KAPITEL 3

Istio im Überblick

Wenn Organisationen im Betrieb von Container-Deployments ausreichend Erfahrungen gesammelt haben, werden viele von ihnen ein Service Mesh in ihrer Umgebung nutzen. Dieses Thema verursacht erheblichen Wirbel im Cloud-native Ökosystem. Derzeit suchen viele Administratoren, Architekten und Entwickler nach einem Verständnis und einer Anleitung dazu, wie, wann und warum ein Service Mesh eingesetzt werden soll. Werfen wir also einen Blick auf Istio.

Wie Sie in Kapitel 2 gelernt haben, führt Istio (wie andere Service Meshes auch) eine neue Schicht in die moderne Infrastruktur ein, die das Potenzial schafft, robuste und skalierbare Anwendungen mit granularer Kontrolle über sie zu implementieren. Falls Sie Microservices ausführen, verschärfen sich diese Herausforderungen, wenn Sie immer mehr Microservices bereitstellen. Allerdings kann es auch sein, dass Sie gar keine Microservices ausführen. Obwohl sich der Wert eines Service-Mesh vor allem in Deployments von Microservices zeigt, ist Istio auch auf Server vorbereitet, die direkt auf dem Betriebssystem Ihres virtuellen Computers und Bare-Metal-Servers laufen.

Service-Mesh-Architektur

Im Großen und Ganzen gesehen, umfassen Service-Mesh-Architekturen häufig zwei Ebenen: eine *Control Plane* (Steuerungsebene) und eine *Data Plane* (Datenebene), während eine dritte Ebene (die Verwaltungsebene oder *Management Plane*) in vorhandenen Infrastruktursystemen existieren kann. Die Architektur von Istio hält sich an dieses Paradigma. Abbildung 3-1 präsentiert die Zuständigkeitsbereiche nach Ebenen.

Eine umfassendere Erläuterung der Service-Mesh-Deployment-Modelle und der Ansätze für evolutionäre Architekturen finden Sie in *The Enterprise Path to Service Mesh Architectures* (*https://oreil.ly/TTO4p*).

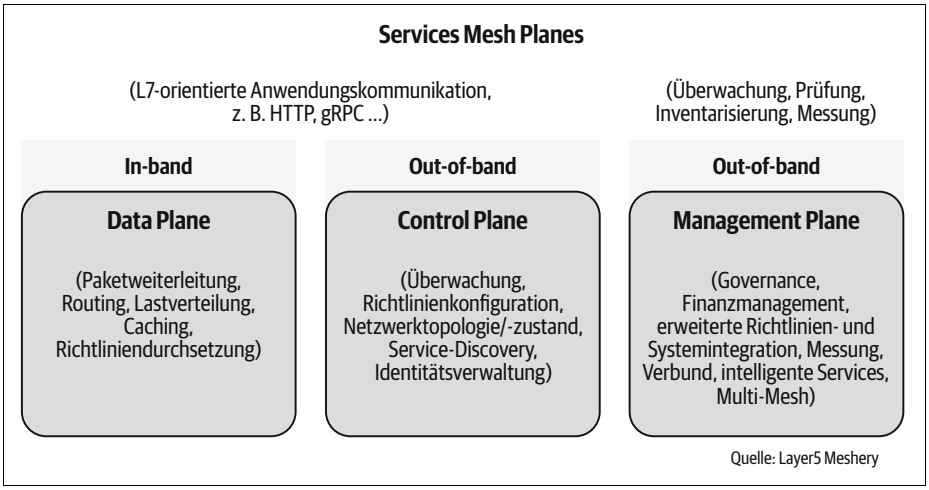

Abbildung 3-1: Istio und andere Service Meshes bestehen aus zwei Ebenen. Häufig wird noch mit einer dritten Ebene zusätzliche Netzwerkintelligenz realisiert und die Verwaltung in heterogenen Umgebungen erleichtert.

Hauptkomponenten

Die Data Plane von Istio fängt jedes Paket in der Anforderung ab und ist verantwortlich für Integritätsprüfung, Routing, Lastverteilung, Authentifizierung, Autorisierung und die Erzeugung von beobachtbaren Signalen. Beim In-band-Betrieb werden Service Proxys transparent eingefügt. Und da Anwendungen Aufrufe von Dienst zu Dienst ausführen, bekommen sie gar nichts davon mit, dass eine Data Plane existiert. Data Planes sind sowohl für clusterinterne Kommunikation als auch für eingehenden (*Ingress*) und ausgehenden (*Egress*) Cluster-Netzwerkverkehr zuständig. Der Service-Datenverkehr der Anwendung – sowohl der in das Mesh eingehende als auch der aus dem Mesh ausgehende – wird zuerst an den Service Proxy zur Bearbeitung geleitet. Bei Istio wird der Verkehr transparent mithilfe von *iptables*-Regeln abgefangen und an den Service Proxy weitergeleitet. Die Istio-Data-Plane berührt jedes Paket/jede Anforderung im System und ist für Service Discovery, Integritätsprüfung, Routing, Lastausgleich, Authentifizierung, Autorisierung und Beobachtbarkeit verantwortlich.

Die Control Plane in Istio bietet einen einzelnen Verwaltungspunkt für Service Proxys, die eine Konfiguration per Programm benötigen, um sie effizient verwalten und ihre Konfiguration in Echtzeit aktualisieren zu können, wenn Services in Ihrer Umgebung (d.h. Container-Cluster) neu geplant werden. Control Planes bieten Richtlinien und Konfiguration für Services im Mesh, wobei eine Reihe isolierter, zustandsloser Proxys in ein Service Mesh umgewandelt wird. Sie berühren keine Netzwerkpakete im Mesh direkt; Control Planes operieren out-of-band. In der Re-

gel verfügen sie über eine Kommandozeile (*Command Line Interface*, CLI) und eine Benutzeroberfläche (*User Interface*, UI), die eine Interaktion ermöglichen. Beide Schnittstellen bieten Zugriff auf eine zentralisierte API für die ganzheitliche Steuerung des Proxy-Verhaltens. Änderungen an der Konfiguration der Control Plane können Sie über ihre APIs automatisieren (z.B. durch eine CI/CD-Pipeline), wobei die Konfiguration in der Praxis meistens über die Versionsverwaltung gesteuert und aktualisiert wird.

Die Istio-Control-Plane realisiert Folgendes:

- Bietet Richtlinien und Konfiguration für Services im Mesh über APIs für Betreiber, um gewünschtes Routing/Stabilitätsverhalten festzulegen.
- Fasst eine Menge isolierter, zustandsloser Sidecar Proxys in einem Service Mesh zusammen:
  - APIs für die Data Plane, um lokalisierte Konfiguration zu konsumieren.
  - Abstraktion der Service Discovery für die Data Plane.
- Verwendet APIs, um Nutzungsrichtlinien über Kontingente und Nutzungsbeschränkungen festzulegen.
- Bietet Sicherheit über Ausstellung und Rotation von Zertifikaten.
- Weist Workload-Identität zu.
- Bearbeitet die Routing-Konfiguration:
  - Fasst keinerlei Pakete/Anforderungen im System an.
  - Spezifiziert die Netzwerkgrenzen und wie auf sie zugegriffen wird.
  - Vereinheitlicht die Erfassung von Telemetriedaten.

Komponenten der Istio-Control-Plane

In diesem Abschnitt stellen wir die Funktionalität der einzelnen Komponenten der Control Plane im Überblick vor. Spätere Kapitel tauchen tiefer in das Verhalten, die Konfiguration und die Fehlerbehebung jeder Komponente ein.

Pilot

Pilot ist sozusagen der Kopf des Schiffs in einem Istio-Mesh. Er bleibt mit der zugrunde liegenden Plattform (z.B. Kubernetes) synchronisiert, indem Zustand und Ort der laufenden Services verfolgt und für die Data Plane dargestellt werden. Pilot bildet eine Schnittstelle zum Service-Discovery-System Ihrer Umgebung und erzeugt Konfigurationsdaten für die Service Proxys der Data Plane (auf `istio-proxy` als Komponente der Data Plane gehen wir später noch im Detail ein).

Mit der Weiterentwicklung von Istio wird sich Pilot mehr auf die skalierbare Übermittlung der Proxy-Konfiguration und weniger auf die Schnittstelle mit zugrunde

liegenden Plattformen konzentrieren. Pilot bedient Envoy-kompatible Konfigurationen, indem er Konfigurations- und Endpunktinformationen aus verschiedenen Quellen zusammenführt und in xDS-Objekte übersetzt. Eine andere Komponente, Galley, übernimmt schließlich die Verantwortung für die direkte Schnittstellenarbeit mit zugrunde liegenden Plattformen.

Galley

Galley ist die Istio-Komponente für die Aggregation und Verteilung von Konfigurationen. Mit der Weiterentwicklung ihrer Rolle wird diese Komponente die anderen Istio-Komponenten von den Konfigurationen der zugrunde liegenden Plattform und den vom Benutzer bereitgestellten Konfigurationsdaten isolieren, indem sie Konfigurationen aufnimmt und validiert. Galley verwendet das *Mesh Configuration Protocol* (MCP) als Mechanismus, um Konfigurationen zu bedienen und zu verteilen.

Mixer

Der *Mixer* ist eine eigenständige Komponente der Control Plane, die Infrastruktur-Backends konzeptionell vom Rest von Istio abstrahieren soll, wobei unter Infrastruktur-Backends solche Dinge wie StackDriver oder New Relic zu verstehen sind. Die Mixer-Komponente ist dafür verantwortlich, Vorbedingungen zu überprüfen, Kontingente zu verwalten und Telemetriedaten zu senden. Im Einzelnen geht es um Folgendes:

- Ermöglicht die Mobilität der Plattform und der Umgebung.
- Bietet granulare Kontrolle über betriebliche Richtlinien und Telemetrie, indem die Verantwortung übernommen wird, Richtlinien auszuwerten und Telemetriedaten zu senden.
- Realisiert ein umfangreiches Konfigurationsmodell.
- Abstrahiert die meisten Infrastrukturbelange durch absichtsbasierte Konfiguration.

Service Proxys und -Gateways rufen Mixer auf, um Vorbedingungen zu überprüfen und um zu bestimmen, ob eine Anforderung weiterverarbeitet werden darf (Check), ob die Kommunikation zwischen dem Aufrufer und dem Service zulässig ist oder Kontingente überschritten hat, und um Telemetriedaten zu senden, nachdem eine Anforderung fertig verarbeitet wurde (Report). Mixer kommuniziert mit Infrastruktur-Backends über einen Satz von nativen und Drittanbieteradaptern. Die Adapterkonfiguration bestimmt, welche Telemetriedaten wann an welches Backend gesendet werden. Die Service-Mesh-Betreiber können die Mixer-Adapter als Integrationspunkt und Vermittlung mit ihren Infrastruktur-Backends verwenden, da sie als Engine zum Verarbeiten und Routen von Attributen arbeiten.

Das derzeit angelaufene Mixer-v2-Design schlägt eine deutlich andere Architektur vor. Umfang und Schwerpunkt sollen jedoch weitgehend dem Design von Mixer v1 gleichen.

Citadel

Mit *Citadel* ist Istio in der Lage, starke Service-to-Service- und Endbenutzerauthentifizierung per mTLS (*mutual Transport Layer Security*) mit integrierter Verwaltung von Identitäten und Anmeldeinformationen zu realisieren. Die CA[1]-Komponente von Citadel genehmigt und signiert Zertifikatsignieranforderungen (*Certificate-Signing Requests*, CSRs), die von Citadel-Agenten gesendet werden, und übernimmt es, Schlüssel und Zertifikate zu generieren, bereitzustellen, zu rotieren und zurückzuziehen. Citadel ist optional in der Lage, während des CA-Prozesses mit einem Identitätsverzeichnis zu interagieren.

Citadel besitzt eine Plug-in-Architektur, in der sich verschiedene CAs verwenden lassen, sodass es keine selbst generierten und selbst signierten Schlüssel und Zertifikate verwendet, um Workload-Zertifikate zu signieren. Die CA-Austauschbarkeit von Istio erlaubt und erleichtert Folgendes:

- Integration mit dem PKI[2]-System Ihrer Organisation.
- Sichere Kommunikation zwischen Istio und Nicht-Istio-Legacy-Services (durch gemeinsame Nutzung derselben Root-of-Trust).
- Speichert den CA-Signierungsschlüssel in einer geschützten Umgebung (z. B. HashiCorp Vault, HSM, *Hardware Security Module*).

Service Proxy

Mithilfe von Service-Mesh-Proxys können Sie den eingehenden Netzwerkverkehr, den Verkehr zwischen den Services und den von Services ausgehenden Verkehr steuern. Istio verwendet Proxys zwischen Services und Clients. Service Proxys werden üblicherweise als Sidecars in Pods bereitgestellt. (Beispiele für andere Deployment-Modelle finden Sie beispielsweise im Buch *The Enterprise Path to Service Mesh Architectures* (*https://oreil.ly/Up2H7*). Die Proxy-zu-Proxy-Kommunikation ist das, was das Mesh wirklich ausmacht. Daraus folgt, dass ein Proxy zwischen der Anwendung und dem Netzwerk platziert werden muss, damit eine Anwendung im Mesh präsent sein kann (siehe Abbildung 3-2).

1 CA – *Certificate Authority* (Zertifizierungsstelle)

2 PKI – *Public Key Infrastructure* (System, das digitale Zertifikate ausstellen, verteilen und prüfen kann. Arbeitet mit öffentlichen Schlüsseln.)

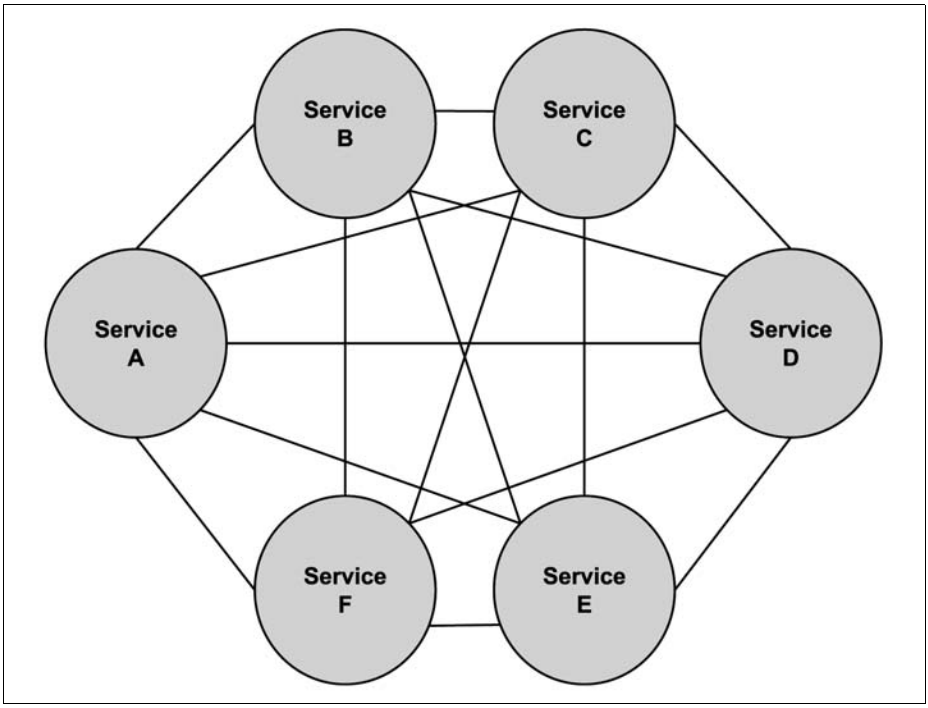

Abbildung 3-2: Vollständig miteinander verbundene Service Proxys bilden das Mesh.

Ein Sidecar fügt einem Container Verhalten hinzu, ohne ihn zu verändern. In diesem Sinne verhalten sich das Sidecar und der Service als einzelne erweiterte Einheit. Die Pods hosten das Sidecar und den Service als einzelne Einheit.

Komponenten der Istio-Data-Plane

Istio verwendet eine erweiterte Version des Envoy-Proxys, eines in C++ entwickelten Hochleistungsproxys, um den gesamten eingehenden und ausgehenden Datenverkehr im Service Mesh zu vermitteln. Dabei greift Istio auf Features von Envoy zurück, wie zum Beispiel dynamische Service Discovery, Lastverteilung, TLS-Terminierung, HTTP/2 und gRPC (*google Remote Procedure Calls*) Proxying, Circuit Breaker, Integritätschecks, gestaffelte Roll-outs mit prozentualer Verkehrsaufteilung, Fehlerinjektion und umfangreiche Metriken.

Envoy wird als Sidecar für den entsprechenden Service im selben Kubernetes-Pod bereitgestellt. Somit kann Istio eine Fülle von Signalen über das Verkehrsverhalten als Attribute extrahieren, die wiederum in Mixer verwendbar sind, um Richtlinienentscheidungen durchzusetzen, und die sich an Überwachungssysteme senden lassen, um Informationen über das Verhalten des gesamten Mesh bereitzustellen.

Injection

Das Sidecar-Proxy-Modell ermöglicht Ihnen auch, Istio-Funktionen einer vorhandenen Bereitstellung hinzuzufügen, ohne das Design verändern oder Code umschreiben zu müssen. Dies ist ein wichtiger Anreiz, Istio zu verwenden. Istio verspricht eine unmittelbare Ansicht von Top-Level-Service-Metriken, eine detaillierte Kontrolle über den Datenverkehr und eine automatisierte Authentifizierung und Verschlüsselung zwischen allen Services, ohne dass Sie Ihren Anwendungscode oder Ihre Deployment-Manifeste ändern müssen.

Die kanonische Beispielanwendung Bookinfo von Istio macht deutlich, wie Service Proxys ins Spiel kommen und ein Mesh bilden. Abbildung 3-3 zeigt die Anwendung Bookinfo ohne die Service Proxys. (In Kapitel 4 sehen wir uns Bookinfo näher an und stellen die Anwendung bereit.)

In Kubernetes ist die automatische Proxy Injection als Webhook über einen Kubernetes-API-Server mit dem Mutating Webhook Admission Controller implementiert. Sie ist zustandslos und hängt nur vom Injection-Template und den Mesh-Konfigurations-ConfigMaps sowie vom zu injizierenden Pod-Objekt ab. Somit lässt sie sich leicht horizontal skalieren, und zwar entweder über das Deployment-Objekt oder automatisch über einen *Horizontal Pod Autoscaler* (HPA).

Bei der Injection des Sidecar-Proxys in einen neu erzeugten Pod dauert es im Mittel 1,5 s (per Microbenchmark), um den Webhook an sich auszuführen. Die gesamte Injektionszeit liegt höher, wenn man die Netzwerklatenz und die Verarbeitungszeit des API-Servers berücksichtigt.

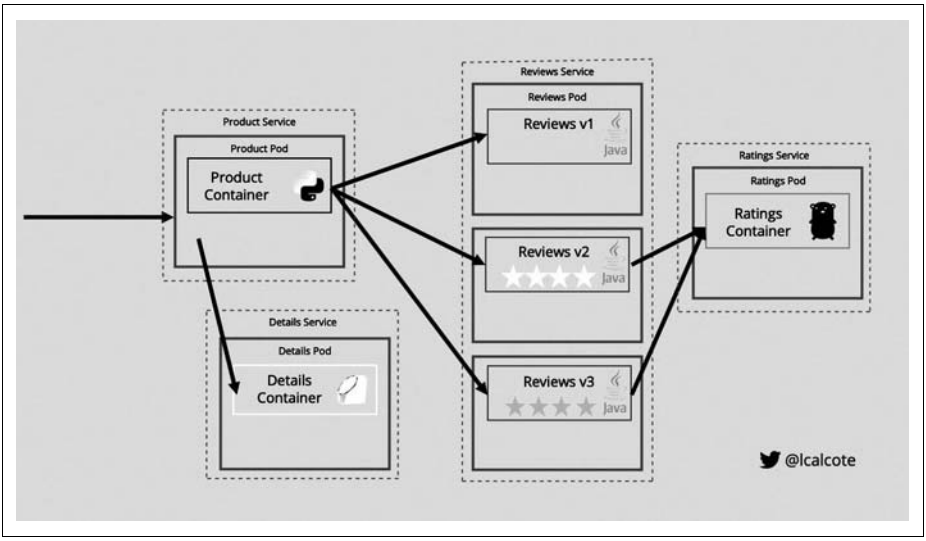

Abbildung 3-3: Die Beispielanwendung Bookinfo von Istio, hier ohne Service Proxys dargestellt

Istio stellt sich der allbekannten Herausforderung verteilter Systeme, die keine homogenen, zuverlässigen und unveränderlichen Netzwerke haben. Dies geschieht über die Bereitstellung von schlanken Proxys zwischen Ihren Anwendungscontainern und dem Netzwerk. Abbildung 3-4 veranschaulicht, wie die vollständige Architektur von Istio die Control und Data Planes jeweils mit ihren internen Komponenten einbindet. Die Bereitstellung eines vollständigen Mesh umfasst auch die Gateways für eingehenden und ausgehenden Service-Datenverkehr.

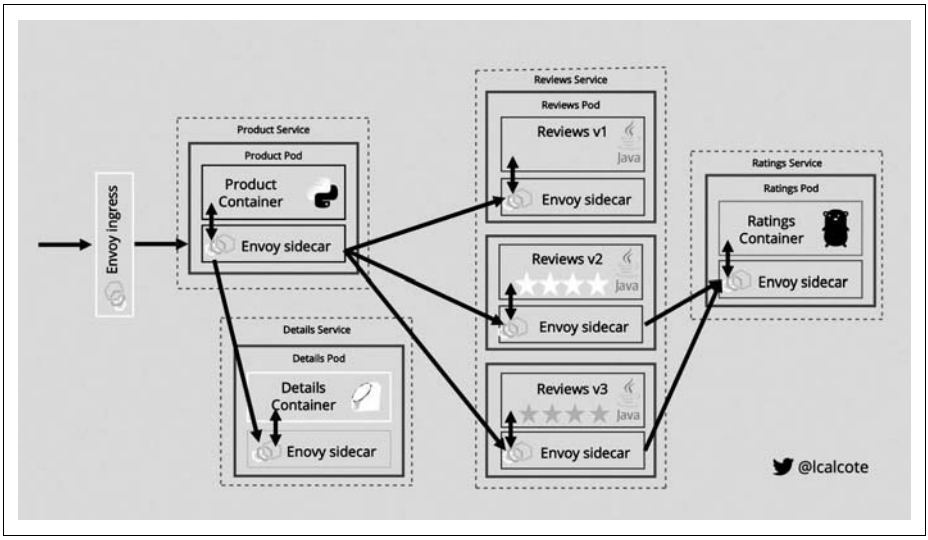

Abbildung 3-4: Bookinfo, mit Service Proxys dargestellt

Gateways

Istio 0.8 hat das Konzept von Ingress- und Egress-Gateways eingeführt. Die symmetrisch ähnlichen Ingress- und Egress-Gateways fungieren als Reverse und Forward Proxys für Datenverkehr, der in das Mesh einläuft bzw. das Mesh verlässt. Wie bei anderen Istio-Komponenten wird das Verhalten eines Istio-Gateways per Konfiguration definiert und gesteuert, sodass Sie die Kontrolle darüber haben, welcher Datenverkehr mit welcher Rate usw. in das Mesh einlaufen und das Mesh verlassen darf.

Ingress

Indem Sie Ingress-Gateways konfigurieren, können Sie für den Datenverkehr in das Service Mesh Zugangswege definieren, durch die der eingehende Verkehr fließt. Bedenken Sie, dass eingehender Datenverkehr in das Mesh eine Reverse-Proxy-Situation darstellt – ähnlich der Lastverteilung bei herkömmlichen Webservern. Die Konfiguration für ausgehenden Datenverkehr aus dem Mesh heraus ist eine Forward-Proxy-Situation, in der Sie identifizieren, welcher Verkehr das Mesh verlassen darf und wo er geroutet werden sollte.

Als Beispiel richtet die folgende Gateway-Konfiguration einen Proxy als Lastverteiler ein, der die Ports 80 und 9080 (HTTP), 443 (HTTPS) und 2379 (TCP) für Ingress zugänglich macht. Das Gateway wird dem Proxy zugeordnet, der auf einem Pod mit den Labels `app: my-gateway-controller` läuft. Selbst wenn Istio den Proxy so konfiguriert, dass er auf diesen Ports lauscht, muss der Benutzer sicherstellen, dass externer Datenverkehr zu diesen Ports in das Mesh zugelassen ist (weitere Einzelheiten finden Sie in der Gateway-Dokumentation von Istio unter *https://oreil.ly/e5pIQ*):

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: my-gateway
spec:
  selector:
    app: my-gateway-controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      httpsRedirect: true # sends 301 redirect for http requests
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - uk.bookinfo.com
    - eu.bookinfo.com
    tls:
      mode: SIMPLE #enables HTTPS on this port
      serverCertificate: /etc/certs/servercert.pem
      privateKey: /etc/certs/privatekey.pem
  - port:
      number: 9080
      name: http-wildcard
      protocol: HTTP
    hosts:
    - "*"
  - port:
      number: 2379 # to expose internal service via external port 2379
      name: mongo
      protocol: MONGO
    hosts:
    - "*"
```

Egress

Der Verkehr kann ein Istio-Service-Mesh auf zwei Arten verlassen: direkt vom Sidecar oder konzentriert über ein Egress-Gateway, auf das Sie Datenverkehrsrichtlinien anwenden können.

Standardmäßig sind Istio-fähige Anwendungen nicht in der Lage, auf URLs außerhalb des Clusters zuzugreifen.

Direkt von einem Service Proxy

Wenn der Datenverkehr, der für eine externe Quelle bestimmt ist, das Egress-Gateway umgehen soll, können Sie die ConfigMap für `istio-sidecar-injector` entsprechend konfigurieren. Mit der folgenden Konfiguration im Sidecar Injector identifizieren Sie die clusterlokalen Netzwerke und halten den Verkehr mit lokalem Ziel innerhalb des Mesh, während der Verkehr für alle anderen Ziele nach außen weitergeleitet wird:

```
--set global.proxy.includeIPRanges="10.0.0.1/24"
```

Nachdem Sie das eingerichtet haben und die Istio-Proxys mit dieser Konfiguration aktualisiert worden sind, umgehen externe Anforderungen das Sidecar und werden direkt zum vorgesehenen Ziel weitergeleitet. Das Istio-Sidecar fängt nur interne Anforderungen innerhalb des Clusters ab und verwaltet sie.

Durch ein Egress-Gateway weiterleiten

Gegebenenfalls müssen Sie ein Egress-Gateway verwenden, um die Konnektivität aus dem privaten IP-Adressraum Ihres Clusters, ein Monitoring oder clusterübergreifende Konnektivität zu gewährleisten.

Durch ein Egress-Gateway ist es Istio möglich, den Verkehr, der das Mesh verlässt, zu überwachen und Weiterleitungsregeln darauf anzuwenden. Außerdem erleichtert es die Kommunikation zwischen Anwendungen, die in einem Cluster laufen, in dem die Knoten keine öffentlichen IP-Adressen besitzen, sodass die Anwendungen im Mesh keinen Zugang zum Internet haben. Wenn man ein Egress-Gateway definiert, den gesamten ausgehenden Verkehr durch dieses Gateway leitet und den Knoten des Egress-Gateways öffentliche IP-Adressen zuordnet, können die Knoten (und die auf ihnen laufenden Anwendungen) kontrolliert auf externe Services zugreifen, wie Abbildung 3-5 veranschaulicht.

Weshalb Istio-Gateways und keine Kubernetes-Ingresses verwenden?

Im Allgemeinen verwenden die Istio-v1alpha3-APIs Gateways wegen einer umfangreicheren Funktionalität, da sich Kubernetes-Ingress für Istio-Anwendungen als unzureichend erwiesen hat. Im Vergleich zu Kubernetes-Ingress können Istio-Gateways als reine L4-TCP-Proxys arbeiten und unterstützen alle Protokolle, die Envoy unterstützt.

Des Weiteren ist die Trennung der Vertrauensdomänen zwischen Organisationsteams zu berücksichtigen. Die Kubernetes-Ingress-API bringt die Spezifikationen für L4 bis Layer 7 (L7) zusammen, wodurch es für verschiedene Teams in Organisationen mit separaten Vertrauensdomänen (wie SecOps, NetOps, ClusterOps und Developers) schwierig ist, eine eigene Verwaltung für eingehenden Verkehr zu realisieren.

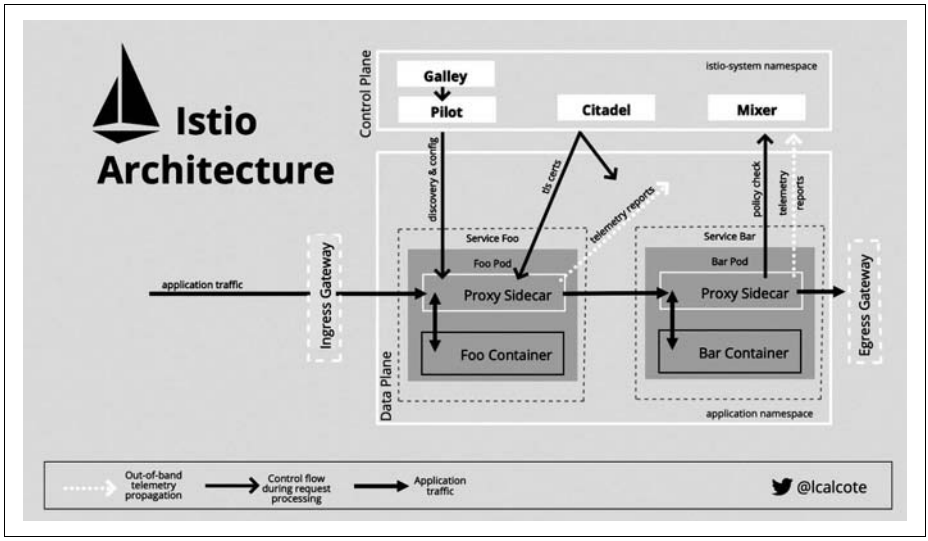

Abbildung 3-5: Architektur und Komponenten von Istio

Erweiterbarkeit

Obwohl es für andere Service Meshes nicht als explizites Ziel formuliert ist – Istio ist so konzipiert, dass es sich anpassen lässt. Als erweiterbare Plattform gibt es hauptsächlich zwei Integrationsformen: austauschbare Sidecar-Proxys und Adapter für Telemetrie/Autorisierung.

Anpassbare Sidecars

Auch wenn Envoy das standardmäßige Proxy-Sidecar ist, lässt sich innerhalb von Istio ein anderer Service Proxy für Ihr Sidecar verwenden. Es gibt zwar mehrere Service Proxys im Ökosystem, doch haben derzeit neben Envoy nur zwei die Integration mit Istio nachgewiesen: Linkerd und NGINX. Linkerd2 ist momentan nicht als Allzweckproxy konzipiert, sondern soll vor allem schlank sein und erst in zweiter Linie erweiterbar, indem eine Erweiterung über das gRPC-Plug-in geboten wird.

Obwohl Sie sich wahrscheinlich eher dafür entscheiden würden, ein ganz anderes Service Mesh auszuführen, kommt auch Istio mit einem der folgenden alternativen Service Proxys infrage:

Linkerd
: Diese Konfiguration können Sie verwenden, wenn Sie Linkerd bereits ausführen und Istio-Steuerungs-APIs wie `CheckRequest` übernehmen möchten, die Ihnen vor der Ausführung einer Aktion grünes oder rotes Licht geben.

NGINX
: Auf der Grundlage Ihrer operativen Erfahrung und dem Bedarf an kampferprobten Proxys könnten Sie sich für NGINX entscheiden. Vielleicht sind Sie

interessiert an Caching, einer Firewall für Webanwendungen oder anderen Funktionen, die zudem in NGINX Plus verfügbar sind.

Consul Connect
: Dieser Proxy bietet sich an, weil er sich leicht bereitstellen lässt und recht anspruchslos ist.

Die Einführung der Wahlmöglichkeit von Service Proxys für Istio hat für viel Wirbel gesorgt. Die Integration von Linkerd wurde bereits in Release 0.1.6 von Istio verwirklicht. In ähnlicher Weise wurde die Möglichkeit, NGINX als Service Proxy über das *nginMesh*-Projekt (*https://oreil.ly/axVOR*) zu verwenden, recht früh im Release-Zyklus von Istio vorgesehen.

Obwohl nginMesh nicht mehr aktiv entwickelt wird, dürfte der Artikel *How to Customize an Istio Service Mesh* (*https://oreil.ly/p72P8*) mit dem entsprechenden *Webcast* (*https://oreil.ly/KHPOG*) hilfreich sein, um die Erweiterbarkeit von Istio in Bezug auf austauschbare Service Proxys besser zu verstehen.

Ohne Konfiguration fehlen den Proxys die Anweisungen, um ihre Aufgaben durchzuführen. Pilot steht sozusagen auf der Brücke des Schiffs in einem Istio-Mesh und hält die Synchronisierung mit der zugrunde liegenden Plattform aufrecht, indem ihre Services verfolgt und für `istio-proxy` dargestellt werden. Als Standard-Proxy enthält `istio-proxy` eine erweiterte Version von Envoy. Typischerweise wird dasselbe `istio-proxy`-Docker-Image vom Sidecar sowie den Ingress- und Egress-Gateways von Istio verwendet. In `istio-proxy` ist nicht nur der Service Proxy enthalten, sondern auch der Istio-Pilot-Agent, der die Konfiguration in kurzen Abständen von Pilot auf den Service Proxy herunterzieht, sodass jeder Proxy weiß, wohin der Datenverkehr weiterzuleiten ist. In diesem Fall führt der Übersetzungsagent von nginMesh das Konfigurieren von NGINX als `istio-proxy` durch. Pilot ist für den Lebenszyklus von `istio-proxy` verantwortlich.

Erweiterbare Adapter

Die Mixer-Komponente für die Control Plane von Istio ist dafür zuständig, die Zugriffssteuerung und die Nutzungsrichtlinien im gesamten Mesh durchzusetzen und Telemetriedaten vom Sidecar-Proxy zu sammeln. Als wichtigster Punkt der Erweiterbarkeit von Istio kategorisiert Mixer die Adapter nach dem Datentyp, den sie verarbeiten, wie Abbildung 3-6 veranschaulicht.

Das prozessinterne Adapter-Authoring-Modell für Mixer-Adapter ist in Istio jetzt ein veraltetes Konzept. Wie andere Open-Source-Projekte davor begann Istio, die Adapter bequem baumartig zu integrieren. Mit der Weiterentwicklung und Reifung von Istio wurde dieses Modell in ein Modell umgewandelt, das Adapter getrennt vom Hauptprojekt hält, um so die Last von den Teams des Kernprojekts zu nehmen und die Eigenverantwortung der normalerweise getrennten Entwicklungs-

teams zu fördern, die eine Integration mit anderen Backend-Systemen geschaffen haben.

Die zukünftige Erweiterbarkeit könnte in Form von sicheren Schlüsselspeichern für HSMs und besserer Unterstützung für den Austausch von Infrastruktur-Backends für verteiltes Tracing erfolgen. Darüber hinaus erwarten wir, dass Verwaltungsebenen eine größere Rolle spielen, wenn sich Istio und andere Service Meshes durchsetzen.

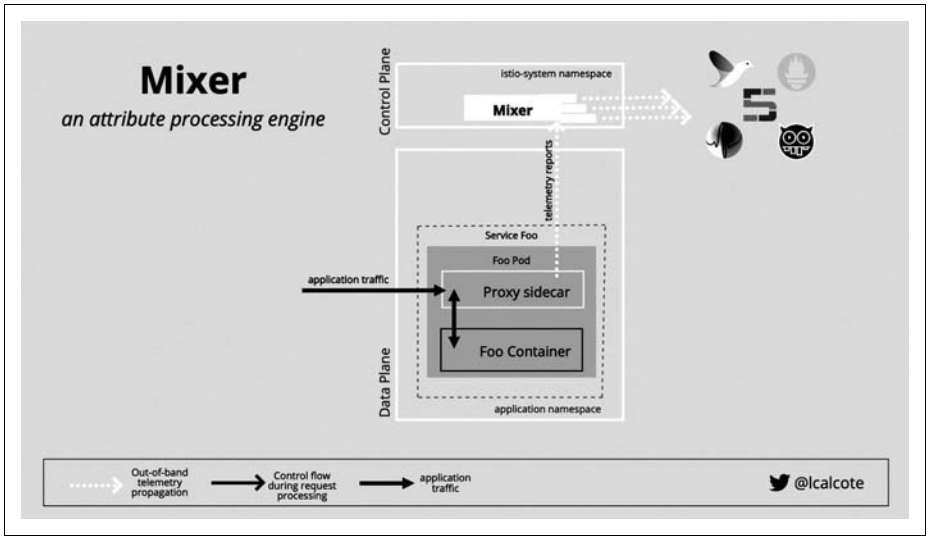

Abbildung 3-6: Mixer fungiert als Engine für die Attributverarbeitung, die Telemetriedaten sammelt, transformiert und überträgt.

Skalierung und Performance

Wie viele andere könnten Sie sich fragen: »Diese Features sind zwar großartig, doch wie sieht es mit dem Overhead aus, ein Service Mesh zu betreiben?« Es stimmt, dass diese Features nicht umsonst zu haben sind. Einen Proxy pro Service auszuführen und jedes Paket abzufangen, beansprucht eine bestimmte Menge an ständigem Overhead. Die Kosten lassen sich nach Data und Control Plane analysieren. Und Antworten auf Performancefragen beginnen immer mit »es kommt darauf an« oder »je nachdem«. Zum Beispiel: Je nachdem, wie viele Features von Istio Sie verwenden, variieren die benötigten Ressourcen. So gibt es etwa folgende Kosten oder Ressourcen:

- Eine virtuelle CPU (vCPU) je Tausend Anforderungen pro Sekunde für das/die Sidecar(s) mit Zugriffsprotokollierung (die in v1.1 standardmäßig ausgeschaltet ist) und 0,5 ohne. Fluentd auf dem Knoten trägt erheblich zu diesen Kosten bei, da es Protokolle erfasst und hochlädt.

- Unter der Annahme einer typischen Cache-Trefferquote (>80%) für Mixer-Überprüfungen: 0,5 vCPU je Tausend Anforderungen pro Sekunde für die Mixer-Pods.
- Eine Latenz von etwa 8 ms wird zur 90% Perzentil Latenz addiert.
- Die mTLS[3]-Kosten sind auf AES-NI-fähiger Hardware sowohl in Bezug auf die CPU als auch in Bezug auf die Latenz vernachlässigbar.

Der Overhead sowohl für die Data Plane als auch für die Control Plane ist ein allgemeines Problem für diejenigen, die ein Service Mesh einsetzen. Was die Data Plane angeht, verstehen die Betreiber von Envoy, dass dieser Service Proxy im kritischen Pfad liegt, und haben alles darangesetzt, seine Performance zu optimieren. Envoy bringt erstklassige Unterstützung für HTTP/2 und TLS in beide Richtungen mit und minimiert den Overhead durch Multiplexing von Anforderungen und Antworten über einzelne langlebige TCP-Verbindungen. Auf diese Weise erreicht seine Unterstützung von HTTP/2 eine geringere Latenz, als sie mit HTTP/1.1 möglich wäre.

Obwohl die Envoy-Projektverantwortlichen derzeit keine offiziellen Performance-Benchmarks veröffentlichen, ermutigen sie Benutzer, Benchmarks in ihren eigenen Umgebungen durchzuführen, und zwar mit einer Konfiguration ähnlich der, die sie für die Produktion planen. Um diese Lücke zu füllen, hat die Open-Source-Gemeinde Tools wie *Meshery* (*https://oreil.ly/MNUQC*) geschaffen. Meshery ist eine Open-Source-Multiservice-Mesh-Verwaltungsebene, die für verschiedene Service Meshes und Beispielanwendungen ausgelegt ist und Benchmarks für die Performance von Service-Mesh-Deployments liefert. Damit ist es leicht, Benchmarks für verschiedene Konfigurationsszenarien von Istio zu erzeugen und die Performance von Services (Anwendungen) innerhalb und außerhalb des Mesh sowie Mesh-übergreifend zu vergleichen. Es durchleuchtet Mesh- und Service-Konfiguration und vergleicht sie mit Deployment-Empfehlungen. Manche Service-Mesh-Projekte verwenden Meshery als Benchmark-Tool für Releases. Es wird ergänzt durch andere Tools zur Lastgenerierung (*https://oreil.ly/pZxU9*), die häufig für das Testen der Service-Mesh-Performance herangezogen werden.

Wenn in größeren Deployments die Flotte der Service Proxys (Envoy) wächst, kann die zentrale Rolle, die die Control Plane spielt, zum Engpass oder zur Ursache von Latenz werden. So können beispielsweise die Trace-Daten je nach Ausführlichkeit der Instrumentierung und der Sampling-Rate das Volumen des eigentlichen Geschäftsverkehrs, das eine Anwendung dauerhaft unterstützt, übersteigen. Diese Telemetriedaten zu sammeln und sie an das Tracing-Backend zu senden (direkt oder über die Control Plane), kann sich tatsächlich auf die Latenz und den Durchsatz der Anwendung auswirken.

3 mTLS – mutual TLS

Deployment-Modelle

Istio unterstützt verschiedene Deployment-Modelle, von denen einige nur ausgewählte Komponenten seiner Architektur bereitstellen. Abbildung 3-7 zeigt ein vollständiges Istio-Deployment in seiner ganzen Pracht.

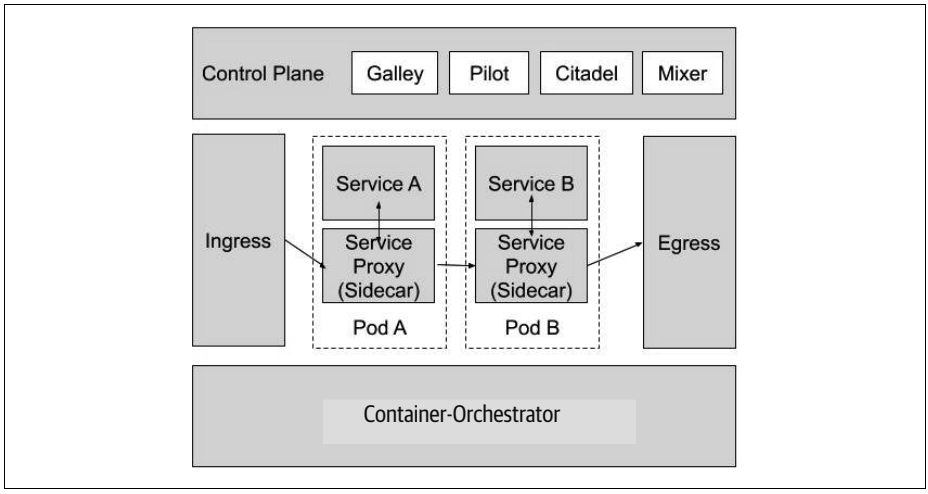

Abbildung 3-7: Istio-Deployment auf Kubernetes

Service Meshes gibt es in verschiedenen Formen und Größen. Um andere Modelle der Mesh-Bereitstellung zu erkunden, sollten Sie sich *The Enterprise Path to Service Mesh Architectures* (*https://oreil.ly/70pu7*) ansehen. Istio zielt darauf ab, die Herausforderungen der Service-Verwaltung direkt anzugehen, indem es eine neue Schicht der Sichtbarkeit, Sicherheit und Steuerung bietet.

Dieses Kapitel hat erläutert, wie die Istio-Control-Plane einen einzelnen Punkt der Sichtbarkeit und Kontrolle bietet, während die Data Plane die Weiterleitung des Datenverkehrs erleichtert. Außerdem ist Istio ein Beispiel für ein Service Mesh, das konzeptionell auf Anpassbarkeit ausgelegt ist. Schließlich haben Sie gelernt, wie die Entkopplung der Steuerung und Vereinheitlichung der Verantwortung über Microservices als neue Schicht der Infrastruktur – Layer 5 (L5) – gegenseitige Schuldzuweisungen zwischen Dev- und Ops-Team vermeidet.