全方位详解Service Mesh（服务网格）

RancherLabs 2019-09-12 10:29:06 855 收藏 4

分类专栏： 云计算 kubernetes

版权

Service mesh是近几年才出现的一个新兴概念。它可以解决微服务之间通信愈发复杂的问题。那么什么是Service mesh？它有什么具体的功能？它的架构又是如何的呢？它与Kubernetes的关系是怎样的？所有答案戳文了解！

在数字化转型的旗帜下，IT界的一大变化是大型单体应用程序被分解为微服务架构，即小型、离散的功能单元，并且这些应用程序在容器中运行。包含所有服务代码以及依赖项的软件包被隔离起来，并且能轻松从一个服务器迁移到另一个。

像这样的容器化架构很容易在云中扩展和运行，并且能够快速迭代和推出每个微服务。然而，当应用程序越来越大并且在同一个服务上同时运行多个实例时，微服务之间通信将会变得愈发复杂。Service mesh的出现将解决这一问题，它是一个新兴的架构形式，旨在以减少管理和编程开销的形式来连接这些微服务。

什么是Service mesh？

关于Service mesh的定义，最为广泛接受的观点是：它是一种控制应用程序不同部分彼此共享数据的方式。这一描述包含了service mesh的方方面面。事实上，它听起来更像是大多数开发人员从客户端-服务器应用程序中熟悉的中间件。

Service mesh也有其独特之处：它能够适应分布式微服务环境的独特性质。在搭建在微服务中的大规模应用程序中，有许多既定的服务实例，它们跨本地和云服务器运行。所有这些移动部件显然使得各个微服务难以找到他们需要与之通信的其他服务。Service mesh可以在短时间内自动处理发现和连接服务，而无需开发人员以及各个微服务自行匹配。

我们可以将service mesh等同为软件定义网络（SDN）的OSI网络模型第7层。正如SDN创建一个抽象层后网络管理员不必处理物理网络连接，service mesh将解耦在抽象架构中的与你交互的应用程序的底层基础架构。

随着开发人员开始努力解决真正庞大的分布式架构的问题，service mesh的概念适时地出现了。这一领域的第一个项目是Linkerd，它一开始是Twitter内部项目的一个分支。Istio是另一个十分流行的service mesh项目，它起源于Lyft，现在这一项目获得了许多企业的支持。

Service mesh负载均衡

Service mesh其中一个关键功能是负载均衡。我们常常将负载均衡视为网络功能——你想要防止服务器或网络链接被流量淹没，因此相应地你会路由你的数据包，而Service mesh在应用程序层面也在执行类似的事情。

本质上，Service mesh的工作之一是跟踪分布在基础设施上的各种微服务的哪些实例是“最健康的”。它可能对他们进行调查来查看它们如何工作的或跟踪哪些实例对服务请求响应缓慢并将后续请求发送到其他实例。此外，service mesh也会为网络路由做类似的工作，如果发现当消息需要很长时间才能送达，那么service mesh将会采用其他路由进行补偿。这些减速可能是由于底层硬件出现问题，或者仅仅是由于服务因请求过载或处理能力不足导致的。但没有关系，service mesh会找到另一个相同服务的实例，然后将其路由以替代响应缓慢的实例，高效利用了整个应用程序的资源。

Service mesh vs Kubernetes

如果你稍微熟悉基于容器的架构，你可能会想Kubernetes这个流行的开源容器编排平台能否适合这种情况。毕竟，Kubernetes不就是管理着你的容器之间如何互相通信的吗？你可将Kubernetes“服务”资源视为非常基础的service mesh，因为它提供服务发现和请求的轮询调度均衡。但是完整的service mesh则提供更丰富的功能，如管理安全策略和加密、“断路”以暂停对缓慢响应的实例的请求以及如上所述的负载均衡等。

请记住，大多数service mesh确实需要像Kubernetes这样的编排系统。Service mesh只是提供扩展功能，而非替代编排平台。

Service mesh vs API 网关

每个微服务都会提供一个API，它会作为其他服务与其通信的手段。这引发了service mesh与其他更传统的API管理形式（如API网关）之间的差异问题。API网关位于一组微服务和“外部”世界之间，它根据需要路由服务请求，以便请求者不需要知道它正在处理基于微服务的应用程序即可完成请求。而service mesh调解微服务应用程序内部的请求，各种组件完全了解其环境。

另一方面，service mesh用于优化集群内东西流量（server-server流量），API网关用于进出集群的南北流量（server-client流量）。但service mesh目前依旧处于早期阶段还在不断发展变化中。许多service mesh（包括Linkerd和Istio）现在已经可以提供南北功能。

Service mesh 架构

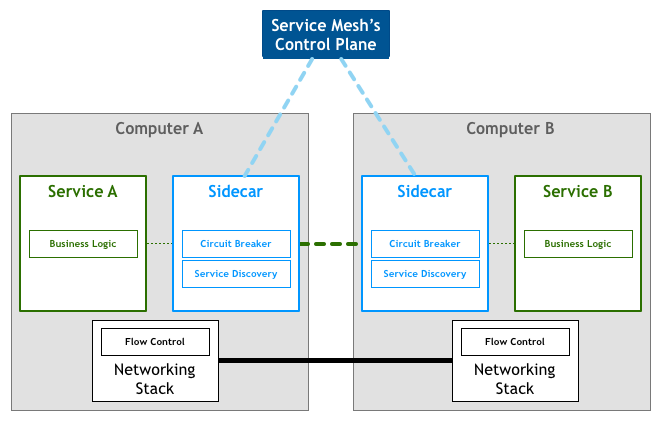
Service mesh这一概念其实出现的时间并不长，并且已经有相当数量的不同的方法来解决“service mesh”的问题，如管理微服务通信。目前，确定了三种service mesh创建的通信层可能存在的位置：

每个微服务导入的library

在特定节点提供服务给所有容器的节点agent

与应用程序容器一起运行的sidecar容器

基于sidecar的模式目前是service mesh最受欢迎的模式之一，以至于它在某种程度上已经成为了service mesh的代名词。尽管这种说法并不严谨，但是sidecar已经引起了很大的关注，我们将在下文更详细地研究这一架构。



Sidecar

Sidecar容器与你的应用程序容器一起运行意味着什么呢？在这类service mesh中每个微服务容器都有另一个proxy容器与之相对应。所有的服务间通信的需求都会被抽象出微服务之外并且放入sidecar。

这似乎很复杂，毕竟你有效地将应用程序中的容器数量增加了1倍。但你使用的这一种设计模式对于简化分布式应用程序至关重要。通过将所有的网络和通信代码放到单独的容器中，将其作为基础架构的一部分，并使开发人员无需将其作为应用程序的一部分实现。

本质上，你所留下的是一个聚焦于业务逻辑的微服务。这个微服务不需要知道如何在其运行的环境中与所有其他服务进行通信。它只需要知道如何与sidecar进行通信即可，剩下的将由sidecar完成。

Service mesh明星项目：Linkerd、Envio、Istio、Consul

那么说了这么多，什么是可用的service mesh呢？目前，这一领域还没有出现完全现成的商业产品。大部分的service mesh只是开源项目，需要通过一定的操作步骤才能实现，现在比较知名的项目有：

Linkerd：2016年发布，是这些项目中最老的。Linkerd是从Twitter开发的library中分离出来的。在这一领域另一位重量型选手，Conduit，已经进入了Linkerd项目并构成了Linkerd 2.0的基础。

Envoy：由Lyft创建，为了能够提供完整的service mesh功能，Envoy占据“数据平面”的部分，与其进行匹配。

Istio：由Lyft、IBM与google联合开发，Istio可以在不修改微服务源代码的情况下，轻松为其加上如负载均衡、身份验证等功能，它可以通过控制Envoy等代理服务来控制所有的流量。此外，Istio提供容错、金丝雀部署、A/B测试、监控等功能，并且支持自定义的组件和集成。

Rancher 2.3 Preview2版本上开始支持Istio，用户可以直接在UI界面中启动Istio并且可以为每个命名空间注入自动sidecar。Rancher内置了一个支持Kiali的仪表盘，简化Istio的安装和配置。这一切让部署和管理Istio变得简单而快速。

HashiCorp Consul：与Consul 1.2一起推出了一项名为Connect的功能，为HashiCorp的分布式系统添加了服务加密和基于身份的授权，可用于服务发现和配置。

哪个service mesh适合你？如果要进行一个全面的比较的话，超出了本文所涉及的范围。但上述的所有产品都已经在大型且严苛的环境中得到验证。目前，Linkerd和Istio包含最丰富的功能集，但一切都还在迅速发展中，现在下定论还为时过早。

————————————————

版权声明：本文为CSDN博主「RancherLabs」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/RancherLabs/article/details/100764041