

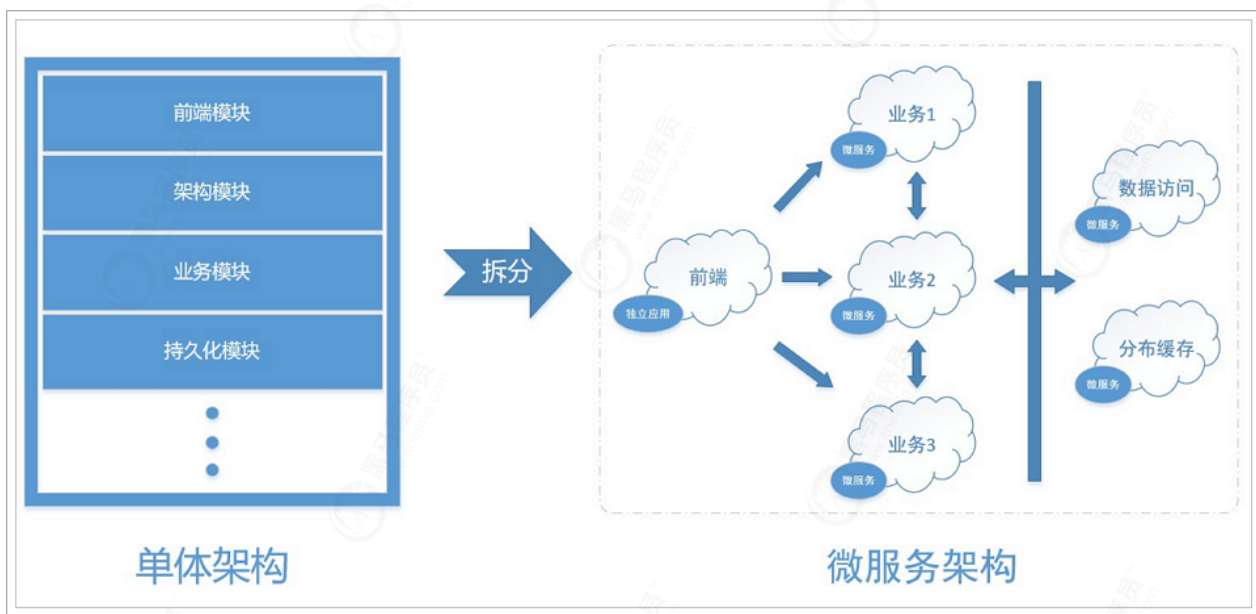
课程说明

- 目前微服务架构面临的一些挑战
- 技术架构演进
- 了解什么是Service Mesh
- Service Mesh产品
- 了解Istio
- Istio快速入门
- 体验Istio

1、Service Mesh简介

1.1、目前微服务架构面临的一些挑战

目前，微服务的架构方式在企业中得到了极大的发展，主要原因是其解决了传统的单体架构中存在的问题。



当单体架构拆分成微服务架构就可以高枕无忧了吗？显然不是的。

微服务架构体系中同样也存在很多的挑战，比如：

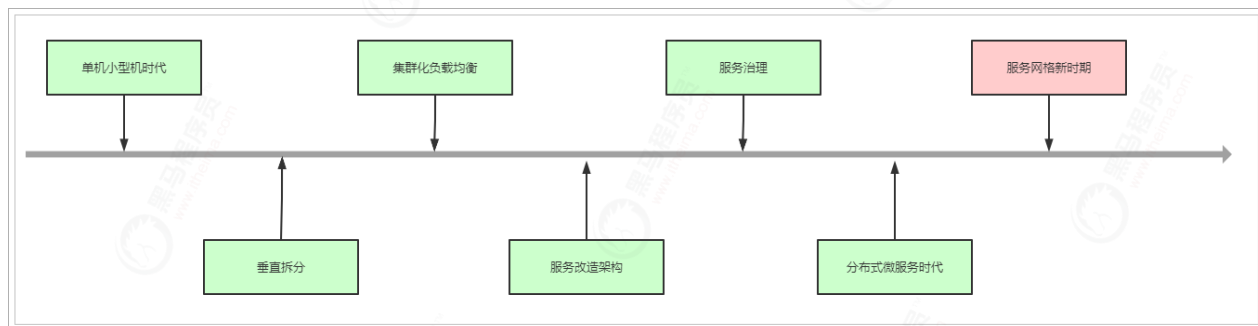
- 原来的单个应用拆分成了许多分散的微服务，它们之间相互调用才能完成一个任务，而一旦某个过程出错（组件越多，出错的概率也就越大），就非常难以排查。
- 如果用户请求的响应太慢，我们就需要知道到底哪些地方比较慢？整个链路的调用各阶段耗时是多少？哪些调用是并发执行的，哪些是串行的？这些问题需要我们能非常清楚整个集群的调用以及流量情况。
- 微服务拆分成这么多组件，如果单个组件出错的概率不变，那么整体有地方出错的概率就会增大。服务调用的时候如果没有错误处理机制，那么会导致非常多的问题。
- 应用数量的增多，对于日常的应用发布来说也是个难题。应用的发布需要非常谨慎，如果应用都是一次性升级的，出现错误会导致整个线上应用不可用，影响范围太大。
- 很多情况我们需要同时存在不同的版本，使用 AB 测试验证哪个版本更好。

- 如果版本升级改动了 API，并且互相有依赖，那么我们还希望能自动地控制发布期间不同版本访问不同的地址。这些问题都需要智能的流量控制机制。
- 为了保证整个系统的安全性，每个应用都需要实现一套相似的认证、授权、HTTPS、限流等功能。

没错，Service Mesh就是为了解决以上问题才出现的。这就是我们学习本套课程的目的。

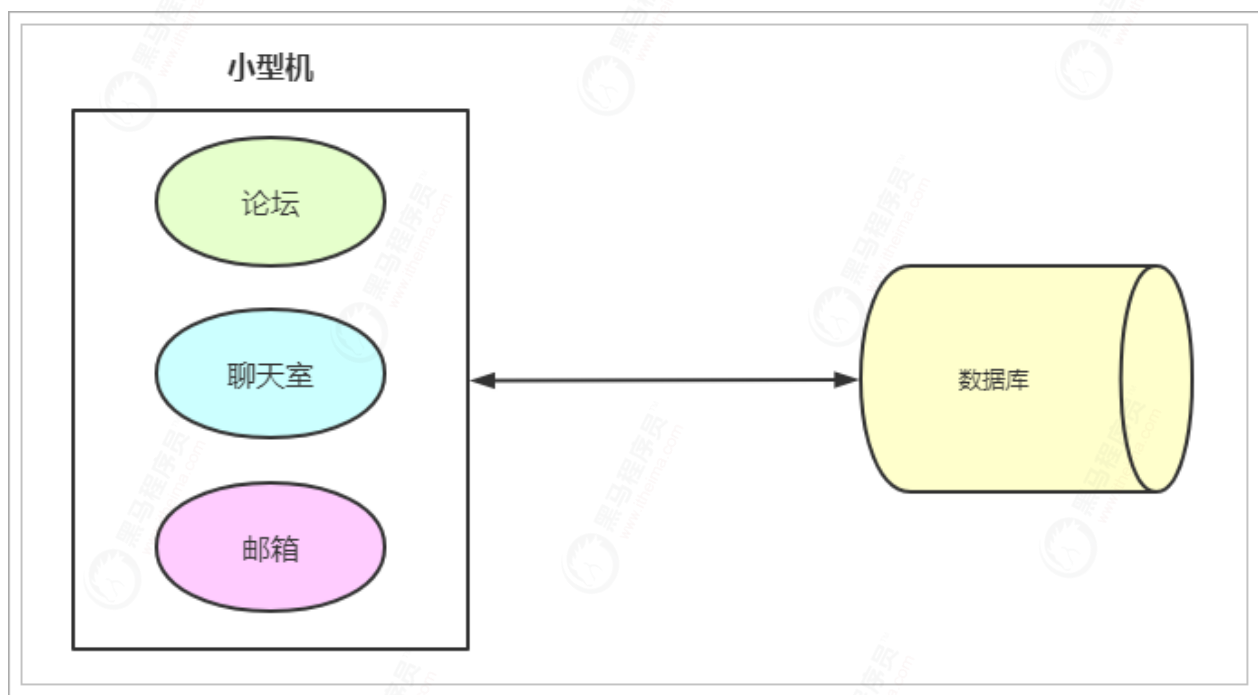
1.2、技术架构演进

1.2.1、发展历史时间轴



1.2.2、单机小型机时代

第一个计算机网络诞生于1969年，也就是美军的阿帕网，阿帕网能够实现与其它计算机进行联机操作，但是早期仅仅是为了军事目的而服务 2000年初，中国的网民大约890万，很多人都不知道互联网为何物，因此大多数服务业务单一且简单，采用典型的单机+数据库模式，所有的功能都写在一个应用里并进行集中部署。



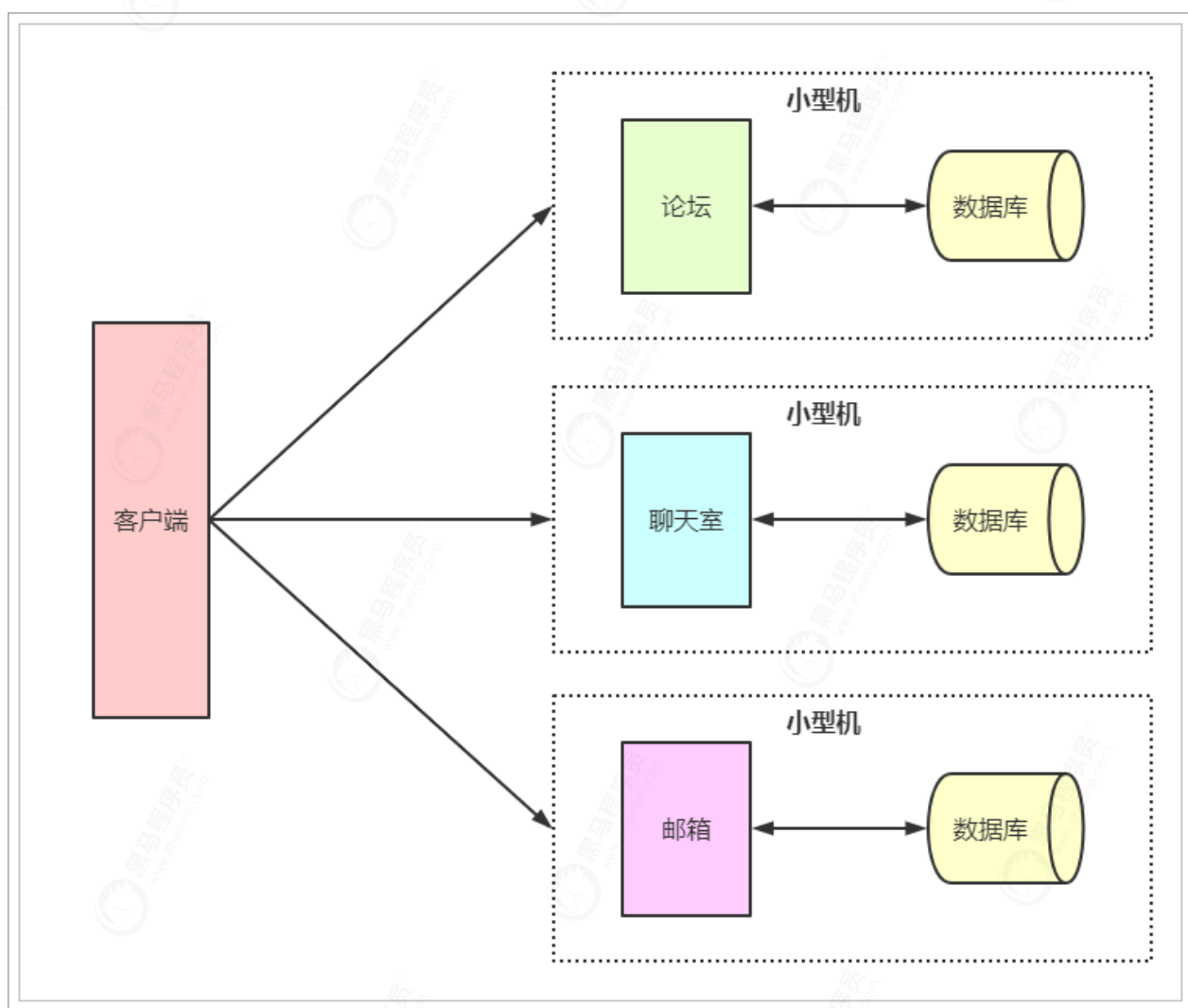
说明：论坛业务、聊天室业务、邮箱业务全部都耦合在一台小型机上面，所有的业务数据也都存储在一台数据库上。

1.2.3、垂直拆分

- 随着应用的日益复杂与多样化，开发者对系统的容灾，伸缩以及业务响应能力有了更高的要求，如果小型机和数据库中任何一个出现故障，整个系统都会崩溃，若某个板块的功能需要更新，那么整

个系统都需要重新发布，显然，对于业务迅速发展的万物互联网时代是不允许的。

- 如何保障可用性的同时快速响应业务的变化，需要将系统进行拆分，将上面的应用拆分成多个子应用。



优点：应用间进行了解耦，系统容错提高了，也解决了独立应用发布的问题。

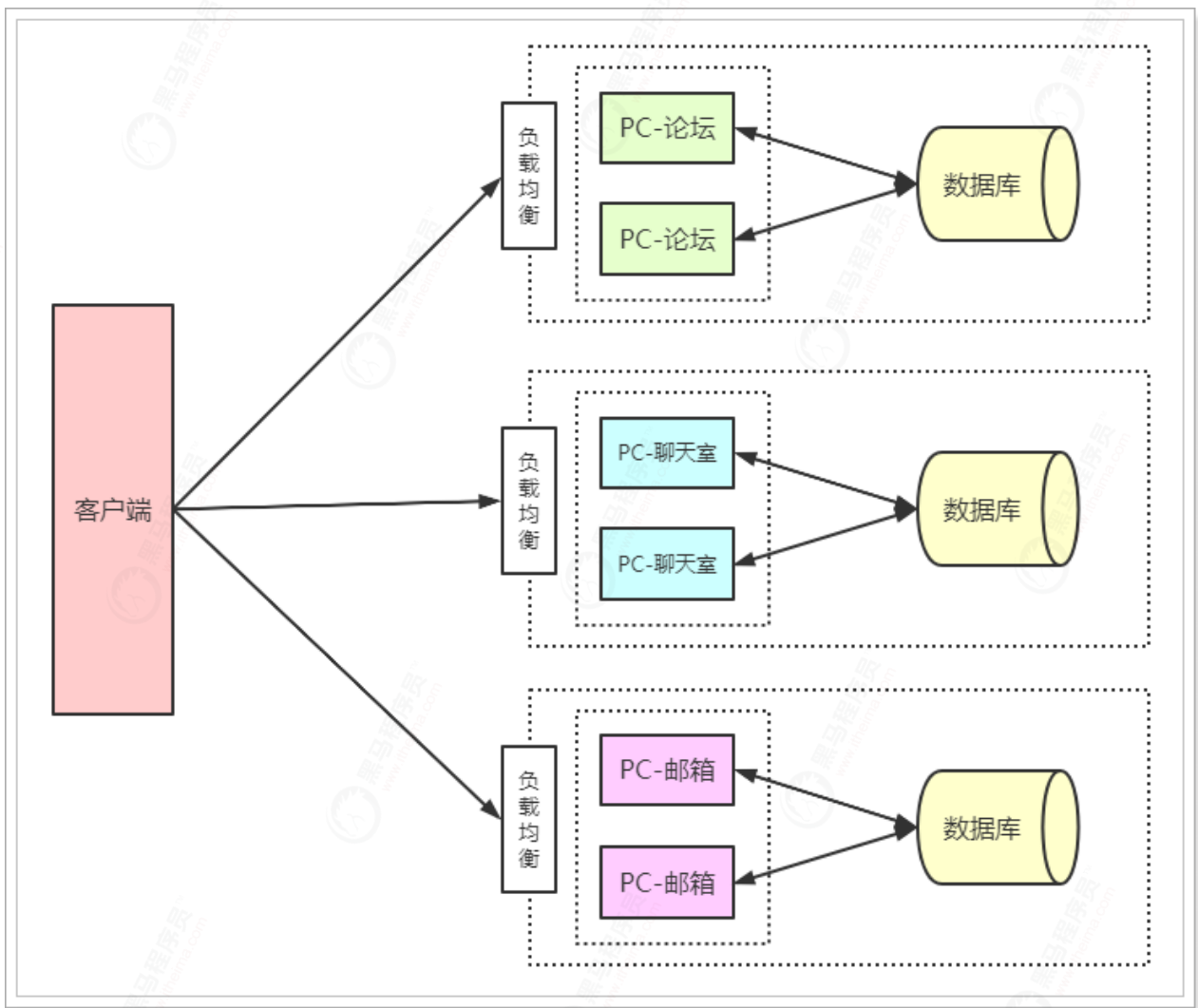
应用垂直拆分解决了应用发布的问题，但是随着用户数量的增加，单机的计算能力依旧是杯水车薪。

1.2.4、集群化负载均衡架构

用户量越来越大，就意味着需要更多的小型机，但是小型机价格昂贵，操作维护成本高。此时更优的选择是采用多台PC机部署同一个应用的方案，但是此时就需要对这些应用做负载均衡，因为客户端不知道请求会落到哪一个后端PC应用上的。

负载均衡可以分为硬件层面和软件层面。硬件层面：F5 软件负载层面：LVS、Nginx、Haproxy
负载均衡的思想：对外暴露一个统一的接口，根据用户的请求进行对应规则转发，同时负载均衡还可以做限流等等

有了负载均衡之后，后端的应用可以根据流量的大小进行动态扩容，我们称为"水平扩展"。

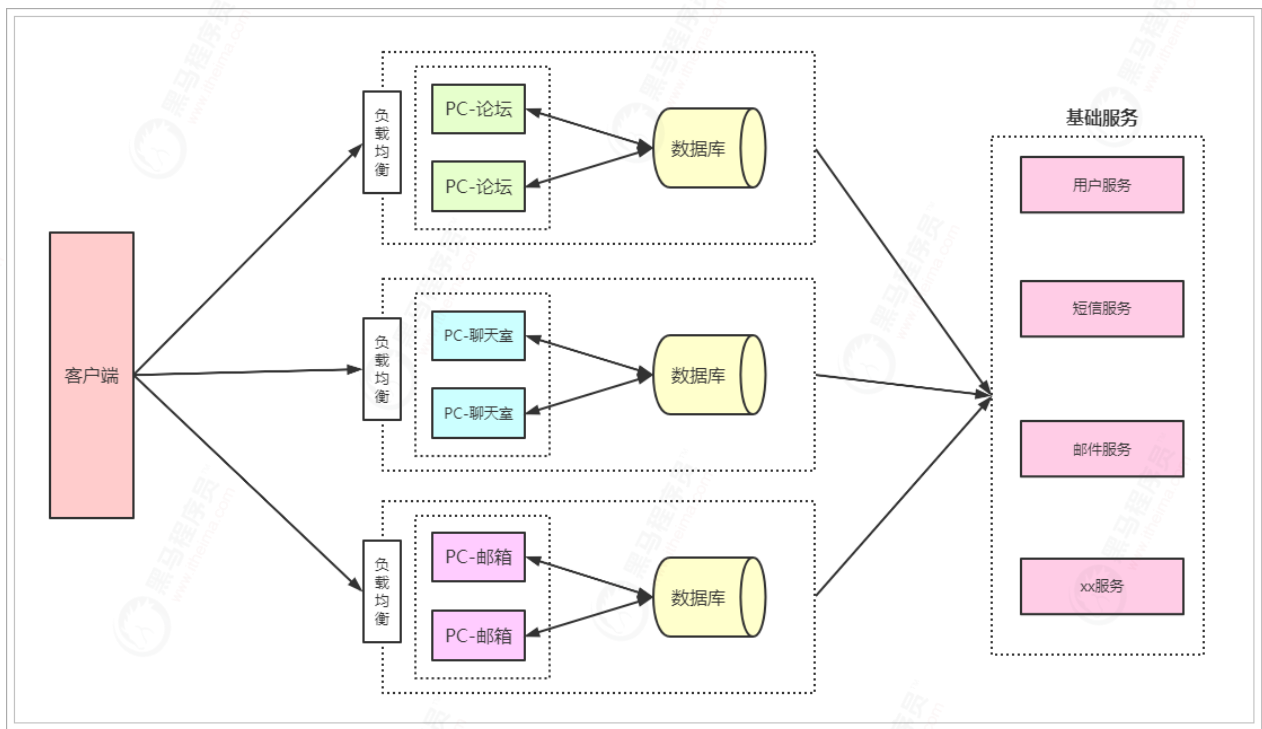


阿里巴巴在2008提出去“IOE”，也就是IBM小型机、Oracle数据库，EMC存储，全部改成集群化负载均衡架构，在2013年支付宝最后一台IBM小型机下线。

优点：通过水平扩展，增强了系统的并发能力。

1.2.5、服务化改造架构

虽然系统经过了垂直拆分，但是拆分之后发现在论坛和聊天室中有重复的功能，比如，用户注册、发邮件等等，一旦项目大了，集群部署多了，这些重复的功能无疑会造成资源浪费，所以会把重复功能抽取出来，名字叫“XX服务（Service）”。



为了解决服务跟服务如何相互调用，需要一个程序之间的通信协议，所以就有了远程过程调用（RPC），作用就是让服务之间的程序调用变得像本地调用一样的简单。

优点：在前面的架构之上解决了业务重用的问题。

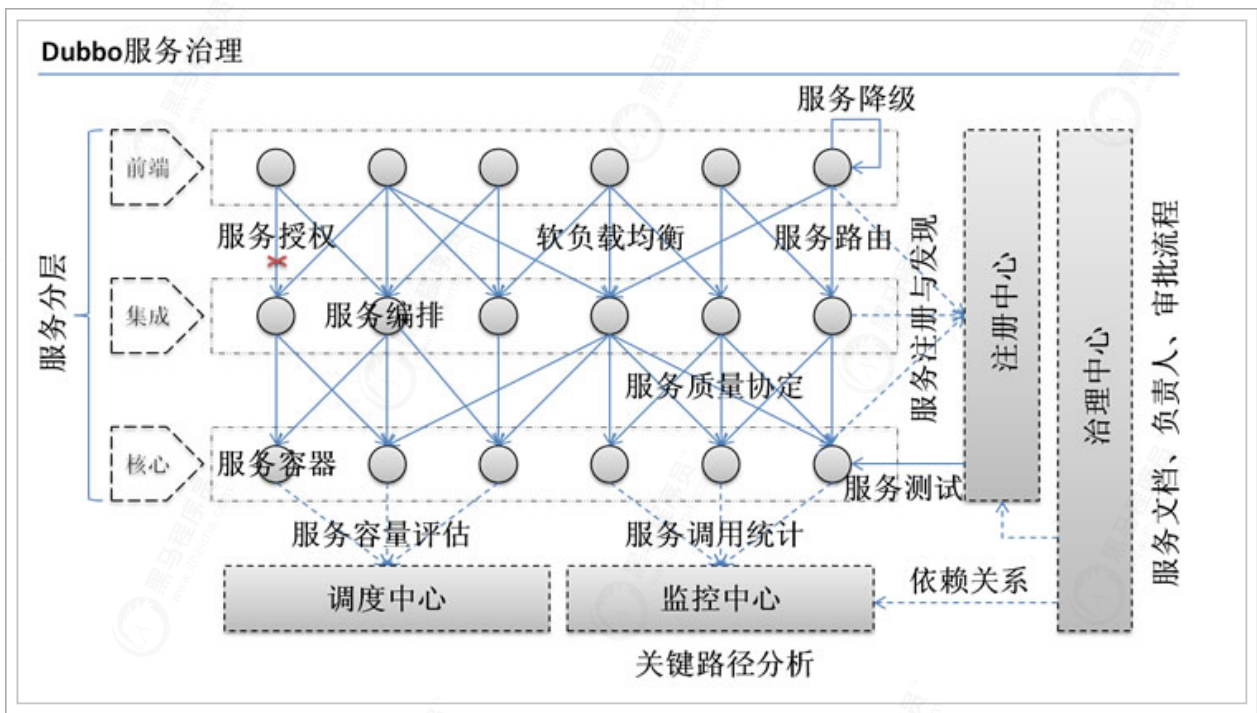
1.2.6、服务治理

随着业务的增大，基础服务越来越多，调用网的关系由最初的几个增加到几十上百，造成了调用链路错综复杂,需要对服务进行治理。

服务治理要求：

- 当我们服务节点数十上百的时候，需要对服务有动态的感知，引入了注册中心。
- 当服务链路调用很长的时候如何实现链路的监控。
- 单个服务的异常，如何能避免整条链路的异常（雪崩），需要考虑熔断、降级、限流。
- 服务高可用：负载均衡。

典型框架比如有：Dubbo，默认采用的是Zookeeper作为注册中心。



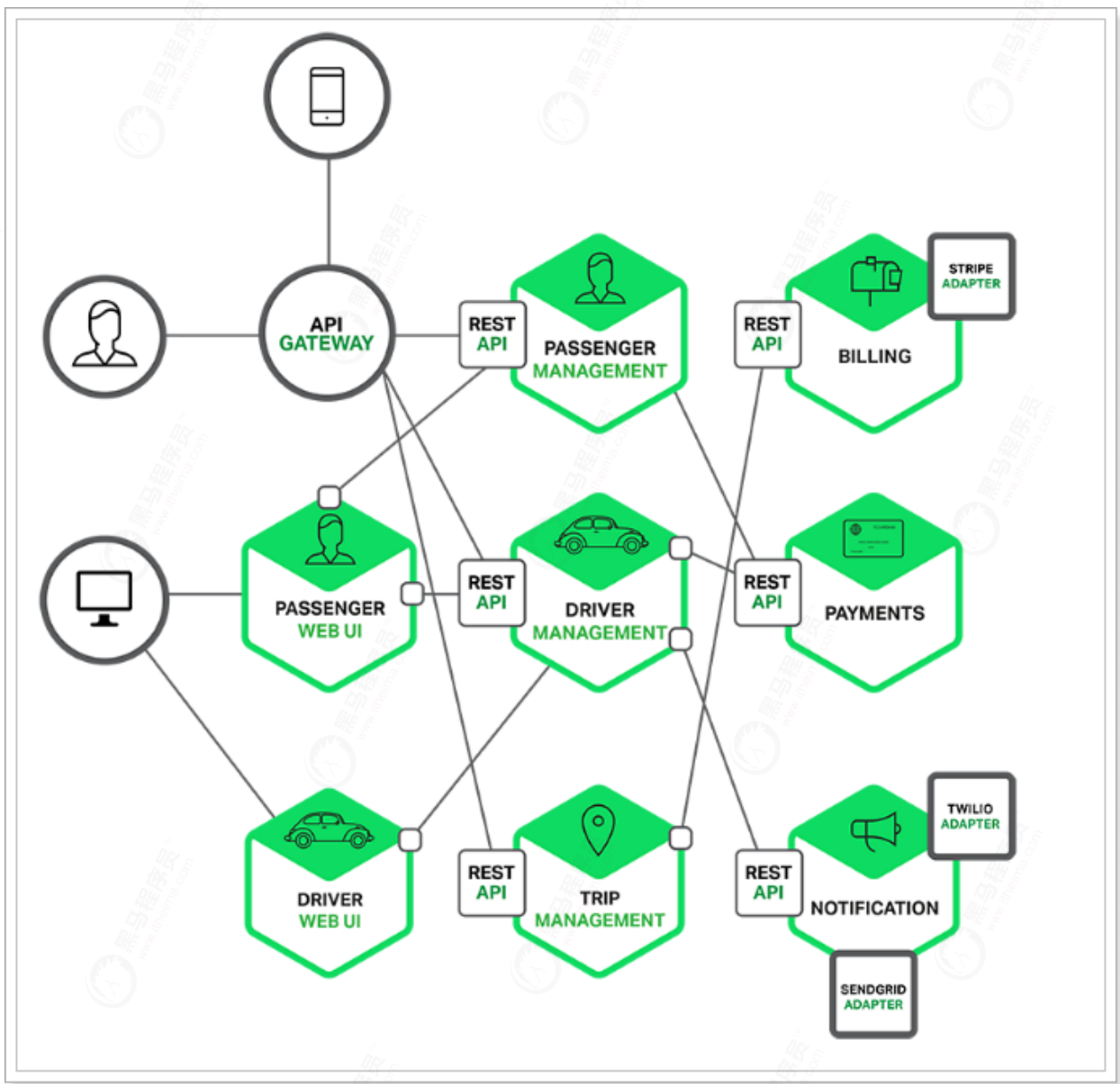
1.2.7、微服务时代

微服务是在2012年提出的概念，微服务的希望的重点是一个服务只负责一个独立的功能。

拆分原则，任何一个需求不会因为发布或者维护而影响到不相关的服务，一切可以做到独立部署运维。

比如传统的“用户中心”服务，对于微服务来说，需要根据业务再次拆分，可能需要拆分成“买家服务”、“卖家服务”、“商家服务”等。

典型代表：Spring Cloud，相对于传统分布式架构，SpringCloud使用的是HTTP作为RPC远程调用，配合上注册中心Eureka和API网关Zuul，可以做到细分内部服务的同时又可以对外暴露统一的接口，让外部对系统内部架构无感，此外Spring Cloud的config组件还可以把配置统一管理。



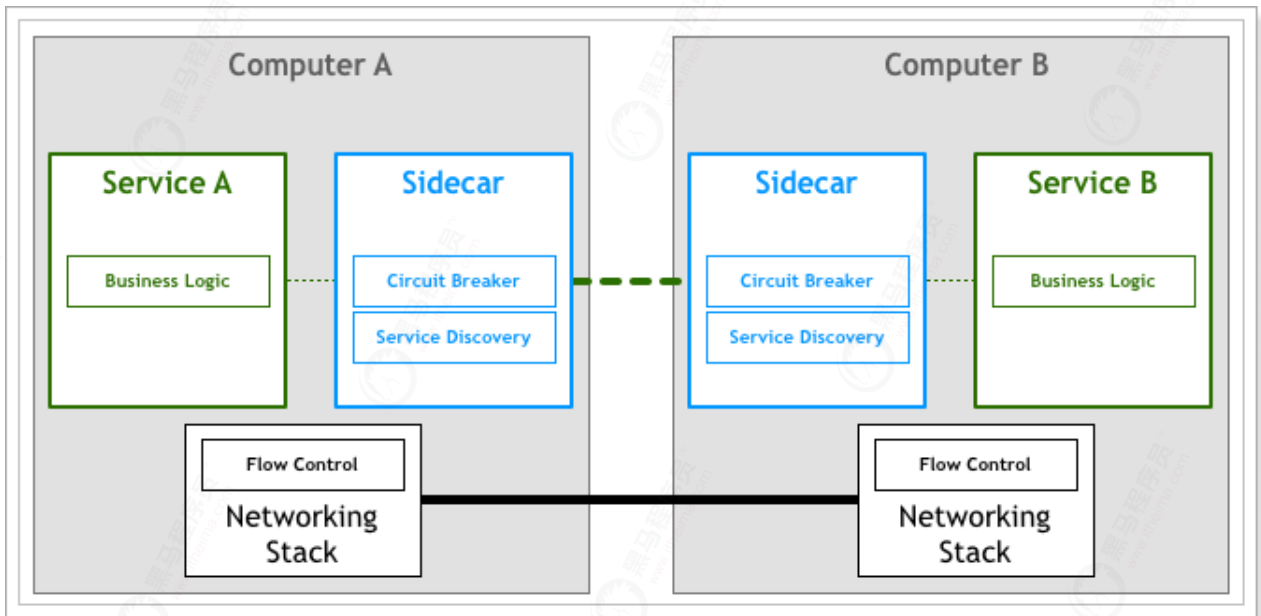
Spring Cloud微服务架构存在的不足：

- Spring Cloud属于侵入式框架，在项目中需要添加spring cloud maven依赖，加上spring cloud组件注解，写配置，打成jar的时候还必须要将非业务的代码也要融合在一起。
- 微服务中的服务支持不同语言开发，也需要维护不同语言和非业务代码的成本；
- 业务代码开发者应该把更多的精力投入到业务熟悉度上，而不应该是非业务上，Spring Cloud虽然能解决微服务领域的很多问题，但是学习成本还是较大的。
- 互联网公司产品的版本升级是非常频繁的，为了维护各个版本的兼容性、权限、流量等，因为Spring Cloud是“代码侵入式的框架”，这时候版本的升级就注定要让非业务代码一起，一旦出现问题，再加上多语言之间的调用，工程师会非常痛苦。
- 我们已经感觉到了，服务拆分的越细，只是感觉上轻量级解耦了，但是维护成本却越高了。

1.2.8、服务网格新时期（Service Mesh）

Service Mesh主要解决的问题就希望开发人员对于业务的聚焦，服务发现、服务注册、负载均衡等对于开发人员透明，可以更加专注业务逻辑的实现。

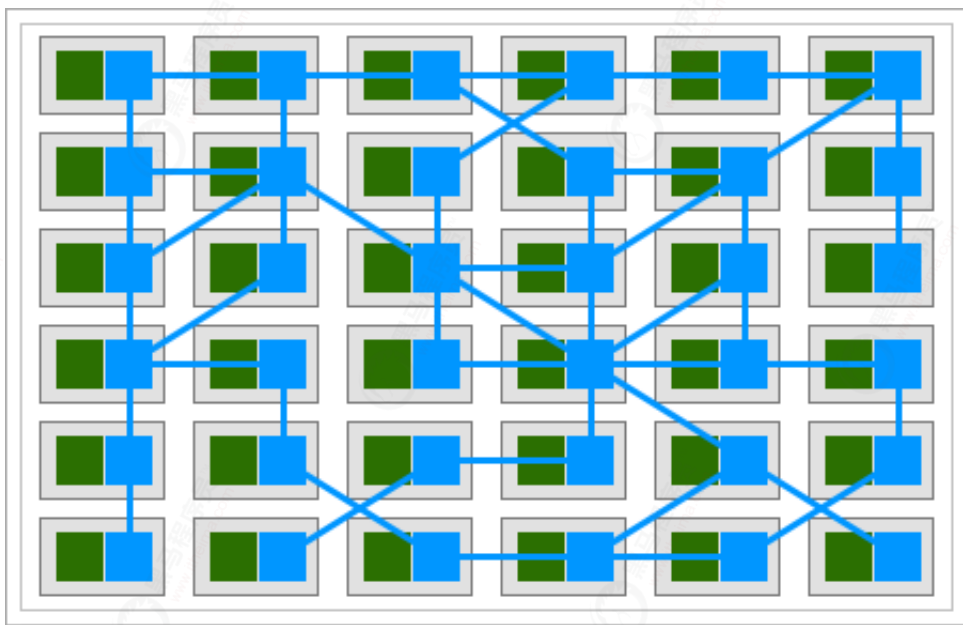
如果将为微服务提供通信服务的这部分逻辑从应用程序进程中抽取出来，作为一个单独的进程进行部署，并将其作为服务间的通信代理，可以得到如下图所示的架构：



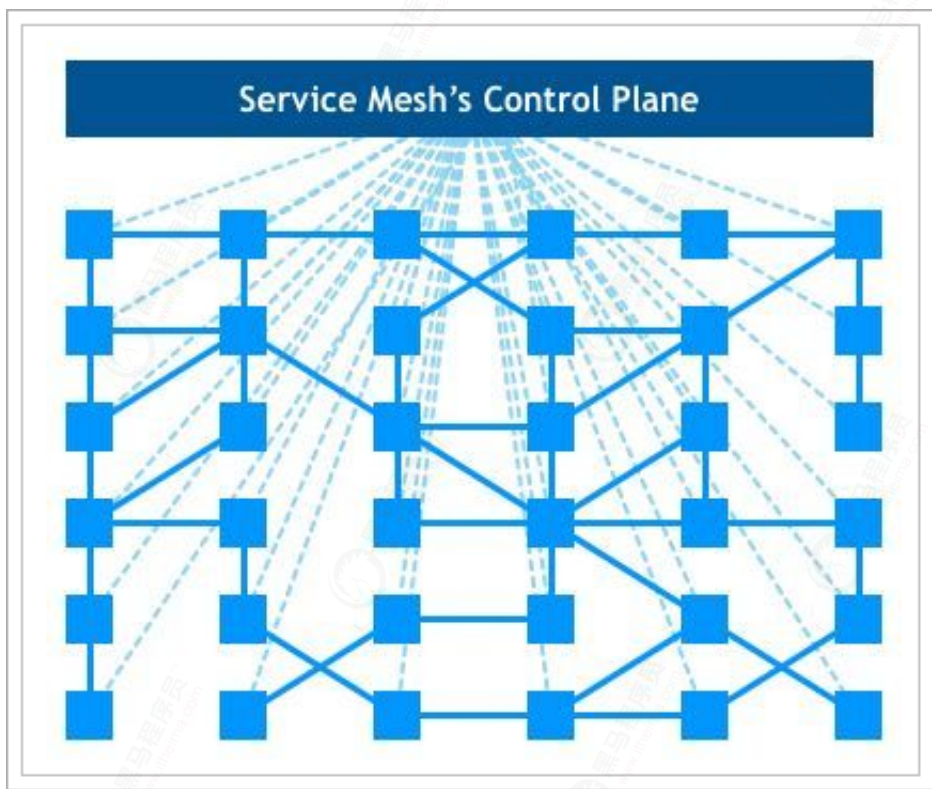
Sidecar，翻译成中文是边车，非常的形象。



当服务大量部署时，随着服务部署的Sidecar代理之间的连接形成了一个如下图所示的网格，该网格成为了微服务的通讯基础设施层，承载了微服务之间的所有流量，被称之为Service Mesh（服务网格）。

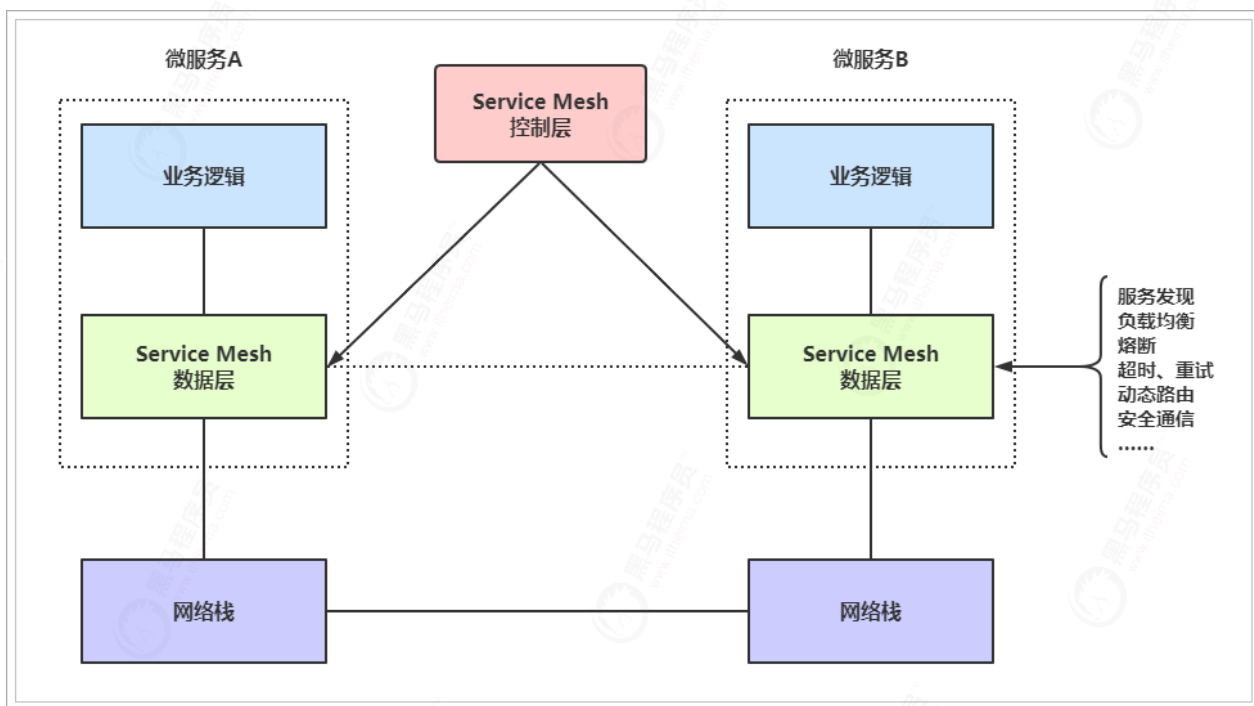


服务网格中有数量众多的Sidecar代理，如果对每个代理分别进行设置，工作量将非常巨大。为了方便地对服务网格中的代理进行统一集中控制，在服务网格上增加了控制面组件。



1.3、什么是Service Mesh

服务网格用来描述组成这些应用程序的微服务网络以及它们之间的交互。随着服务网络的规模和复杂性不断增长，它将会变得越来越难以理解和管理。它的需求包括服务发现、负载均衡、故障恢复、度量和监控等。服务网格通常还有更复杂的运维需求，比如 A/B 测试、金丝雀发布、速率限制、访问控制和端到端认证。



1.4、Service Mesh产品

1.4.1、CNCF

CNCF 是一个开源软件基金会，致力于使云原生计算具有普遍性和可持续性。云原生计算使用开源软件技术栈将应用程序部署为微服务，将每个部分打包到自己的容器中，并动态编排这些容器以优化资源利用率。云原生技术使软件开发人员能够更快地构建出色的产品。

官网：<https://www.cncf.io/>

常用的已经毕业的云原生项目：

- Kubernetes
 - Kubernetes 是世界上最受欢迎的容器编排平台也是第一个 CNCF 项目。Kubernetes 帮助用户构建、扩展和管理应用程序及其动态生命周期。
- Prometheus
 - Prometheus 为云原生应用程序提供实时监控、警报包括强大的查询和可视化能力，并与许多流行的开源数据导入、导出工具集成。
- Jaeger
 - Jaeger 是由 Uber 开发的分布式追踪系统，用于监控其大型微服务环境。Jaeger 被设计为具有高度可扩展性和可用性，它具有现代 UI，旨在与云原生系统（如 OpenTracing、Kubernetes 和 Prometheus）集成。
- Envoy
 - Envoy 是最初在 Lyft 创建的 Service Mesh（服务网格），现在用于 Google、Apple、Netflix 等公司内部。Envoy 是用 C++ 编写的，旨在最大限度地减少内存和 CPU 占用空间，同时提供诸如负载均衡、网络深度可观察性、微服务环境中的跟踪和数据库活动等功能。
- Containerd
 - Containerd 是由 Docker 开发并基于 Docker Engine 运行时的行业标准容器运行时组件。作为容器生态系统的选择，Containerd 通过提供运行时，可以将 Docker 和 OCI 容器镜像作为新平台或产品的一部分进行管理。

1.4.2、Linkerd



Linkerd是Buoyant公司2016年率先开源的高性能网络代理程序，是业界的第一款Service Mesh产品，甚至可以说Linkerd的诞生标志着Service Mesh时代的开始，其引领后来Service Mesh的快速发展。

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，比如网络不可靠、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，每秒以最小的时延及负载处理万级请求，易于水平扩展，经过生产线测试及验证，可运行任何平台的产线级Service Mesh工具。

1.4.3、Envoy



Envoy也是一款高性能的网络代理程序，于2016年10月份由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，当然，也作为内部服务间通信代理，实现服务间可靠通信。

Envoy的实现借鉴现有产线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公司产线实践证明，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，实现平台及语言无关。

1.4.4、Istio



2017年5月24日，Google, IBM 和 Lyft 共同发布 Istio 的第一个公开版本(0.1)。Istio为一款开源的为微服务提供服务间连接、管理以及安全保障的平台软件，支持运行在Kubernetes、Mesos等容器管理工具，但不限于Kubernetes、Mesos，其底层依赖于Envoy。

Istio提供一种简单的方法实现服务间的负载均衡、服务间认证、监控等功能，而且无需应用层代码调整。其控制平面由Pilot、Citadel 和 Galley组成，数据平面由Envoy实现，通常情况下，数据平面代理Envoy以sidecar模式部署，使得所有服务间的网络通信均由Envoy实现，而Istio的控制平面则负责服务间流量管理、安全通信策略等功能。

1.4.5、Conduit



Conduit于2017年12月发布，作为由Buoyant继Linkerd后赞助的另一个开源项目。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。

1.4.6、国内产品

国内很多团队也已经在着手研究了，这些团队主要分为四类体系：

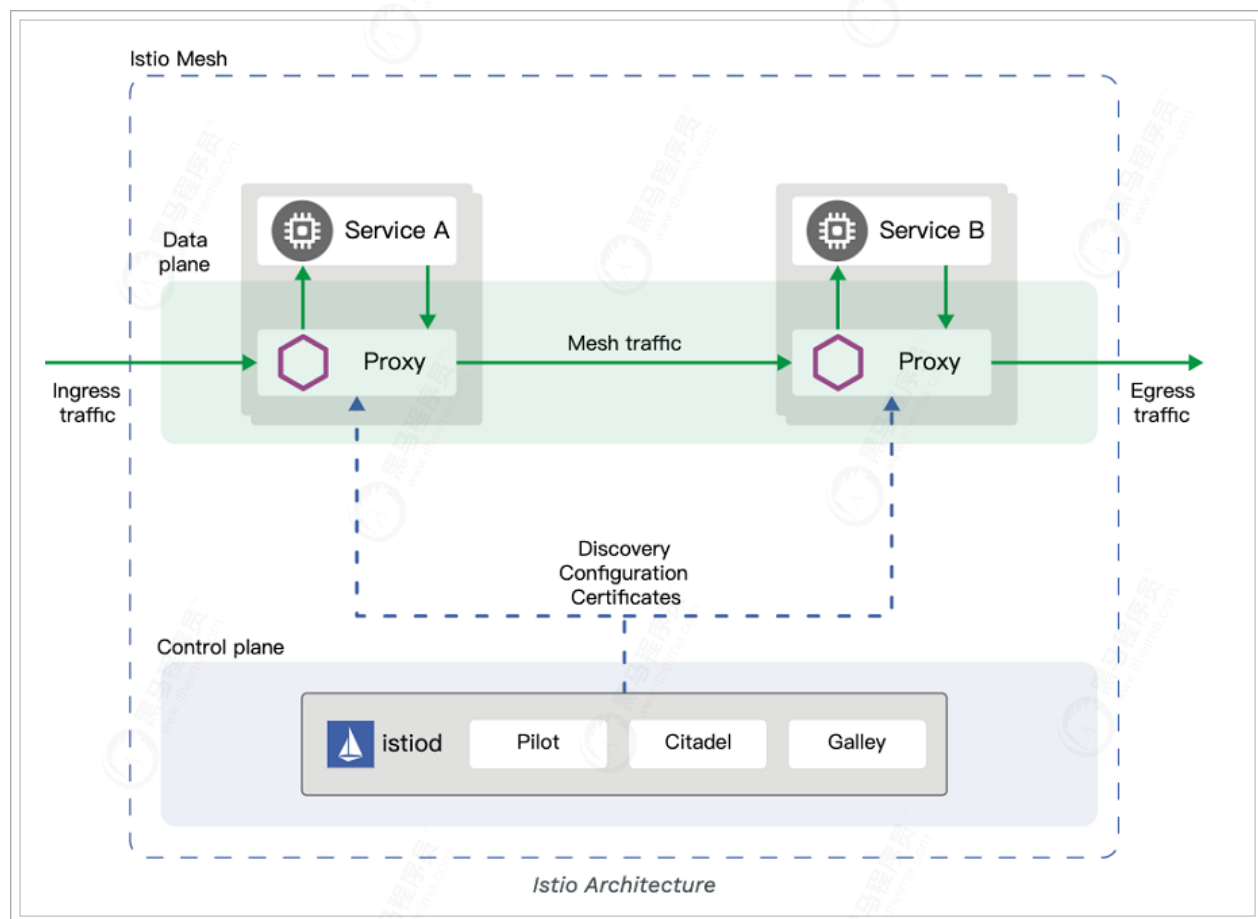
- **以蚂蚁金服为首的开源系：**蚂蚁金服自研的 SOFA（Scalable Open Financial Architecture）Mesh 在开始的时候走的就是开源路线，他们参考了 Istio 及 Envoy 的设计思想，重新实现了自己的 Service Mesh 系统，旁路网关（Sidecar）基于 Go 语言，该系统的前身是已经开源的 SOFA RPC 框架。蚂蚁金服于 2018 年 7 月正式将其开源，正式的可以用于生产的框架可能还需要一些时间。
- **以华为为代表的自研系：**华为可能在 Service Mesh 概念出来前就已经有类似的想法了，只是没有抽取出一个公共的概念。无论是华为早期的 HSA 还是之后出现的 CSE Mesher，都是对 Service Mesh 的探索。CSE Mesher 的整个架构都是基于华为自身微服务架构经验研发的，其 Sidecar 也是用 Go 语言编写的。如其官方文档所述，其资源占用非常小，常规状态下仅为 30MB。
- **以腾讯为代表的拿来主义系：**腾讯的 Tencent Service Mesh对开源的产品（如 Istio）进行定制，强化吸收后再加入自身特殊的业务逻辑。腾讯选择的Sidecar是Envoy，使用 C++编写，比较符合

腾讯的技术栈。其公开的技术并不多，仍然以内部小范围使用为主。

- 以 **UCloud** 为代表的适配系：主要也是依赖开源方案，但不是完全将其产品引入，只是对其中几个关键部分添加适配器，以适应企业现有产品，以最小的变更成本引入 Service Mesh 体系。

2、Istio简介

2.1、istio架构



实际上Istio 就是 Service Mesh 架构的一种实现，服务之间的通信（比如这里的 Service A 访问 Service B）会通过代理（默认是 Envoy）来进行。

而且中间的网络协议支持 HTTP/1.1, HTTP/2, gRPC 或者 TCP，可以说覆盖了主流的通信协议。代理这一层，称之为数据平面。

控制平面做了进一步的细分，分成了 Pilot、Citadel 和 Galley，它们的各自功能如下：

- Pilot：为 Envoy 提供了服务发现，流量管理和智能路由（AB 测试、金丝雀发布等），以及错误处理（超时、重试、熔断）功能。
- Citadel：为服务之间提供认证和证书管理，可以让服务自动升级成 TLS 协议。
- Galley：Galley 是 Istio 的配置验证、提取、处理和分发组件。它负责将其余的 Istio 组件与从底层平台（例如 Kubernetes）获取用户配置的细节隔离开来。

数据平面会和控制平面通信，一方面可以获取需要的服务之间的信息，另一方面也可以汇报服务调用的 Metrics 数据。

2.1、为什么使用 Istio?

通过负载均衡、服务间的身份验证、监控等方法，Istio 可以轻松地创建一个已经部署了服务的网络，而服务的代码只需很少更改甚至无需更改。通过在整个环境中部署一个特殊的 sidecar 代理为服务添加 Istio 的支持，而代理会拦截微服务之间的所有网络通信，然后使用其控制平面的功能来配置和管理 Istio，这包括：

- 为 HTTP、gRPC、WebSocket 和 TCP 流量自动负载均衡。
- 通过丰富的路由规则、重试、故障转移和故障注入对流量行为进行细粒度控制。
- 可插拔的策略层和配置 API，支持访问控制、速率限制和配额。
- 集群内（包括集群的入口和出口）所有流量的自动化度量、日志记录和追踪。
- 在具有强大的基于身份验证和授权的集群中实现安全的服务间通信。

Istio 为可扩展性而设计，可以满足不同的部署需求。

2.2、核心特性

Istio 以统一的方式提供了许多跨服务网络的关键功能。

2.2.1、流量管理

Istio 简单的规则配置和流量路由允许您控制服务之间的流量和 API 调用过程。

Istio 简化了服务级属性（如熔断器、超时和重试）的配置，并且让它轻而易举的执行重要的任务（如 A/B 测试、金丝雀发布和按流量百分比划分的分阶段发布）。

有了更好的对流量的可视性和开箱即用的故障恢复特性，就可以在问题产生之前捕获它们，无论面对什么情况都可以使调用更可靠，网络更健壮。

2.2.2、安全

Istio 的安全特性解放了开发人员，使其只需要专注于应用程序级别的安全。

Istio 提供了底层的安全通信通道，并为大规模的服务通信管理认证、授权和加密。有了 Istio，服务通信在默认情况下就是受保护的，可以让您在跨不同协议和运行时的情况下实施一致的策略——而所有这些都只需要很少甚至不需要修改应用程序。

Istio 是独立于平台的，可以与 Kubernetes（或基础设施）的网络策略一起使用。但它更强大，能够在网络和应用层面保护 pod 到 pod 或者服务到服务之间的通信。

2.2.3、可观察性

Istio 健壮的追踪、监控和日志特性让您能够深入的了解服务网格部署。

通过 Istio 的监控能力，可以真正的了解到服务的性能是如何影响上游和下游的；而它的定制 Dashboard 提供了对所有服务性能的可视化能力，并让您看到它如何影响其他进程。

Istio 的 Mixer 组件负责策略控制和遥测数据收集。它提供了后端抽象和中介，将一部分 Istio 与后端的基础设施实现细节隔离开来，并为运维人员提供了对网格与后端基础设施之间交互的细粒度控制。

所有这些特性都使您能够更有效地设置、监控和加强服务的 SLO。当然，底线是您可以快速有效地检测到并修复出现的问题。

2.3、平台支持

Istio 独立于平台，被设计为可以在各种环境中运行，包括跨云、内部环境、Kubernetes、Mesos 等等。您可以在 Kubernetes 或是装有 Consul 的 Nomad 环境上部署 Istio。Istio 目前支持：

- Kubernetes 上的服务部署
- 基于 Consul 的服务注册
- 服务运行在独立的虚拟机上

3、Istio快速入门

下面我们将Istio进行部署安装，来体验下它的魅力。

3.1、搭建kubernetes集群

Istio运行在kubernetes平台是最佳的选择，所以我们先搭建kubernetes环境。

3.1.1、环境准备

准备3台Centos7虚拟机：

名称	IP	角色	CPU	内存	硬盘
node1	192.168.31.106	master	2核	4GB	100GB
node2	192.168.31.107	node	2核	4GB	100GB
node3	192.168.31.108	node	2核	4GB	100GB

3.1.2、前置工作

搭建K8S之前，需要一些前置的准备工作，否则不能完成集群的搭建。

#修改主机名

```
hostnamectl set-hostname node2
hostnamectl set-hostname node3
```

#更新yum源，并且完成yum update操作

```
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
wget -O /etc/yum.repos.d/CentOS-Base.repo
https://mirrors.aliyun.com/repo/Centos-7.repo
yum makecache
yum -y update
```

#安装docker

```
yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo https://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
yum makecache fast
yum -y install docker-ce
```

#启动docker服务

```
systemctl start docker.service
```

#开机自启

```
systemctl enable docker.service
```

#添加docker阿里云加速器

```
sudo mkdir -p /etc/docker
```

```
sudo tee /etc/docker/daemon.json <<- 'EOF'
```

```
{
```

```
  "registry-mirrors": ["https://c6n8vys4.mirror.aliyuncs.com"]
```

```
}
```

```
EOF
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl restart docker
```

#测试一下，看下载速度怎么样

```
docker pull redis
```

```
docker rmi redis:latest
```

#关闭防火墙

```
systemctl stop firewalld.service
```

```
systemctl disable firewalld.service
```

#添加hosts映射

```
vim /etc/hosts
```

```
192.168.31.106 node1
```

```
192.168.31.107 node2
```

```
192.168.31.108 node3
```

```
scp /etc/hosts node2:/etc/
```

```
scp /etc/hosts node3:/etc/
```

#设置node1到node2、node3免登陆

```
ssh-keygen #一路下一步操作
```

```
ssh-copy-id node2
```

```
ssh-copy-id node3
```

#测试

```
ssh node2
```

```
ssh node3
```

3.1.3、搭建集群

#修改系统参数

将 SELinux 设置为 permissive 模式（相当于将其禁用）

```
setenforce 0
```



```
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

禁用swap文件, 编辑/etc/fstab, 注释掉引用 swap 的行

```
vim /etc/fstab
```

```
swapoff -a
```

设置网桥参数

```
vim /etc/sysctl.conf
```

```
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.bridge.bridge-nf-call-iptables = 1
```

```
net.ipv4.ip_forward = 1
```

```
net.ipv4.tcp_tw_recycle = 0
```

```
scp /etc/sysctl.conf node2:/etc/
```

```
scp /etc/sysctl.conf node3:/etc/
```

#立即生效

```
sysctl -p
```

#安装kubect1

```
vim /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
```

```
name=Kubernetes
```

```
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
```

```
enabled=1
```

```
gpgcheck=0
```

```
yum list kubect1 --showduplicates
```

```
yum install -y kubect1.x86_64
```

```
kubect1 version
```

```
yum install -y kubelet kubeadm --disableexcludes=kubernetes
```

#拉取所需要的镜像

```
kubeadm config images pull --image-repository=registry.cn-
```

```
hangzhou.aliyuncs.com/itcast --kubernetes-version=v1.18.6
```

#如果拉取失败, 尝试这个: kubeadm config images pull --image-repository=lank8s.cn --kubernetes-version=v1.18.6

#开始初始化, 如果使用lank8s.cn拉取的镜像, 需要指定--image-repository=lank8s.cn

```
kubeadm init --apiserver-advertise-address 192.168.31.106 --pod-network-
```

```
cidr=10.244.0.0/16 --image-repository=registry.cn-hangzhou.aliyuncs.com/itcast
```

```
--kubernetes-version=v1.18.6
```

#当看到 Your Kubernetes control-plane has initialized successfully! 说明初始化成功了!

#拷贝admin.conf到home目录, 否则出错: The connection to the server localhost:8080 was refused - did you specify the right host or port?

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

#设置网络 kube-flannel.yml 文件在资料中

```
kubectl apply -f kube-flannel.yml
```

#测试

```
[root@node1 k8s]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
node1	Ready	master	23m	v1.18.6

#将node2、node3加入到集群, token要替换成自己的

```
kubeadm join 192.168.31.106:6443 --token ekw4eu.cfi77sjiljyczhj6 --discovery-token-ca-cert-hash sha256:21de4177eaf76353dd060f2a783c9dafe17636437ade020bc40d60a8ab903483
```

#测试

```
[root@node1 k8s]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
node1	Ready	master	31m	v1.18.6
node2	Ready	<none>	6m46s	v1.18.6
node3	Ready	<none>	2m21s	v1.18.6

#说明集群组件成功了

#如果需要删除集群, 执行 kubeadm reset , 3台机器都需要执行

#查看正在执行的pod

```
kubectl get pod --all-namespaces -o wide
```

查看正在执行的pod, kubectl get pod --all-namespaces -o wide

```
[root@node1 k8s]# kubectl get pod --all-namespaces -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
kube-system	coredns-6b7d9985d9-85gfl	1/1	Running	0	39m	10.244.0.2	node1	<none>	<none>
kube-system	coredns-6b7d9985d9-96fkv	1/1	Running	0	39m	10.244.0.3	node1	<none>	<none>
kube-system	etcd-node1	1/1	Running	0	39m	192.168.31.106	node1	<none>	<none>
kube-system	kube-apiserver-node1	1/1	Running	0	39m	192.168.31.106	node1	<none>	<none>
kube-system	kube-controller-manager-node1	1/1	Running	0	39m	192.168.31.106	node1	<none>	<none>
kube-system	kube-flannel-ds-amd64-gm5dm	1/1	Running	0	15m	192.168.31.107	node2	<none>	<none>
kube-system	kube-flannel-ds-amd64-njr2r	1/1	Running	0	10m	192.168.31.108	node3	<none>	<none>
kube-system	kube-flannel-ds-amd64-tjjqv	1/1	Running	0	19m	192.168.31.106	node1	<none>	<none>
kube-system	kube-proxy-42hnb4	1/1	Running	0	15m	192.168.31.107	node2	<none>	<none>
kube-system	kube-proxy-dzjnd	1/1	Running	0	39m	192.168.31.106	node1	<none>	<none>
kube-system	kube-proxy-lrp48	1/1	Running	0	10m	192.168.31.108	node3	<none>	<none>
kube-system	kube-scheduler-node1	1/1	Running	0	39m	192.168.31.106	node1	<none>	<none>

3.2、搭建Istio环境

3.2.1、下载 Istio

下载 Istio, 下载内容将包含: 安装文件、示例和 istioctl 命令行工具。

1. 访问 [Istio release](https://istio.io/downloadIstio) 页面下载与您操作系统对应的安装文件。在 macOS 或 Linux 系统中，也可以通过以下命令下载最新版本的 Istio：

```
$ curl -L https://istio.io/downloadIstio | sh -
```

2. 切换到 Istio 包所在目录下。例如：Istio 包名为 `istio-1.6.5`，则：

```
$ cd istio-1.6.5
```

安装目录包含如下内容：

- `samples/` 目录下，有示例应用程序
 - `bin/` 目录下，包含 `istioctl` 的客户端文件。`istioctl` 工具用于手动注入 Envoy sidecar 代理。
3. 将 `istioctl` 客户端路径增加到 path 环境变量中，macOS 或 Linux 系统的增加方式如下：

```
$ export PATH=$PWD/bin:$PATH
```

安装 bash 自动补全文件

如果您使用 bash，`istioctl` 自动补全的文件位于 `tools` 目录。通过复制 `istioctl.bash` 文件到您的 home 目录，然后添加下行内容到您的 `.bashrc` 文件执行 `istioctl` tab 补全文件：

```
source ~/istioctl.bash
```

如果 `istioctl` 补全文件已经正确安装，在您输入 `istioctl` 命令时通过按 Tab 键，它会返回一组推荐命令供您选择：

```
$ istioctl proxy-<TAB>
proxy-config proxy-status
```

3.2.2、安装Istio

请按照以下步骤在您所选的平台上使用 `demo` 配置文件安装 Istio。

1. 安装 `demo` 配置

```
$ istioctl manifest apply --set profile=demo
```

2. 为了验证是否安装成功，需要先确保以下 Kubernetes 服务正确部署，然后验证除 `jaeger-agent` 服务外的其他服务，是否均有正确的 `CLUSTER-IP`：

```
$ kubectl get svc -n istio-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S)			
			AGE

grafana 3000/TCP	ClusterIP	172.21.211.123	<none>	2m
istio-citadel 8060/TCP,15014/TCP	ClusterIP	172.21.177.222	<none>	2m
istio-egressgateway 80/TCP,443/TCP,15443/TCP	ClusterIP	172.21.113.24	<none>	2m
istio-galley 443/TCP,15014/TCP,9901/TCP	ClusterIP	172.21.132.247	<none>	2m
istio-ingressgateway 15020:31831/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30318/TCP,15030:32645/TCP,15031:31933/TCP,15032:31188/TCP,15443:30838/TCP	LoadBalancer	172.21.144.254	52.116.22.242	2m
istio-pilot 15010/TCP,15011/TCP,8080/TCP,15014/TCP	ClusterIP	172.21.105.205	<none>	2m
istio-policy 9091/TCP,15004/TCP,15014/TCP	ClusterIP	172.21.14.236	<none>	2m
istio-sidecar-injector 443/TCP,15014/TCP	ClusterIP	172.21.155.47	<none>	2m
istio-telemetry 9091/TCP,15004/TCP,15014/TCP,42422/TCP	ClusterIP	172.21.196.79	<none>	2m
jaeger-agent 5775/UDP,6831/UDP,6832/UDP	ClusterIP	None	<none>	2m
jaeger-collector 14267/TCP,14268/TCP	ClusterIP	172.21.135.51	<none>	2m
jaeger-query 16686/TCP	ClusterIP	172.21.26.187	<none>	2m
kiali 20001/TCP	ClusterIP	172.21.155.201	<none>	2m
prometheus 9090/TCP	ClusterIP	172.21.63.159	<none>	2m
tracing 80/TCP	ClusterIP	172.21.2.245	<none>	2m
zipkin 9411/TCP	ClusterIP	172.21.182.245	<none>	2m

如果集群运行在一个不支持外部负载均衡器的环境中（例如：minikube），`istio-ingressgateway` 的 `EXTERNAL-IP` 将显示为 `<pending>` 状态。请使用服务的 `NodePort` 或端口转发来访问网关。

请确保关联的 Kubernetes pod 已经部署，并且 `STATUS` 为 `Running`：

```
$ kubectl get pods -n istio-system
```

NAME			READY
STATUS	RESTARTS	AGE	
grafana-f8467cc6-rbjlg			1/1
Running	0	1m	
istio-citadel-78df5b548f-g5cpw			1/1
Running	0	1m	
istio-egressgateway-78569df5c4-zwtb5			1/1
Running	0	1m	
istio-galley-74d5f764fc-q7nrk			1/1
Running	0	1m	
istio-ingressgateway-7ddcfd665c-dmtqz			1/1
Running	0	1m	
istio-pilot-f479bbf5c-qwr28			1/1
Running	0	1m	
istio-policy-6fccc5c868-xhblv			1/1
Running	2	1m	
istio-sidecar-injector-78499d85b8-x44m6			1/1
Running	0	1m	
istio-telemetry-78b96c6cb6-ldm9q			1/1
Running	2	1m	
istio-tracing-69b5f778b7-s2zvw			1/1
Running	0	1m	
kiali-99f7467dc-6rvwp			1/1
Running	0	1m	
prometheus-67cdb66cbb-9w2hm			1/1
Running	0	1m	

3.3、Bookinfo示例

3.3.1、应用说明

这个示例部署了一个用于演示多种 Istio 特性的应用，该应用由四个单独的微服务构成。这个应用模仿在线书店的一个分类，显示一本书的信息。页面上会显示一本书的描述，书籍的细节（ISBN、页数等），以及关于这本书的一些评论。

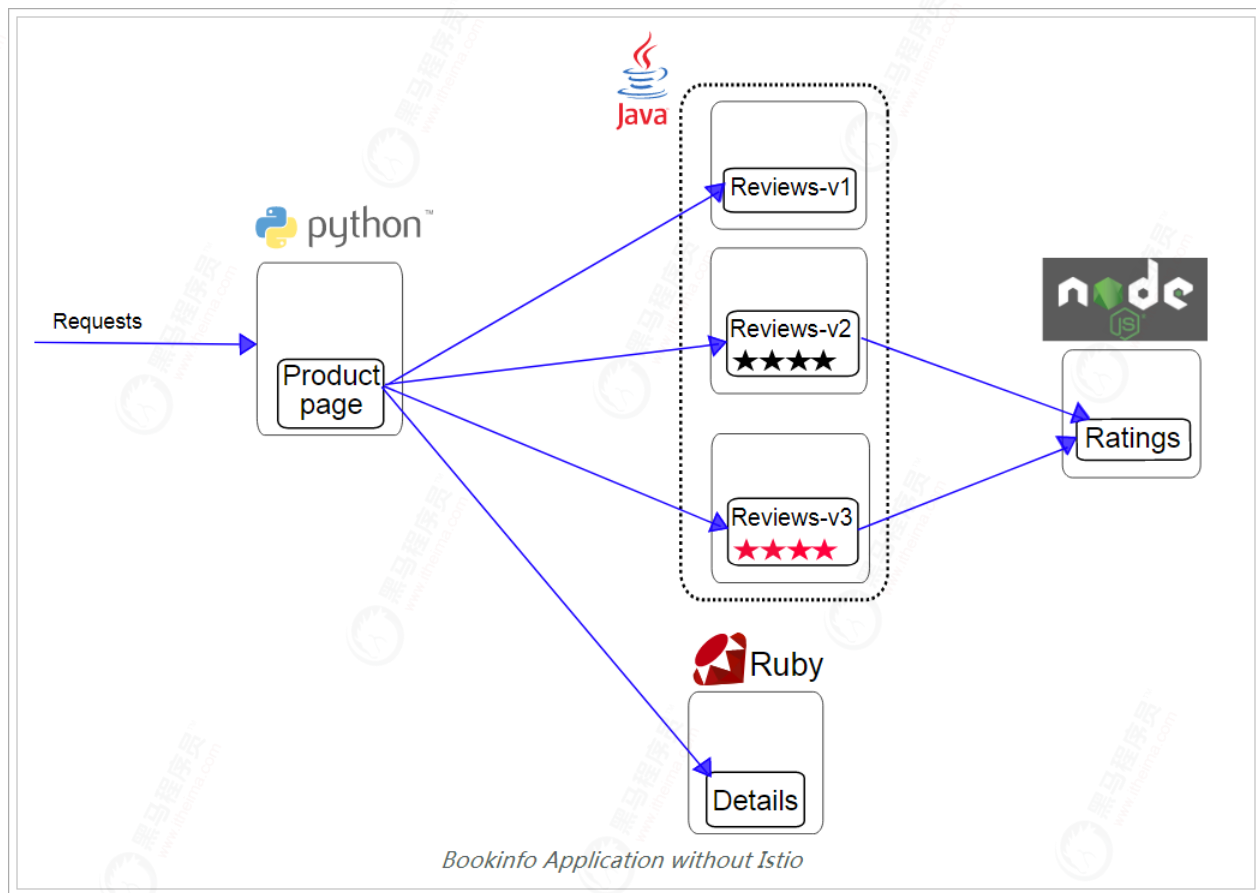
Bookinfo 应用分为四个单独的微服务：

- `productpage`. 这个微服务会调用 `details` 和 `reviews` 两个微服务，用来生成页面。
- `details`. 这个微服务中包含了书籍的信息。
- `reviews`. 这个微服务中包含了书籍相关的评论。它还会调用 `ratings` 微服务。
- `ratings`. 这个微服务中包含了由书籍评价组成的评级信息。

`reviews` 微服务有 3 个版本：

- v1 版本不会调用 `ratings` 服务。
- v2 版本会调用 `ratings` 服务，并使用 1 到 5 个黑色星形图标来显示评分信息。
- v3 版本会调用 `ratings` 服务，并使用 1 到 5 个红色星形图标来显示评分信息。

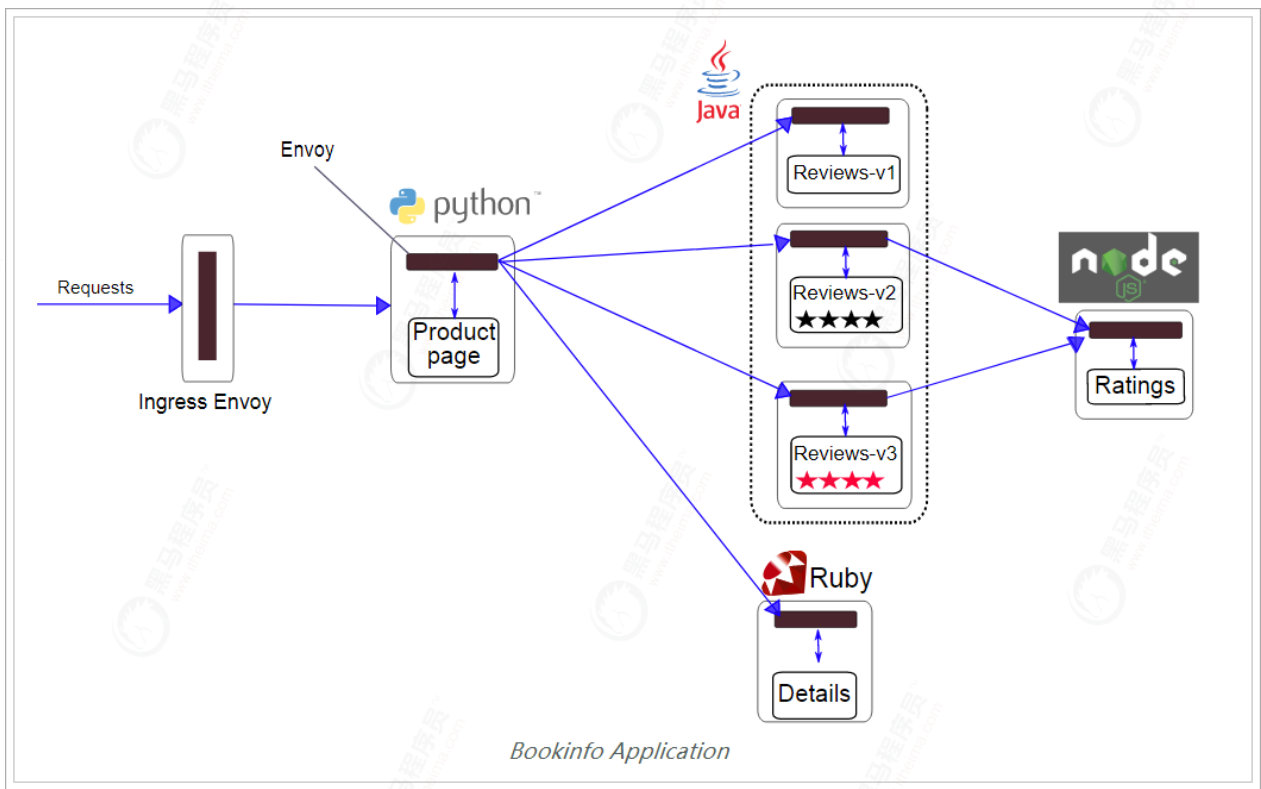
下图展示了这个应用的端到端架构。



Bookinfo 应用中的几个微服务是由不同的语言编写的。这些服务对 Istio 并无依赖，但是构成了一个有代表性的服务网格的例子：它由多个服务、多个语言构成，并且 `reviews` 服务具有多个版本。

3.3.2、部署应用

要在 Istio 中运行这一应用，无需对应用自身做出任何改变。您只要简单的在 Istio 环境中对服务进行配置和运行，具体一点说就是把 Envoy sidecar 注入到每个服务之中。最终的部署结果将如下图所示：



所有的微服务都和 Envoy sidecar 集成在一起，被集成服务所有的出入流量都被 sidecar 所劫持，这样就为外部控制准备了所需的 Hook，然后就可以利用 Istio 控制平面为应用提供服务路由、遥测数据收集以及策略实施等功能。

3.3.3、启动应用服务

1. 进入 Istio 安装目录。
2. Istio 默认自动注入 Sidecar. 请为 `default` 命名空间打上标签 `istio-injection=enabled`：

```
$ kubectl label namespace default istio-injection=enabled
```

3. 使用 `kubectl` 部署应用：

```
$ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

如果您在安装过程中禁用了 Sidecar 自动注入功能而选择手动注入 Sidecar，请在部署应用之前使用 `istioctl kube-inject` 命令修改 `bookinfo.yaml` 文件。

```
$ kubectl apply -f <(istioctl kube-inject -f  
samples/bookinfo/platform/kube/bookinfo.yaml)
```

上面的命令会启动全部四个服务，其中也包括了 reviews 服务的三个版本（v1、v2 以及 v3）。

在实际部署中，微服务版本的启动过程需要持续一段时间，并不是同时完成的。

4. 确认所有的服务和 Pod 都已经正确的定义和启动：

```
$ kubectl get services
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE			
details	10.0.0.31	<none>	9080/TCP
6m			
kubernetes	10.0.0.1	<none>	443/TCP
7d			
productpage	10.0.0.120	<none>	9080/TCP
6m			
ratings	10.0.0.15	<none>	9080/TCP
6m			
reviews	10.0.0.170	<none>	9080/TCP
6m			

还有：

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS
AGE			
details-v1-1520924117-48z17	2/2	Running	0
6m			
productpage-v1-560495357-jk1lz	2/2	Running	0
6m			
ratings-v1-734492171-rnr5l	2/2	Running	0
6m			
reviews-v1-874083890-f0qf0	2/2	Running	0
6m			
reviews-v2-1343845940-b34q5	2/2	Running	0
6m			
reviews-v3-1813607990-8ch52	2/2	Running	0
6m			

5. 要确认 Bookinfo 应用是否正在运行，请在某个 Pod 中用 `curl` 命令对应用发送请求，例如 `ratings`：

```
$ kubectl exec -it $(kubectl get pod -l app=ratings -o  
jsonpath='{.items[0].metadata.name}') -c ratings -- curl  
productpage:9080/productpage | grep -o "<title>.*</title>"  
<title>Simple Bookstore App</title>
```

3.3.4、确定 Ingress 的 IP

现在 Bookinfo 服务启动并运行中，您需要使应用程序可以从外部访问 Kubernetes 集群，例如使用浏览器。可以用 Istio Gateway 来实现这个目标。

1. 为应用程序定义 Ingress 网关：

```
$ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

2. 确认网关创建完成:

```
$ kubectl get gateway
NAME                AGE
bookinfo-gateway    32s
```

3. 设置访问网关的 `INGRESS_HOST` 和 `INGRESS_PORT` 变量。确认并设置。

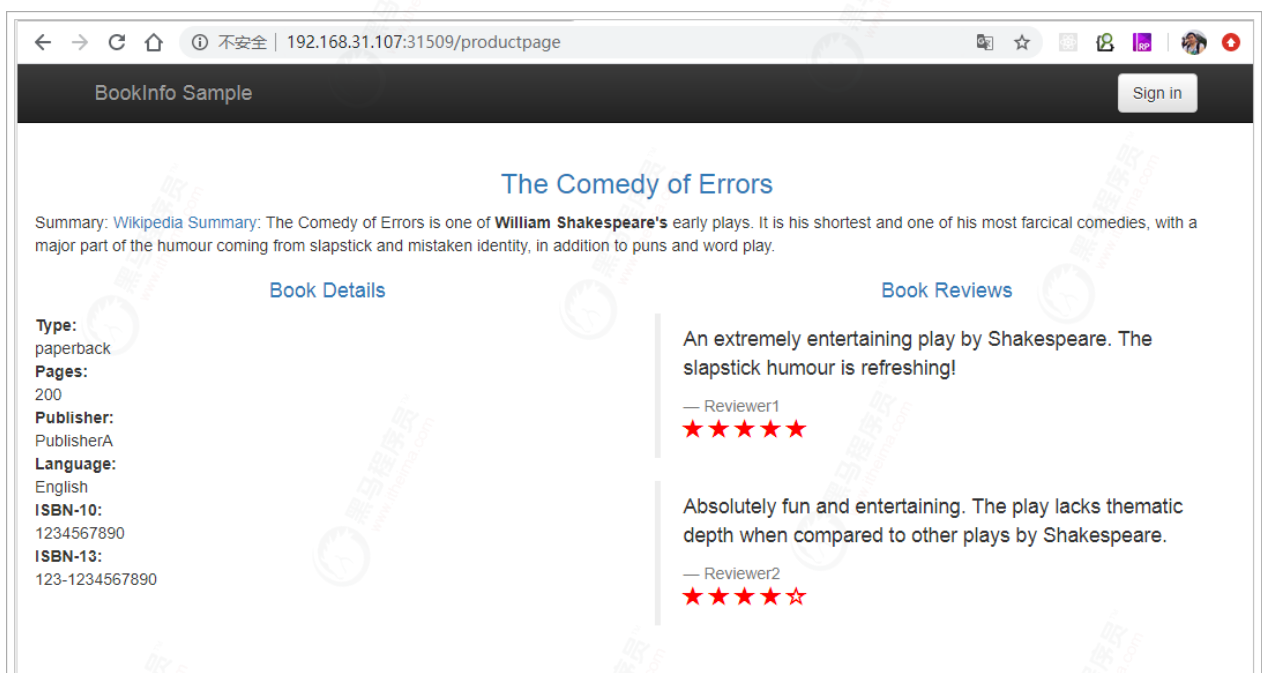
```
#设置 ingress 端口
export INGRESS_PORT=$(kubectl -n istio-system get service istio-
ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-
ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')

#设置 ingress IP
export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-
system -o jsonpath='{.items[0].status.hostIP}')
```

4. 设置 `GATEWAY_URL`:

```
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

可以用浏览器打开网址 `http://$GATEWAY_URL/productpage`, 来浏览应用的 Web 页面。如果刷新几次应用的页面, 就会看到 `productpage` 页面中会随机展示 `reviews` 服务的不同版本的效果(红色、黑色的星形或者没有显示)。`reviews` 服务出现这种情况是因为我们还没有使用 Istio 来控制版本的路由。



3.3.5、应用默认目标规则

在使用 Istio 控制 Bookinfo 版本路由之前，您需要在目标规则中定义好可用的版本，命名为 *subsets*。

#设置

```
kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml
```

#查询

```
kubectl get destinationrules -o yaml
```

至此，Istio 完成了全部的接管，第一个示例部署完成。

3.4、体验Istio

3.4.1、按照版本路由

reviews有三个版本，默认情况下，会进行轮询，也就是我们看到的，每一次刷新都会有不同的效果。

如果，我们需要将请求全部切换到某一个版本，需要Istio是非常简单的，只需要添加虚拟服务即可。

示例：

#virtual-service-all-v1.yaml是官方提供的示例文件

```
kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml
```

其内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
```

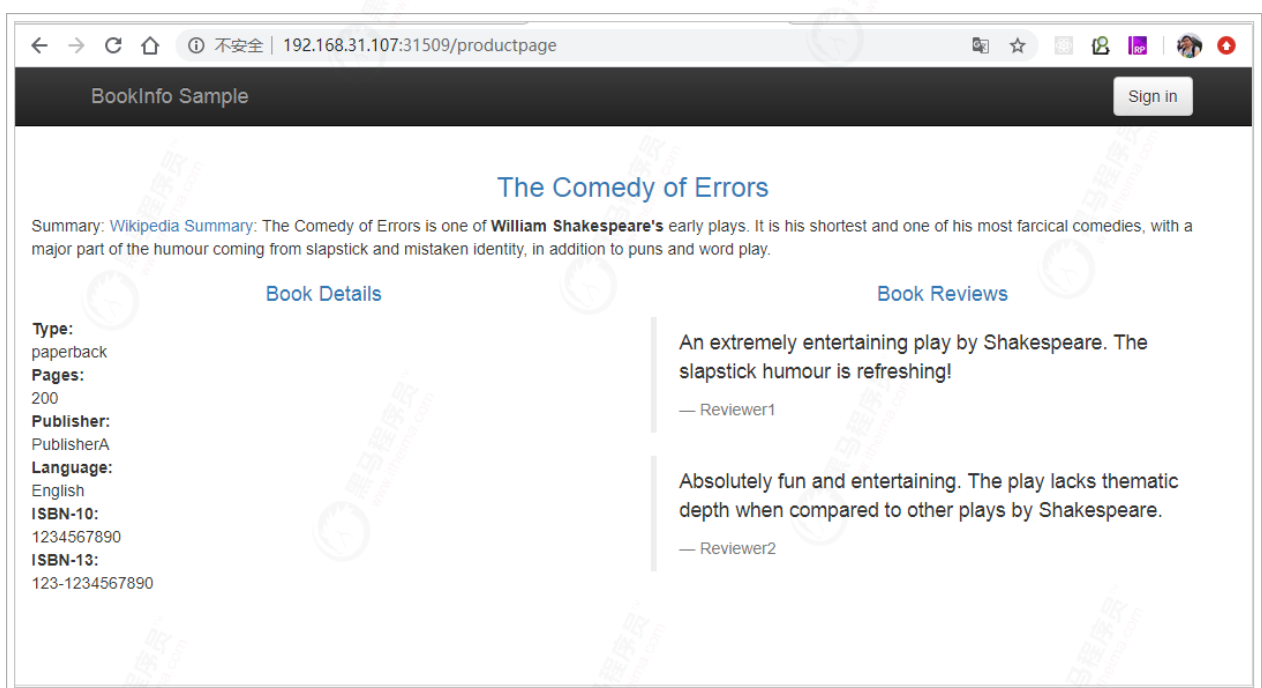


```

    host: reviews
    subset: v1 #在这里指定了所有的http请求都通过v1完成，而v1在默认的规则中有定义
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
spec:
  hosts:
  - details
  http:
  - route:
    - destination:
        host: details
        subset: v1
---

```

经过测试，发现reviews不再切换样式。



还可以将部分流量转移到v3版本，基于此可以实现灰度发布、A/B测试等：

#将50%的流程转移到v3

```
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml
```

内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
    - route:
        - destination:
            host: reviews
            subset: v1
            weight: 50
        - destination:
            host: reviews
            subset: v3
            weight: 50
```

刷新浏览器中的 /productpage 页面，大约有 50% 的几率会看到页面中出带 红色 星级的评价内容。这是因为 v3 版本的 reviews 访问了带星级评级的 ratings 服务，但 v1 版本却没有。

如果认为 reviews:v3 微服务已经稳定，可以通过应用此 virtual service 规则将 100% 的流量路由到 reviews:v3：

```
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-v3.yaml
```

这样，所有的请求都转向到了v3了。

如果需要删除虚拟网络，可以执行：

```
kubectl delete -f samples/bookinfo/networking/virtual-service-all-v1.yaml
```

3.4.2、按照不同用户身份路由

接下来，您将更改路由配置，以便将来自特定用户的所有流量路由到特定服务版本。在这，来自名为 Jason 的用户的所有流量将被路由到服务 `reviews:v2`。

请注意，Istio 对用户身份没有任何特殊的内置机制。事实上，`productpage` 服务在所有到 `reviews` 服务的 HTTP 请求中都增加了一个自定义的 `end-user` 请求头，从而达到了本例子的效果。

请记住，`reviews:v2` 是包含星级评分功能的版本。

1. 运行以下命令以启用基于用户的路由：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

2. 确认规则已创建：

```
$ kubectl get virtualservice reviews -o yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
    - route:
        destination:
          host: reviews
          subset: v1
```

3. 在 Bookinfo 应用程序的 `/productpage` 上，以用户 `jason` 身份登录。

刷新浏览器。你看到了什么？星级评分显示在每个评论旁边。

4. 以其他用户身份登录（选择您想要的任何名称）。

刷新浏览器。现在星星消失了。这是因为除了 Jason 之外，所有用户的流量都被路由到 `reviews:v1`。

您已成功配置 Istio 以根据用户身份路由流量。