服务网格

服务网格简介

2017 年底，服务网格依托其非侵入式特性在微服务技术中崭露头角，Service Mesh 又译作“服务网格”，作为微服务间通信的基础设施层，Buoyant 公司的 CEO William Morgan 在文章《WHAT'S A SERVICE MESH? AND WHY DO I NEED ONE?》 1中解释了什么是 Service Mesh，为什么云原生应用需要使用 Service Mesh。

服务网格通常通过一组轻量级网络代理实现，这些代理与应用程序一起部署，而无需感知应用程序本身，图 1.1 为服务网格的架构图：

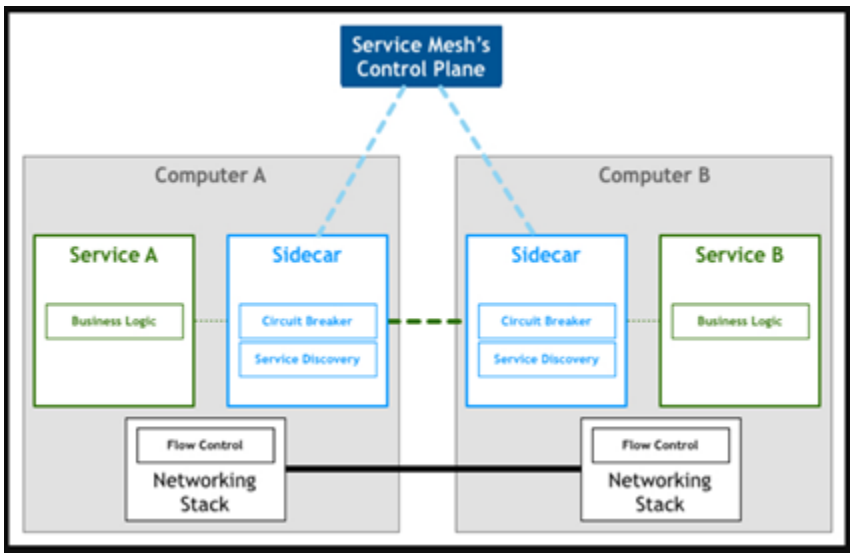

图 1.1 服务网格架构图

可以看出服务网格作为 Sidecar 运行在服务旁，对应用来说是透明的，所有通过应用的流量均会经过 Sidecar，因此 Sidecar 实现了流量控制功能，包括服务发现、负载均衡、智能路由、故障注入、熔断器、TLS 终止等。服务网格的出现将微服务治理从应用自身中抽离出来，通过 Sidecar 的形式极大降低了代码耦合度，使得微服务管理不再复杂。

1 https://buoyant.io/2020/10/12/what-is-a-service-mesh/

Istio 介绍

Istio 架构

Istio 是一款微服务管理框架，也被认为是服务网格的代表实现，由 Google、IBM、Lyft 联合开发的开源项目，其架构图如图 1.2 所示：

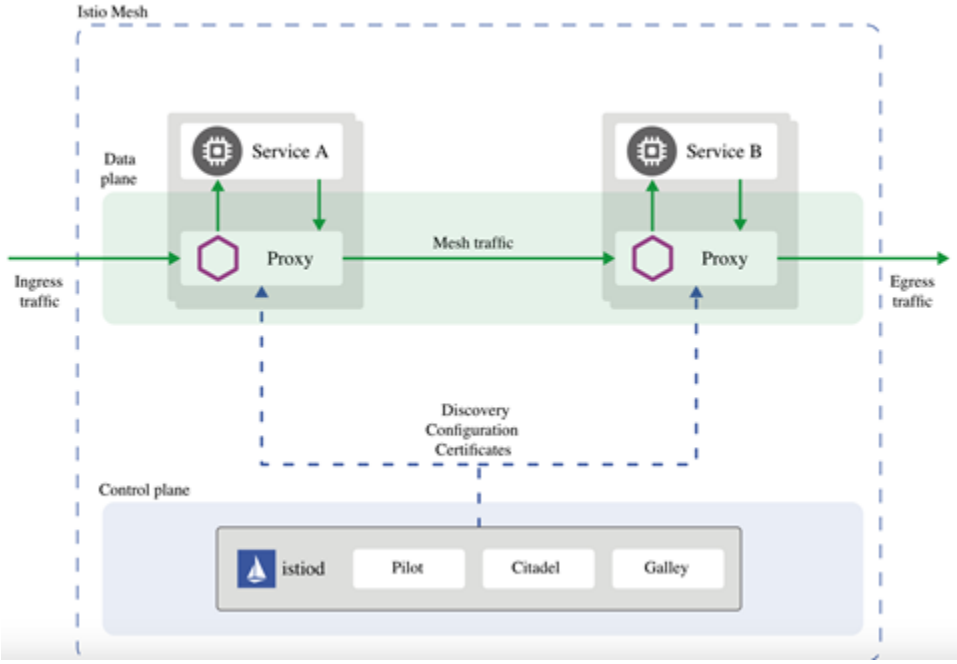

图 1.2 Istio 架构图2

Istio 架构的设计类似软件定义网络（Software-defined networking, SDN），主要分为控制平面和数据平面两个部分，其中数据平面由一组代理组成并以 Sidecar 的形式部署在服务旁，这些代理负责管理服务间的所有网络通信；控制平面用于对数据平面的代理进行管理和配置。这种设计的好处有很多，首先无需更改业务容器代码，从而可以最大程度的透明化使应用程序对代理注入无感知；另一方面，Istio 的开放化接口和灵活的部署机制在很大程度提升了可扩展性和可移植性；最后 Istio 的统一策略管理也将基于服务的诸多策略从 Sidecar 内部解耦，并通过 API 进行统一管理，从而简化了操作成本。

2 图片引用自 https://istio.io/latest/docs/ops/deployment/architecture/

数据平面

Istio 的数据平面由一组网络代理构成，Istio 默认使用的代理为 Envoy，当然这里也可以替换为其它代理，例如 linkerd，nginmesh、moison 或开发者自己编写的代理。 Envoy 是以 C++ 为基础开发的高性能代理，用于调解服务网格中服务的所有入站和出站流量。Envoy 代理是唯一与 Istio 数据平面流量进行交互的组件。

Envoy 的核心为 L3/L4 和 L7 层网络代理，通过 Envoy 内部的 Filter 机制及开放化的 Filter 编程接口，开发人员可使用 Envoy 现有的 Filter 或自定义 Filter 对业务流量进行细粒度化管理。

Envoy 的内置功能包括 “服务发现”、“负载均衡”、“TLS 终止”、“HTTP/2”、GRPC 代理”、“熔断器”、“健康检查”、“基于百分比流量拆分的灰度发布”、“故障注入” 等。

Envoy 在 Istio 中通常以 Sidecar 的方式部署，Sidecar 与业务容器部署在同一个 pod 中，如图 1.3 所示：

图 1.3 Envoy 代理部署拓扑

这种部署模式允许 Istio 提取大量流量相关的信息作为元数据。Istio 可以利用这些元数据实施相应策略，并同时将元数据发送至外部监控系统，例如 Prometheus，Zipkin，Jaeger 等，以将服务网格中发生的行为可视化。

控制平面

Istiod 为 Istio 的控制平面组件，istiod 下又包括 Pilot、Citadel、Galley 三个子组件。

Pilot

Pilot 负责为 Envoy Sidecar 提供服务发现功能，为智能路由和弹性（A/B 测试，金丝雀部署，超时，重试，熔断等）提供流量管理功能，将控制流量的高级路由规则转换为特定于 Envoy 的配置，并在运行时传播至 Envoy/Sidecar 容器中。

图 1.4 展示了 Pilot 和 Envoy 代理如何进行交互：

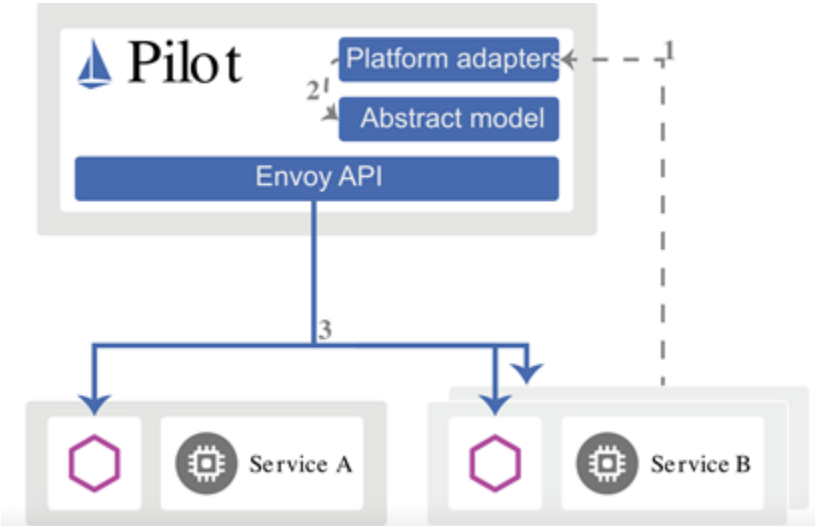

图 1.4 Pilot 与 Envoy 交互图[3]

由上图我们可以看出交互流程主要分为以下三个步骤：

1. 平台启动一个服务的新实例，该实例通知 Pilot 的平台适配器；
2. 平台适配器使用 Pilot 的抽象模型注册实例；
3. Pilot 将流量规则和配置下发至数据平面的 Envoy 代理；

可以看出这种松耦合设计允许 Istio 在 Kubernetes、Consul 或 Nomad 等多种环境中运行，同时维护相同的 Operator 接口来进行流量管理。

3 图片来自 https://istio.io/latest/zh/docs/ops/deployment/architecture/discovery.svg

Citadel

Citadel 是 Istio 的安全模块，通过内置的身份和证书管理，可支持强大的服务到服务以及最终用户的身份验证，同时，我们可以使用 Citadel 来升级服务网格中未经加密流量。关于 Citadel 如何实现服务的认证、授权可参考本书第 28 章的认证及授权部分。

Galley

Galley 是 Istio 配置过程中进行验证、提取、处理和分发的核心组件，其并不直接向数据平面提供业务能力，而是在控制平面上向其它组件提供支持，这样其它组件只用和 Galley 打交道，从而与底层平台（例如 Kubernetes）解耦。

Istio 特性

Istio 流量管理

Istio 拥有丰富的流量管理功能，其中主要包括“请求路由”、“故障注入”、“流量迁移”三种。

请求路由

Istio 支持用户下发请求路由策略，该策略主要用于将访问服务流量重定向至特定版本，从而实现金丝雀部署[4]，这种方式在测试新项目上线时带来了诸多好处。关于此特性的实验部分可以参考官方教程[5]。

故障注入

Istio 的故障注入机制主要用于测试整个应用的故障恢复能力，最终可以确保故障策略不会出现不兼容以及限制较多的情况，与在其它网络层引入故障机制不同，Istio 是在应用

4 金丝雀发布指的是在生产环境中分阶段逐步更新后端应用的版本（需要具备流量控制能力），在小范围验证符合预期之后，再推广至整个生产环境

5 https://istio.io/latest/docs/tasks/traffic-management/request-routing/

层注入故障，从而可以获得更多结果，例如使用 http 状态码来标明故障结果。关于此特性的实验部分可以参考官方教程[6]。

流量迁移

流量迁移指将流量从应用的一个版本逐步迁移到另一个版本，在 Istio 中，可以通过配置一系列规则来实现这一目标，例如可以将请求中 50%的流量发送到某服务 A 的 v1 版本，剩余的 50%流量发送到服务 A 的 v2 版本。然后，通过调整阈值最终将 100%的流量发送至服务 A 的 v2 版本来完成迁移。关于此特性的实验部分可以参考官方教程[7]。

Istio 安全性

Istio 通过 Citadel 组件可为服务网格中的服务提供签名证书从而保证微服务间的通信安全，此外，Istio 的认证及授权机制为服务网格服务间提供了相对安全的访问环境，关于 Istio 的安全风险和防护可以参考书中相关章节。

Istio 可观察性

由于 Istio 整个架构中包含 kiali、Prometheus、Grafna 等监控及可视化工具，因而可对服务网格中的服务调用链和请求数进行可视化，从而使服务网格具有较好的可观察性。关于 Istio 可观察性实验部分可以参考书中相关章节。

小结

本附录对服务网格进行了较为全面的介绍，读者可通过简明实践部分快速部署 Istio 平台，并理解如何使用 Istio 进行流量及监控管理，服务网格虽然在近几年非常火，但是其相比于容器、容器编排、微服务依然是新兴领域，目前国内外真正落地的产品并不多，不过伴随着云原生发展也是不可或缺的一环。

6 https://istio.io/latest/docs/tasks/traffic-management/fault-injection/
7 https://istio.io/latest/docs/tasks/traffic-management/traffic-shifting/