integrates with active directory (AD) or LDAP.

Universal extensibility

It's understood that service mesh started in microservices applications. However, enterprises have mission critical applications running in traditional architectures both in on-prem data centers and in clouds. Some applications are still running as monoliths in VM's or bare-metal. Most of the stateful applications have not and will likely not been re-architected. In the telco space, the move to microservices-based network function virtualization (NFV) architecture is still a work in progress. We believe that this mixed reality of multi-infrastructure is here to stay.

On average, enterprises operate with 3 to 5 clouds (private and public) and 81% of them have a multi-cloud strategy[1]. A few reasons for organizations to have a multi-cluster/multi-cloud strategy are as follows:

1. High availability across clusters
2. Reduced dependency on one cloud Infrastructure
3. Multi-tenancy - one tenant per cluster
4. Shared/common applications - where the shared applications are running on one cluster, shared by applications running on other clusters
5. Stateful applications - some organizations prefer to deploy them on separate clusters for better manageability, and resource mapping
6. Legacy applications - still running as monolithic applications on VMs and bare metal servers

To address these workloads, you need a solution that truly extends across:

- Multi-cluster: stretched across clusters with isolation & security
- Multi-infra: extended beyond Kubernetes containers to VM/bare metal
- Multi-cloud: automated for all environments across multiple regions

[1] RightScale 2018 State of the Cloud Report

SERVICE MESH DEEP DIVE

The service mesh architectural pattern is designed to provide application services with three core capabilities: traffic management, security and observability (see Figure 5). These core pillars of the architectural pattern address application services needs throughout the lifecycle of an application. This section will explore in depth what is required for each and how to use the framework to evaluate various solutions.

Figure 5: Core Pillars of a Production Ready Service Mesh

Traffic Management

The key functionality of a service mesh is traffic management, which includes routing traffic from external sources into the cluster through an ingress gateway or out of the cluster through an egress gateway, and within the cluster(s) to communicate between microservices. These are called north-south and east-west traffic management respectively. Before we dive into the use cases in the later section, let's look at some fundamental capabilities.

Before traffic can be routed, you first need to discover what services are available. As you can imagine, the process has to be dynamic and automated due to the transient nature of containers and the needs from DevOps practice. Between the data plane and control plane, there has to be a *service discovery* mechanism, either through DNS lookups or an automatic registration process with a service registry. Some solutions send data to a centralized controller and visualize the connectivity into a dynamic application map.

Another capability is *traffic splitting* according to service versions, or certain load balancing algorithms. This is useful in cases where you wish to route traffic based on requests (see Figure 6). For example, you are testing out version 2 of the software but you only want to roll out to 10% of the pods of the new version while keeping the rest on stable production version 1. Common scenarios of *request routing* include Blue-Green deployments, A/B testing, or canary rollouts.

To handle failures, *timeouts and retries* need to be built into the rule configurations, including circuit breaking and error injection. You can define the intervals of timeout and maximum number of retries before an error is incurred. The built-in *circuit breaker* protects overall cluster health. Some solutions take advantage of health scores and eject "bad" instances from the load balancing pool if certain thresholds are met.

In more advanced cases, you can perform *fault injection* by introducing a 1-minute delay to 1% of your requests, resulting in error code 500. It helps put the cluster resiliency to test without affecting all users. This is a critical capability to look for when you are considering running your production workloads in container clusters.

These fundamental capabilities of traffic management are key to a fully functional service mesh but to get to an enterprise-grade solution, advanced features like ingress integration with IPAM / DNS, rate limiting, and full-featured L4-L7 load balancing with SSL offload are also important.

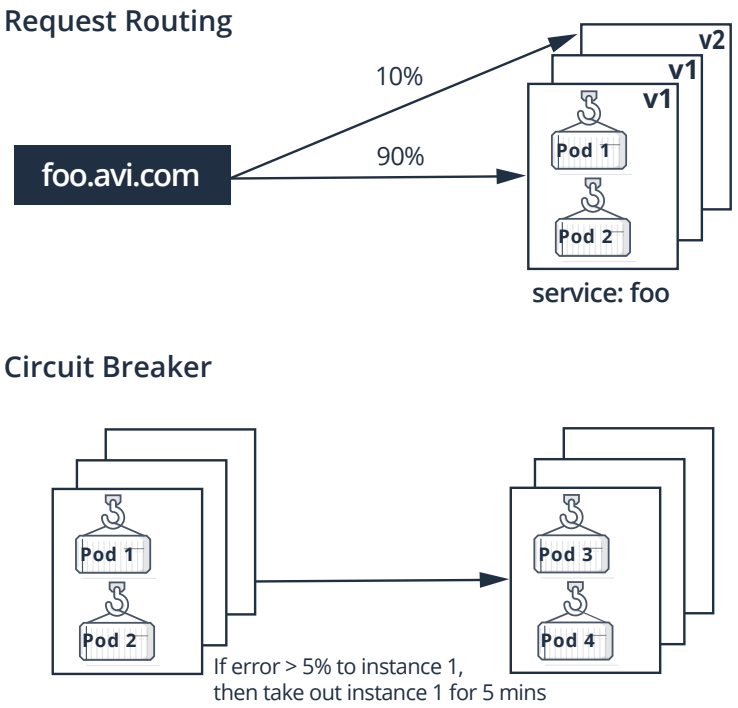

Figure 6: Service Mesh Traffic Management

Security

The next logical step of an application lifecycle is to secure the application, especially in the case of thousands of microservices. In these cases, the connectivity is dynamic, and each service-to-service communication needs to be encrypted, authenticated, and authorized.

Let's deep dive into four sections:

- **Zero-trust security model**
  Service mesh should be built on top of a zero-trust security model, which starts out with the assumption that the infrastructure especially network is not secure. It requires a full set of continuous checks for user authentication, traffic inspection, identity authorization, access and privilege audits, and so on.

  We recommend the following in the service mesh implementation:

  - Nothing is accessible by default
  - Port scans should be performed
  - Server should block cURL requests

- **Imposter prevention**
  To prevent identity fraud for service mesh, application and infrastructure owners need to have a zero-trust security mindset. Every user or service owner should be authenticated for access control. For example, Figure 7 describes imposter prevention in service-to-service transactions . The *green* service is authenticated to communicate with the *blue* service via an explicit whitelist policy. An unauthorized *red* service might pretend to be a *green* service and these attempts need to be blocked. An enterprise-grade service mesh solution should provide integration into SSO solutions with multi-factor authentication (MFA), and into RBAC solutions for authorization that integrate with enterprise AD or LDAP.

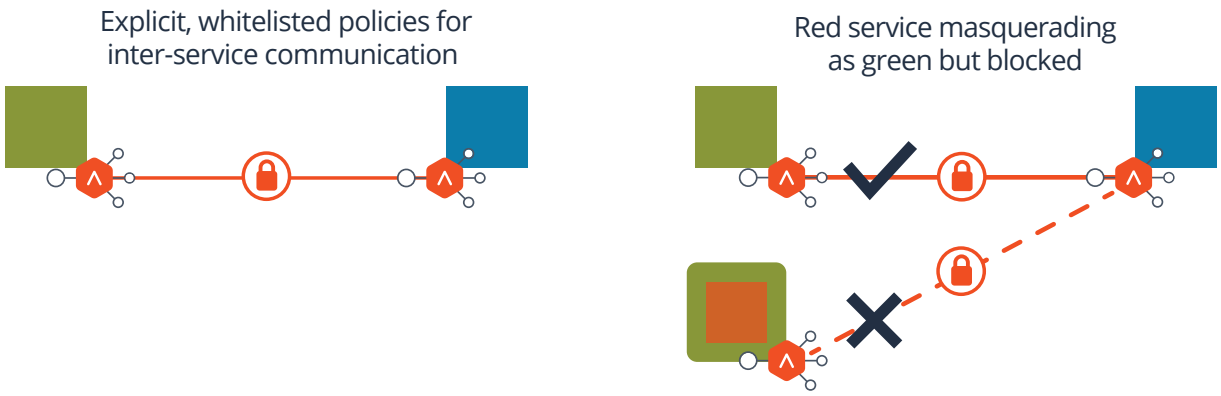

Figure 7: Imposter Attack and Prevention

- **Encryption and certification**
  Transparent encryption without any app modifications is a fundamental need for service communications to be secure. The service mesh should be able to provide flexible models to encrypt messages between services, using *TLS/mTLS* policies. It should also provide built-in certificate management layer, which would transparently distribute certificates to the microservices, based on the same identity.

- **Policy and telemetry**
  Service mesh should also provide a flexible way to enforce authorization policies and be able to collect telemetry for the services. The policy enforcement can be either done at the proxy/sidecar level or could also be done externally with caching enabled. This should include policies like *quota* and *rate limiting*.

Observability

Monitoring and auditing at the container level based on real-time telemetry from the service mesh is a requirement, not a nice-to-have. And deploying a solution with a built-in analytics engine instead of add-on adapters ensures real-time application performance monitoring and the ability to pinpoint latency.

A few of the key functionalities of "integrated observability" are depicted in Figure 8:

- **Metrics**
  It's important to ensure that proxies are injected as close to the service instances as possible. Istio has taken the approach of a sidecar, which collects real-time telemetry on performance, security, and infrastructure. For usability, it's also very useful to aggregate all metrics in one dashboard (see Figure 8a). There are different open source projects or components involved in monitoring and displaying metrics data. The integration efforts and potential complexities in nested API calls should not to be neglected.

Performance, Security, Infrastructure

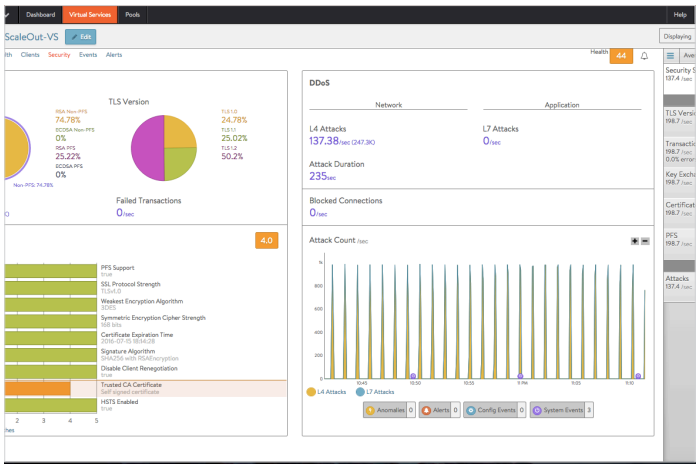

Figure 8a: Metrics

- **Mapping**
  While metrics are an important first step, they are just data points without the visualization of services and associated connections. Service graph or application map is a graphical way to gain visibility and understand the dependency mapping between microservices (see Figure 8b). The graph can show the microservices that are authorized to talk to each other and be dynamically updated as services are added, deleted or updated. It can also monitor the health of services with color coding, and the performance of services with node sizing.

Visibility, Dependency

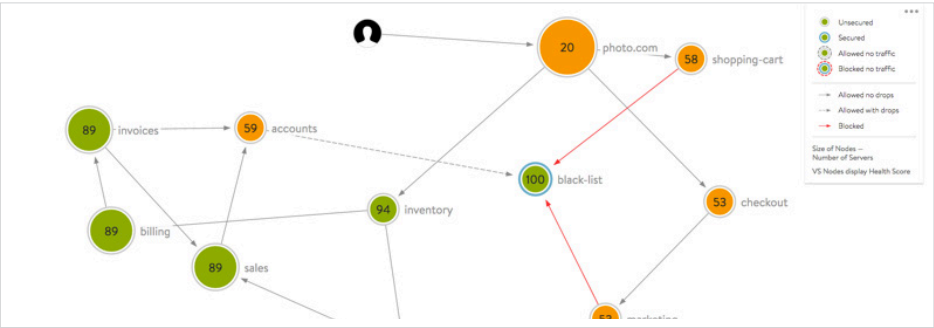

Figure 8b: Application Map

- **Tracing**
  Once the mapping is figured out, it allows distributed tracing to track end-to-end latency across microservices (see Figure 8c). With a sidecar proxy deployment, all network traffic going in and out of your application, including HTTP/1.1, HTTP/2.0 and gRPC requests can be automatically traced.

End-to-end tracing across microservices

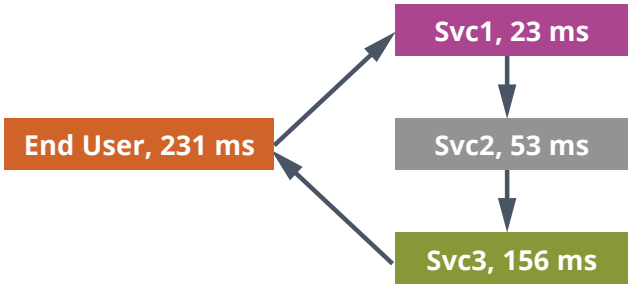

Figure 8c: Application Tracing

- **Logging**
  It's important to track and log every transaction as containers are constantly spinning up and down. The challenges of tracing or troubleshooting microservices are two-fold: how to quickly identify issues in real-time and how to root cause the problem. Searchable application logs and intelligent log analytics that surface anomalies and performance issues should be a part of every service mesh implementation (see Figure 8d).

Log Analytics

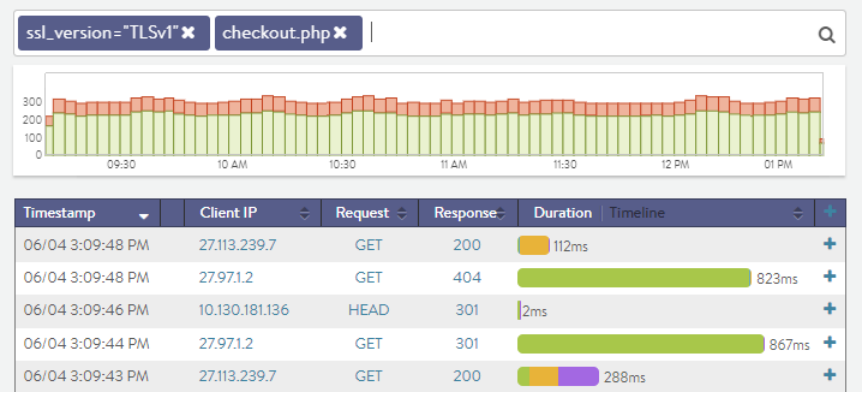

Figure 8d: Application Logging

AVI UNIVERSAL SERVICE MESH

With a distributed architecture, Avi's Universal Service Mesh - a part of the Avi Vantage Platform - resembled service mesh even before the formal industry emergence of that term. Avi Vantage (see Figure 9) is built on the following 5 fundamental building blocks:

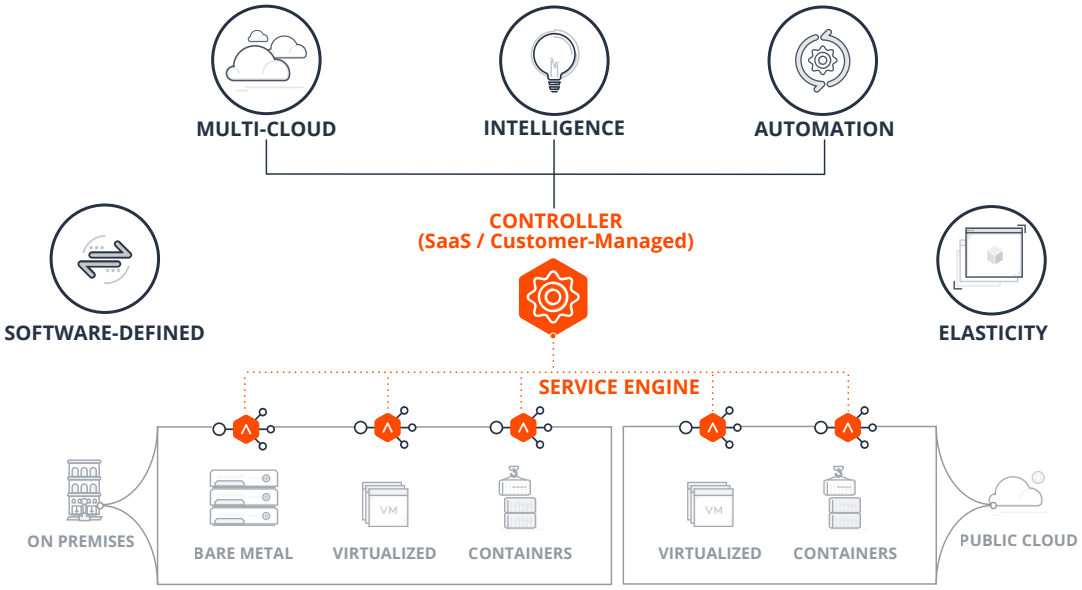

Figure 9: Avi Vantage Platform High-Level Overview

Software-defined

Separation of control and data planes allows centralized control and a single repository of policies, instead of managing disparate hardware or virtual appliances that tightly couple control and data planes.

Elasticity

The data plane (composed of a single fabric of Avi Service Engines) scales out and in elastically. These application service proxies are deployed close to applications regardless of the underlying infrastructure.

Multi-cloud

A modern architecture that spans both on-prem data centers and public clouds and across bare metal servers, virtual machines, and containers. In other words, you have a solution that supports both traditional and cloud-native applications.

Intelligence

The mesh of Avi Service Engines (service proxies) collect and send application telemetry from network traffic, enabling the controller to make intelligent infrastructure decisions such as virtual service placement, recovery from failures, autoscaling, and security.

Automation

The platform's RESTful APIs enable integration into any ecosystem, end-to-end automation through the lifecycle of an application, and autoscaling of application services.

Avi's approach is complementary to Istio in a way that allows service mesh to extend beyond containers as an architectural pattern. It integrates the Istio control plane inside Avi Controller, giving enterprises the option to deploy Envoy proxies or Avi Service Engines as the data plane for all environments. Figure 10 shows how the following core groups of features are achieved:

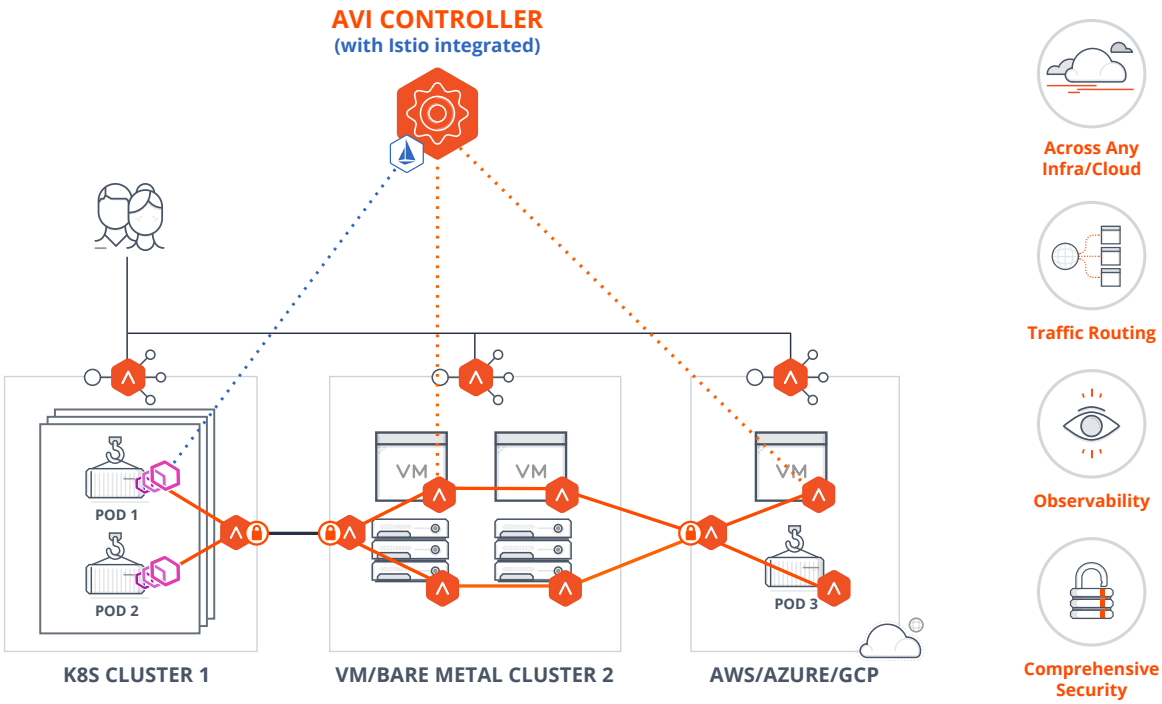

Figure 10: Universal Service Mesh Extends Istio

Universality

- Multi-Infra: Traditional and cloud-native apps in VMs/bare metal/containers
- Multi-Cluster: Inter/intra container cluster management and secure gateways
- Multi-Cloud: Across on-premises data centers and multi-region public clouds

Traffic Management

- Ingress gateway with integrated IPAM/DNS, blacklist/whitelist and rate limiting
- L4-7 load balancing with SSL/TLS offload
- Automated service discovery and application map

Observability

- Real-time application and container performance monitoring with tracing
- Big data driven connection log analytics
- Machine learning-based insights and app health analytics

Security

- Zero trust security model and encryption
- Distributed WAF for application security
- SSO for enterprise-grade authentication and authorization

SERVICE MESH USE CASES

The paper focuses on two use cases: ingress gateway and multi-cluster deployment.

Ingress Gateway (see Figure 11)

1. Ingress is deployed at entry point into service mesh, making sure external accesses are authenticated and authorized
2. Each ingress provides integrated DNS, IPAM, and GSLB capabilities
3. Gateways deployed at the edge of the cluster allow for communications between clusters while preserving authentication policies and isolation
4. Gateways preserve identity and security policies for communication without compromising isolation
5. Ingress security includes B/W lists, rate limiters, DoS, distributed WAF, and SSO
6. In this deployment model, Istio control plane and Envoy data plane operate separately from Avi ingress load balancers

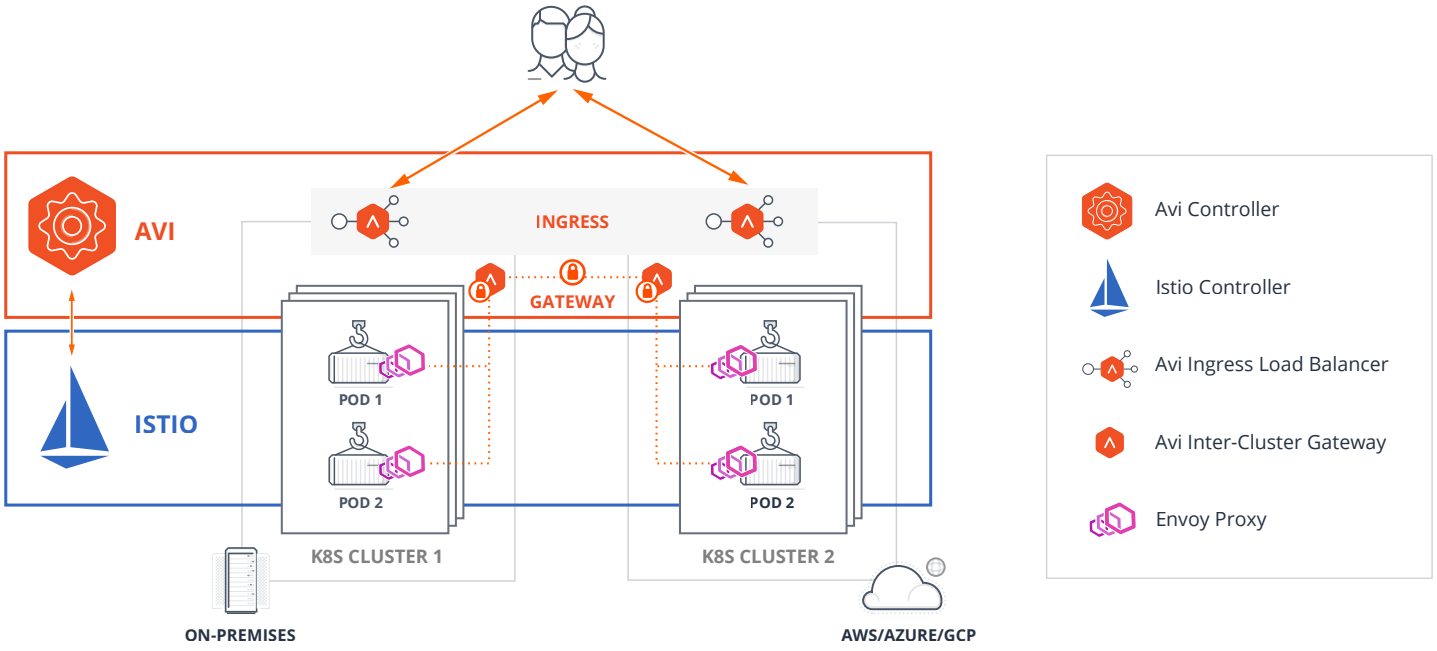

Figure 11: Use Case 1: Ingress Gateway

Multi-Cluster (see Figure 12)

1. Istio's control plane is integrated into Avi's Controller - a central point to configure service mesh across multiple clusters, including non-Kubernetes deployments
2. Observability: end-user has full visibility into performance and security insights
3. Automation: native Istio constructs integrated with Avi using RESTful APIs
4. Multi-cluster: clusters and services are isolated, secure, scalable, available
5. Multi-cloud: across public and private clouds, including VMware, OpenStack, AWS, Azure, and Google Cloud environments

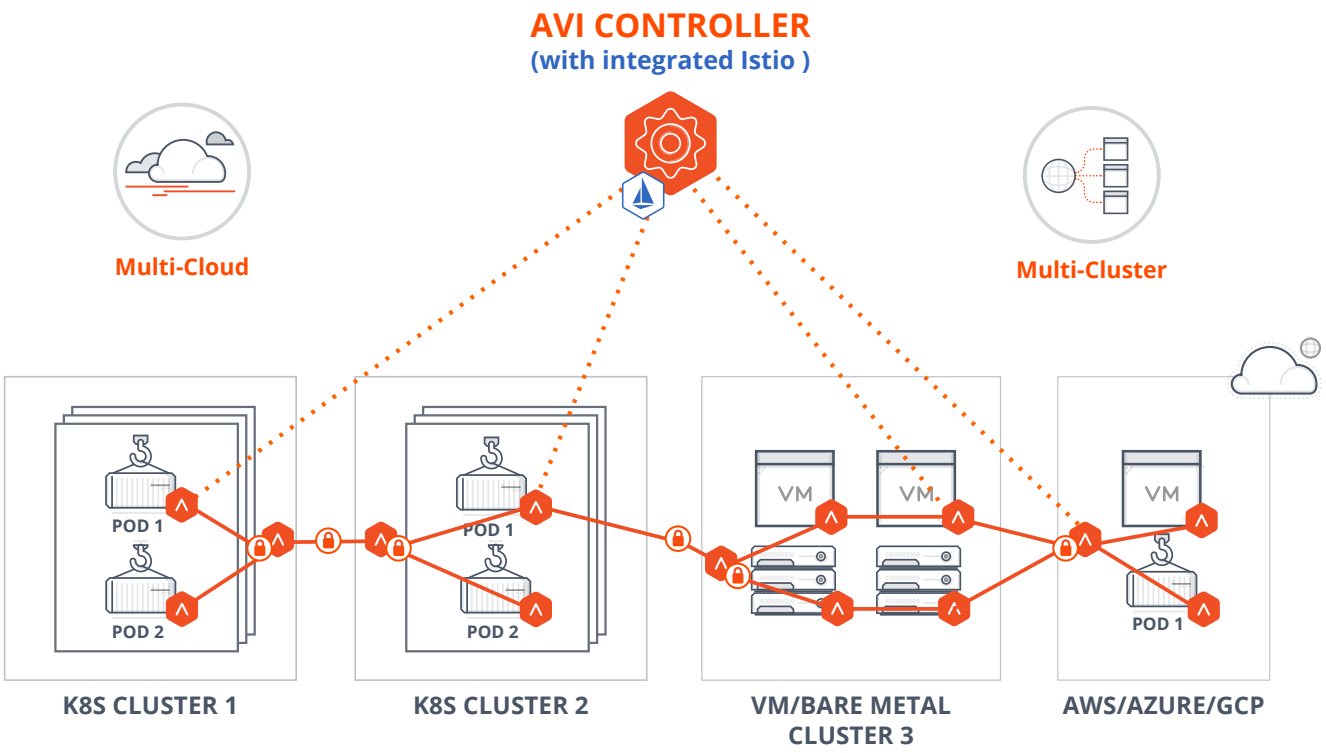

Figure 12: Use Case 2: Multi-Cluster and Multi-Cloud Service Mesh

MORE INFORMATION

1. Avi Universal Service Mesh Overview
2. Avi Universal Service Mesh Walkthrough