9.1.1 两个平面的定义

在日常工作中，我们常见两个术语：数据平面和控制平面。两个平面的定义，最早见于高端路由器。顾名思义，数据平面负责数据的转发，控制平面负责执行路由选择协议。将两个平面分离是为了消除单点故障。例如：当数据平面的业务由于数据量过多而出现性能问题时，并不影响控制平面的路由策略；当控制平面由于路由策略负载过重时，也不会影响数据平面的转发。

随着 IT 的发展，两个平面的架构被广泛应用于软件定义网络和软件定义存储。Istio 作为新一代微服务治理框架，同样也分为数据平面和控制平面。如图 8-1 中 Proxy 为数据平面；Pilot、Galley 和 Citadel 组成了控制平面。我们接下来从两个平面的定义入手，揭开 Istio 架构的面纱。

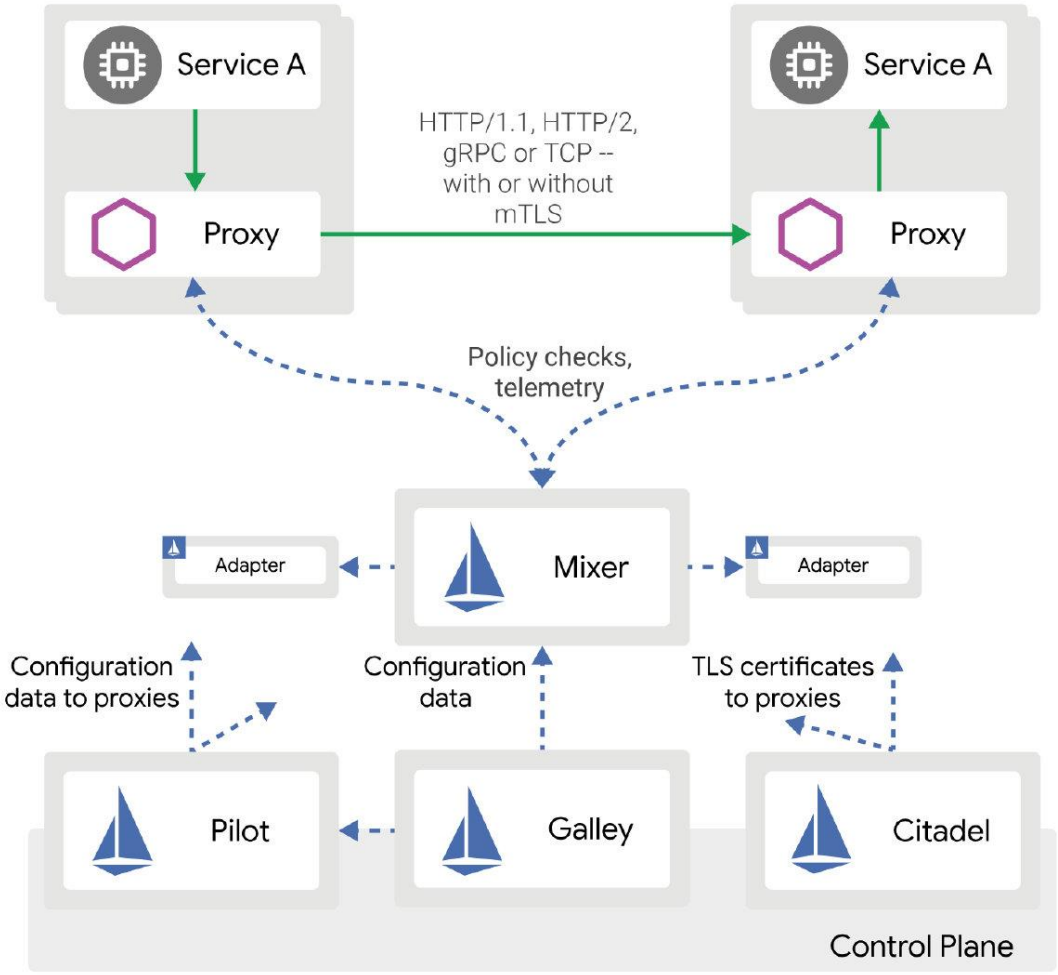


图 8-1 Istio 官方架构图

9.1.2 控制平面

Isito 控制平面主要负责管理和配置数据平面，控制数据平面的数据转发，如路由流量、转发策略、收集遥测数据、加密认证等。目前包含四个核心组件：Pilot、Citadel、Mixer 和 Galley。接下来我们对这四个组件进行讲解。

1. Pilot 解析

Pilot 是流量管理的核心组件，在 Istio 中承担的主要职责是向 Envoy proxy 提供服务发现，以及为高级路由（如：A / B 测试、金丝雀部署等）提供流量管理功能和异常控制（如：超时、重试、断路器等），如图 8-2 所示。

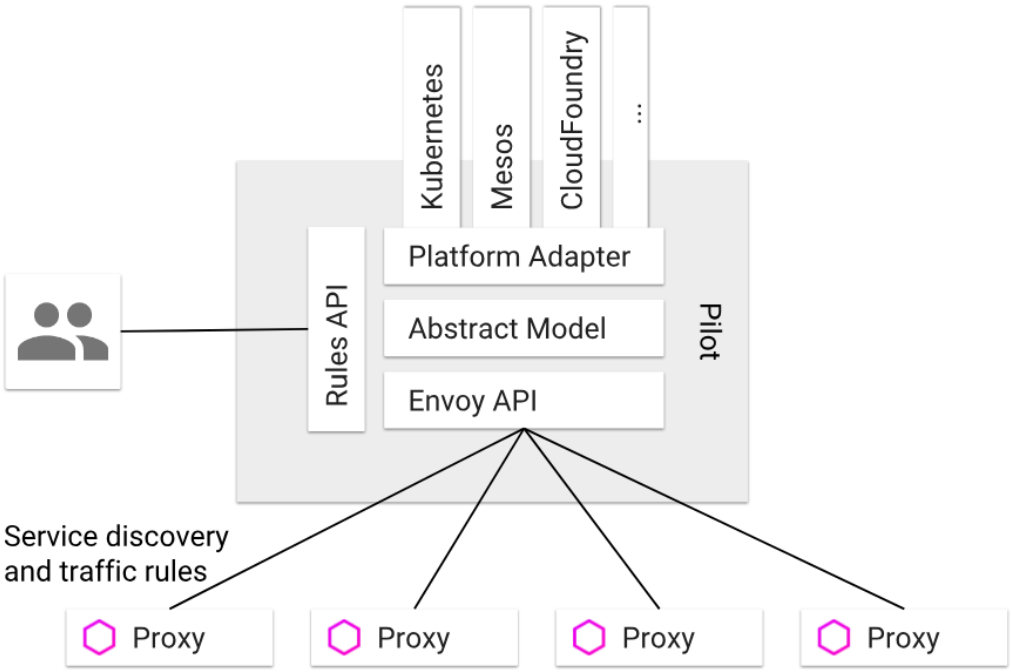


图 8-2 Pilot 内部组件架构图

从上图中可以看出，Pilot 内部主要分为 4 个组成部分：

- Platform Adapter：通过实现不同平台的适配器，满足对接不同的外部平台，目前支持 Kubernete、Mesos、CloudFoundry。Istio 本身不提供服务注册能力，而是通过 Adapter 来适配不同平台，来获取平台中服务注册表中的信息。OpenShift 基于 Kubernetes，所以 Platform Adapter 会与 OpenShift 进行适配，获取到 OpenShift 服务注册表中的信息。

- Abstract Model：抽象统一的服务模型，屏蔽跨平台差异性，为跨平台提供基础。
- Envoy API：负责和 Envoy 的通信, 主要是发送服务发现信息和流量控制规则给 Envoy。
- Rules API：提供对外管理规则的接口, 包括命令行工具 istioctl 以及管理界面 Kiali（Istio 微服务调用可观测工具，后文会介绍到）。

了解了 Pilot 的组成部分之后，Pilot 的工作流程也就不言而喻了。首先，Pilot 通过 Platform Adapter 从平台获取服务信息（在 Kubernetes/OpenShift 中为 Service）；然后，用户创建的控制流量行为的高级路由规则，转换为 Envoy 的规则配置，通过 Envoy API 将这些规则实时下发到 Envoy proxy 中实现服务之间的流量管理。

（1） 服务发现

如果在 OpenShift 上运行 Spring Cloud，往往需要一个独立的服务注册中心，如 Eureka，但这种模式完全未使用 PaaS 平台自身的服务注册能力。Istio 并未重新实现服务注册，而是使用平台本身的注册中心。

OpenShift 的服务注册中心为 Etcd。Isito 中的 Envoy 实例执行服务发现，通过 Pilot 的 Kubernetes Adapter 获取记录在 Etcd 中的服务信息，并相应地动态更新其负载均衡池，完成服务发现的过程，如图 8-3 所示。

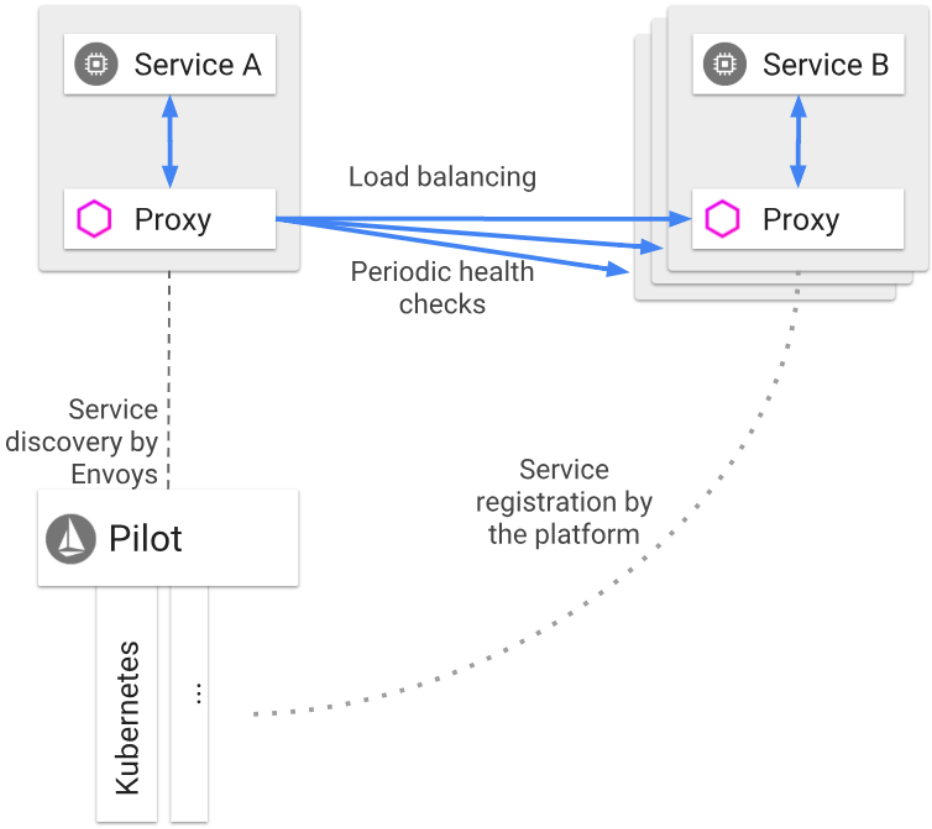


图 8-3 Pilot 与 Envoy 服务注册发现交互图

Istio 中的服务使用 Service 名称完成服务间调用，这就需要用到第二章中介绍的 OpenShift 内部 DNS 的解析。

（2） 路由控制

Pilot 还会读取 Istio 的各种策略配置，最终传递到 Envoy 进行路由控制。也就是说，用户在 OpenShift 集群上，通过 oc/istioctl 命令创建 Pilot 相关的 CRD 资源的方式进行配置变更时，Pilot 会监听 CRD 中的资源（CRD 是 Kubernetes 为提高可扩展性，开发者可以自定义资源），在检测到变更后，针对其中涉及的服务，生成对应的 Envoy proxy 配置文件，随后 Envoy 就根据这些配置信息对微服务的通信进行路由控制。如服务 A 调用服务 B，分配 1%的流量访问测试版本 version： v2.0-alpha 的服务，如图 8-4 所示。

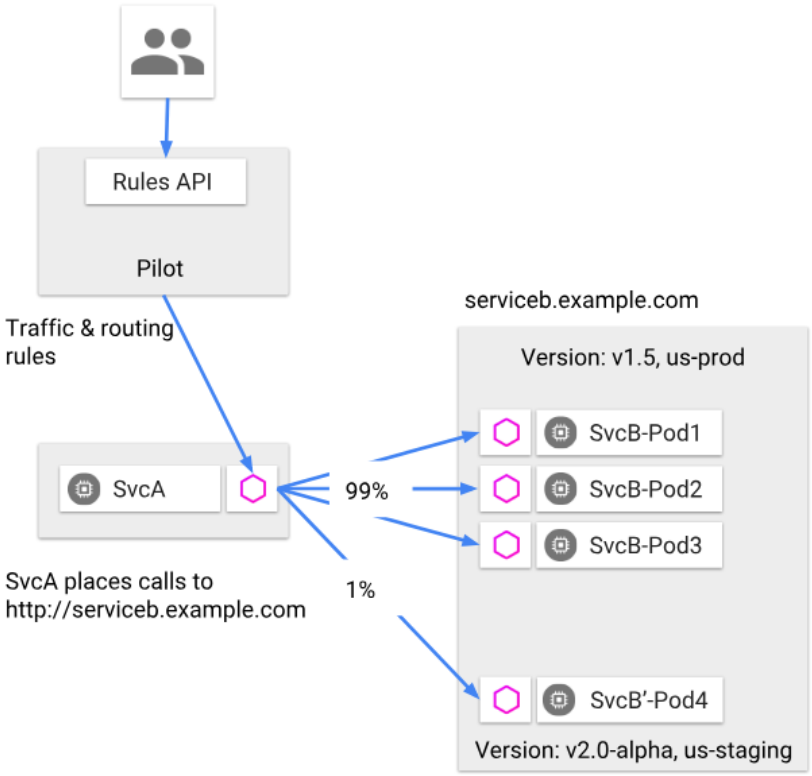


图 8-4 Pilot 实现服务路由控制示例

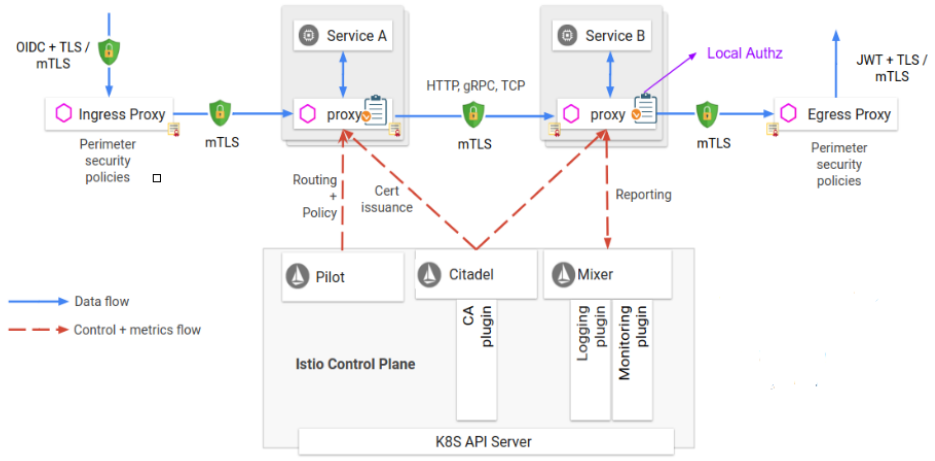
2. Citadel 解析

将单一应用程序分解为微服务可提供各种好处，如更好的灵活性、可伸缩性以及服务复用的能力。但是，在安全方面也带来了比单一应用程序更多的需求，毕竟不同服务间调用由单体架构中的方法调用变成了微服务间的远程调用。这些新增的安全需求包括：

- 加密：为了抵御中间人攻击、不泄露信息，需要对服务间通信进行加密。
- 访问控制：需要提供灵活的访问控制，如双向 TLS 和细粒度的访问策略。
- 审计：提供审计功能，审核系统中用户做了什么。

这些安全需求适用于所有的微服务体系，大部分微服务框架需要自己实现服务通信的安全和加密，且安全加密耦合在业务代码中。但是在 Istio 中，通过多个组件的配合，在应用程序无感知的情况下，为现有应用提供微服务之间以及微服务和最终用户身份验证和加密。

Istio 的安全架构如图 8-5 所示：



8-5 istio 安全架构图

如图 8-5，实现微服务安全，需要 Citadel 和其他多个组件配合实现：

- Citadel：用于密钥管理和证书管理，下发到 Envoy 等负责通信转发的组件。
- Envoy：使用从 Citadel 下发而来的密钥和证书，实现服务间通信的安全。
- Pilot：将授权策略和安全命名信息分发给 Envoy。
- Mixer：负责管理授权，完成审计等。

（1） 身份认证

istio 提供了两种类型的身份认证：

- 传输身份认证：也称为服务到服务的身份认证。Istio 提供双向 TLS 作为传输身份认证的解决方案。
- 来源身份认证：也称为最终用户身份认证，用于验证请求的最终用户或设备。常用的是 JWT（JSON Web Token）验证。

在 Istio 启用微服务认证以后，Citadel 负责为 OpenShift 集群中的每个服务账户（如 Kubernetes serviceaccount）创建证书和密钥对，并颁发给各个微服务中的 Envoy proxy，然后微服务之间的 TLS 会依赖这些证书完成加密和认证。需要注意的是，认证是对服务受到的请求生效的。

认证架构图如图 8-6 所示：

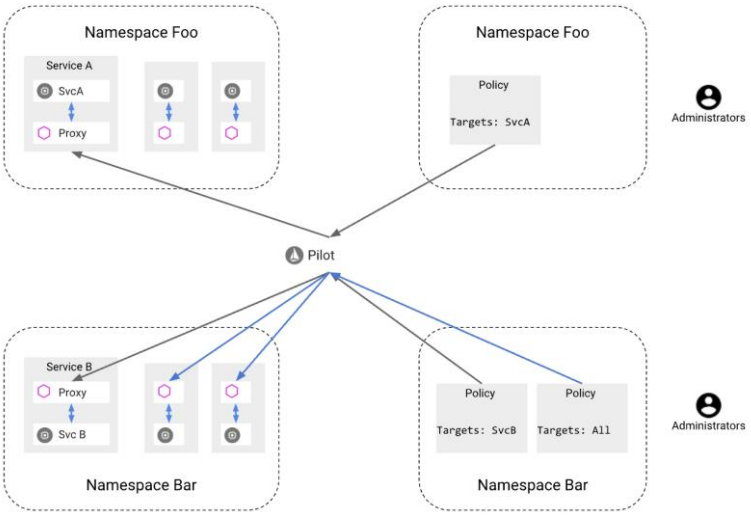


图 8-6 Istio 认证架构图

在 Kubernetes 平台上，Istio 认证实现的过程中，各组件功能如下：

- Citadel 监视 Kubernetes apiserver，负责为每个服务帐户创建证书和密钥对，并将证书和密钥对存储为 Kubernetes Secrets 对象。微服务 Pod 在启动时以 Volume 的形式将其服务账户的证书和密钥挂载。除了第一次创建证书外，Citadel 还会监视每个证书的生命周期，并通过重写 Secret 自动轮换证书。
- Pilot 在认证过程中，监视 Kubernetes API，生成安全命名信息，该信息定义了哪些服务账户可以运行哪些服务，避免恶意用户伪装服务账户获取数据。Pilot 将安全命名信息下发到 Envoy 中。另外，Pilot 还会监视 Istio 配置存储中的认证策略，通知 Envoy 如何执行身份认证机制。
- Envoy 在收到客户端的出站流量后，开始与服务端 Envoy 进行双向 TLS 握手。在握手期间，客户端 Envoy 还会执行安全命名检查，验证服务器证书中提供的服务账户是否有权运行目标服务。验证通过后，客户端 Envoy 和服务端 Envoy 建立一个双向的 TLS 连接，并将流量从客户端 Envoy 发送到服务端 Envoy。

Istio 身份验证通过之后，将两种类型的身份验证以及凭证中的其他声明输出到授权层，进行请求鉴权。

（2） 授权和鉴权

Istio 使用基于角色的访问控制（Role-Based Access Control）实现授权功能。并且具有多级别、灵活、高性能等特点。

授权架构图如图 8-7 所示：

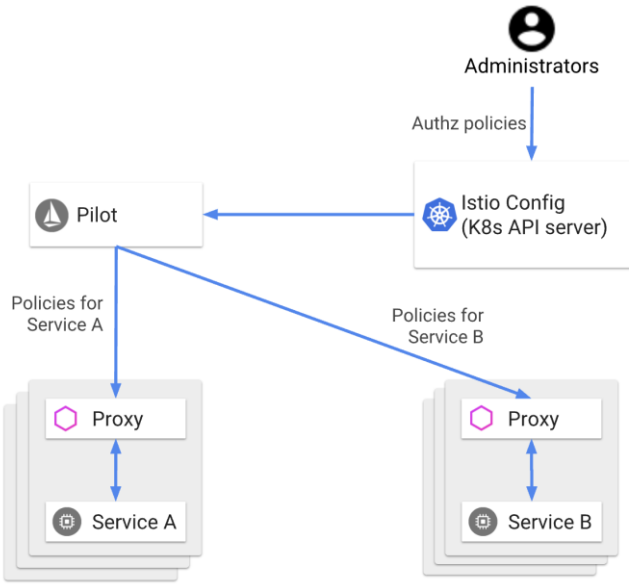


图 8-7 istio 授权架构图

图 8-7 显示了基本的授权模型，Istio 中授权实现的过程中，各组件功能如下：

- 管理员通过 Kubernetes 客户端创建授权策略。
- Pilot 监视 Istio 授权策略的创建和变更。如果发现任何更改，它将获取更新的授权策略。Pilot 将 Istio 授权策略分发给与服务实例位于同一位置的 Envoy Proxy。
- 每个 Envoy 代理都运行一个授权引擎，该引擎在运行时授权请求。当请求到达代理时，授权引擎根据当前授权策略评估请求上下文，并返回授权结果 ALLOW 或 DENY。

3. Galley 解析

Galley 在 Istio 中，承担配置的导入、处理和分发任务，为 Istio 提供了配置管理服务，提供在 Kubernetes 服务端验证 Istio 的 CRD 资源的合法性。

4. Mixer 解析

Mixer 负责执行访问控制、策略控制 (如授权、速率限制、配额、身份验证、请求跟踪等)和从 Envoy proxy 或者其他服务中采集遥测数据。Mixer 主要负责提供三个核心功能：

- 前置条件检测：发生在服务响应请求之前，验证一些前提条件，如认证、黑白名

单、ACL 检查等。如果检查不通过则终止响应。

- 配额管理：分配服务的配额，如根据条件对请求实施速率限制。
- 遥测数据报告：采集遥测数据，通常包括 Metrics、Logging、Distribution Trace 等。

Mixer 与 Envoy proxy 交互如图 8-8 所示：

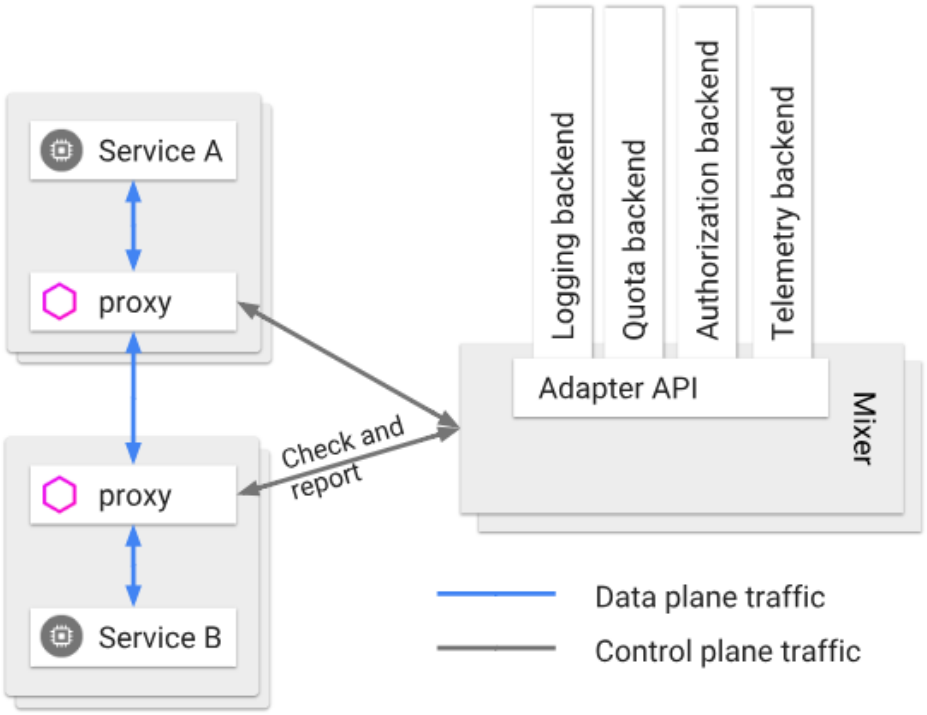


图 8-8 Mixer 与 Envoy proxy 交互

从图 8-8 可以看出，Mixer 的基本工作流程如下：

- 某一服务的调用请求被 Envoy 拦截，Envoy 根据请求向 Mixer 发起 Check rpc 请求。
- Mixer 进行前置条件检查和配额检查，调用相应的检测 adapter 做处理，并返回相应结果。
- Envoy 分析返回结果，决定是否执行请求或拒绝请求。若请求可以执行，则在请求执行完成后再向 Mixer gRPC 服务发起 Report rpc 请求，上报遥测数据。
- Mixer 后端的遥测 adapter 基于上报的遥测数据做进一步处理，记录在后端相应的服务中，如日志。

在 Istio 中，为了避免应用程序的微服务和基础设施的后端服务之间的直接耦合，

Mixer 使用适配器模型实现，屏蔽底层差异。Mixer 通过不同的 Adapter 与不同的后端服务对接，如日志、监控、配额、ACL 检查等 Endpoint 连接。Mixer 常见的 Adapter 如图 8-9 所示：

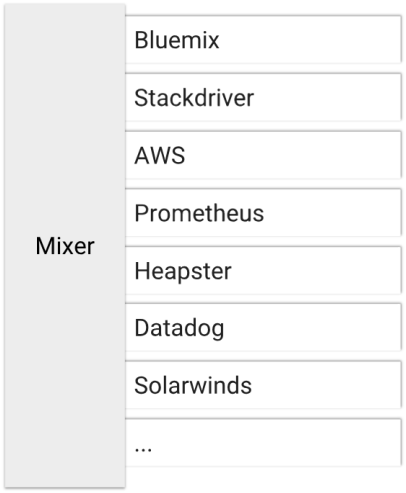


图 8-9 Mixer 常见 Adapter

9.1.3 数据平面

Istio 数据平面由一组代理（Envoy）组成。这些代理以 Sidecar 的方式与每个应用程序协同运行，负责调解和控制微服务之间的所有网络通信，并且与控制平面的 Mixer 通信，接受调度策略。正是有了 Envoy 代理，才使 Istio 不必像 Spring Cloud 框架那样，需要将微服务治理架构以 annotation 的方式写到应用源代码中（如 Spring Boot 使用 Hystrix），从而做到零代码侵入。如果用一种形象的方式比喻，Sidecard 就像挂斗摩托车的挂斗，如图 8-10 所示。挂斗与主车身组成了整个车，也就是 Pod。

图 8-10 Sidecar 示意图

Envoy 是一个基于 C ++开发的高性能代理，在 Istio 中，使用的是 Envoy 的扩展版本，被称为 Istio Proxy，可以理解为在标准版本的 Envoy 基础上，扩展了 Istio 独有的功能，典型如和 Mixer 的集成。

在 Istio 中，用于管理 Istio 中所有服务的所有入站和出站流量。 Istio 利用 Envoy 的许多内置功能，例如：

- 动态服务发现
- 负载均衡
- TLS 终止
- HTTP / 2 和 gRPC 代理
- 断路器
- 健康检查
- 流量分割
- 故障注入
- 监控指标