



(<https://www.kubernetes.org.cn/peixun>)

您目前处于：社区首页 (<https://www.kubernetes.org.cn>) > 容器 (<https://www.kubernetes.org.cn/container>) >
企业应用架构演化探讨：从微服务到Service Mesh

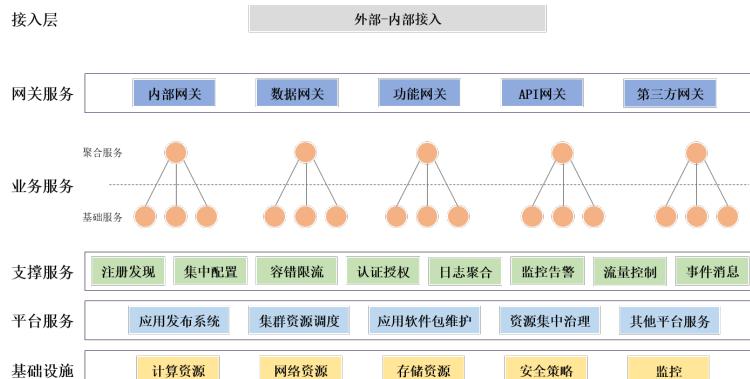
企业应用架构演化探讨：从微服务到Service Mesh (<https://www.kubernetes.org.cn/5349.html>)

2019-04-28 13:53 BoCloud (<https://www.kubernetes.org.cn/author/bocloud>) 分类：容器
(<https://www.kubernetes.org.cn/container>) 阅读(696) 评论(0)

导读

当下微服务的实践方案中，Spring Cloud, Dubbo作为主流的落地方案，在企业应用架构中发挥越来越重要的作用。本文探讨企业应用架构如何从微服务架构向Service Mesh架构演化，并形成落地方案。需要特别说明：本文讨论的架构目前适用于普通的企业级应用，其他行业（例如互联网）需要进一步扩展。

在讨论之前，我们需要明确一个事实：**企业应用一定是围绕业务进行的**。无论采用什么的架构落地，都是为了更好的为应用业务进行服务。从企业应用的特性考虑，主要包括：稳定性，安全性，扩展性，容错性。围绕着企业应用的这些特点，我们来看一个典型的微服务企业架构模型，如图所示：



- **服务接入层：**企业暴露到外部访问的入口，一般通过防火墙等。
- **网关层：**服务网关是介于客户端和服务端的中间层，所有的外部请求会先经过服务网关，为企业应用提供统一的访问控制入口。服务网关是微服务架构下的服务拆分，聚合，路由，认证以及流控综合体现。
- **支撑服务层：**为企业应用提供运行所需的支撑环境，包括注册发现，集中配置，容错限流，认证授权，日志聚合，监测告警，消息服务等
- **业务服务层：**业务服务是企业应用的核心所在，为企业领域应用的具体实现，一般进一步拆分为基础服务（基础功能）和聚合服务（综合场景）。
- **平台服务层：**为企业应用提供运行所需的软件资源，包括应用服务器，应用发布管理，应用镜像包管理，服务治理。

Kubernetes 1.13 版本

2018年最后一次大版本Kubernetes 1.13，此版本继续关注Kubernetes的稳定性和可扩展性，其中在存储和群集生命周期领域的三个主要功能实现普遍可用 (GA) ...
(<https://www.kubernetes.org.cn/tags/kubernetes1-13>)



关注「K8S中文社区」微信公众号

回复“文档” 获取K8S文档下载链接

回复“加群” 加入K8S微信技术交流群

最新文章



浅析 Kubernetes 原生 NetworkPolicy 网络策略，让更安全的容器运行环境唾手可得

2019-05-29 评论(0)
(<https://www.kubernetes.org.cn>)



搞搞 Prometheus:
Alertmanager

2019-05-27 评论(0)
(<https://www.kubernetes.org.cn>)



容器监控之 kube-state-metrics

2019-05-27 评论(0)
(<https://www.kubernetes.org.cn>)



微服务网关实战——Spring Cloud Gateway

2019-05-26 评论(0)
(<https://www.kubernetes.org.cn>)

热门推荐



kubernetes v1.14.0高可用 master集群部署 (使用 kubeadm, 离线安装)

2019-04-13
(<https://www.kubernetes.org.cn>)

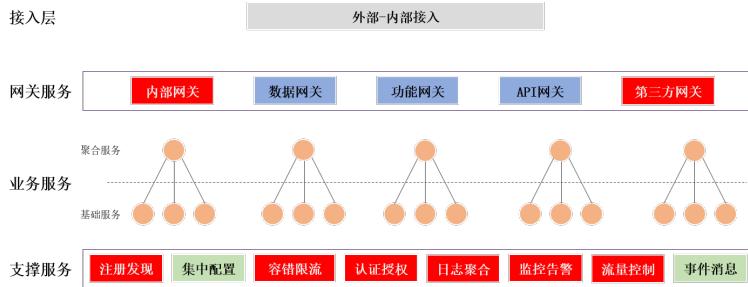


kubeadm HA master(v1.14.0) 离线包 + 自动化脚本 + 常用插件 For CentOS/Fedora

2019-03-27
(<https://www.kubernetes.org.cn>)

- 基础设施层：为企业应用提供运行所需的硬件资源，包括计算资源，网络资源，存储资源，基本的安全策略控制等。

从这个典型的服务架构体系中，能够清晰的表明层级架构以及各层涵盖的职责说明。我们暂不考虑基础设施层和平台服务两层，重点关注**网关服务**，**业务服务**，**支撑服务**，突出其中的一些**基础支撑功能组件**，这也是我们本篇探讨的重点内容。如下图所示：



根据图中红色标识，我们会发现这样一个事实：**在微服务架构下，无论是哪种落地实现方式，都集中在网关服务、支撑服务两个层面**。无论是Spring Cloud “套装组件”，Dubbo “套件”还是其他开源组件，都为支撑服务的实现提供了数量众多的选择。功能完整、选择性多这是业内喜闻乐见的事情，但是也无形中增加了开发，测试，运维人员的压力。大家需要掌握越来越多的“使用工具”以更“方便”、“快捷”地应对业务服务。有时候，可能为了实现单一功能，而必须引入一堆组件，这时候我们希望能够有一个完整的平台来为应用业务提供一体化的支撑服务，而不是一系列“套装组件”与业务的集成。那么如何基于一个平台来实现这些企业应用需要的能力呢？经过一定阶段的技术调研，我们认为Service Mesh能够帮助我们初步达到这个目标。我们都知道Service Mesh以解决“服务通信”的问题作为其设计初衷，聚焦基础设施“网络层”，并以此做技术基础，解决业务通信场景面临的问题。那么**如何把它应用在企业应用架构中来取代“微服务套装组件”**呢？那接下来让我们针对网关服务，业务服务，支撑服务分别来看一下，如何从原来的微服务“套装组件”中抽离出来，实现Service Mesh方向的转变。

网关服务

前面提到过：服务网关是介于客户端和服务端的中间层。从功能上不难理解，对内屏蔽内部细节，对外提供统一服务接口。从场景聚焦角度考虑，网关根据不同的场景承载不同的职责，包括认证，授权，路由，流控，负载等。（之前我们也聊过网关组件的对比及具体实现，感兴趣的同学可点击微服务五种开源API网关实现组件对比 (http://mp.weixin.qq.com/s?__biz=Mzg3OTA4NDgzOQ==&mid=2247483846&idx=1&sn=f4026386671097f5ba4e9679227e5a8&chksm=cf08958ef87f1c9814cfce4245dc2bbf9024f06f4c5a61cae2298d8bddcebd8931be90e5bdb&scene=21#wechat_redirect)）。由此可见，服务网关是企业应用架构下一些列功能的综合体现。那么在Service Mesh情况下如何处理网关服务呢？在展开之前首先需要说明一个前提：**目前为止Service Mesh跟真正企业网关相比还存在一定的不足之处**，例如“协议转化”，“安全策略”，“性能要求”等方面。在这里我们也是探讨这样的可能性。下面以Istio为例，我们来看一下，如何提供网关层面的服务。Istio在网关层面提供两种类型的网关服务：Ingress Gateway，Egress。

Ingress Gateway

Ingress Gateway用于接收传入的HTTP/TCP连接，它配置暴露端口，协议供外部统一接入，但是自身不提供任何的路由配置，而是完全依赖 Istio 的控制规则来进行流量路由。从而与内部服务请求统一到同一个控制层面上。

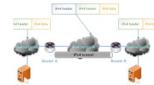
Egress

在企业应用与外部应用之间，有时候为了业务需要会出现内部服务调用外部服务的情况，此时一般会从企业内部接入第三方网关来获取服务数据。在 Istio 中你同样可以基于Egress来达到目的。Istio中提供两种方式：一种基于ServiceEntry + VirtualService的配置，实现第三方服务的访问，一种扩大sidecar的访问地址列表。（参考文档：<https://preliminary.istio.io/zh/docs/tasks/traffic-management/egress/>）。

基于上述两种场景，我们可以看出，在 Service Mesh 的体系下，网关层面变成一个可以动态生成和销毁的组件，能够通过控制层面实现统一规则管理，并且实时生效。基于Service Mesh的网关服务如下图所示：



每个人都必须遵循的九项
Kubernetes安全最佳实践
2019-01-15
(<https://www.kubernetes.org.cn>)



Calico on Kubernetes 从入门
到精通
2018-12-22
(<https://www.kubernetes.org.cn>)



kubeadm HA master(v1.13.0)
离线包 + 自动化脚本 + 常用插
件 For CentOS/Fedora
2018-12-10
(<https://www.kubernetes.org.cn>)



Kubernetes 暴惊天大漏洞，请
升级生产环境的集群！
2018-12-05
(<https://www.kubernetes.org.cn>)



(<https://www.kubernetes.org.cn/peixun>)

社区标签

BoCloud博云
(<https://www.kubernetes.org.cn/tags/bocloud%e5%e5%calico>)

CI/CD
(<https://www.kubernetes.org.cn/tags/cicd>)

CNCF
(<https://www.kubernetes.org.cn>)
CoreOS (<https://www.kubernetes.org.cn/tags/coreos>)

DevOps
(<https://www.kubernetes.org.cn/tags/devops>)

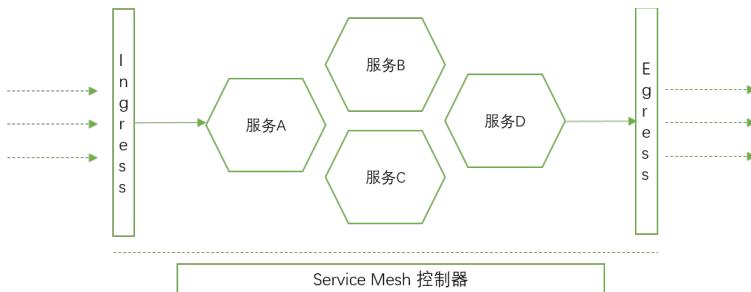
Docker
(<https://www.kubernetes.org.cn/tags/docker>)
ha (<https://www.kubernetes.org.cn/tags/ha>)

Helm
(<https://www.kubernetes.org.cn/tags/helm>)
Istio (<https://www.kubernetes.org.cn/tags/istio>)

Jenkins
(<https://www.kubernetes.org.cn/tags/jenkins>)
k8s代码解读
(<https://www.kubernetes.org.cn/tags/k8s>)

kubeadm
(<https://www.kubernetes.org.cn/tags/kubeadm>)
KubeCon
(<https://www.kubernetes.org.cn/tags/kubecon>)
Kubelet (<https://www.kubernetes.org.cn/tags/kubelet>)

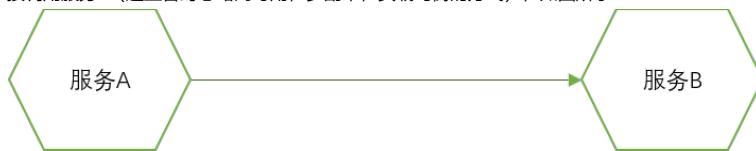
Kubernetes1.4
(<https://www.kubernetes.org.cn/tags/kubernetes1-4>)



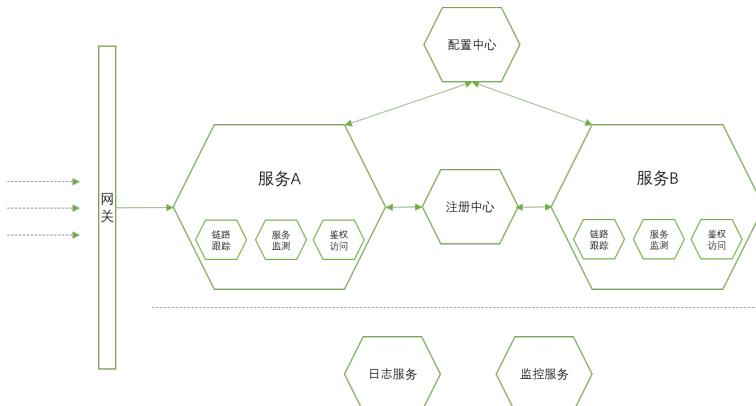
从实现原理上分析，传统的网关实现基于 Servlet 的 filter 的模式，实现服务请求转移过程中的层层过滤和处理。区别在于采用同步或者异步处理机制，用来解决网关的性能瓶颈。而 Service Mesh 的网关完全是基于网络代理的请求转发与控制，本质上作用在服务的 Iptables 上，通过对 Iptables 的规则控制达到同样的效果。

业务服务

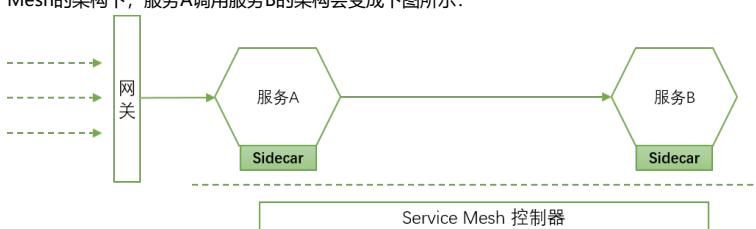
业务是企业应用的“重中之重”，无论哪种企业架构，最终都是为了更好地为业务提供服务，那么我们如何在 Service Mesh 的体系下，重构业务服务呢？我们以两个简化的服务调用来说明整个架构的转变过程。假如要实现服务 A、服务 B 的相互调用，最原始的方式是服务 A 基于协议层直接调用服务 B（这里暂时忽略高可用，多副本，负载均衡的方式），如图所示：



由图可见，服务 A 基于某种协议完成对服务 B 的请求，相对比较简单。但是我们知道这样虽然能够快速完成业务关联，但是无法确保业务正常稳定的运行，因此我们需要引入更多的服务来保证业务的稳定，可靠，可控。此时我们最容易想到的是引入微服务的支撑组件来达到目标。以 Spring Cloud 方案为例，我们来说明当前微服务架构的实现方式。为了满足企业应用对服务 A、服务 B 的管理控制，需要额外引入“注册中心”，“网关”，“配置中心”，“服务监控”，“事件消息”，“链路跟踪”，“日志服务”等众多与直接业务无关的“旁路保障服务”，简化一下，如下图所示：



从图中可以看出，每个服务都引入了大量与业务无关的“保障服务”，这些“旁路保障服务”消耗的资源，与比业务本身消耗的资源成“倍数关系”。随着服务数目的增多，业务服务本身占用的资源会越来越少，此时开发人员会把大量的精力花费在维护这些“旁路保障服务”上，而忽略业务本身。这对于企业应用而言，有些本末倒置的意思。我们再来看一下 Service Mesh 体系下，我们如何解决上述问题。Service Mesh 为了解决企业应用的“通信问题”重点做了四个方面的工作，以 Istio 为代表，提供了包括流量管理，安全配置，策略控制以及外围组件支撑的遥测功能（需要的朋友，可以参考官方文档：<https://preliminary.istio.io/zh/docs>），在 Service Mesh 的架构下，服务 A 调用服务 B 的架构会变成下图所示：



Kubernetes1.5

(<https://www.kubernetes.org.cn/tags/1.5>)

Kubernetes1.6

(<https://www.kubernetes.org.cn/tags/1.6>)

Kubernetes1.7

(<https://www.kubernetes.org.cn/tags/1.7>)

Kubernetes1.8

(<https://www.kubernetes.org.cn/tags/1.8>)

Kubernetes1.9

(<https://www.kubernetes.org.cn/tags/1.9>)

Kubernetes1.10

(<https://www.kubernetes.org.cn/tags/kubernetes1.10>)

Kubernetes1.13

(<https://www.kubernetes.org.cn/tags/kubernetes1.13>)

Linkerd (<https://www.kubernetes.org.cn/tags/linkerd>)

OpenStack (<https://www.kubernetes.org.cn/tags/openstack>)

PaaS

(<https://www.kubernetes.org.cn/tags/paas>)

Pod

(<https://www.kubernetes.org.cn/tags/pod>)

Prometheus

(<https://www.kubernetes.org.cn/tags/prometheus>)

Rancher

(<https://www.kubernetes.org.cn/tags/rancher>)

Serverless

(<https://www.kubernetes.org.cn/tags/serverless>)

Service (<https://www.kubernetes.org.cn/tags/service>)

service mesh

(<https://www.kubernetes.org.cn/tags/service-mesh>)

Spring Cloud (<https://www.kubernetes.org.cn/tags/spring-cloud>)

Traefik (<https://www.kubernetes.org.cn/tags/traefik>)

企业案例

(<https://www.kubernetes.org.cn/tags/case>)

存储

(<https://www.kubernetes.org.cn/tags/storage>)

安全

(<https://www.kubernetes.org.cn/tags/security>)

容器

(<https://www.kubernetes.org.cn/tags/container>)

容器云

(<https://www.kubernetes.org.cn/tags/container-cloud>)

微服务

(<https://www.kubernetes.org.cn/tags/microservices>)

微软 Azure (<https://www.kubernetes.org.cn/tags/azure>)

日志

(<https://www.kubernetes.org.cn/tags/logging>)



通过上图我们可以发现，与Spring Cloud的实现方式相比，似乎简单了很多，我们不再需要在业务服务中引入众多的“旁路保障服务”，而是保障了业务服务本身的简化。那么Service Mesh是如何处理的呢？

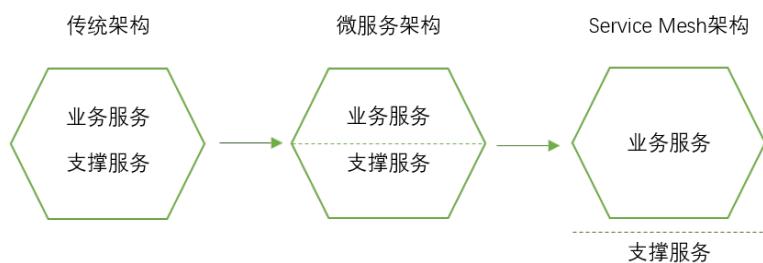
第一，引入了Sidecar代理模型，作为服务流量控制的入口和出口，保证能够对网络层面数据做实时监控和调整；第二，控制器根据具体业务情况，分发控制状态和控制指令，Sidecar获取控制信息后，及时更新缓存信息，保证策略有效性。第三，Sidecar由于能够拦截所有请求的流量信息，定期把收集的数据向控制层进行上报，从而完成服务状态和应用链路的监测。

第四，所有的这些都是在应用部署阶段完成，在开发层面不需要花费大量的精力。

基于以上四点我们可以发现，Service Mesh 仅仅通过方式转变就达到了同样的效果，还极大的解放了开发人员。通过业务服务调用方式的实现转变，我们发现这样一个事实：Service Mesh 在保证业务简化有效的同时，进一步屏蔽了多种开发语言带来的障碍。它完全基于网络层面和协议层面作为出发点，达到“以不变应万变”的效果。

支撑服务

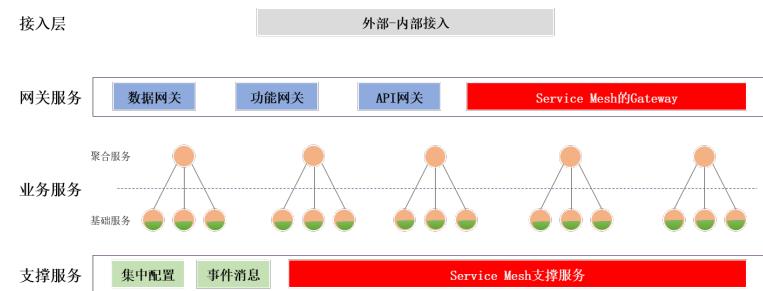
从企业业务的价值角度考虑，其实支撑服务更多属于“资源消耗”品，虽然如此，它却是企业应用架构不可或缺的一部分。从单一的业务调用—>微服务体系业务调用—>**Service Mesh 体系业务调用**的转变方式中，可以看出支撑服务处于一个层级不断下降的过程。而依赖于 ServiceMesh 的定位，最终一些支撑服务会作为基础设施的形态呈现出来，这也是未来发展的趋势所在。支撑服务在企业架构的形态转变如图所示：



- **传统架构**: 传统架构下，支撑服务，业务服务基本上没有明确的边界区分，实现方式上都通过代码杂糅在一起。
 - **微服务架构**: 微服务架构下，支撑服务，业务服务能够初步分离，但是需要保证代码框架的统一性和依赖性，跨语言受限比较严重。
 - **Service Mesh架构**: Service Mesh架构下，支撑服务，业务服务能够彻底分离，不收语言限制，唯一需要考虑的是不同协议的支持情况。

通过支撑服务的转变形态可以看出，支撑服务与业务服务分离是必然趋势，而最终的受限取决于多元化的网络协议的处理分析能力。当然我们需要明确一个事实：就Service Mesh目前的发展趋势和定位而言，并不能够完全取代所有的支撑服务，例如事件消息、配置管理等。我们更多期望它能够帮助解决应用服务在网络层面需要面对的场景和问题。这也是它发挥价值的地方所在。

通过对网关服务、业务服务、支撑服务在不同体系架构下的转变，我们清晰的认识到Service Mesh能够帮助我们重点解决微服务体系下繁琐的“旁路支撑”服务，保证业务服务的简单有效性。通过演化分析，最终基于Service Mesh的企业应用架构如下图：



从图中可以看到 Service Mesh 架构下重点做了三件事情：

- 网关层的接入工作，无论是外部请求接入，还是内部服务请求转发，都可以基于 Service Mesh 提供的不同类型的 gateway 实现，同时还可以保证策略的统一控制和管理。省略了独立的网关管理控制台。

灵雀云

(<https://www.kubernetes.io>)

监控

(<https://www.kubernetes.org.cn/tar>)

网络

(<https://www.kubernetes.or>

Kubernetes 版本资讯

- Kubernetes v1.14 正式版已发布
(<https://www.kubernetes.org.cn/5204.html>)
 - Kubernetes v1.13 正式版已发布
(<https://www.kubernetes.org.cn/4896.html>)
 - Kubernetes v1.13 beta.1 发布
(<https://github.com/kubernetes/kubernetes/releases/tag/v1.13%2Bbeta.1>)
 - Kubernetes v1.12 已发布
(<https://www.kubernetes.org.cn/4617.html>)
 - Kubernetes v1.11.2 版本已发布
(<https://github.com/kubernetes/kubernetes/releases/tag/v1.11.2>)

最新评论

暴走少年哈尔 2天前说：

楼下你好，看了你的帖子安装之后node节点在master上显示NotReady，查看报错为【6月 01 23:38:52 k8s-master kubelet[19475]: W0601 23:38:
(<https://www.kubernetes.org.cn/5213.html#comment-1144>)

VPS234 4天前说·

好像国内的网络下载包有点问题
(<https://www.kubernetes.org.cn/5425.html#comment-1143>)

lentil1016 4天前说：

那就不是一路顺风了，那是半路就出问题了
(<https://www.kubernetes.org.cn/5213.html#comment-1142>)

lentil1016 4天前说：

书名：人生の本質
作者：井上正樹
出版社：新星出版社
出版地：中国
出版时间：2018年

- 针对业务服务，增加了 Sidecar 的代理模型，用来处理所有的入站和出站流量，并且配合支撑服务的控制策略，实现业务服务的旁路控制功能。
- 统一面向网络的支撑服务，基于控制与数据分离的思想，根据业务的运行情况，提高企业应用运行过程中的动态控制能力。

同样我们也意识到，利用 Service Mesh 处理服务通信的能力，替换需要不同组件支撑的“注册发现”，“容错限流”“认证授权”“日志搜集”，“监控告警”“流量控制”等功能。在减少组件代码开销的同时，将企业应用的支撑服务进一步下移。无论是开发人员，还是领域专家，可以集中精力用来处理应用业务，而不用在维护第三方的不同的功能组件上“浪费时间”。而业务运维人员，通过 Service Mesh 的控制平台，能够实时监测企业服务的运行状态，而不需要向之前那样花费精力维护不同的工具和组件。最后让我们一起讨论一下，Service Mesh 是如何做到这些的。**Service Mesh 本质上并没有采用任何技术上的创新，更多是思想层面的变革**。我们认为有几个转变是需要提出来的：

- 层级转变**: Service Mesh 在设计思路上，把自己不再定位成企业应用组件，而是把自己下沉到基础设施层，成为基础设施的一部分，这样层级的转变就与企业业务本身划清楚界限。
- 方式转变**: Service Mesh 在实现思路上，高度抽象，聚焦于通信链路本身，而不是聚焦于组件功能上，从网络层入手，抓住了服务通信交互的本质。
- 控制转变**: Service Mesh 将控制和实现分离，提供统一，灵活的控制入口，能够快速方便的针对业务场景进行自定义处理。

最后的最后需要说明 Service Mesh 还不完善，还有很多问题需要在实际的企业应用过程中逐步去解决，让我们一起拭目以待吧。

本文由博云研究院原创发表，转载请注明出处。



关注微信公众号，加入社区



(<http://www.kubernetes.org.cn/service-mesh/>)

上一篇: Telepresence本地化部署 (https://www.kubernetes.org.cn/5313.html)
下一篇: (原创)docker容器搭建prometheus+grafana+cadvisor+elasticsearch镜像 (https://www.kubernetes.org.cn/5358.html)

标签: service mesh (<https://www.kubernetes.org.cn/tags/service-mesh>)

微服务 (<https://www.kubernetes.org.cn/tags/%e5%be%ae%e6%9c%8d%e5%8a%a1>)

相关推荐

- 服务迁移之路 | Spring Cloud向Service Mesh转变 (<https://www.kubernetes.org.cn/5418.html>)
- 基于事件驱动机制，在Service Mesh中进行消息传递的探讨 (<https://www.kubernetes.org.cn/5386.html>)
- Linkerd or Istio? 哪个Service Mesh框架更适合你? (<https://www.kubernetes.org.cn/5370.html>)
- Telepresence: 让微服务本地开发不再难 (<https://www.kubernetes.org.cn/5313.html>)
- Spring Cloud 微服务部署在Rainbond的优势 (<https://www.kubernetes.org.cn/5272.html>)
- Service Mesh在企业级应用的生存之道 (<https://www.kubernetes.org.cn/5263.html>)
- Rainbond v5.1.2发布，微服务架构应用便捷管理和交付 (<https://www.kubernetes.org.cn/5242.html>)
- 微服务五种开源API网关实现组件对比 (<https://www.kubernetes.org.cn/5224.html>)





社区交流

2 4 4 6 3 .

提交评论

昵称

昵称 (必填)

邮箱

邮箱 (必填)

网址

网址

© 2019 Kubernetes中文社区 鲁ICP备16060255号-2 (<http://www.miitbeian.gov.cn/>) 版权说明 (<https://www.kubernetes.org.cn/版权声明>) 联系我们
(<https://www.kubernetes.org.cn/联系我们>) 广告投放 (<https://www.kubernetes.org.cn/广告投放>) 法律声明：本网站不隶属于谷歌或 Alphabet 公司 | kubernetes、kubernetes 标识及任何相关标志均为 Google LLC 公司的商标。

