1. 什么是服务网格（ServiceMesh）？

2016年9⽉29⽇，Buoyant的CEO William Morgan，也就是Service Mesh这个词的发明人，对Service Mesh的定义：

服务网格是一个基础设施层，用于处理服务间通信。云原生应用有着复杂的服务拓扑，服务网格保证请求在这些拓扑中可靠地穿梭。在实际应用当中，服务网格通常是由一系列轻量级的网络代理组成的，它们与应用程序部署在一起，但对应用程序透明。

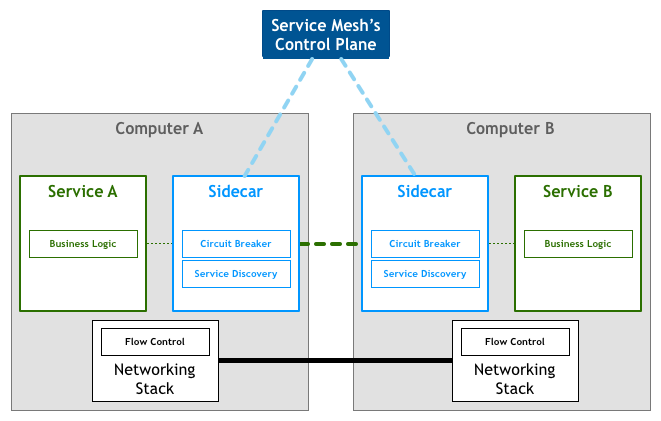
这个定义中，有四个关键词：

***基础设施层*：**这描述了Service Mesh的定位。

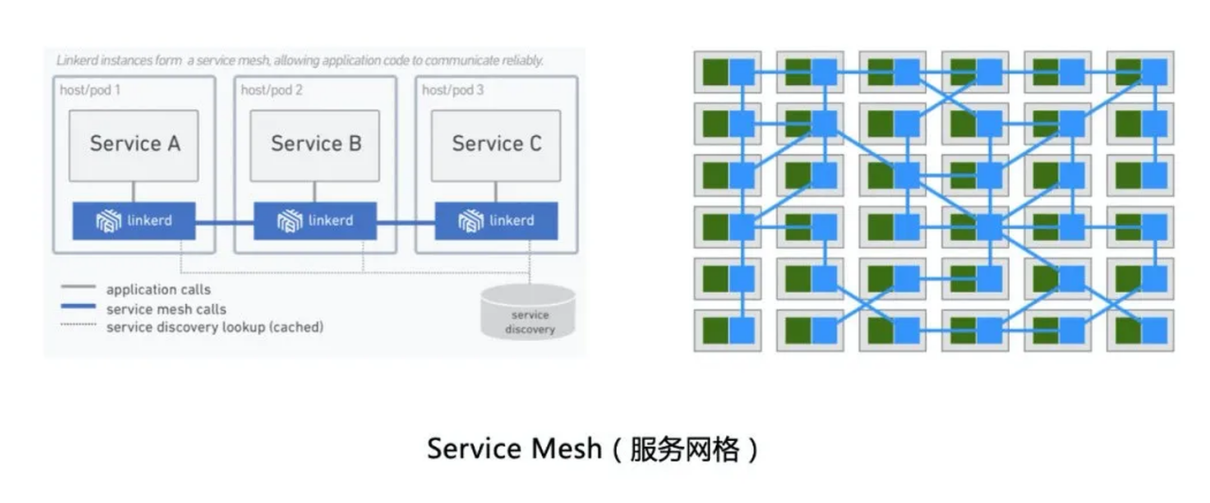
***请求在这些拓扑中可靠穿梭***：这描述了Service Mesh的功能。

***网络代理***：这描述了Service Mesh的实现形态。

***对应用透明***：这描述了Service Mesh的关键特点。



在服务网格中，所有的服务之间彼此不直接通信，而是通过服务所在容器内部的代理进程（边车sidecar）来通信。而所有容器组成的集群中，不同容器间通过sidecar连接起来，形成一个网格结构，这就是所谓的服务网格。



基于以上特性，Service Mesh具有如下优点：

屏蔽分布式系统通信的复杂性(负载均衡、服务发现、认证授权、监控追踪、流量控制等等)，服务只用关注业务逻辑；真正的语言无关，服务可以用任何语言编写，只需和Service Mesh通信即可；对应用透明，Service Mesh组件可以单独升级。

当然，Service Mesh目前也面临一些挑战：

Service Mesh组件以代理模式计算并转发请求，一定程度上会降低通信系统性能，并增加系统资源开销；Service Mesh组件接管了网络流量，因此服务的整体稳定性依赖于Service Mesh，同时额外引入的大量Service Mesh服务实例的运维和管理也是一个挑战。

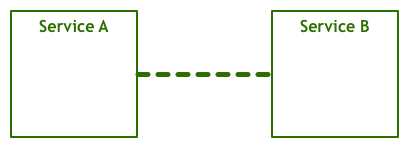
1. 为什么服务网格是微服务的下一阶段？

提到服务网格（Service Mesh），就不得不提微服务。根据维基百科的定义：

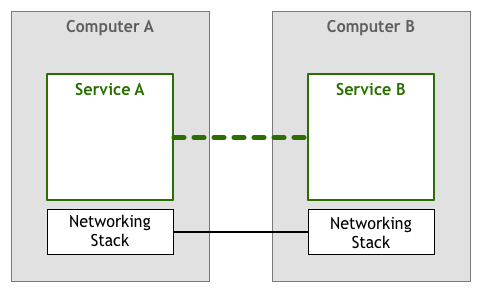
微服务 (Microservices) 是一种软件架构风格，它是以专注于单一责任与功能的小型功能区块 (Small Building Blocks) 为基础，利用模块化的方式组合出复杂的大型应用程序，各功能区块使用与语言无关 (Language-Independent/Language agnostic) 的 API 集相互通信。

Phil Calçado的文章[《Pattern: Service Mesh》](http://philcalcado.com/2017/08/03/pattern_service_mesh.html)，详细的介绍了从开发者视角来看，服务开发模式，微服务和Service Mesh技术的演化过程。

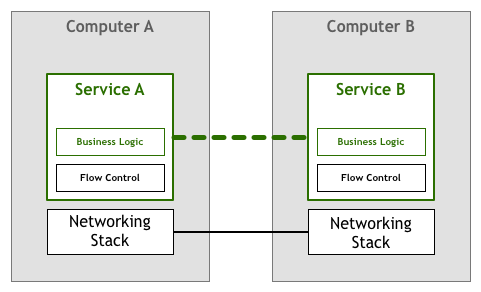
首先来看一个最简单的服务调用：Service A在本机同一个进程直接调用Service B。



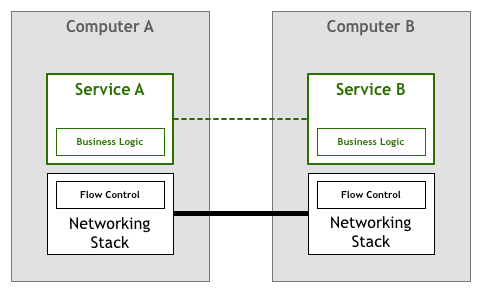
随着网络和分布式技术的发展，出现了远程服务调用（RPC）技术，Service A部署到了机器A，Service B部署到了机器B，这时候Service A调用Service B就需要通过网络调用了。



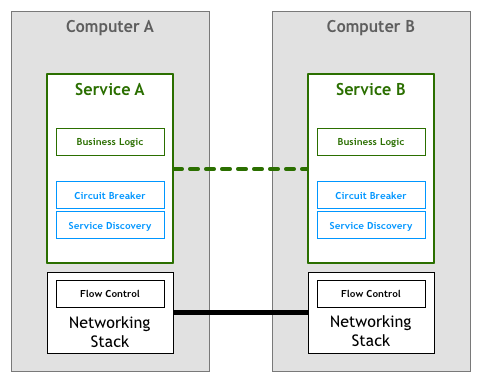
随着业务规模的快速发展，分布式服务化也得到了进一步发展，微服务兴起，大规模的应用服务下对系统可用性、稳定性等，提出了更高架构要求，所以业务逻辑和流量控制逐渐在服务层面体现出来。



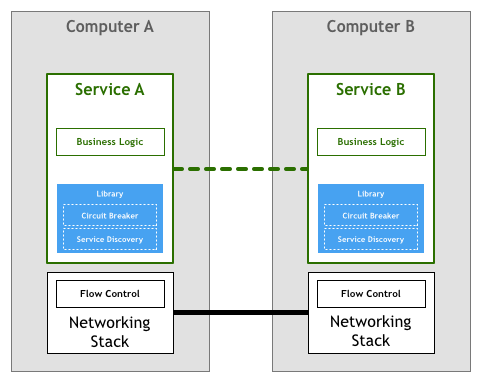
随着技术进一步的发展，流量控制可以下沉到网络技术栈层面，这样服务层就可以跟流量控制这种非业务能力进行解耦。



随着业务和技术的进一步发展，我们需要考虑在微服务层面下，对应用流量采取一定的策略实现服务的注册发现、降级熔断、权限认证、负载均衡等等进行治理管控。



当我们的技术再进一步发展，不少类库封装了通用性的一些服务治理管控策略，就出现了微服务框架。比如Spring Cloud，Vert.x，Go-Micro等等。这些框架的出现，大大简化了开发一个大规模的微服务应用系统。我们可以看到，此时一个明显的特征就是，服务框架的逻辑代码，最终和业务逻辑代码都运行在同一个进程里，比如JVM或Go进程。相应的所有策略也都配合在应用系统侧。这样的话，应用系统本身比较重。

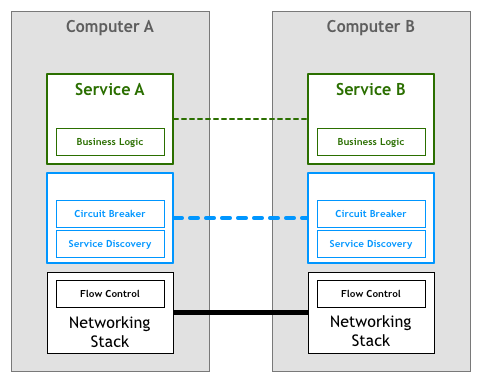


所以这个架构模式存在一些问题：

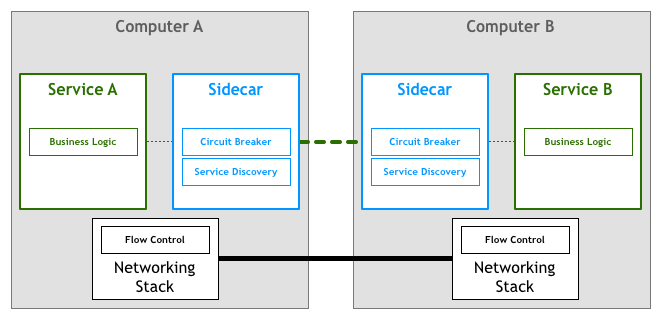
其一，虽然框架本身屏蔽了分布式系统通信的一些通用功能实现细节，但开发者却要花更多精力去掌握和管理复杂的框架本身，在实际应用中，去追踪和解决框架出现的问题也绝非易事；

其二，开发框架通常只支持一种或几种特定的语言，回过头来看文章最开始对微服务的定义，一个重要的特性就是语言无关，但那些没有框架支持的语言编写的服务，很难融入面向微服务的架构体系，想因地制宜的用多种语言实现架构体系中的不同模块也很难做到；

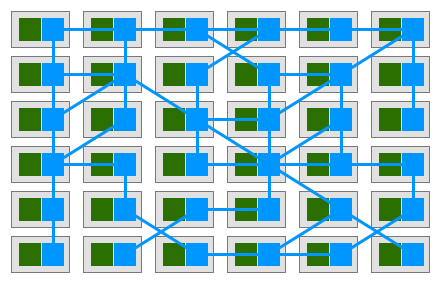
其三，框架以lib库的形式和服务联编，复杂项目依赖时的库版本兼容问题非常棘手，同时，框架库的升级也无法对服务透明，服务会因为和业务无关的lib库升级而被迫升级。



因此以Linkerd，Envoy，NginxMesh为代表的代理模式（边车模式）应运而生，这就是第一代Service Mesh，它将分布式服务的通信抽象为单独一层，在这一层中实现负载均衡、服务发现、认证授权、监控追踪、流量控制等分布式系统所需要的功能，作为一个和服务对等的代理服务，和服务部署在一起，接管服务的流量，通过代理之间的通信间接完成服务之间的通信请求，这样上边所说的三个问题也迎刃而解。



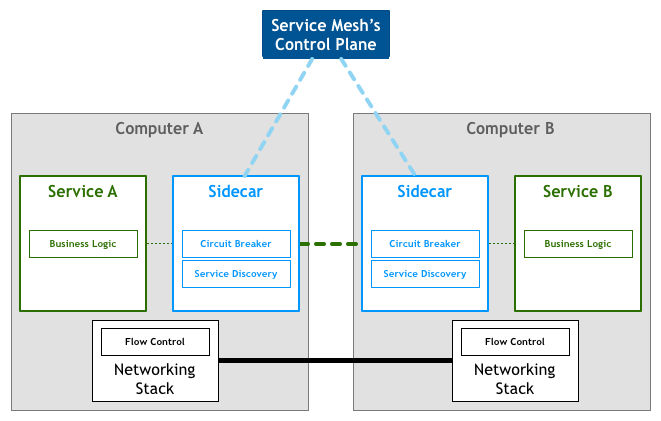
此时我们就得到了一个如下图所示的第一代服务网格：



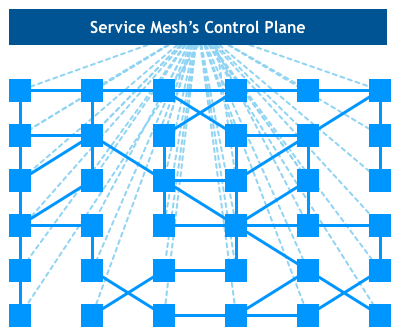
或者我们可以换一种表述方式：微服务应用中的非功能性的策略部分能力，下沉到网络的基础设施中，形成一个网格状的基础设施，而业务服务能力都通过容器化的方式部署到这个基础设施上，这就是服务网格。

实际上，第一代Service Mesh由一系列独立运行的单机代理服务构成，为了提供统一的上层运维入口，演化出了集中式的控制面板，所有的单机代理组件通过和控制面板交互进行网络拓扑策略的更新和单机数据的汇报。这就是以Istio为代表的第二代Service Mesh。

在Istio的体系里，每个Sidecar是否能与另外一个Sidecar通信，也就是对应的流量控制策略，都是由控制面板进行管理的。而部署了微服务和Sidecar的容器集群，则被称为数据面板。所有的控制面板和数据面板都是由Kubernetes进行管理的。



集群里所有容器中Sidecar都是由控制面板进行策略的管理，并且在发生变化时同步到整个集群。此时我们再拉回全局视图，就可以看到第二代服务网格的整体结构如下图所示：



全局的网格拓扑关系，都是形成了所谓的数据面板网格，而这些网格的关系由控制面板来管理。

1. 服务网格有什么用途？

Service Mesh 主要解决用户如下 3 个维度的痛点需求：

* 1. 完善的微服务基础设施

通过将微服务通信下沉到基础设施层，屏蔽了微服务处理各种通信问题的复杂度，形成微服务之间的抽象协议层。开发者无需关心通信层的具体实现，也无需关注 RPC 通信（包含服务发现、负载均衡、流量调度、流量降级、监控统计等）的一切细节，真正像本地调用一样使用微服务，通信相关的一切工作直接交给 Service Mesh。

* 1. 语言无关的通信和链路治理

功能上，Service Mesh 并没有提供任何新的特性和能力，Service Mesh 提供的所有通信和服务治理能力在 Service Mesh 之前的技术中均能找到，比如 Spring Cloud 就实现了完善的微服务 RPC 通信和服务治理支持。

Service Mesh 改变的是通信和服务治理能力提供的方式，通过将这些能力实现从各语言业务实现中解耦，下沉到基础设施层面，以一种更加通用和标准化的方式提供，屏蔽不同语言、不同平台的差异性，有利于通信和服务治理能力的迭代和创新，使得业务实现更加方便。

Service Mesh 避免了多语言服务治理上的重复建设，通过 Service Mesh 语言无关的通信和服务治理能力，助力于多语言技术栈的效率提升。

* 1. 通信和服务治理的标准化

微服务治理层面，Service Mesh 是标准化、体系化、无侵入的分布式治理平台。

标准化方面，Sidecar 成为所有微服务流量通信的约束标准，同时 Service Mesh 的数据平台和控制平面也通过标准协议进行交互。

体系化方面，从全局考虑，提供多维度立体的微服务可观测能力（Metric、Trace、Logging），并提供体系化的服务治理能力，如限流、熔断、安全、灰度等。

通过标准化，带来一致的服务治理体验，减少多业务之间由于服务治理标准不一致带来的沟通和转换成本，提升全局服务治理的效率。

1. 服务网格有哪些开源技术？



Service Mesh 的主流实现包括：

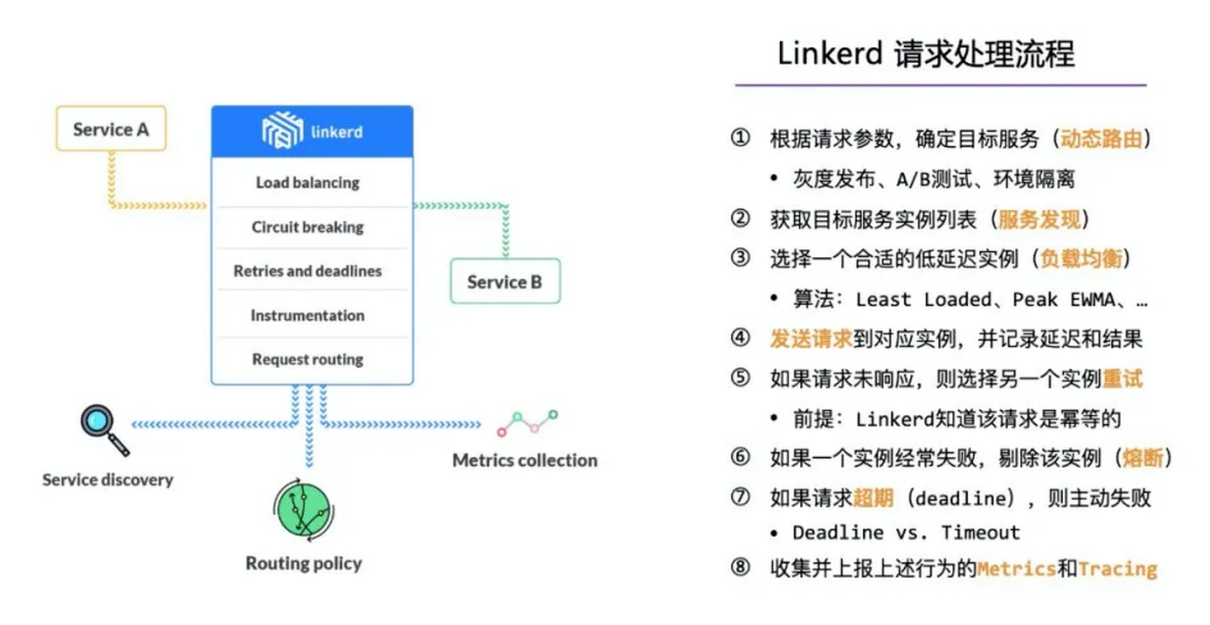
Linkerd：背后公司是Buoyant，开发语⾔使用Scala，2016年1⽉15日初次发布，2017年1⽉23日加入CNCF，2018年5⽉1⽇发布1.4.0版本。

Envoy：背后公司是Lyft，开发语言使用C++ 11，2016年9月13日初次发布，2017年9⽉14日加⼊CNCF，2018年3月21日发布1.6.0版本。

Istio：背后公司是Google和IBM，开发语言使用Go，2017年5⽉月10日初次发布，2018年3⽉31日发布0.7.1版本。

Conduit：背后公司也是Buoyant，开发语言使用Rust和Go，2017年12月5日初次发布，2018年4⽉27日发布0.4.1版本。

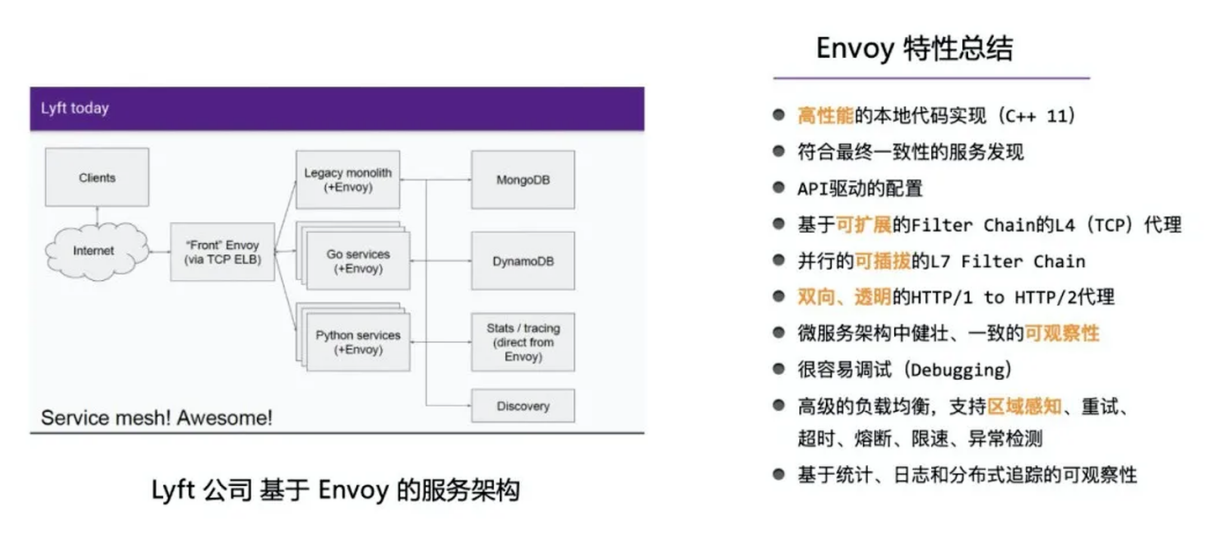
* 1. Linkerd 简介



Linkerd的核心组件就是一个服务代理，因此只要理清它的请求处理流程，就掌握了它的核心逻辑：

动态路由：根据上游服务请求参数，确定下游目标服务；除了常规的服务路由策略，Linkerd还可以通过这一层动态路由能力，支持灰度发布、A/B测试、环境隔离等非常有价值的场景。服务发现：确定目标服务后，下一步就是获取对应的实例的地址列表（e.g. 查询service registry）。负载均衡：如果列表中有多个地址，Linkerd会通过负载均衡算法（e.g. Least Loaded、Peak EWMA）选择其中⼀个合适的低延迟实例。执行请求：发送请求到上一步所选择的实例，并记录延迟和响应结果。重试处理：如果请求未响应，则选择另⼀个实例重试（前提：Linkerd知道该请求是幂等的）。熔断处理：如果发往某个实例的请求经常失败，则主动从地址列表中剔除该实例。超时处理：如果请求超期（在给定的deadline时间点之前仍未返回），则主动返回失败响应。可观测性：Linkerd会持续收集和上报上述各种行为数据，包括Metrics和Tracing。

* 1. Envoy 简介

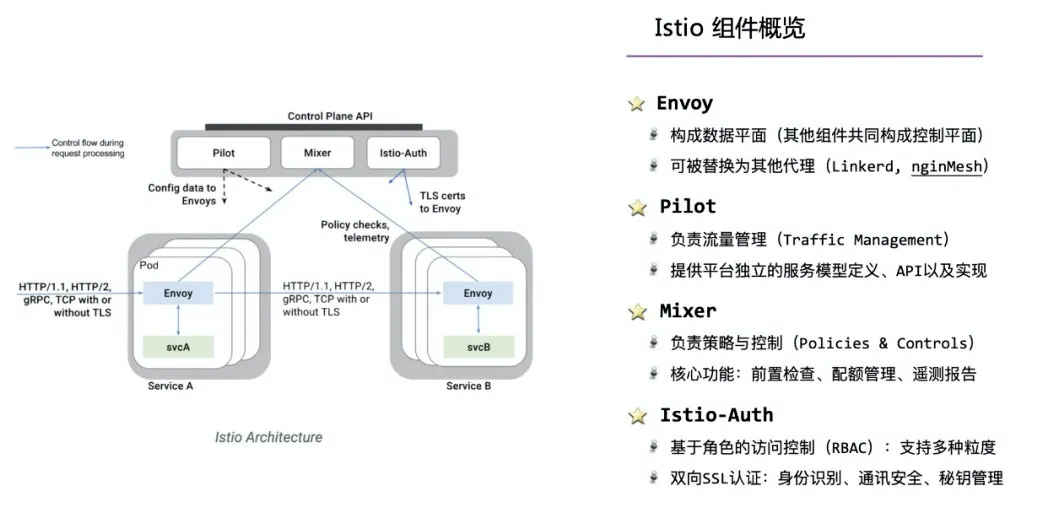


Envoy是一个高性能的Service Mesh软件，主要包含如下特性：

高性能：基于本地代码（C++ 11）实现；相比之下，Linkerd是基于Scala编写，肯定要慢不少。

可扩展：L4和L7层代理功能均基于可插拔的 Filter Chain 机制（类比 netfilter、servlet filter）。协议升级：支持双向、透明的 HTTP/1 to HTTP/2 代理能力。其他能力：服务发现（符合最终一致性）、负载均衡（支持区域感知）、稳定性（重试、超时、熔断、限速、异常检测）、可观测性（统计/日志/追踪）、易于调试等。

* 1. Istio 简介



Istio是一个管控/数据平面分离的完整Service Mesh套件，包含如下组件：

* Envoy：构成数据平⾯（其他组件共同构成控制平⾯）；可被替换为其他代理（e.g. Linkerd, nginMesh）。Pilot：负责流量管理（Traffic Management），提供平台独⽴的服务模型定义、API以及实现。Mixer：负责策略与控制（Policies & Controls），核⼼功能包括：前置检查、配额管理、遥测报告。Istio-Auth：支持多种粒度的RBAC权限控制；支持双向SSL认证，包括身份识别、通讯安全、秘钥管理。

Istio 首先是一个服务网络，但是Istio又不仅仅是服务网格: 在 Linkerd， Envoy 这样的典型服务网格之上，Istio提供了一个完整的解决方案，为整个服务网格提供行为洞察和操作控制，以满足微服务应用程序的多样化需求。

Istio在服务网络中统一提供了许多关键功能(以下内容来自官方文档)：

流量管理：控制服务之间的流量和API调用的流向，使得调用更可靠，并使网络在恶劣情况下更加健壮。

可观察性：了解服务之间的依赖关系，以及它们之间流量的本质和流向，从而提供快速识别问题的能力。

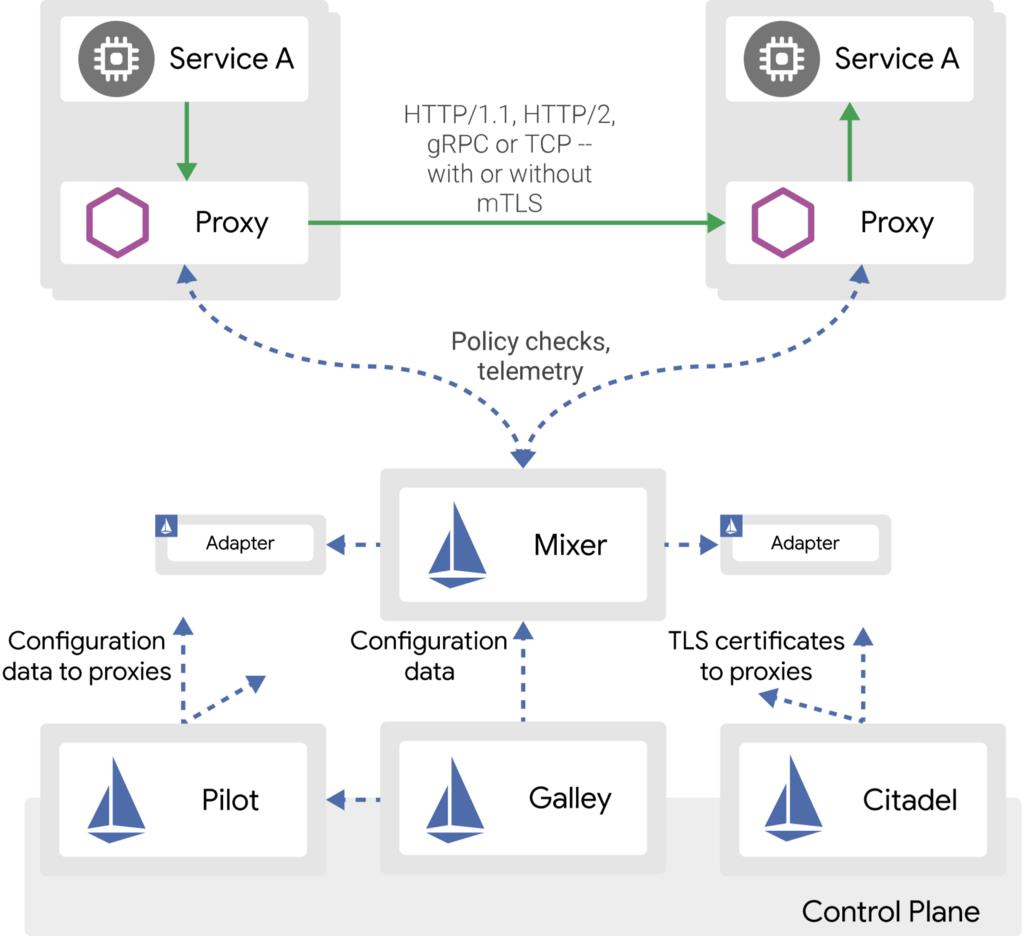
策略执行：将组织策略应用于服务之间的互动，确保访问策略得以执行，资源在消费者之间良好分配。策略的更改是通过配置网格而不是修改应用程序代码。

服务身份和安全：为网格中的服务提供可验证身份，并提供保护服务流量的能力，使其可以在不同可信度的网络上流转。

除此之外，Istio针对可扩展性进行了设计，以满足不同的部署需要：

平台支持：Istio旨在在各种环境中运行，包括跨云， 预置，Kubernetes，Mesos等。最初专注于Kubernetes，但很快将支持其他环境。

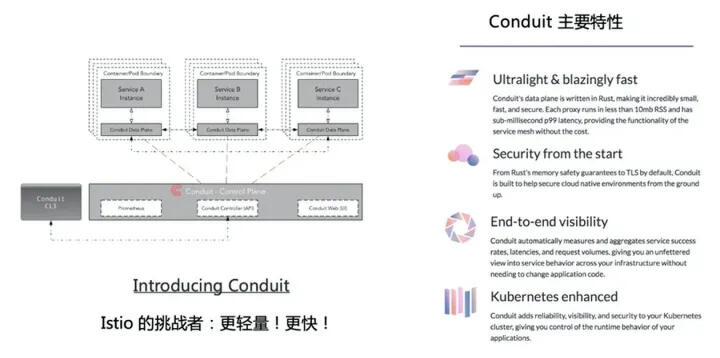
集成和定制：策略执行组件可以扩展和定制，以便与现有的ACL，日志，监控，配额，审核等解决方案集成。



这些功能极大的减少了应用程序代码，底层平台和策略之间的耦合，使微服务更容易实现。所以说，Istio 大幅降低微服务架构下应用程序的开发难度，势必极大的推动微服务的普及。

目前来看，Istio基本上已经成为了服务网格实践的事实标准，无论是各大公司的内部服务网格实践，还是各种大型公司、创业公司的服务网格平台产品，都是基于Istio来进行封装或者定制开发的。

* 1. Conduit 简介



Conduit是由Buoyant公司出品的下一代 Service Mesh。作为Istio的挑战者，Conduit的整体架构与Istio类似也明确区分了管控平面和数据平面，但同时它还具备如下关键特性：

轻量快速：Conduit的数据平面是基于原生的Rust语言编写，非常轻量、快速和安全（Rust相比C/C++的最大改进点就是安全性）。单个代理的实际内存消耗（RSS）小于10mb，延迟的p99分位点小于1ms，基本相当于能为应用程序提供免费（无额外开销）的Service Mesh功能。

安全保障：Conduit构建之初就考虑了云原生环境的安全性，包括Rust语言内存安全性、默认TLS加密等。

端到端可见性：Conduit可以自动测量和聚合服务的成功率、延迟与请求容量数据，让开发者在无需变更应用代码就能获取到服务的完整行为视图。

Kubernetes增强：Conduit为K8s集群添加了可靠性、可见性和安全性，同时给予了开发者对自己应用程序运行时行为的完全控制。

* 1. Linkerd、Envoy、Istio、Conduit对比

《云计算及虚拟化技术丛书Service Mesh实战》

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **功能** | **Linkerd** | **Envoy** | **Istio** | **Conduit** |
| 代理 | Finagle + Jetty | Envoy | Envoy | Conduit |
| 熔断 | 支持。基于连接的熔断器Fast Fail和基于请求的熔断器Failure Accrual。 | 支持。通过特定准则，如最大连接数、 最大请求数、最大挂起请求数或者最大重试数的设定。 | 支持。通过特定准则，如最大连接数和最大请求数等的设定。 | 暂不支持。 |
| 动态路由 | 支持。通过设置Linkerd的dtab规则实现不同版本服务请求的动态路由。 | 支持。通过服务的版本或环境信息实现。 | 支持。通过服务的版本或环境信息实现。 | 暂不支持。 |
| 流量分流 | 支持。以增量和受控的方式实现分流。 | 支持。以增量和受控的方式实现分流。 | 支持。以增量和受控的方式实现分流。 | 暂不支持。 |
| 服务发现 | 支持。支持多种服务发现机制，如基于文件的服务发现、Consul、Zookeeper、Kubernetes等。 | 支持。通过提供平台无关的服务发现接口实现与不同服务发现工具集成。 | 支持。通过提供平台无关的服务发现接口实现与不同服务发现工具集成。 | 只支持Kubernetes。 |
| 负载均衡 | 支持。提供多种负载均衡算法。 | 支持。提供多种负载均衡算法，如Round Robin、加权最小请求、哈希环、Maglev等。 | 支持。提供多种负载均衡算法，如Round Robin、加权最小请求、哈希环、Maglev等。 | 支持。当前只有HTTP请求支持基于P2C+ least-loaded的负载均衡算法。 |
| 安全通信 | 支持TLS。 | 支持TLS。 | 支持TLS。 | 支持TLS。 |
| 访问控制 | 不支持。 | 不支持。 | 支持。基于RBAC的访问控制。 | 暂不支持。 |
| 可见性 | 分布式追踪(Zipkin)、运行时指标(InfluxDB、Prometheus、statsd) | 分布式追踪(Zipkin)、运行时指标(statsd) | 分布式追踪(Zipkin)、运行时指标(Prometheus、statsd)、监控(NewRepic、Stackdriver) | 运行时指标(Prometheus) |
| 部署模式 | sidecar或者per-host模式 | sidecar模式 | sidecar模式 | sidecar模式 |
| 控制平面 | Namerd | 没有，但可通过API实现。 | Pilot、Mixer、Citadel | Conduit |
| 协议支持 | HTTP/1.x、HTTP/2、gRPC | HTTP/1.x、HTTP/2、gRPC、TCP | HTTP/1.x、HTTP/2、gRPC、TCP | HTTP/1.x、HTTP/2、gRPC、TCP |
| 运行平台 | 平台无关 | 平台无关 | 目前支持Kubernetes，平台无关是最终实现目标。 | 只支持Kubernetes。 |

1、Linkerd

Linkerd是Buoyant公司2016年率先开源的高性能网络代理，是业界的第一款Service Mesh框架。其主要用于解决分布式环境中服务之间通信面临的一些问题，如网络不可靠、不安全、延迟丢包等问题。

Linkerd使用Scala语言编写，运行于JVM，底层基于Twitter的Finagle库，并对其做了相应的扩展。最主要的是Linkerd具有快速、轻量级、高性能等特点，每秒以最小的延迟及负载处理万级请求，易于水平扩展。除此之外，还有以下功能：

支持多平台：可运行于多种平台，比如Kubernetes、DC/OS、Docker，甚至虚拟机或物理机。

无缝集成多种服务发现工具。

支持多协议，如gRPC、HTTP/1.x、HTTP/2，甚至可通过linkerd-tcp支持TCP协议。

支持与第三方分布式追踪系统Zipkin集成。

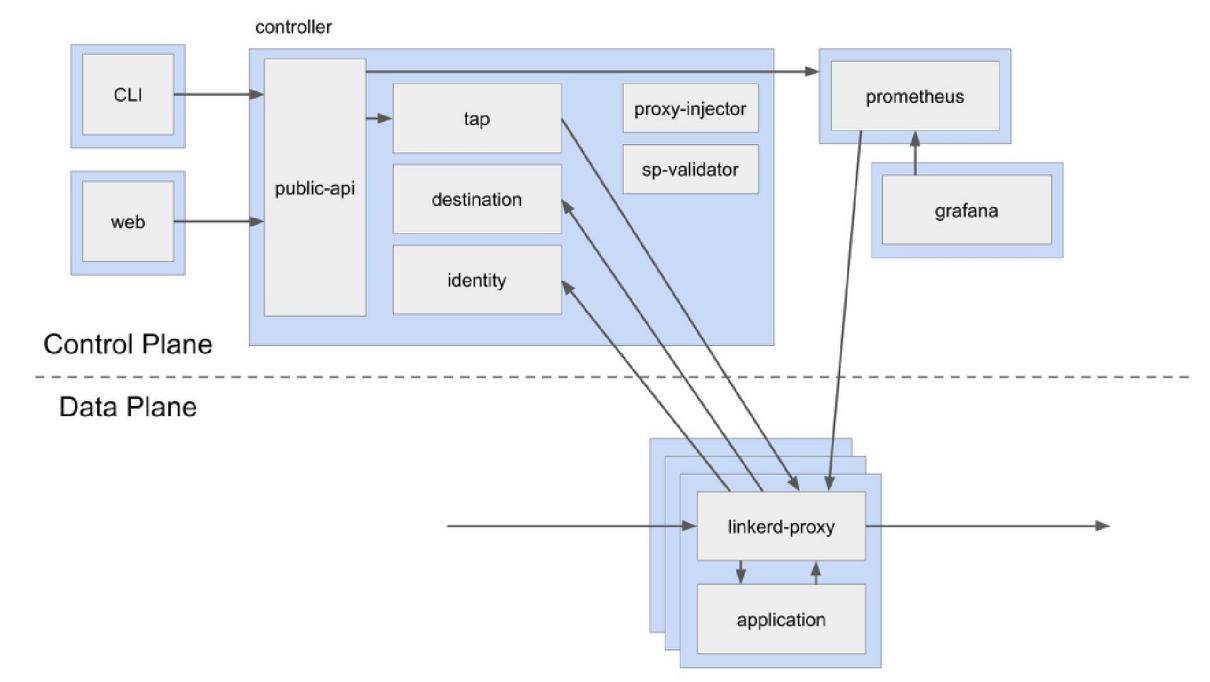
灵活性、扩展性高，可通过其提供的接口开发自定义插件。

目前，Linkerd和Linkerd2并行开发，其情况如下：

Linkerd：Linkerd使用\*\*Scala语言编写\*\*，运行于JVM，底层基于Twitter的Finagle库，并对其做了相应的扩展。

Linkerd2：使用Go语言和Rust语言完全重写了Linkerd，专门用于Kubernetes。

Linkerd本身是数据平面，负责将数据路由到目标服务，同时保证数据在分布式环境中传输是安全、可靠、快速的。另外，Linkerd还包括控制平面组件Namerd，通过控制平面Namerd实现中心化管理和存储路由规则、中心化管理服务发现配置、支持运行时动态路由以及暴露Namerd API管理接口。



控制平面

是在Kubernetes特定命名空间中运行的一组服务。这些服务可以完成各种事情：聚集遥测数据，提供面向用户的API，向数据平面代理提供控制数据等。

由以下部分组成：

Controller：由public-api容器组成，该容器为CLI和dashboard提供接口API。

Destination：数据平面中的每个代理都使用此组件来查找将请求发送到哪里。还用于获取服务配置信息，如：路由指标，重试和超时等。

Identity：该组件提供了证书的颁发，接受来自代理的[CSRs](https://en.wikipedia.org/wiki/Certificate_signing_request)并返回正确身份签名的证书。这些证书由代理在启动时获取，并且必须在代理准备就绪之前发出。随后，它们可用于Linkerd代理之间的任何连接以实现mTLS。

Proxy Injector：是一个注入程序，每次创建一个pod时，它都会接收一个webhook请求。该注入程序检查资源以查找特定于Linkerd的注释（linkerd.io/inject: enabled）。当存在该注释时，注入器将更改容器的规范，并添加 initContainer包含代理本身的以及附属工具。

Service Profile Validator：用于在保存新服务配置文件之前先对其进行验证。

Tap：从CLI和dashboard接收请求，以实时监视请求和响应。

数据平面

由轻量级代理组成，这些代理作为sidecar容器与服务代码的每个实例一起部署。为了将服务“添加”到Linkerd服务网格，必须重新部署该服务的Pod，以在每个Pod中包含数据平面代理。

2、Envoy

同Linkerd一样，Envoy也是一款高性能的网络代理，于2016年10月份有Lyft公司开源，为云原生应用而设计，可作为边界入口，处理外部流量，此外，也作为内部服务间通信代理，实现服务间可靠通信。Envoy的实现借鉴现有生产级代理及负载均衡器，如Nginx、HAProxy、硬件负载均衡器及云负载均衡器的实践经验，同时基于C++编写及Lyft公司生产实践证明，Envoy性能非常优秀、稳定。

Envoy既可用作独立代理层运行，也可作为Service Mesh架构中数据平面层，因此通常Envoy跟服务运行在一起，将应用的网络功能抽象化，Envoy提供通用网络功能，实现平台及语言无法性。除此之外，还有以下功能：

优先支持HTTP/2和gRPC，同时支持Websocket和TCP代理。

API驱动的配置管理方式，支持动态管理、更新配置以及无连接和请求丢失的热重启功能。

L3/L4层过滤器形成Envoy核心的连接管理功能。

通过与多种指标收集工具及分布式追踪系统集成，实现运行时指标收集、分布式追踪，提供整个系统及服务的运行时可见性。

内存资源使用率低，sidecar是Envoy最常用的部署模式。

3、Istio

Istio是由Google、IBM和Lyft发起的开源的Service Mesh框架。该项目在2017年推出，并在2018年7月发布了1.0版本。

Istio是Service Mesh目前的实现的典型代表，如果Sidecar是整个Service Mesh的数据面，那么Istio主要在控制面上做了更多的改进，Istio使用Envoy作为Sidecar，控制面相关全部使用Golang编写，性能上有了很大的提升。

Istio 首先是一个服务网格，但是Istio又不仅仅是服务网格：在 Linkerd，Envoy 这样的典型服务网格之上，Istio提供了一个完整的解决方案，为整个服务网格提供行为洞察和操作控制，以满足微服务应用程序的多样化需求。

Istio在服务网络中统一提供了许多关键功能：

流量管理：控制服务之间的流量和API调用的流向，使得调用更可靠，并使网络在恶劣情况下更加健壮。

可观察性：了解服务之间的依赖关系，以及它们之间流量的本质和流向，从而提供快速识别问题的能力。

策略执行：将组织策略应用于服务之间的互动，确保访问策略得以执行，资源在消费者之间良好分配。策略的更改是通过配置网格而不是修改应用程序代码。

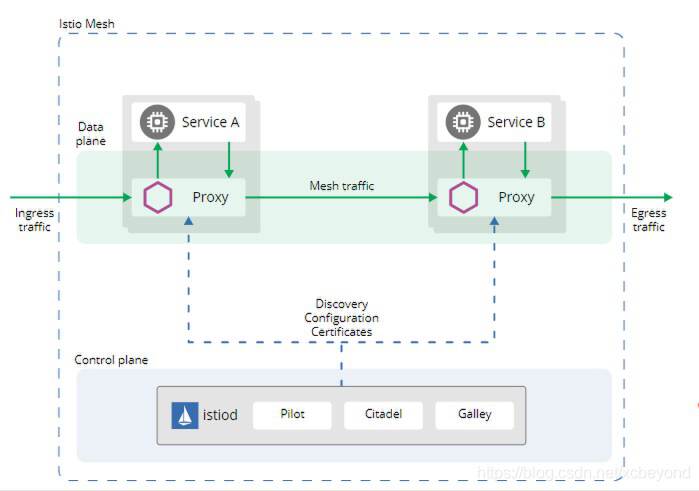
服务身份和安全：为网格中的服务提供可验证身份，并提供保护服务流量的能力，使其可以在不同可信度的网络上流转。

除此之外，Istio针对可扩展性进行了设计，以满足不同的部署需要。

平台支持：Istio旨在在各种环境中运行，包括跨云， 预置，Kubernetes，Mesos等。最初专注于Kubernetes，但很快将支持其他环境。

集成和定制：策略执行组件可以扩展和定制，以便与现有的ACL，日志，监控，配额，审核等解决方案集成。

这些功能极大的减少了应用程序代码，底层平台和策略之间的耦合，使微服务更容易实现。



Istio架构图中各个子模块功能如下：

Envoy：负责各个应用服务之间通信。

Pilot：管理和配置Envoy，提供服务发现、负载均衡和智能路由，保证弹性服务（服务超时次数、重试、熔断策略）。

Mixer：信息监控检查。

Istio-Auth：提供服务和服务、用户和服务之间的认证服务，实现访问控制，解决是谁访问的是哪个 API 的问题。

其中，图中的通信代理组件为Envoy，这是Istio原生引入的，但Linkerd也能够集成对接Istio。

4、Conduit

Conduit于2017年12月发布，作为由Buoyant继Linkerd后赞助的另外一个开源项目，作为Linkerd面向Kubernetes的独立版本。Conduit旨在彻底简化用户在Kubernetes使用服务网格的复杂度，提高用户体验，而不是像Linkerd一样针对各种平台进行优化。

Conduit的主要目标是轻量级、高性能、安全并且非常容易理解和使用。同Linkerd和Istio，Conduit也包含数据平面和控制平面，其中数据平面由Rust语言开发，使得Conduit使用极少的内存资源，而控制平面由Go语言开发。Conduit依然支持Service Mesh要求的功能，而且还包括以下功能：

超级轻量级和极快的性能。

专注于支持Kubernetes平台，提高运行在Kubernetes平台上服务的可靠性、可见性及安全性。

支持gRPC、HTTP/2和HTTP/1.x请求及所有TCP流量。

Conduit以极简主义架构，以零配置理念为中心，旨在减少用户与Conduit的交互，实现开箱即用。

1. 服务网格在业内有哪些平台级实践？
   1. 蚂蚁 SOFAStack Mesh/MOSN

服务网格（SOFAStack Mesh）是蚂蚁集团自主研发的基于金融级生产实践的增强版服务网格平台，将传统微服务和 Service Mesh 技术进行了深度融合，其核心技术经过了蚂蚁集团的大规模生产实践验证。它深度、无缝对接了 SOFAStack 经典应用服务和容器应用服务，为客户提供了简单易用的 Service Mesh 架构的支撑平台。

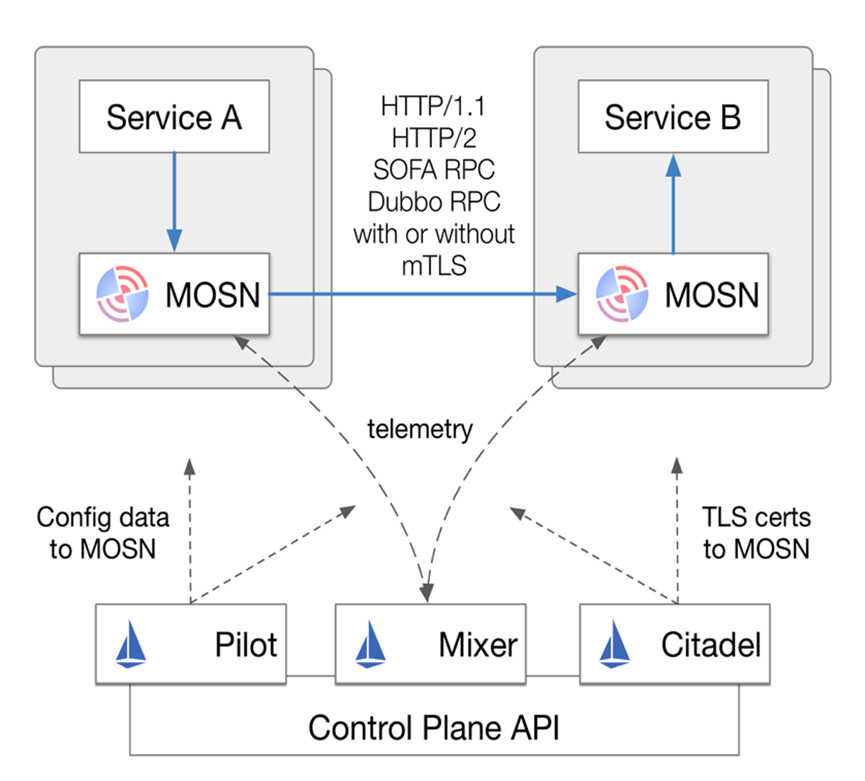
SOFAStack 服务网格（简称服务网格）= 传统微服务 + Service Mesh。

服务网格旨在提供与平台无关、语言无关、轻量无侵入的云原生分布式架构解决方案，在兼容 Kubernetes 和 Istio 生态的基础上还能支持虚拟机场景，同时支持 SOFA、Dubbo、SpringCloud 微服务框架的服务治理能力。在整个接入过程中，无需修改业务代码，帮助企业以最小的成本完成云原生落地，加速数字化转型。

服务网格的产品架构逻辑上分为控制层面和数据层面。

控制层面：管理代理（蚂蚁自研 MOSN），用于管理流量路由、运行时策略执行等。

数据层面：由一系列代理（蚂蚁自研 MOSN）组成，用于管理和控制服务之间的网络通信。

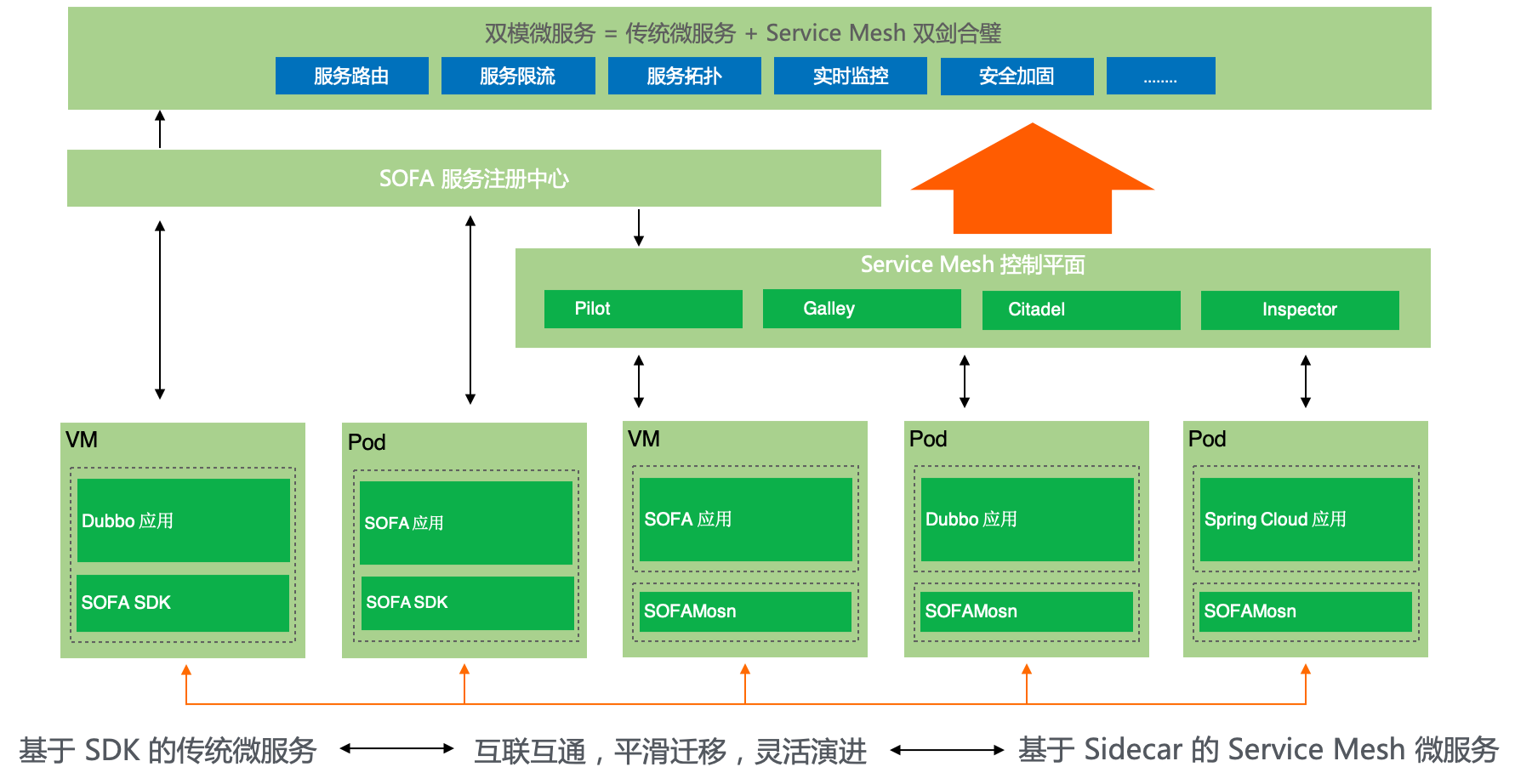


SOFAStack 服务网格结合了 SOFAStack 微服务的优势，将 SOFAStack 微服务和 Service Mesh 双剑合璧，即基于 SDK 的传统微服务可以和基于 Sidecar 的 Service Mesh 微服务实现下列目标：

互联互通：两个体系中的应用可以相互访问。

平滑迁移：应用可以在两个体系中迁移，对于调用该应用的其他应用，做到透明无感知。

灵活演进：在互联互通和平滑迁移实现之后，我们就可以根据实际情况进行灵活的应用改造和架构演进。



在控制面上，引入了 Pilot 实现配置的下发（如服务路由规则），在服务发现上保留了独立的 SOFA 服务注册中心。

在数据面上，使用了自研的 SOFA MOSN，不仅支持 SOFA 应用，同时也支持 Dubbo 和 Spring Cloud 应用。

在部署模式上，我们不仅支持容器/k8s，同时也支持虚拟机场景。

ylq……

准备工作

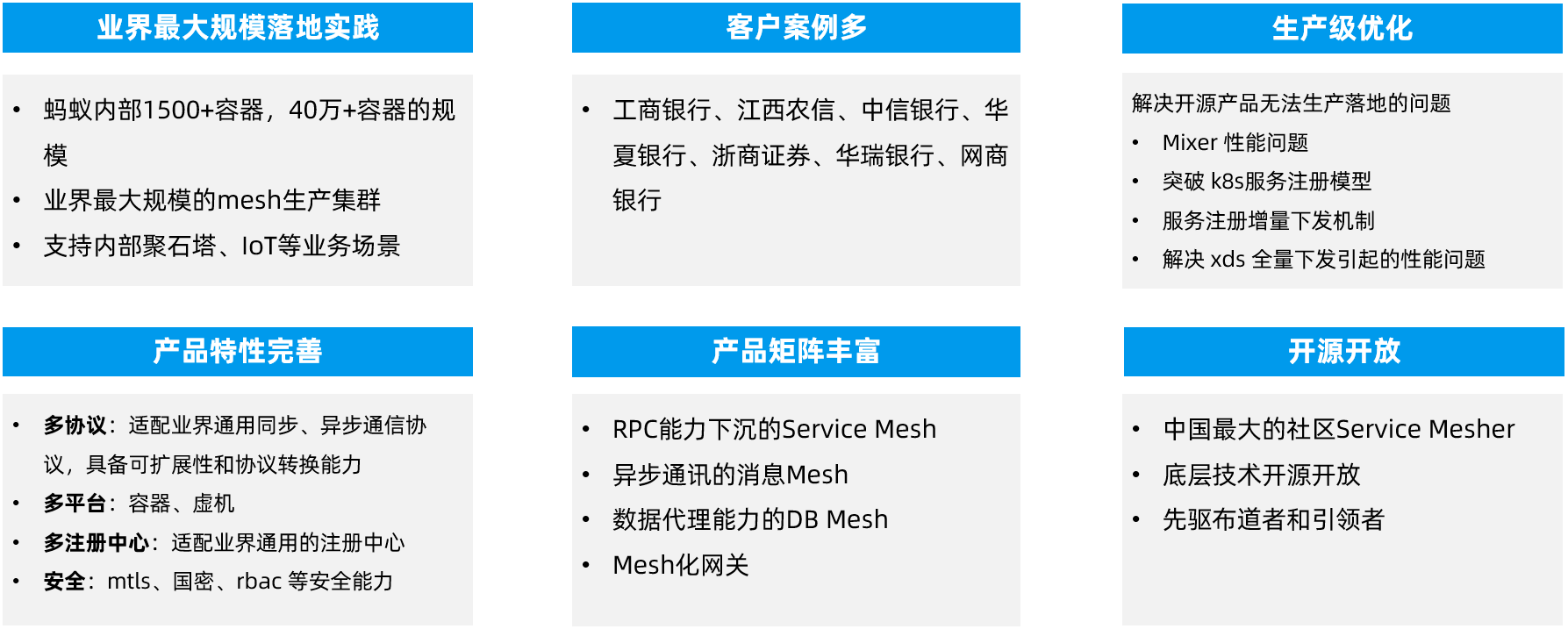
若要使用 SOFAStack 托管应用，首先需要先准备好部署环境，包括资源隔离的工作空间、安全的专有网络VPC，以及ECS服务器的安全组。

**前提条件**

* 已注册阿里云账号，并完成实名认证。
* 已 [开通 SOFAStack 服务](https://help.aliyun.com/document_detail/137419.htm)。

产品优势

在传统单体式架构向微服务架构迁移的过程中，随着应用微服务数量的增加，微服务间的通信、监控以及安全性的管理成为新的挑战。服务网格作为应用与基础设施的桥梁，突破传统的 SDK 接入方式，以对应用透明的方式处理服务之间、服务与基础设施间的通信，实现应用研发和基础设施最大程度地解耦，让应用研发只需专注于业务开发。



**技术业务分离**

在之前，微服务体系都是由中间件团队提供一个 SDK 给业务应用使用，在 SDK 中会集成各种服务治理的能力，如：服务发现、负载均衡、熔断限流、服务路由等。有了服务网格之后，我们就可以把 SDK 中的大部分能力从应用中剥离出来，拆解为独立进程，以 Sidecar 的模式部署。通过将服务治理能力下沉到基础设施，可以让业务更加专注于业务逻辑，中间件团队则更加专注于各种通用能力，真正实现独立演进，透明升级，提升整体效率。

**异构服务治理**

既能借助蚂蚁集团久经考验的微服务框架 SOFA 在云上构建微服务应用，也可以支持原生 Dubbo 和 Spring Cloud 上云，无需构建 ZooKeeper、Eureka、Consul 等微服务依赖的自建服务，极大降低运维成本。

**轻量无侵入接入**

业务应用系统通过 Service Mesh 技术架构轻量级接入，实现对应用无侵入的服务注册与服务治理方案，减少改造成本。

**跨平台支持**

该方案支持容器平台、虚拟机平台，能够满足企业用户未容器化的场景对 Service Mesh 架构转型的需求。

**网络安全**

通过网格安全，可以更方便地实现应用的身份标识和访问控制，辅之以数据加密，就能实现全链路可信，从而使得服务可以运行于零信任网络中，提升整体安全水位。

**简单易用易维护**

SOFAStack 服务网格提供集中式图形化易操作的管理平台，支持多租户管理能力，满足企业级高级特性需求，简化分布式应用的服务管理、服务治理、可观察性等能力，让用户便捷的对应用服务统一管理和治理。

* 1. 腾讯 TSF Service Mesh

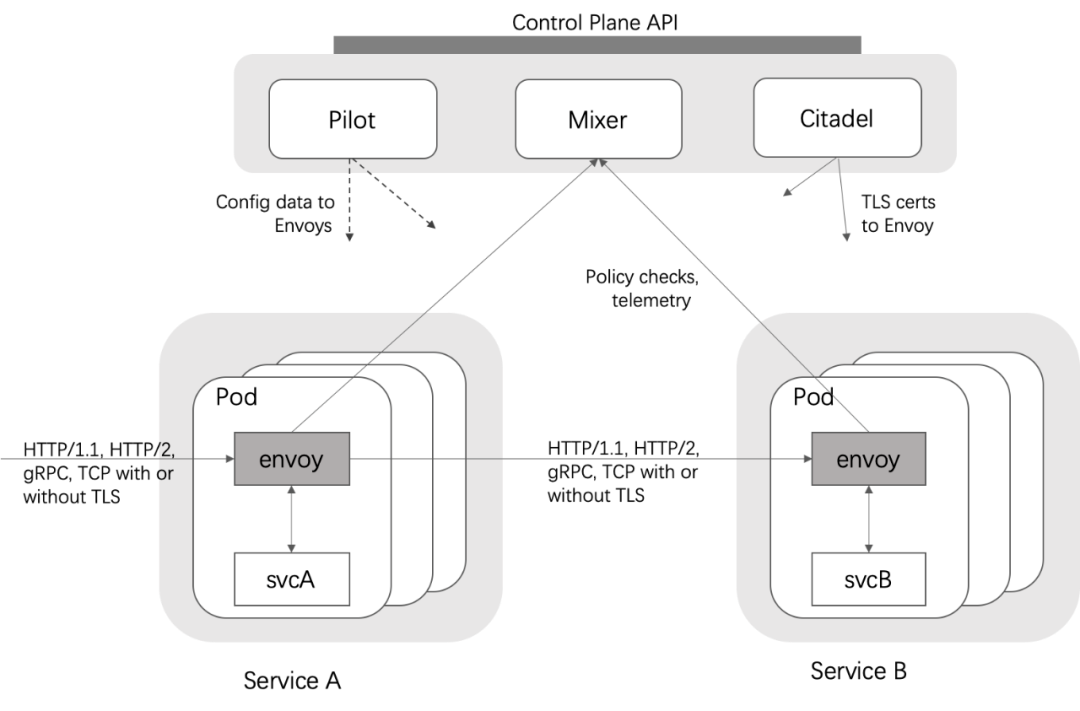
早在 2017 年，腾讯云中间件团队就选定 Istio 为技术路线，开始 Service Mesh 的相关预研和研发工作。作为腾讯微服务平台（TSF）的无侵入式微服务框架的核心实现，于 18 年初在腾讯广告平台投入，打磨稳定后，陆续开始对外输出，目前在金融、工业互联网等领域都有落地案例。

Mesh 微服务平台（Tencent Service Mesh Framework，简称 TSF Mesh），基于 Service Mesh 的理念，为应用提供服务自动注册与发现、服务路由、鉴权、限流、熔断等服务治理能力，且应用无需对源代码进行侵入式改造，即可与该服务框架进行集成。

在开发选型上，基于业界达到商用标准的开源软件 Istio 进行构建，主要原因如下：

* Istio 功能相对完备，mesh 该有的能力都有。
* 社区活跃，资源丰富，CNCF 成员，代表云原生标准化。
* Golang（Istio）& C++14（envoy）都是高性能语言，且运行起来资源使用灵活，独立性好，无 JVM 等外部依赖。

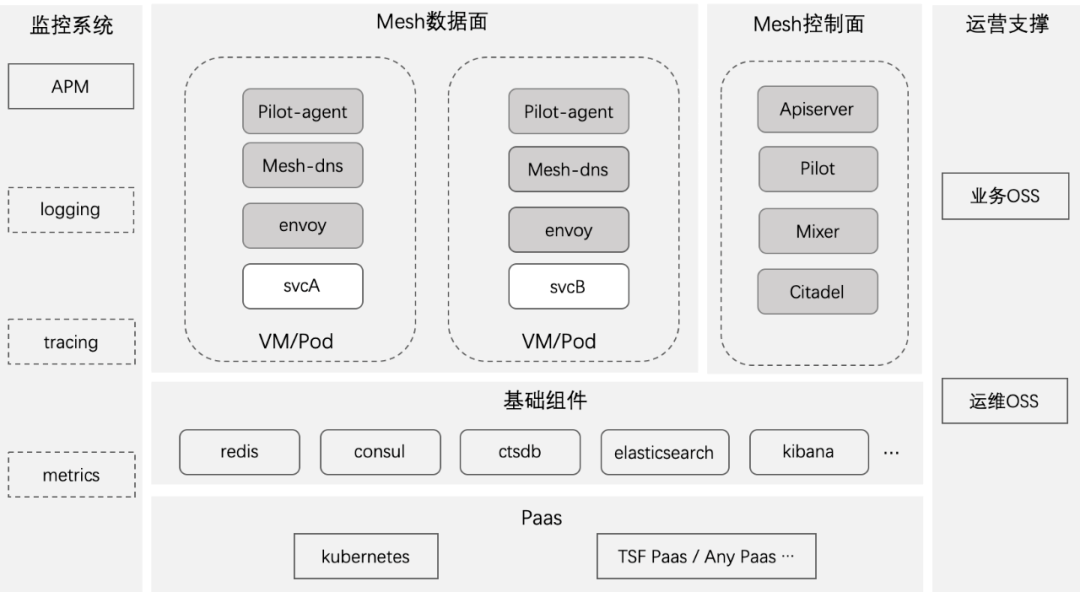
在了解 TSF Mesh 架构之前，先回顾一下 Istio 的架构图，如下图所示：



在上面的架构图中，Istio Mesh 分为两块：数据面板和控制面板。envoy 在 Istio 中扮演数据面板的角色，作为服务的代理，被部署为 sidecar，服务无需感知 envoy 的存在；控制面板包含Pilot，Mixer，Citadel等组件。这些组件的主要功能如下：

* Envoy: 作为底层的代理，通常选用其扩展版本 istio-proxy，用于调度服务网格中所有服务的出入站流量。包含了丰富的内置功能，例如动态服务发现，负载均衡，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，性能指标等。
* Pilot: 控制面的核心组件，为 Envoy 提供服务发现、智能路由（如 AB 测试、灰度发布）和弹性流量管理功能（如超时、重试、熔断），负责将高层的抽象的路由规则转化成低级的 envoy 的配置。
* Mixer: 提供策略检查和遥测功能。
* Citadel: 安全组件，提供了自动生成、分发、轮换与撤销密钥和证书功能。

TSF Mesh 整体架构上，其核心能力与开源的 Istio 保持一致，同时对 envoy、Pilot、Mixer、Pilot-agent 组件做了增强，并且新增组件 Apiserver 和 Mesh-dns。外围能力聚焦在安全性、易用性、可维护性和可观测性，如下图所示：



运营支撑提供了运营端管理和租户端管理，比如租户端的角色管理，集群管理，命名空间管理，应用管理，服务治理等；运营端提供资源管理等。监控系统提供了日志功能，链路追踪，调用链拓扑图，指标监控等。基础组件为限流、注册中心、配置中心、日志采集和实时监控提供支撑。Paas为应用部署提供支撑，比如aPaas等。

TSF Mesh 保留 Istio 所有的原生特性，同时对 Service Mesh 叠加了部分商业特性，如下：

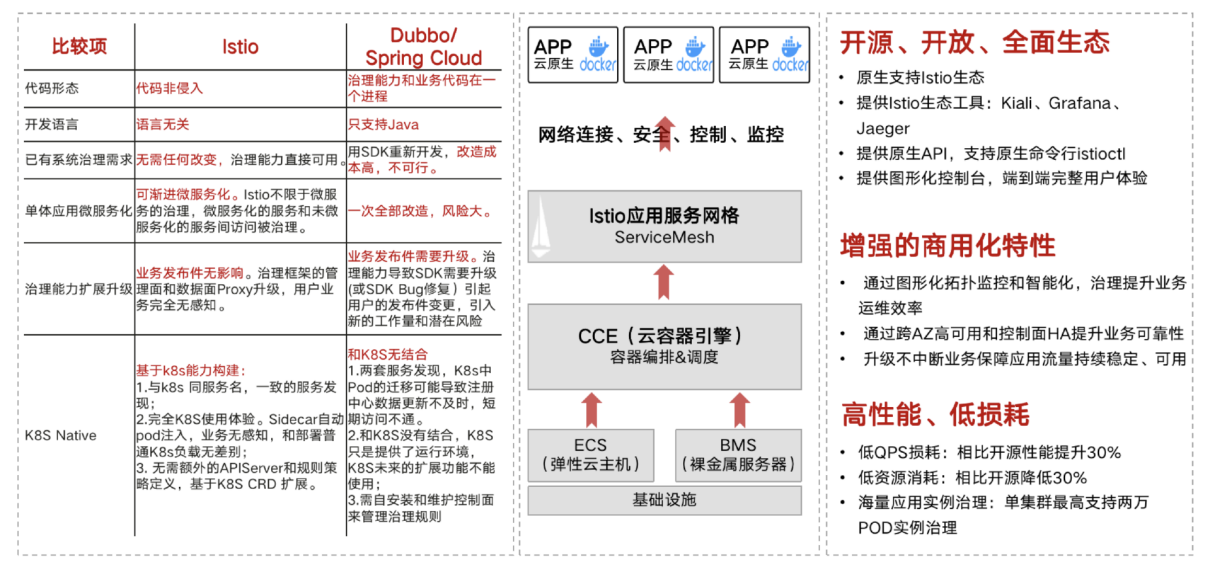
* 平台解耦：支持K8S/VM/裸金属服务器环境
* 新旧兼容：支持 Spring Cloud 应用、Service Mesh 应用互通，统一治理
* 多租户隔离、管理支持
* 调用链、日志、监控落盘
* 高可用性

TSF Mesh 作为腾讯微服务平台 TSF 的 Service Mesh 解决方案，在持续交付中，帮助企业客户解决传统集中式架构转型的困难，打造大规模高可用的分布式系统架构，实现业务、产品的快速落地。

TSF Mesh 团队拥抱开源协同，努力跟进 Service Mesh 的技术发展趋势，积极参与社区贡献。就技术发展趋势，有些点仍值得后续探讨，比如控制面单体化，UDPA（通用数据平面API）的标准化演进，wasm 在 envoy 中扮演的角色，mixer 下沉，协议扩展，性能优化等等。

* 1. 华为 ASM

应用服务网格ASM提供非侵入式的微服务治理解决方案，支持完整的生命周期管理和流量治理，兼容Kubernetes和Istio生态，功能包括负载均衡、熔断、限流等多种治理能力。并内置金丝雀、蓝绿灰度发布流程，提供一站式自动化的发布管理。



应用服务网格控制面组件（Istio-master）每Kubernetes集群部署一套，可以独占用户节点，也可以与用户节点共享，都使用租户内的ECS或BMS节点。数据面Envoy和业务容器部署在同一个Pod中，在创建Pod时自动注入。

简单来说，ASM是封装了Istio，产品功能支持：

多种灰度发布策略，流量治理，安全认证策略，可观察性，多集群服务治理等等。

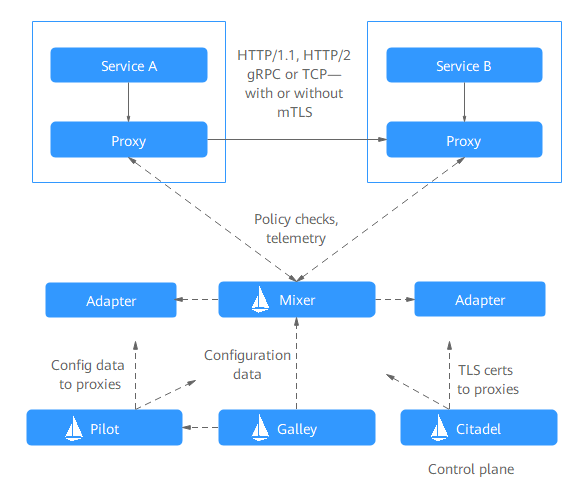
网格数据面服务框架上支持：

Springcloud：解决方案支持Springcloud SDK开发的服务在网格上统一管理。

Dubbo：解决方案支持DubboSDK开发的服务在网格上统一管理。

ylq……

应用服务网格架构



* 1. 美团 OCTO Mesh

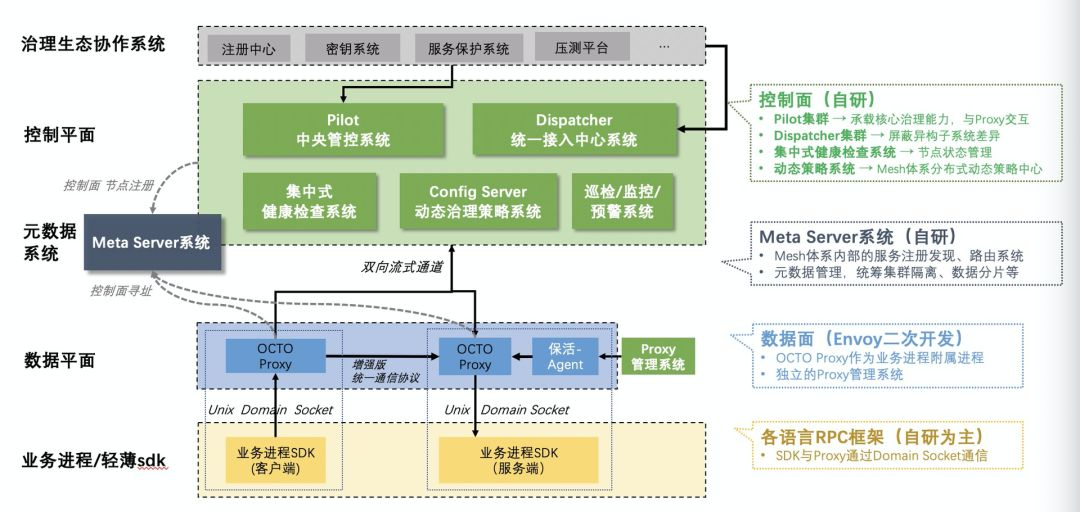
美团 OCTO Mesh 采用数据面基于 Envoy 二次开发、控制面自研为主、SDK 协同升级的方案。

架构简介如下：

各语言轻薄的 SDK 与 Proxy 通过 UDS（Unix Domain Socket）交互，主要出发点是考虑到相比透明流量劫持，UDS 性能与可运维性更好。控制面与 Proxy 通过双向流通信，控制面与治理生态的多个子系统交互，并将计算处理过的治理数据及策略数据下发给 Proxy 执行，协同配合完成路由、限流等所有核心治理功能。

控制面内部的 5 个模块都是自研的独立服务。Pilot 承载核心治理能力，与 Proxy 直接交互。Dispatcher 负责屏蔽异构子系统差异。集中式健康检查管理节点状态。Config Server 管理 Mesh 体系内相关的策略，并将 Pilot 有状态的部分尽量迁移出来。

监控及巡检系统负责提升稳定性。自研了的 Meta Server 系统实现 Mesh 体系内部的节点注册和寻址，通过管理控制面与数据面的链接关系，也实现了按事业群隔离、水平扩展等能力。



美团在 Service Mesh 落地进程中面临的“兼容性”、“异构性”、“规模化”、“交易属性业务容错性低”这四类复杂性挑战，针对上述挑战，最终实现了大规模私有云集群场景下的优化及实践方案。

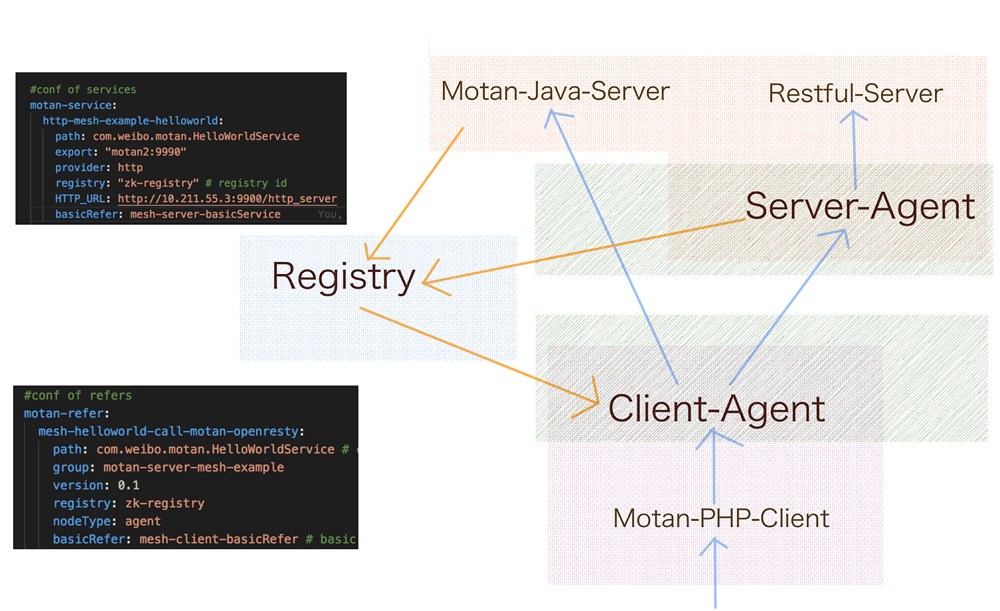
基于上述实践，目前美团线上落地服务数超过 600，线下服务数超过 3500+，初步验证了模式的可行性。短期价值方面，支持了摩拜等异构治理体系的快速融合、多语言治理能力的统一；长期价值方面，在标准化服务治理运行时并与业务解耦、中心化管控下更丰富的治理能力输出两个方面，则继续在实践中探索与验证。

* 1. 新浪微博 Weibo Mesh

在微博单体架构走向微服务改造的过程中，自研了微博自己的 RPC 框架 Motan，结合注册中心，实现了业务的解耦和服务的高效治理。之后， Motan 经历了多次热点事件和三节高峰的严峻考验，稳定性和可靠性都得到了实际场景的验证。

随着大量按业务拆分的服务独立部署，使得服务的扩缩容操作缓慢，直接拉低了峰值流量的应对能力。为了低成本高效率地应对各种极端峰值，微博在容器化的基础上探索了公有云和私有云混合部署的模式，研发了微博 DCP 混合云平台，实现了资源的动态扩缩容，结合 Motan RPC 的服务治理实现了对流量的弹性调度。至此，微博平台在 Java 技术栈形成了配套完善的一整套服务体系，有完善的服务治理相关组件、明确的 SLA 指标、完备的 Trace、监控等体系保障微博平台的高性能高可用运转。

从 2016 年开始，微博开始探索跨语言服务化的道路。Java 和 PHP 是微博内部使用最多的两种语言，所以我们起初的跨语言是立足于微博平台的 Java 体系，探索 Java 与 PHP 之间的跨语言调用。此时就希望引入一个 Agent，来统一解决服务治理的问题，Client 只需要实现 Motan 协议解析，直接通过本机 Agent 调用远端服务即可。这便是 WeiboMesh 的雏形，也就是目前被大家所熟知的 SideCar 模式代理的 Service Mesh 实现。



在实践中，服务网格通常实现为轻量级网络代理，与应用程序部署在一起，但是对应用程序透明。我们在解决跨语言服务化中服务间调用和统一服务治理所引入的 Agent 就是这个 Mesh 层。

最终基于 Motan-go 实现了客户端服务端的双向代理，基于微博注册中心 Vintage 实现了对 Agent 的动态指令控制，完成了传输与控制的重新定义，并结合 OpenDCP 平台实现了动态流量调度和弹性扩缩容，保障了服务的高可用。

目前已经有很多核心业务完成了基于 WeiboMesh 的升级改造，比如大家经常使用的微博热搜、热门微博等。微博从 2016 年开始起步，一路摸索，最终完成了 WeiboMesh 的主体建设。

WeiboMesh数据面板主要以 SideCar 模式与 Service 部署在一起，保障请求数据在 Service 间进行可靠传输，提供服务治理以及对请求状态数据的提取与检测等。而控制面板主要控制服务间流量流向、请求路由，提供一种对数据面板访问策略实施的干预机制以及对服务依赖、服务状态的监控等功能。

WeiboMesh 的数据面板Motan-go 的 Agent 作为 WeiboMesh 的数据面板，由 Server Agent 和 Client Agent 两部分组成，Server Agent 作为服务提供方的反向代理使用，Client Agent 则作为服务依赖方调用服务的正向代理，复用了 Motan 原有的服务治理体系，比如服务注册与发现、负载均衡、高可用、服务监控、跟踪、依赖关系刻画、指令调度系统等。

WeiboMesh 的控制面板前身是 Motan 的指令控制体系，为了实现对服务流量的切换、降级、分组以及支持各种发布方式，Motan 原本就结合注册中心实现了一套完善的指令控制系统，以提供一种对流量进行干预的机制。WeiboMesh 基于这种机制并配合决策系统实现了对流量的动态控制和策略执行，同时又扩展了 Filter 机制，实现了身份验证、安全管理、监控和可视化数据收集等，收集到的遥测数据实时反馈到决策系统作为决策依据。

1. 服务网格总结

Service Mesh是2016年之后兴起的云原生技术发展方向，与之类似的还有Data Mesh，Event/Message Mesh, Serverless Mesh等，其目的都是为了能让业务开发更快，业务与中台、底层的技术间进行充分的解耦，重复利用云计算和容器化大趋势下的基础设施能力，使得大规模微服务的设计开发、运维监控、治理管控更简单高效，成本更低。

1. 参考文献

1.Phil Calçado，《Pattern: Service Mesh》，<https://philcalcado.com/2017/08/03/pattern_service_mesh.html>

2.阿里技术团队，《Service Mesh：起源、发展和现状》，<https://zhuanlan.zhihu.com/p/164730174>

3.美团技术团队，《复杂环境下落地Service Mesh的挑战与实践》，<https://zhuanlan.zhihu.com/p/338971268>

4.K8S生态，《什么是服务网格》，<https://zhuanlan.zhihu.com/p/61901608>

5.阿里云云栖号，《Dapr 在阿里云原生的实践》，<https://zhuanlan.zhihu.com/p/394661997>

6.腾讯云中间件团队，《腾讯云中间件团队在Service Mesh中的实践与探索》，<https://cloud.tencent.com/developer/article/1805886>

7.阿里云文档，《金融分布式架构 SOFAStack》，<https://help.aliyun.com/document_detail/200294.html>

8.华为云文档，《什么是应用服务网格》，<https://support.huaweicloud.com/productdesc-istio/istio_productdesc_0001.html>