ISSN: 2755-0109

Journal of Media & Management

SCIENTIFIC Research and Community

Review Article

Open Access

Kubernetes Traffic Management using Istio

Pallavi Priya Patharlagadda

United States of America

ABSTRACT

When infrastructures transition from monolithic to microservices, containers might provide consistency and granularity in resource management. We can now treat one computer as multiple computers, thanks to microservices. This decreased resource waste by enabling programs to scale up and down computers in response to demand. But how can traffic be directed to services that are hosted on-premises, in a hybrid cloud, or on multi cloud deployments? The idea is to use service mesh to handle such scenarios in microservices. The idea of service mesh is to increase Kubernetes' efficacy and security. One use of this service mesh concept is Istio. In this paper, we also discuss why we need Istio even though Kubernetes has Ingress.

***Corresponding author**

Pallavi Priya Patharlagadda, United States of America.

Received: January 11, 2022; **Accepted:** January 18, 2022; **Published:** January 25, 2022

Problem Statement

In 2014, Google made Kubernetes open-source, and during the following three years, its growth was exponential. It evolved into a container scheduling solution that helped with distributed application deployment and scheduling by treating several computers as if they were a single machine. The flexibility of computing resources must be strong since a single machine's resources are limited and Internet applications may experience traffic floods at different periods (due to the rapid increase of user scale or diverse user qualities). Large-scale applications are too complex for a single computer to handle, while small-scale applications using the entire host would be a waste. To put it briefly, Kubernetes allows the system to automatically attain and maintain the desired state of the service. How then can you control the flow of traffic via the service after the application is live? We'll examine how Kubernetes handles service management and how Istio has modified it below.

Introduction

Service Mesh

By adding observability, stability, and security capabilities to applications at the platform level as opposed to the application level, Kubernetes service meshes are a useful tool. Interest in this technology has increased due to the popularity of microservices and Kubernetes, with several enterprises using Kubernetes service mesh solutions.

The architecture of microservices heavily relies on the network. Traffic management between application services is possible using a service mesh. Although there are alternative methods for managing network traffic, they are not as long-lasting as a service mesh as they require manual, prone to mistake effort that puts more of an operational load on the DevOps team.

A Kubernetes service mesh is often implemented as a collection of network proxies that are deployed inside of containers together with an application code sidecar. These sidecar proxies oversee communication between containerized microservices and serve as an entry point for service mesh capability. Proxies under the control of the control plane make up the data plane of the Kubernetes service mesh.

Kubernetes and service mesh architectures came into being together with the rise of cloud-native apps. A single containerized service may have hundreds of instances, while an application may have thousands of instances of each service. Since each of these instances varies often, dynamic scheduling is necessary, and Kubernetes assists in handling this.

Istio

Istio is a service mesh solution that works with any vendor or platform that enables developers to execute, connect, secure, control, and monitor distributed microservices systems. In workloads based on virtual machines and containers, it controls how services communicate with one another.

Istio is helpful on any platform because it is independent and open-source, but it works best with Kubernetes. By integrating the two, you can protect communication between pods and services both on the network and in applications.

Software for container orchestration gains more functionality by integrating with Istio. Failure and traffic management are not tasks that Kubernetes typically solves, but rather multi-container workloads like microservices. Istio fills this void by building dependable and more effective solutions.

Advantages of Using Istio with Kubernetes

Organizations may offer decentralized apps at scale with the help of Istio. Network activities like inter-service traffic management, encryption, authorization, debugging, and auditing are made easier with its assistance.

In addition to the functions that standard Kubernetes delivers, Istio supports the following

- **Cloud-Native Application Security:** encryption, robust identity-driven authorization, and authentication allows you to concentrate on application-level security.

- **Efficient Traffic Management:** use error injection, failover, retries, and extensive routing rules to provide fine-grained control over network traffic and behavior. The incorporation of Chaos Monkey enables SREs to introduce delays and errors during post-production testing, hence enhancing resilience.
- **Service Mesh Monitoring:** You can track, monitor, and resolve issues with the help of Itsio's service-level visibility. In the absence of detailed information, bottlenecks are difficult to fix quickly. Maintaining API responsiveness and turning off faulty services and replicas is simple with a service mesh.
- **Easy Deployment with Kubernetes:** Virtual machines and containers are only two examples of the classic and new workloads for which Istio provides network visibility and management.
- **Simplified Load Balancing:** Advanced features allow canary installations, offer client-based routing and automate load balancing.
- **Policy Enforcement:** Policies including access control, quotas, and rate limitations are enforced with the use of a configuration API and policy layer.

How Istio Works?

Istio employs a proxy to snoop on all of your network traffic, enabling a wide range of application-aware functionality contingent on your setup.

The control plane programs the proxy servers dynamically, adjusting them in response to changes in the environment or the rules based on your intended configuration and its perception of the services. The means of communication between services is the data plane. The network is unable to distinguish between different types of traffic, determine who is sending or receiving it, or make any choices regarding it in the absence of a service mesh.

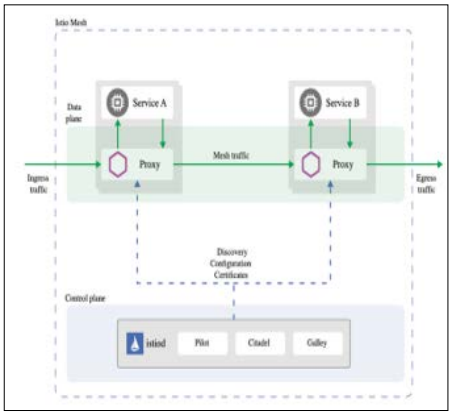
Istio Supports Two Data Plane Modes

- **Sidecar Mode:** sidecar mode, which runs alongside VM-based services and launches an Envoy proxy with each pod you launch in your cluster.
- **Ambient Mode:** ambient mode makes use of a per-node Layer 4 proxy and, for Layer 7 functionality, a per-namespace Envoy proxy.

Istio Architecture & Components

Two planes make up the Istio service mesh: the control plane and the data plane

- The data plane is made up of several intelligent proxies that are installed as sidecars next to the application and are based on the Envoy open source project.
- To route traffic, the control plane sets up and maintains proxies.



Istio Data Plane

Envoy proxies, installed as sidecars and running alongside application instances in Kubernetes pods, make up the data plane. The Envoy proxies oversee network connections between microservices and manage traffic for the system's services.

With Istio, developers may add proxies to their applications without requiring any modifications to the code by deploying Envoy as a sidecar. By implementing transport layer security (TLS) and traffic encryption policies, as well as listing routing rules (supporting HTTP, gRPC, and TCP), the Envoy proxy manages traffic.

All application traffic passes via these Envoy proxies, which promote observability by gathering a lot of data and providing insightful information about traffic.

Istio Control Plane

Three essential parts make up the Istio control plane

- **Pilot:** Talks to the Envoy sidecar using the Envoy API. The pilot is in charge of route, traffic control, and service inspection.
- **Citadel:** Manages credential and certificate management, user authentication, and secure communication between services.
- **Galley:** In charge of processing, distributing, and managing configurations.

Envoy

To mediate inbound and outgoing traffic for services in a service mesh, Istio uses Envoy, a high-performance proxy. It restricts data plane traffic interaction to envoy proxies exclusively. Istio runs Envoy proxies alongside services in a pod by deploying them as sidecar containers. This makes sense since it enhances services with Envoy's inherent qualities, which include

- Load balancing
- Dynamic service discovery
- TLS termination
- Circuit breakers
- HTTP/2 and gRPC proxies
- Health checks
- Fault injection
- Staged rollouts with %- based traffic split
- Rich metrics

Istio can extract rich telemetry and impose policies via a sidecar deployment. Istio can provide this information to monitoring systems so they can understand the behavior of the entire mesh. You may also add Istio functionality to current deployments using this sidecar proxy paradigm without having to rewrite or re-architect the code.

Envoy proxies facilitate the following important Istio functionality and tasks

- **Traffic Control:** it allows you to control TCP, gRPC, HTTP, WebSocket, fine-grained traffic management, and extensive routing rules.
- **Network Resilience:** features such as setup retries, circuit breakers, fault injection, and failovers.
- **Security and Authentication:** Istio may impose rate limits, access control, and security controls thanks to Envoy proxies. By utilizing the configuration API, you may specify it.
- **Pluggable Extensions:** this WebAssembly-based architecture allows you to generate telemetry for your mesh traffic and impose custom policies

Citation: Pallavi Priya Patharlagadda (2022) Kubernetes Traffic Management using Istio. Journal of Media & Management. SRC/JMM-E101.
DOI: doi.org/10.47363/JMM/2022(4)E101

Istiod

Istiod offers configuration, service discovery, and certificate management. During runtime, it translates the high-level routing rules governing traffic behavior into Envoy-specific settings and distributes them to sidecars.

- **Service Discovery:** The pilot is in charge of taking platform-specific techniques for service discovery and combining them into a common format that sidecars that follow the Envoy API may use. Istio facilitates discovery across several contexts, such as Kubernetes and virtual machines. The Traffic Management API of Istio can assist Istiod in fine-tuning an Envoy setup to impose more detailed control over traffic inside a service mesh.
- **Security:** To accomplish robust service-to-service and end-user authentication, Istiod security offers integrated identity and credential management. Istio can also assist you in improving unencrypted traffic inside a service mesh. Istio allows you to apply regulations based on service identification, rather than comparatively erratic layer three or layer four network identities. To manage service access, you may also take advantage of Istio's authorization features.
- **Certificate Management:** To provide secure mTLS communication within the data plane, Istiod performs the role of a certificate authority (CA), creating certificates.

The Need for an Istio Management Plane

Even though Istio has had plenty of time to develop and is still releasing new features every quarter, a lot of organizations are looking to improve Istio by adding a management plane. This will make it easier to configure Istio and give you access to features like enhanced multi-cluster capabilities, software lifecycle management, advanced/federated security with builds that are ready for FIPS, backported CVEs, and everything else you need for successful Day 2 operations [1].

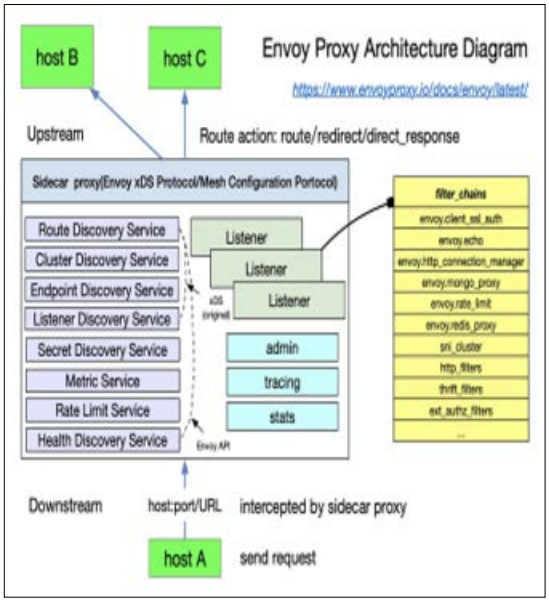
Kubernetes Ingress vs Istio Gateway

The only traffic that kube-proxy can transport is inside a Kubernetes cluster. A CNI-created network houses the pods of a Kubernetes cluster. The creation of an ingress, a Kubernetes resource object, allows for communication with devices outside the cluster. The Kubernetes edge nodes that govern north-south traffic are home to an ingress controller that powers it. It is necessary to dock ingress to different ingress controllers, such as traffic and the nginx ingress controller. Ingress is easy to use, however, it only applies to HTTP traffic. A restricted set of fields, including service, port, HTTP path, etc., must match for it to be able to route traffic. Routing TCP traffic, including that from MySQL, Redis, and other RPCs, is thus not viable. For this reason, you'll find individuals annotating ingress resources with nginx configuration language. The service's Load Balancer or Node Port are the only options for directly routing traffic north-south; the former requires support from cloud vendors, while the latter requires extra port management [2-5].

Istio Gateway manages traffic moving north-south into and out of the cluster, much as Kubernetes Ingress. To transport connections to and from the mesh's edge, Istio Gateway defines a load balancer. The standard outlines several open ports, the protocols they employ, the SNI load balancing settings, etc. The CRD extension known as Gateway takes advantage of the sidecar proxy's capabilities as well.

Envoy

In Istio, Envoy is the sidecar proxy by default. Istio expands its control plane using the xDS protocol from Enovy. Before delving into Envoy's xDS protocol, let's brush up on some fundamental vocabulary. The words below are basic terms in Envoy together with their corresponding data structures [3-6].



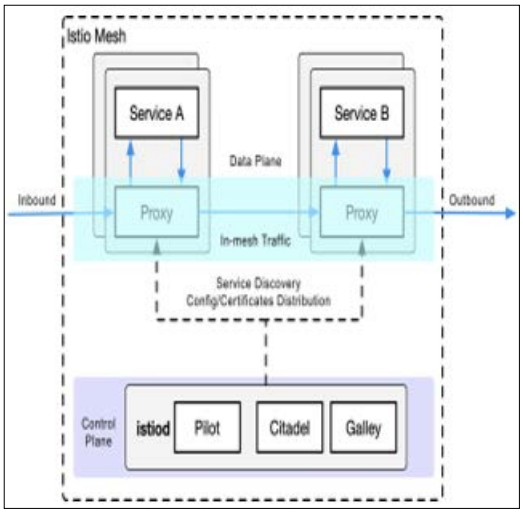
Basic Terminology

These are the fundamental phrases of Envoy that you ought to be aware of

- **Downstream:** The host that made the request connects to Envoy, makes the request, and gets the answer. This host is known as the downstream host.
- **Upstream:** The host that receives the requests, or the upstream host, accepts connections and requests from Envoy and responds to them.
- **Listener:** A listener is a specified network address that downstream clients can connect to, such as a port or UNIX domain socket. Envoy allows connections with downstream hosts by exposing one or more listeners.
- **Cluster:** An Envoy cluster is a collection of logically equivalent upstream hosts. Envoy uses service discovery to find out who is in a cluster. Proactive health checks are one way to potentially find out how well each cluster member is doing. Envoy uses a load-balancing strategy to choose which cluster member to route requests through.

Envoy allows for the configuration of many listeners, the setup of a filter chain (filter chain table) by each listener, and scalable filtering, making it easier to control traffic behavior through the use of private RPC, encryption, and other features [7-9].

The MOSN open-source project by Ant Group is one example of how the xDS protocol, which was introduced by Envoy and is the default sidecar proxy in Istio, can conceivably be used as a sidecar proxy in Istio as long as it is implemented [10].



The following are only a few of the features that make Istio a very feature-rich service mesh.

- **Traffic Management:** This is the most basic feature of Istio.
- **Policy Control:** Permits the use of telemetry capture, charging, quota management, and access control systems.
- **Observability:** The sidecar proxy implements this.
- **Security Authentication:** The Citadel component handles certificate and key management.
- **Traffic Management in Istio:** Istio provides users with traffic control assistance by defining the following CRDs.
- **Gateway:** A gateway is an HTTP/TCP connection receiver that operates at the network's edge.
- **Virtual Service:** This is the real virtual service that links the Istio Gateway with the Kubernetes service. It is also capable of carrying out other tasks, such as creating a set of traffic routing rules that will be used when a host is addressed.
- **Destination Rule:** After traffic has been routed, the access policy is decided by the policy specified by the Destination Rule. It specifies the flow of traffic, to put it simply. Load-balancing setups, connection pool sizes, and external detection configurations (which identify and remove problematic hosts from the load-balancing pool) are a few examples of these rules.
- **Envoy Filter:** Istio Pilot generates a proxy configuration, the Envoy Filter object defines filters for proxy services that allow customization of that configuration. Primary users often don't utilize this setup very often.
- **Service Entry:** Istio service mesh services are not by default able to find services outside of the mesh. By enabling the insertion of new entries to the Istio service registry, Service Entry allows automatically detected services inside the mesh to connect to and route to manually added services.

Conclusion

It just requires using Kubernetes to manage microservices and then implementing service mesh if the item managed by Kubernetes is a pod and the object managed in service mesh is a service. Use a serverless platform like Knative if you don't even want to maintain a service; however, that is an afterthought.

References

1. Available at: https://www.solo.io/topics/istio/istio-architecture/.
2. Available at: https://thenewstack.io/why-do-you-need-istio-when-you-already-have-kubernetes/.
3. Available at: https://www.opsmx.com/blog/what-is-istio-and-why-is-it-necessary-for-kubernetes/.
4. Available at: https://www.solo.io/topics/istio/istio-kubernetes/.
5. Available at: https://tetrate.io/blog/what-is-istio-and-why-does-kubernetes-need-it/.
6. Available at: https://kubebyexample.com/learning-paths/istio/intro.
7. Available at: https://istio.io/latest/docs/concepts/traffic-management/.
8. Available at: https://dev.to/techworld_with_nana/step-by-step-guide-to-install-istio-service-mesh-in-kubernetes-d6d.
9. Available at: https://birthday.play-with-docker.com/istio-docker-desktop/.
10. Available at: https://jimmysong.io/en/blog/service-mesh-the-microservices-in-post-kubernetes-era/.