O'REILLY®

Compliments of IBM

Istio Explained

Getting Started with Service Mesh

Lin Sun & Daniel Berg

Preview Edition

REPORT

Build

Smart

Run Istio on the IBM Cloud to connect, manage and secure microservices at scale. Explore tutorials, sample code and tech talks to learn more.

ibm.biz/oreilly-istio-tech

Istio Explained

Getting Started with Service Mesh

This Preview Edition of *Istio Explained*, Chapter 1, is a work in progress. The final report is currently scheduled for release in November 2019 and will be available at *learning.oreilly.com* and through other retailers once it is published.

Lin Sun and Daniel Berg

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Istio Explained

by Lin Sun and Daniel Berg

Copyright © 2020 IBM Corporation. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Virginia Wilson and John Devins
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2019: First Edition

Revision History for the Early Release

2019-09-30: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492073956* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. Istio Explained, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors, and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and IBM. See our statement of editorial independence.

978-1-492-07393-2

[LSI]

Table of Contents

Preface. v

1. Introduction to Service Mesh. 7

Challenges Managing Microservices 7
What is a Service Mesh anyway? 8
How Does a Service Mesh Work? 9
Service Mesh Ecosystem 11
Conclusion 16

Preface

Microservices can be complicated and difficult to manage. The increase use of containers and cloud have increased the distributed nature of applications and has further complicated development teams' ability to understand and control interactions between services within these environments. These complexities have given rise to a new solution called a service mesh which helps teams manage the interactions between microservices.

Who This Book is For

We wrote this short book for developers and operators, however, anyone responsible for the delivery of microservices will find the material valuable. We assume you are just learning about Istio and service mesh or you have been recently introduced to Istio and looking for a getting started guide.

What You Will Learn

In the pages that follow we will give you a solid background into the challenges of microservices, explain what a service mesh is, describe how a service mesh works, and explore the current service mesh landscape. Starting with Chapter 2, we'll use Istio as our main service mesh implementation to explain how to set up and use a service mesh. We'll describe the Istio architecture and explore Istio's observability, traffic management and security capabilities.

You don't have to understand and consume all service mesh features at once. You can instead adopt features incrementally and still enjoy some of the benefits service mesh offers. To that end, we take an

incremental approach to teaching you how to adopt a service mesh like Istio, our goal being to set you up for gradual adoption so you can see benefits as quickly as possible.

Why Istio?

While there are many service mesh options to choose from, as you'll see in Chapter 1, we are personally most familiar with Istio and so chose to illustrate the benefits service mesh can offer through Istio's features. We encourage you to use the information we provide in this book to evaluate and choose the right solution for your needs.

Prerequisites

The working examples in the book build on Kubernetes for managing the sample's containers and Kubernetes serves as the platform for Istio itself. To get the most out of the working examples, it would be helpful for you to have a basic understanding of Kubernetes. To get quickly up to speed, we recommend that you check out this Kubernetes tutorial: Kubernetes 101. As with any adoption process of a new technology, you are likely to run into trouble with configuration or setup. We provide a chapter that introduces techniques and commands that may help you troubleshoot issues that you may encounter on your journey to adopt service mesh.

CHAPTER 1

Introduction to Service Mesh

In this chapter we explore the notion of a service mesh and the vast ecosystem that has emerged in support of service mesh solutions. Organizations face many challenges when managing services especially in a cloud-native environment. We are introducing service mesh as a key solution on your cloud-native journey because we believe that a service mesh should be a serious consideration for managing complex interactions between services. An understanding of service mesh and its ecosystem will help you choose an appropriate implementation for your cloud solution.

Challenges Managing Microservices

Microservices are an architecture and development approach that breaks down business functions into individually deployable and managed services. We view microservices as loosely coupled components of an application that communicate with each other using well defined APIs. A key characteristic of microservices is that you should be able to update them independent from one another which enables smaller and more frequent deployments. Having many loosely coupled services that are independently and frequently changing does promote agility but it also presents a number of management challenges.

1. Observing interactions between services can be complex when you have many distributed, loosely coupled components.

2. Traffic management at depth becomes more important to enable specialized routing for A/B or canary deployments without impacting clients within the system.
3. Securing communication by encrypting the data flows is more complicated when the services are decoupled and not part of the same binary process.
4. Managing timeouts and communication failures between the services can lead to cascading failures and is more complicated to get correct when the services are distributed.

Many of these challenges can be resolved directly in the services code. However, adjusting the service code puts a massive burden on you to properly code solutions to these problems and it requires each microservice owner to agree on the same solution approach to assure consistency. Solutions to these types of problems are complex and it is extremely error prone to rely on application code changes to provide the solutions. Removing the burden of codifying solutions to these problems is a primary reason we have seen the introduction of the service mesh.

What is a Service Mesh anyway?

A service mesh is a programmable framework that allows you to observe, secure, and connect microservices. A service mesh doesn't establish connectivity between microservices but, instead, it has policies and controls that are applied on top of an existing network to govern how microservices interact. Generally a service mesh is agnostic to the language of the application and can be applied to existing applications usually with little to no code changes.

A service mesh, ultimately, shifts responsibilities out of the application and moves that responsibility to the network. This is accomplished by injecting behavior and controls within the application that are then applied to the network. This is how things such as metrics collecting, communication tracing, and secure communication can take place without changing the applications themselves. As stated earlier, a service mesh is a programmable framework. This means that you can declare your intentions and the mesh will ensure that your declared intentions are applied to the services and network. A service mesh simplifies the application code making it easier to write, support, and develop by removing complex logic that

normally has to be bundled in the application itself. Ultimately a service mesh allows you to innovate faster.

How did we get here?

Netflix was a pioneer in developing the early frameworks for managing microservices as part of the Netflix OSS or Open Source Software using components such as Asgard (control plane), Eureka (service registry), Zuul (load balancer gateway), and Ribbon (client-side load balancer). These early frameworks were implemented as a series of libraries written in Java. To make use of their features, you had to modify your Java application by adding the necessary Netflix OSS libraries and adjusting the application logic to make use of the types and methods within these libraries. This approach worked well if you were developing Java applications and if you were willing to adjust the code.

Recent years have given rise to the use of containers and Kubernetes (container orchestration) as the basis for microservices. Using containers has made it simpler to develop applications using multiple languages (e.g., Python, Golang, Java, etc). Containers also provided opportunities for standardizing operational capabilities as more features were being abstracted out of the application code and moved into the container platform. These conditions helped to usher in the modern service management system which we now call a service mesh.

How Does a Service Mesh Work?

Many service mesh implementations have the same general reference architecture (see Figure 1-1). A service mesh will have a *control plane* to program the mesh and client-side proxies in the *data plane* (shown below the dashed line) which serve as the control point for securing, observing, and routing decisions between services. The control plane transfers configurations to the proxies in their native format. The client-side proxies are attached to each application within the mesh. Each proxy intercepts all inbound and outbound traffic to and from its associated application. By intercepting traffic, the proxies have the ability to inject behavior on the communication flows between services. Here is a list of behaviors that commonly found in a service mesh implementation.

- Traffic shaping with dynamic routing controls between services
- Resiliency support for service communication such as circuit breakers, timeouts, and retries
- Observability of traffic between services
- Tracing of communication flows
- Secure communication between services

In Figure 1-1, you can see that the communication between two applications such as App1 and App2 is executed via the proxies versus directly between the applications themselves as indicated by the red arrows. By having communication routed between the proxies, the proxies serve as a key control point for performing complicated tasks such as initiating TLS handshaking for encrypted communication (shown on the red line with the lock in Figure 1-1) Since the communication is performed between the proxies, there is no need to embed complex logic in the applications themselves. Each service mesh implementation option has various features but they all share this general approach.

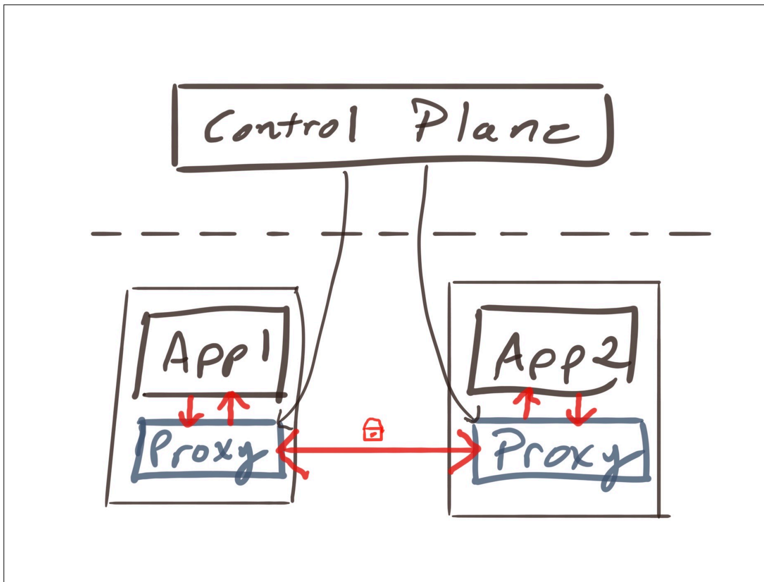

Figure 1-1. Anatomy of a Service Mesh

Service Mesh Ecosystem

You may find navigating the service mesh ecosystem to be a bit daunting because there are many different implementation choices. While most choices share the same reference architecture shown in Figure 1-1, there are variations to approaches and project structure you should consider when making your service mesh selection. Here are some questions to ask yourself when selecting a service mesh implementation.

- Is it an open-source project governed by a diverse contributor base?
- Does it use a proprietary proxy?
- Is the project part of a foundation?
- Does it contain the feature set that you need and want?

The fact that there are many service mesh options validates the interest of service mesh and it shows that the community has not selected a de facto standard as we have seen with other projects such as Kubernetes for container orchestration. Your answers to these questions will have an impact on the type of service mesh that you prefer whether it is a single vendor controlled or a multi-vendor, open source project. Let's take a moment to review the service mesh ecosystem and describe each implementation so that you have a better understanding of what is available.

Envoy

Envoy proxy is an open source project originally created by the folks at Lyft. Envoy proxy is an edge and service proxy that was custom built to deal with the complexities and challenges of cloud native applications. While Envoy itself does not constitute a service mesh, it is definitely a key component of the service mesh ecosystem. What you will see from exploring the service mesh implementations is that the client-side proxy from the reference architecture in Figure 1-1 is often implemented using an Envoy proxy.

Envoy is one of the six Graduated projects in the Cloud Native Computing Foundation (CNCF). The CNCF is part of the Linux foundation and it hosts a number of open source projects that are used to manage modern cloud native solutions. The fact that Envoy is a CNCF graduated project is an indicator that it has a strong community with adopters using Envoy proxy in production settings.

While Envoy was originally created by Lyft, the open source project has grown to a diverse community as depicted in the company contributions in the CNCF DevStats graph shown in Figure 1-2.

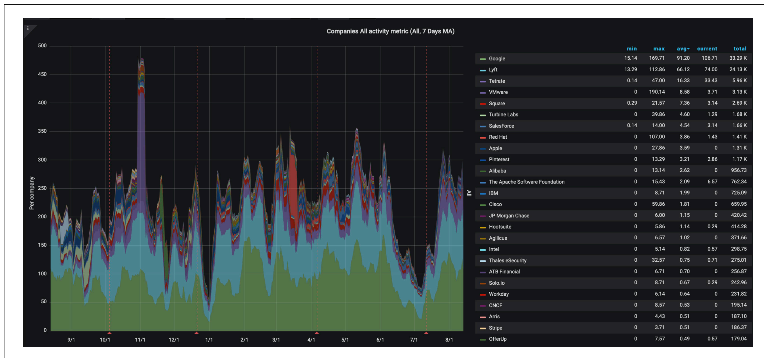

Figure 1-2. Envoy Twelve Month Contribution Distribution

Istio

The Istio project is an open source project co-founded by IBM and Google in 2017. Istio makes it possible to connect, secure, and observe your microservices while being language agnostic. Istio has grown to have contributions beyond just IBM and Google with contributions from VMware, Red Hat, and Aspen Mesh. Figure 1-3 shows the company contributions over the past twelve months using the CNCF DevStats tool. The recent open-source project named Knative builds upon Istio as well to provide tools and capabilities to build, deploy, and manage serverless workloads. At the time of writing this book, Istio was not contributed to the Cloud Native Computing Foundation (CNCF) but many of the components that Istio uses are in the CNCF such as Envoy, Kubernetes, Jaeger, and Prometheus. Istio is listed as part of the CNCF Cloud Native Landscape under the Service Mesh category.

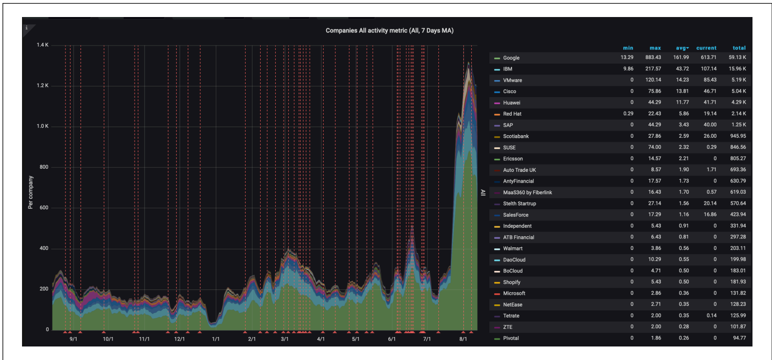

Figure 1-3. Istio Twelve Month Contribution Distribution

The Istio control plane extends the Kubernetes API server and utilizes the popular Envoy proxy for its client-side proxies. Istio supports mTLS communication between services, traffic shifting, mesh gateways, monitoring and metrics with Prometheus and Grafana, as well as custom policy injection. Istio has installation profiles such as demo and production to make it easier to provision and configure the Istio control plane for specific use cases.

Consul Connect

Consul Connect is a service mesh that is developed by Hashicorp. Consul Connect extends Hashicorp's existing Consul offering which has service discovery as a primary feature as well as other built-in features such as a key-value store, health checking, and service segmentation for secure TLS communication between services. Consul Connect is available as an open-source project with Hashicorp itself being the predominate contributor. Hashicorp has an enterprise offering for Consul Connect for purchase with support. At the time of writing this book, Consul Connect was not contributed to the CNCF or another foundation. Consul is listed as part of the CNCF Cloud Native Landscape under the Service Mesh category.

Consul Connect uses Envoy as the sidecar proxy and the Consul server as the control plane for programming the sidecars. Consul Connect includes secure mTLS support between microserves and observability using Prometheus and Grafana projects. The secure connectivity support leverages the Hashicorp Vault product for managing the security certificates. Recently, Hashicorp has intro-

duced L7 traffic management and mesh gateways into Consul Connect as beta features.

Linkerd

The Linkerd service mesh project is an open-source project as well as a CNCF incubating project. The predominate contributors to the Linkerd project are from Buoyant as shown in the twelve month CNCF DevStats company contribution graph shown in Figure 1-4. Linkerd has the key capabilities of a service mesh which includes observability using Prometheus and Grafana, secure mTLS communication, and the project recently added support for service traffic shifting. The client-side proxy used with Linkerd is proprietary to the Linkerd project itself written in Rust. Linkerd provides an injector to inject proxies during a Kubernetes pod deployment based on an annotation to the Kubernetes pod specification. Linkerd also includes a UI dashboard to view and configure settings of the mesh.

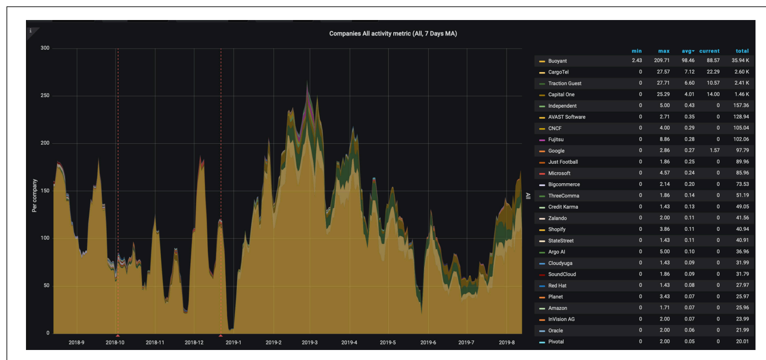

Figure 1-4. Linkerd One Year Contribution Distribution

App Mesh

App Mesh is a cloud service hosted by AWS to provide a service mesh with application-level networking support for compute services within AWS such as Amazon ECS, AWS Fargate, Amazon EKS, and Amazon EC2. As the project URL suggests, AWS App Mesh is not open source and is proprietary to Amazon. App Mesh does utilize the Envoy proxy for the sidecar proxies within the mesh which does have the benefit that it may be compatible with other AWS partners and open source tools. It appears that App Mesh has similar routing concepts as the Istio control plane which is not surprising since Istio serves as a control plane for Envoy

proxy. There is no mention of secure mTLS communication support between services at this time. The focus of App Mesh appears to be primarily traffic routing and observability.

Kong

Kong's service mesh builds upon the Kong edge capabilities for managing APIs and has delivered these capabilities throughout the entire mesh. While Kong is an open source project, it appears that the contributions are heavily dominated by Kong members. Kong is not a member of a foundation but it is listed as part of the CNCF Cloud Native Landscape under the API Gateway category. Kong does provide Kong Enterprise which is a paid product with support.

Much like all the other service mesh implementations, Kong has both a control plane to program and manage the mesh as well as a client-side proxy. In Kong's case the client side proxy is unique to the Kong project. Kong includes support for end-to-end mTLS encryption between services. Kong promotes their extensibility feature as a key advantage. You can extend the Kong proxy using Lua plugins to inject custom behavior at the proxies.

AspenMesh

AspenMesh is unlike the other service mesh implementations because ApsenMesh is a supported distribution of the Istio project. AspenMesh does have many open source projects on Github but their primary direction is not to build a new service mesh implementation but rather to harden and support an open source service mesh implementation through a paid offering. AspenMesh hosts components of Istio such as Prometheus and Jaeger making it easier to get started and use over time. They have features above and beyond the Istio base project such as a UI and dashboard to view and manage Istio resources. AspenMesh has introduced additional tools such as Istio Vet which is used to detect and resolve misconfigurations within an instance of Istio. AspenMesh is an example where there are new markets emerging to offer support and build upon source service mesh implementation such as Istio.

Service Mesh Interface

Service Mesh Interface or SMI is a relatively new specification that was announced at KubeCon EU 2019. SMI is spearheaded by Micro-

soft with a number of backing partners such as Linkerd, Hashicorp, solo.io, and VMware. SMI is not a service mesh implementation; however, SMI is attempting to be a common interface or abstraction for other service mesh implementations. If you are familiar with Kubernetes then SMI is similar in concept to what Kubernetes has with CRI or the Container Runtime Interface which provides an abstraction for the container runtime in Kubernetes with implementations such as docker and containerd. While SMI may not be immediately applicable for you, it is an area worth watching to see where the community may head as far as finding a common ground for service mesh implementations.

Conclusion

The service mesh ecosystem is vibrant and you have learned that there are many open source projects as well as vendor specific projects that provide implementations for a service mesh. As we continue to explore service mesh more deeply, we will turn our attention to the Istio project. We have selected the Istio project because it uses the Envoy proxy, it is rich in features, it has a diverse open source community, and, most importantly, we both have experience with the project.

About the Authors

Lin Sun is a Senior Technical Staff Member and Master Inventor at IBM. She is a maintainer on the Istio project and also serves on the Istio Steering Committee and Technical Oversight Committee. She is passionate about new technologies and loves to play with them. She holds 150+ patents issued with USPTO.

Daniel Berg is an IBM Distinguished Engineer responsible for the technical architecture and delivery of the IBM Cloud Kubernetes Service and Istio. Daniel has deep knowledge of container technologies including Docker and Kubernetes and has extensive experience building and operating highly available cloud-native services. Daniel is a member of the Technical Oversight Committee for the Istio.io open source service mesh project and he is responsible for driving the technical integration of Istio into IBM Cloud.