Service Mesh

Sep 19th, 2018

申恒恒

openstack™

INTRODUCTION

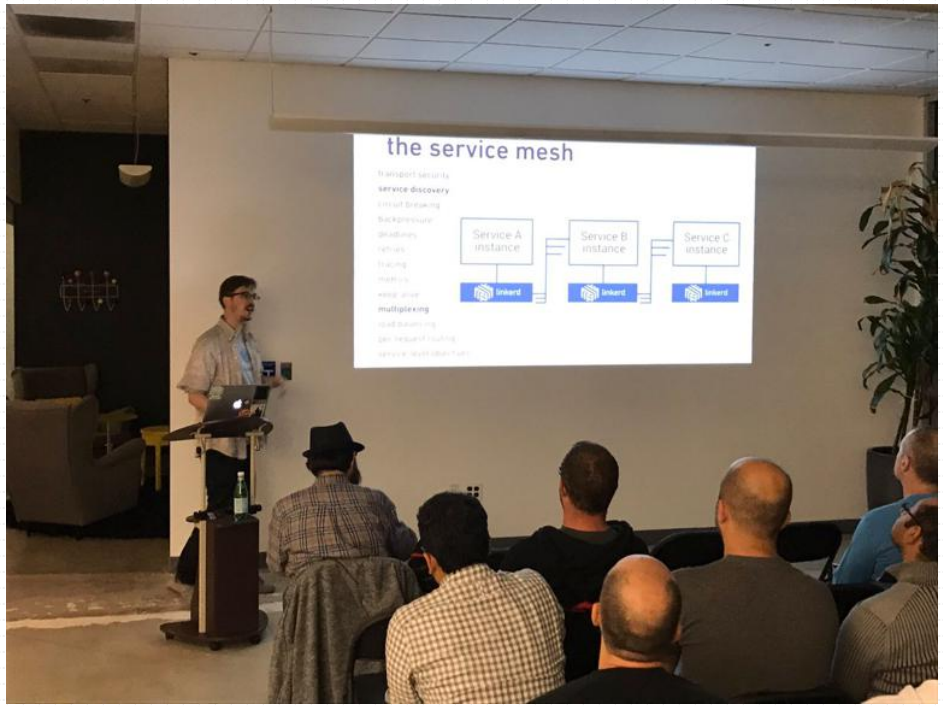
- Service Mesh
  - 中文名：服务网格
- 被誉为下一代的“微服务架构”.
- 云原生（Cloud Native）技术栈的关键组件之一.

服务网格是一个**基础设施层**，用于处理服务间通讯。云原生应用有着复杂的服务拓扑，服务网格负责在这些拓扑中**实现请求的可靠传递**。在实践中，服务网格通常实现为一组**轻量级网络代理**，它们与应用程序部署在一起，而**对应用程序透明**。

openstack™

HISTORY I

- Engieers of Leave Twitter: 2016.01
  - William Morgan && Oliver Gould
    - Create a release version for linkerd 0.0.7
    - Buoyant - 业界第一个 Service Mesh 项目
- SF Microservice Meetup: 2016.09
  - “Service Mesh”这个词汇第一次在公开场合被使用.
  - 藉此，Buoyant 由公司走向社区

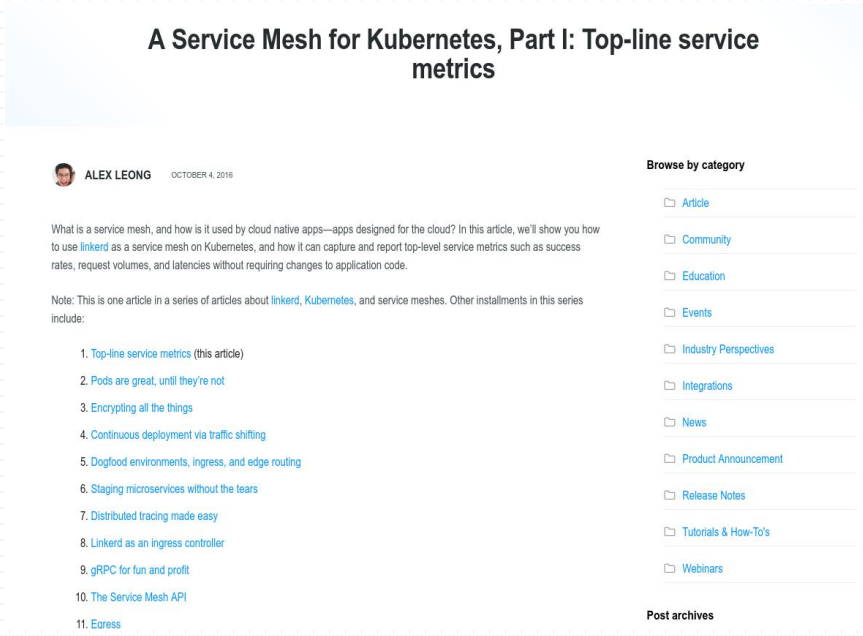


HISTORY II

- 2016年10月，Alex Leong开始在Buoyant公司的官方Blog中开始“A Service Mesh for Kubernetes”系列博客的连载。随着”The services must mesh”口号的喊出，buoyant和Linkerd开始service mesh概念的布道。
- 2017年1月23日，Linkerd加入CNCF，类型为“Service Mesh”。这是Service Mesh技术非常重要的历史事件，代表着CNCF社区对Service Mesh理念的认同和赞赏。
- 2018年7月，CNCF社区正式发布了Cloud Native的定义1.0版本，非常明确的指出云原生代表技术包括容器、服务网格、微服务、不可变基础设施和声明式API，将Service Mesh技术放在了一个前所未有的高度。

CLOUD NATIVE COMPUTING FOUNDATION

Kubernetes Orchestration
Prometheus Monitoring
OpenTracing Distributed Tracing API
Fluentd Logging
linkerd Service Mesh
gRPC Remote Procedure Call
CoreDNS Service Discovery
containerd Container Runtime
rkt Container Runtime
CNI Networking API
Envoy Service Mesh
Jaeger Distributed Tracing
Notary Security
TUF Software Update Spec


A Service Mesh for Kubernetes, Part I: Top-line service metrics

ALEX LEONG OCTOBER 4, 2016

What is a service mesh, and how is it used by cloud native apps—apps designed for the cloud? In this article, we'll show you how to use linkerd as a service mesh on Kubernetes, and how it can capture and report top-level service metrics such as success rates, request volumes, and latencies without requiring changes to application code.

Note: This is one article in a series of articles about linkerd, Kubernetes, and service meshes. Other installments in this series include:

1. Top-line service metrics (this article)
2. Pods are great, until they're not
3. Encrypting all the things
4. Continuous deployment via traffic shifting
5. Dogfood environments, ingress, and edge routing
6. Staging microservices without the tears
7. Distributed tracing made easy
8. Linkerd as an ingress controller
9. gRPC for fun and profit
10. The Service Mesh API
11. Egress

Browse by category

- Article
- Community
- Education
- Events
- Industry Perspectives
- Integrations
- News
- Product Announcement
- Release Notes
- Tutorials & How-To's
- Webinars

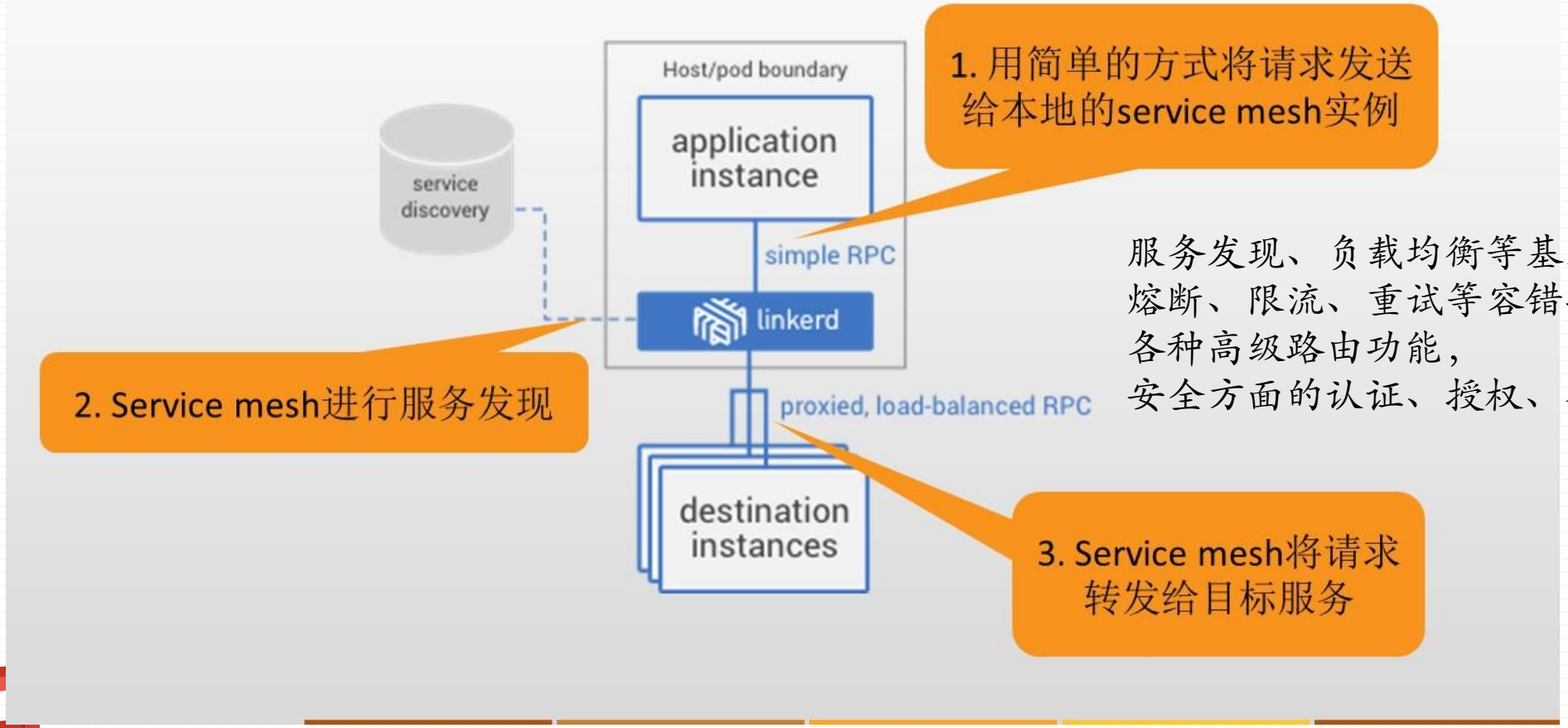
Post archives

openstack

Service Mesh详解

场景1. 单个服务调用

部署模型：单个服务调用，表现为sidecar

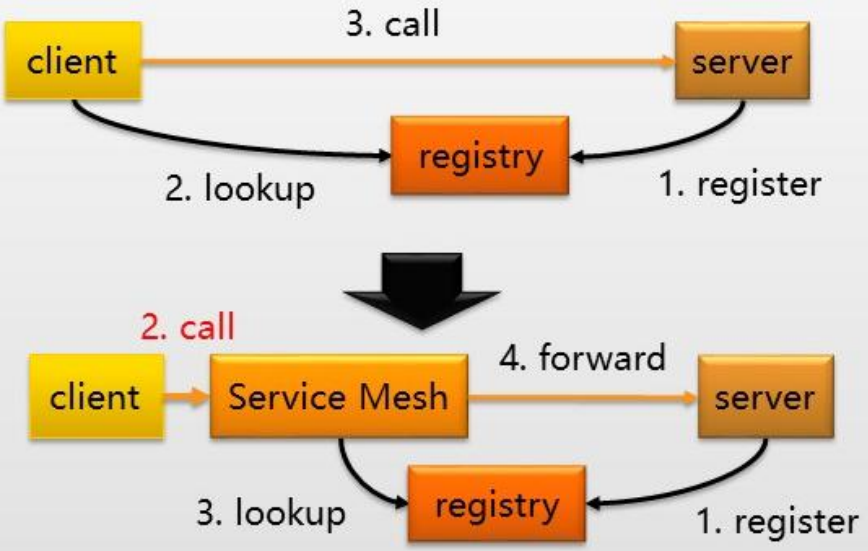


服务发现、负载均衡等基本功能，
熔断、限流、重试等容错功能，
各种高级路由功能，
安全方面的认证、授权、鉴权和加密等

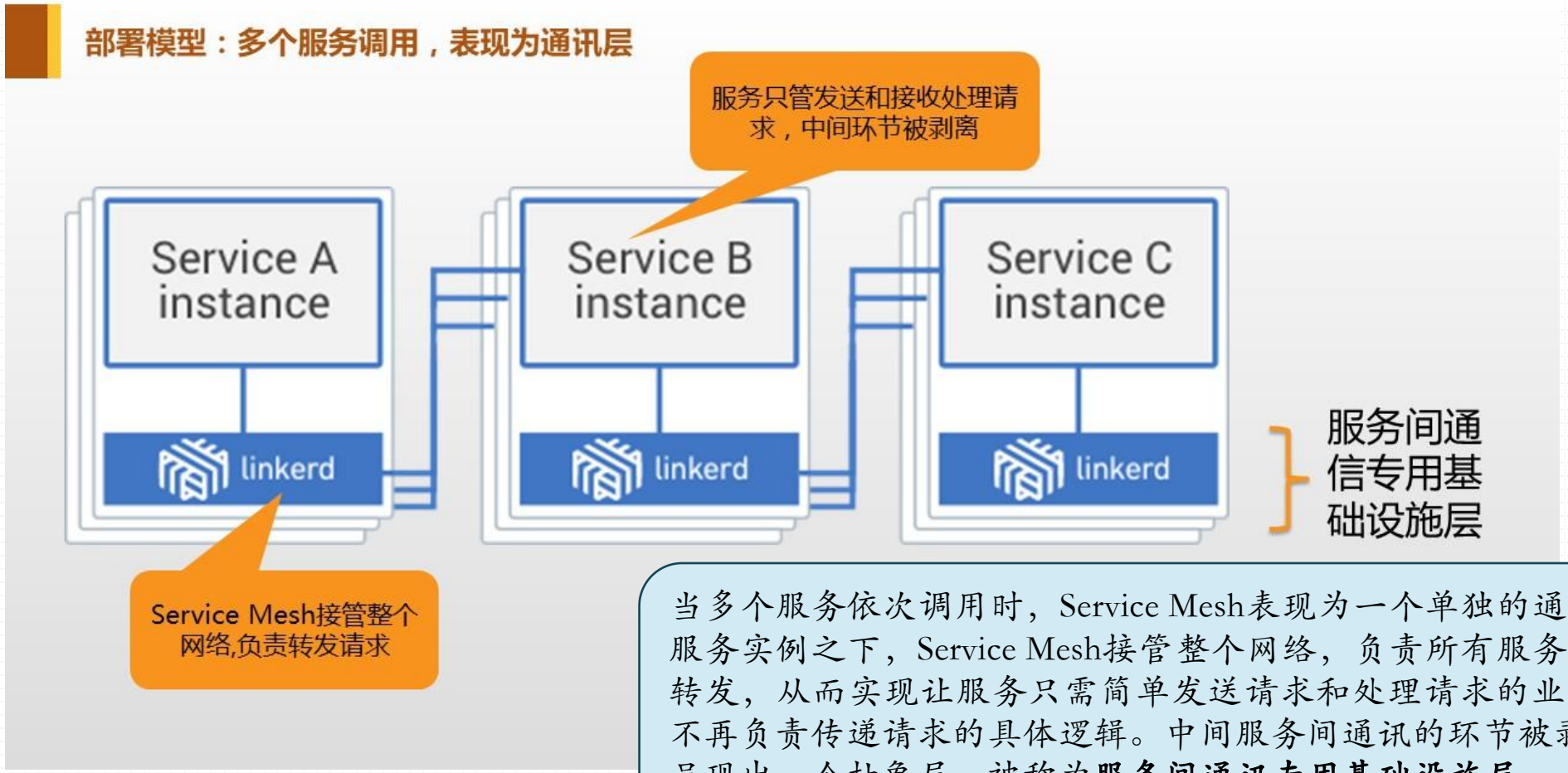
Why Sidecar?

Service Mesh 原理

- Service Mesh通过在请求调用的路径中增加Sidecar，将原本由客户端（通常通过类库）完成的复杂功能，转移到Sidecar中，实现对客户端的简化和服务间通讯控制权的转移。



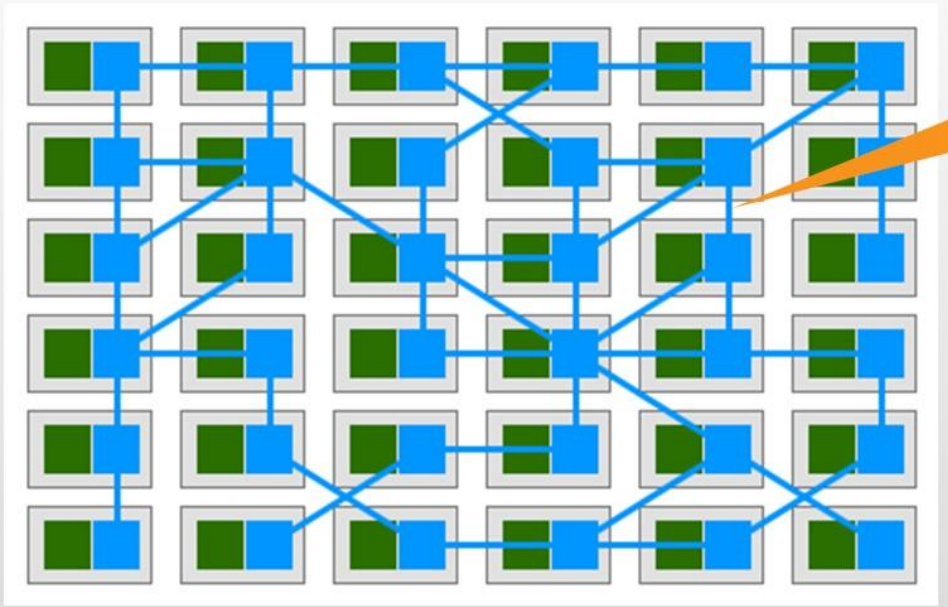
场景2.多个服务调用



当多个服务依次调用时，Service Mesh表现为一个单独的通讯层。在服务实例之下，Service Mesh接管整个网络，负责所有服务间的请求转发，从而实现让服务只需简单发送请求和处理请求的业务处理，不再负责传递请求的具体逻辑。中间服务间通讯的环节被剥离出来，呈现出一个抽象层，被称为**服务间通讯专用基础设施层**。

openstack

场景3.大量服务调用



此时Service Mesh体现出来的依然是一个通讯层，只是这个通讯层内部更加复杂，不是简单的顺序调用关系，而是彼此相互调用，形成网状。

openstack

Service Mesh 深入

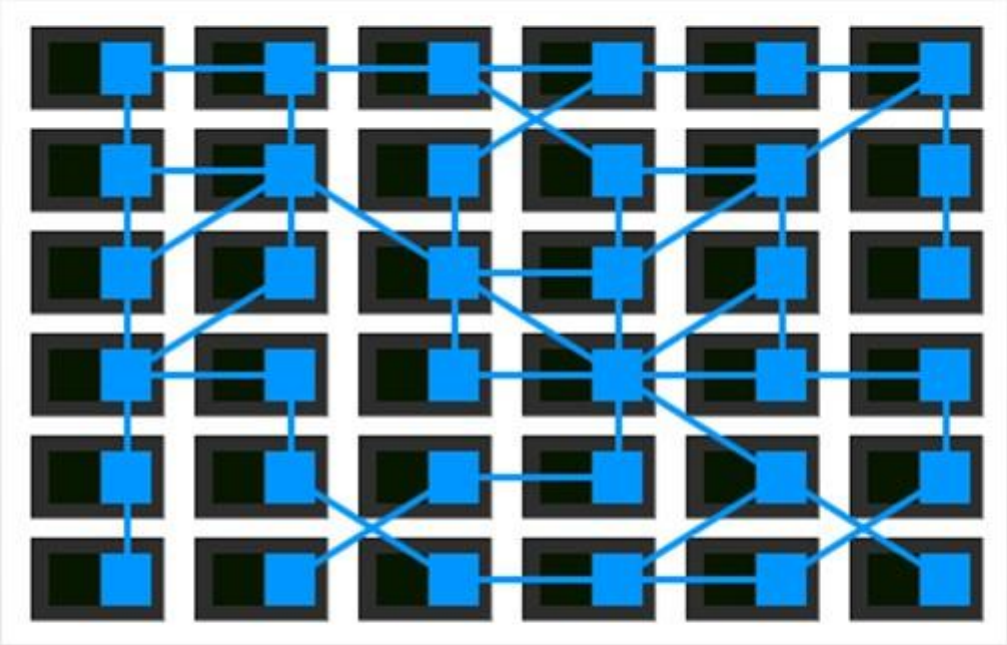
服务网格是一个**基础设施层**，用于处理服务间通讯。云原生应用有着复杂的服务拓扑，服务网格负责在这些拓扑中**实现请求的可靠传递**。在实践中，服务网格通常实现为一组**轻量级网络代理**，它们与应用程序部署在一起，而**对应用程序透明**。

- 抽象：Service Mesh是一个抽象层，负责完成服务间通讯。但是和传统类库方式不同的是，Service Mesh将这些功能从应用中剥离出来，形成了一个单独的通讯层，并将其下沉到基础设施。
- 功能：Service Mesh负责实现请求的可靠传递，从功能上说，和传统的类库方式并无不同，原有的功能都继续提供，甚至可以做的更多更好。
- 部署：Service Mesh在部署上体现为轻量级网络代理，以Sidecar的模式和应用程序一对一部署在一起，两者之间的通讯是远程调用，但是走的是localhost。
- 透明：Service Mesh是应用程序是透明的，其功能实现完全独立于应用程序。应用程序无需关注Service Mesh的具体实现细节，甚至对Service Mesh的存在也可以无感知。带来的一个巨大优势是Service Mesh可以独立的部署升级，扩展功能修复缺陷而不必改动应用程序。

Service Mesh定义回顾

Sidecar和调用关系形成完整的网络，代表服务间复杂的调用关系，承载着系统内的所有应用。

- 抽象：基础设施层
- 功能：实现请求的可靠传递
- 部署：轻量级网络代理
- 关键：对应用程序透明

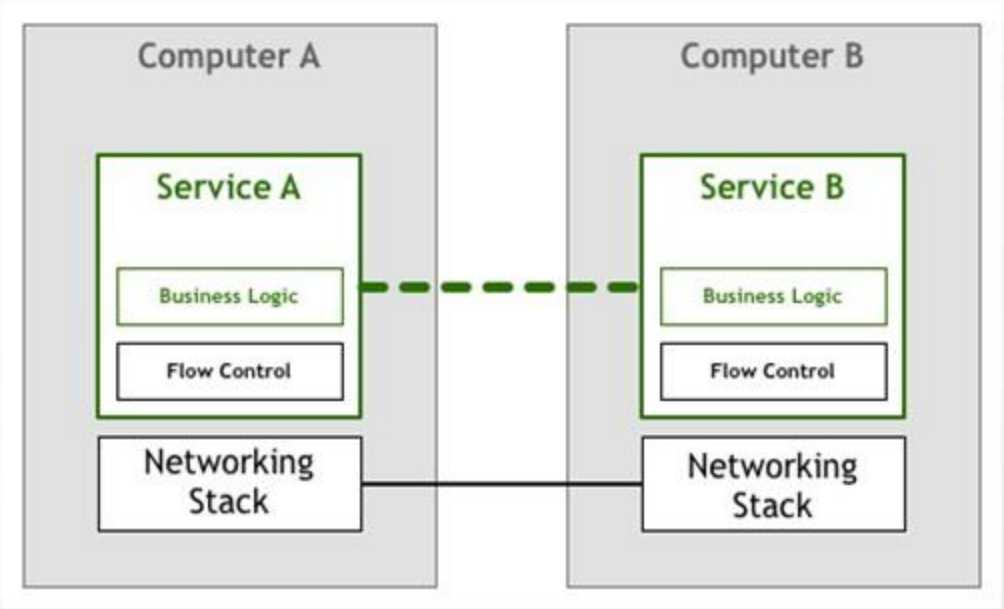


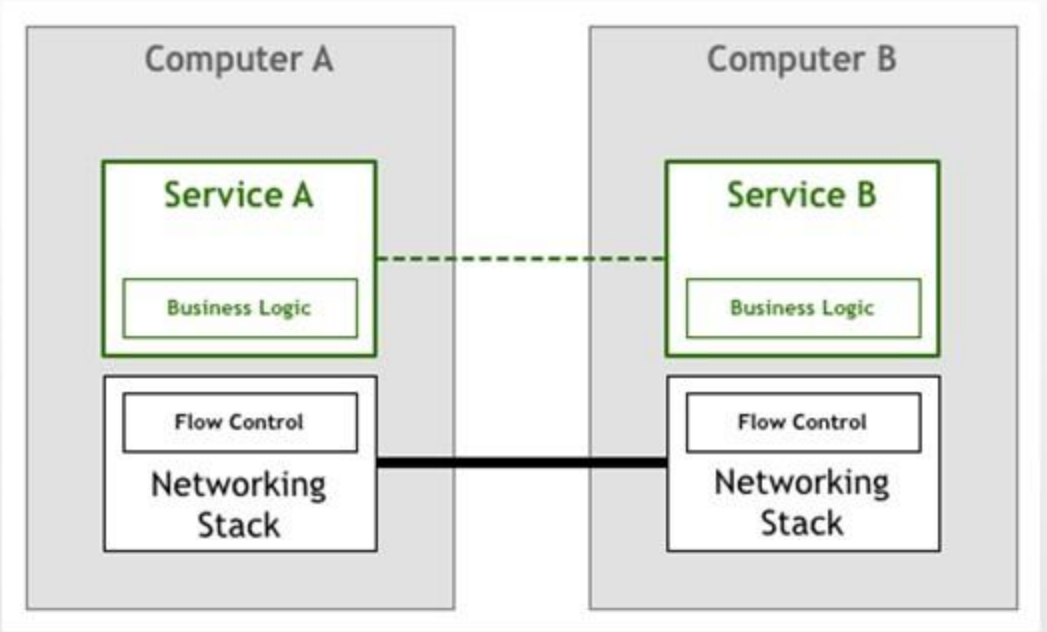
不再将代理视为单独的组件，而是强调由这些代理连接而形成的网络

Service Mesh演进历程

Hint: Service Mesh这个词汇直到2016年9才出现，但是和Service Mesh一脉相承的技术很早就出现了，经过了长期的发展和演变，才形成了今天的Service Mesh，并且这个演进的过程还在继续。

Service Mesh的演进1：微服务之前





- **远古时代：第一代网络计算机系统**

开发人员需要在自己的代码里处理网络通讯的细节问题，如数据包顺序，流量控制等。结果就是应用程序需要处理网络逻辑，导致网络逻辑和业务逻辑混杂在一起

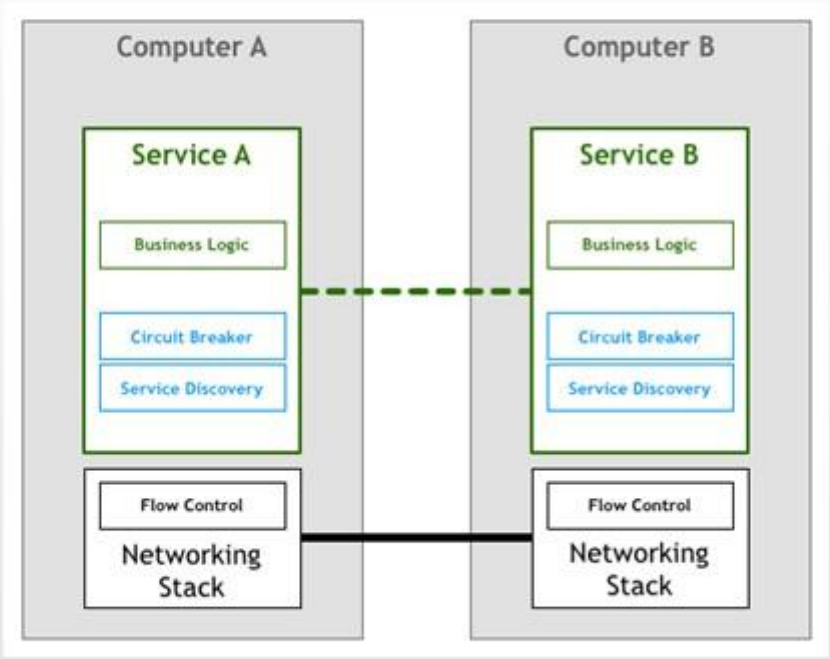
- **TCP/IP出现**

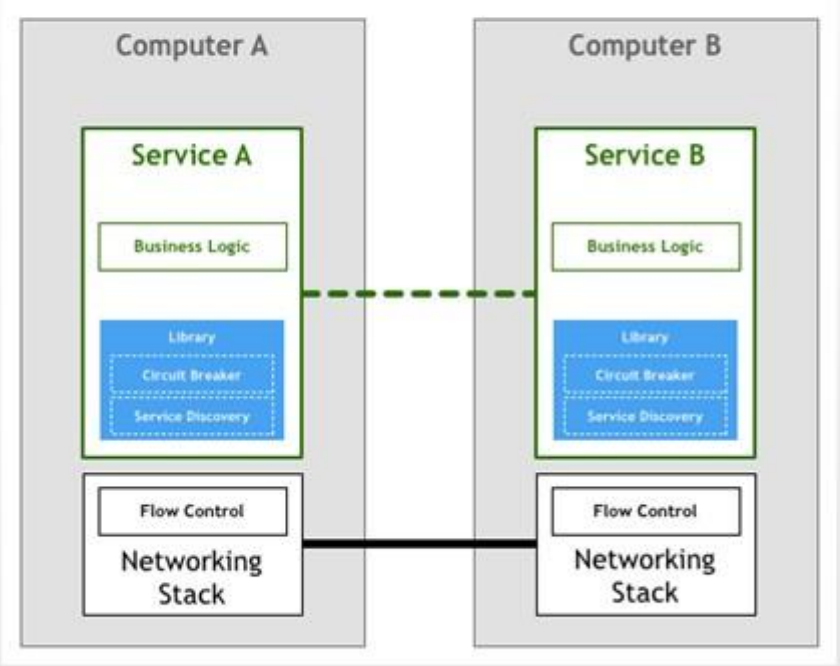
解决了流量控制等问题。

尽管网络逻辑的代码依然存在，但已经从应用程序里抽取出来，成为操作系统网络层的一部分。

应用程序和开发人员得以解脱☺

Service Mesh的演进2：微服务时代





- **第一代微服务架构**

开发人员需要在自己的代码里处理一系列问题，如服务发现，负载均衡，熔断，重试等。导致应用程序中，在业务逻辑外混杂大量非功能的代码。

- **类库和框架出现**

典型如Netflix OSS套件，Spring Cloud框架，开发人员只要写少量代码，甚至几个注解就搞定。Spring Cloud因此风靡一时，几乎成为微服务的代名词。**好像一切都很完美的样子？:)**

侵入式框架的痛点

以Spring Cloud/Dubbo为代表的传统微服务框架，是以类库的形式存在，通过重用类库来实现功能和避免代码重复。但在以运行时操作系统进程的角度来看，这些类库还是渗透进了打包部署之后的业务应用程序，和业务应用程序运行在同一进程内。所谓**侵入式框架**的称谓由此而来。

痛点1：内容多，门槛高

- **spring cloud**
  - spring-cloud-commons
  - spring-cloud-Netflix
  - spring-cloud-sleuth
  - spring-cloud-gateway
  - spring-cloud-bus
  - spring-cloud-consul
  - spring-cloud-config
  - spring-cloud-security
  - spring-cloud-zookeeper
  - spring-cloud-aws
  - spring-cloud-cloudfoundry

- **Netflix OSS**
  - eureka
  - hystrix
  - Turbine
  - archaius
  - Atlas
  - Feign
  - Ribbon
  - zuul

需要多长时间，才能让整个开发团队掌握并熟练使用？

- 业务开发团队的强项往往不是技术，而是对业务的理解，对整个业务体系的熟悉程度
- 业务应用的核心价值在于业务实现，微服务是手段而不是目标，在学习和掌握框架上投入太多精力，在业务逻辑的实现上的投入必然受影响。
- 业务团队往往承受极大的业务压力，时间人力永远不足

痛点2：服务治理功能不够齐全

基本功能

- 服务注册与服务发现
  - 主动健康检查
- 负载均衡
  - 随机轮询之外的高级算法
- 故障处理和恢复
  - 超时
  - 熔断
  - 限流
  - 重试
- RPC支持
- HTTP/2支持
- 协议转换/提升

高级功能

- 加密
  - 密钥和证书的生成，分发，轮换和撤销
- 认证/授权/鉴权
  - OAuth
  - 多重授权机制
    - ABAC
    - RBAC
    - 授权钩子
- 分布式追踪/APM
- 监控
  - 日志
  - 度量（Metrics）
  - 仪器仪表（instrumentation）

运维测试类

- 动态请求路由
  - 服务版本
  - 分段服务(staging service)
  - 金丝雀(canaries)
  - A/B测试
  - 蓝绿部署(blue-green deploy)
  - 跨DC故障切换
  - 黑暗流量(dark traffic)
- 故障注入
- 高级路由支持
  - 高度可定制：script, DSL
  - 可灵活配置的规则，即时生效

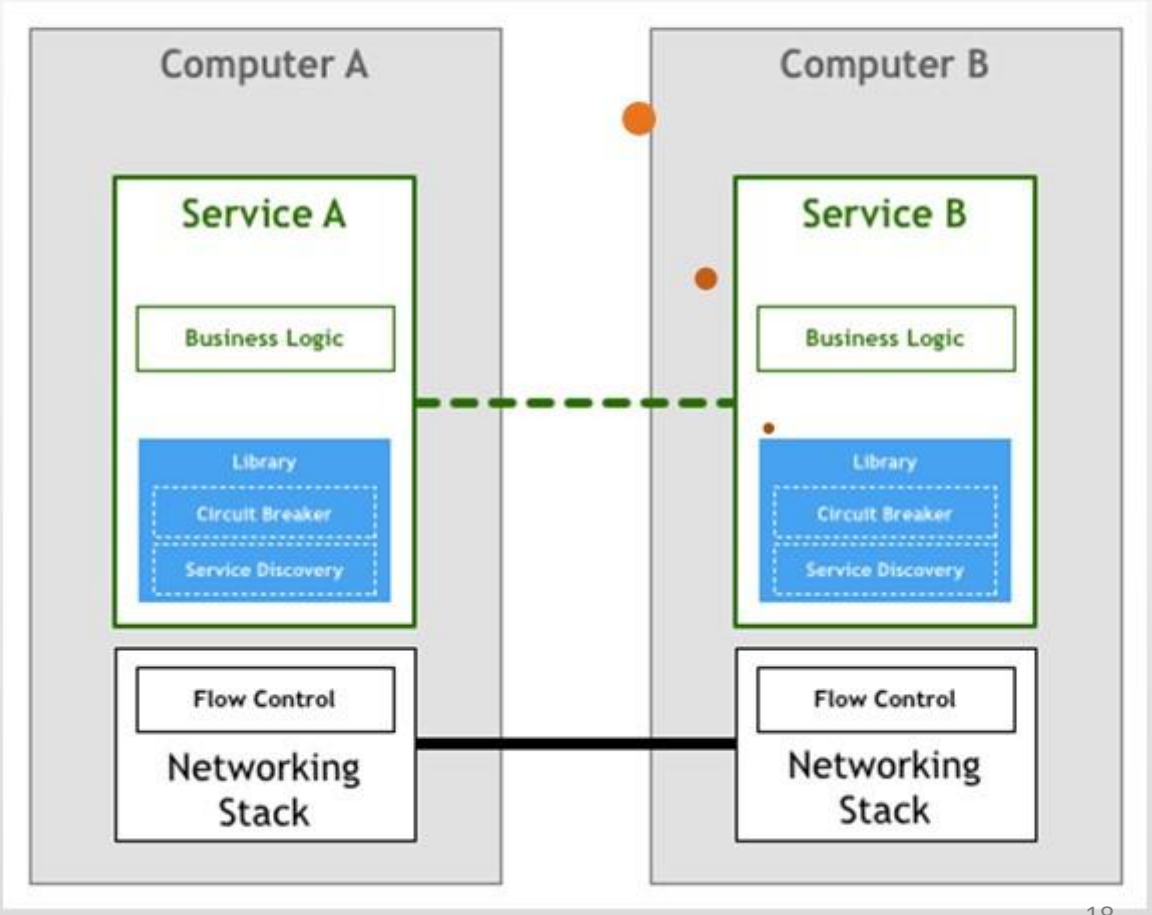
你打算投多少时间和精力进去？

痛点3：说好的跨语言呢？

微服务带来的一个巨大优势，就是容许不同的服务根据实际需要采用不同的编程语言。但是，当我们将代码封装到类库和框架时，有个小问题冒出来了☺：

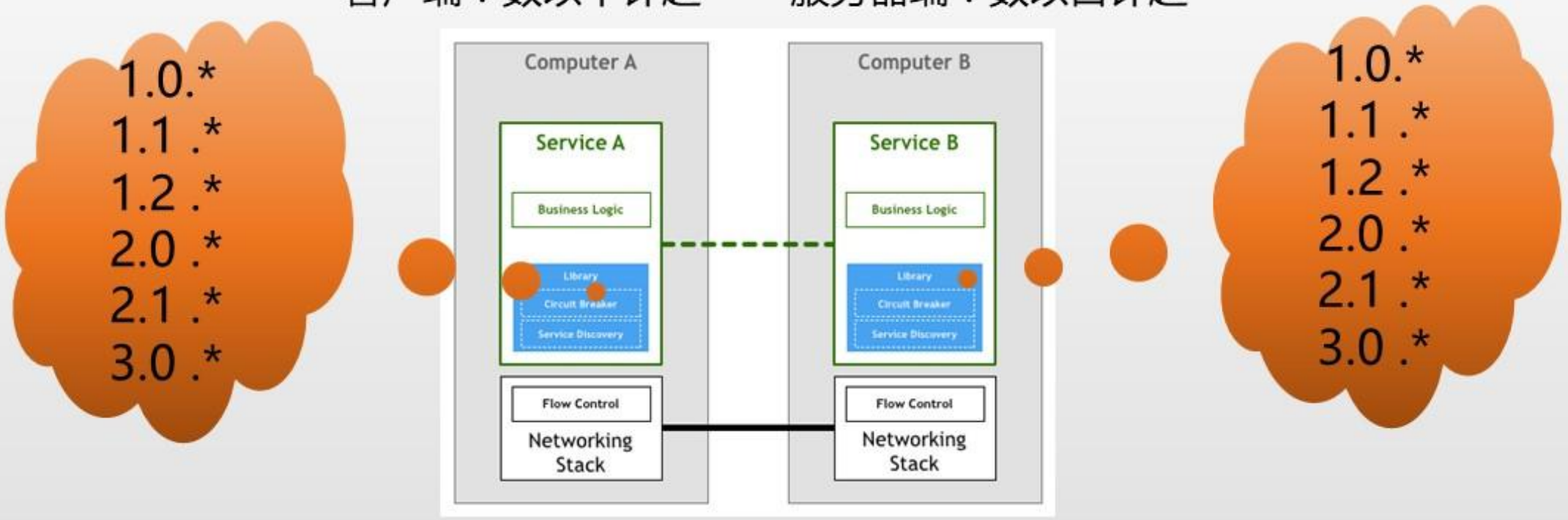
我们需要为多少种语言提供类库？

- **主流编程语言**
  - Java
    - Scala
    - Groovy
    - Kotlin
  - C
  - C++
  - C#
  - Python
  - PHP
  - Ruby
- **新兴编程语言**
  - Golang
  - Node.js
  - Rust
  - R
  - Lua/OpenResty



痛点4：升级怎么办？

客户端：数以千计起　　服务器端：数以百计起



别忘了：编程语言，再*N ☺

思考

所有的努力，都是为了保证将请求发到正确的地方

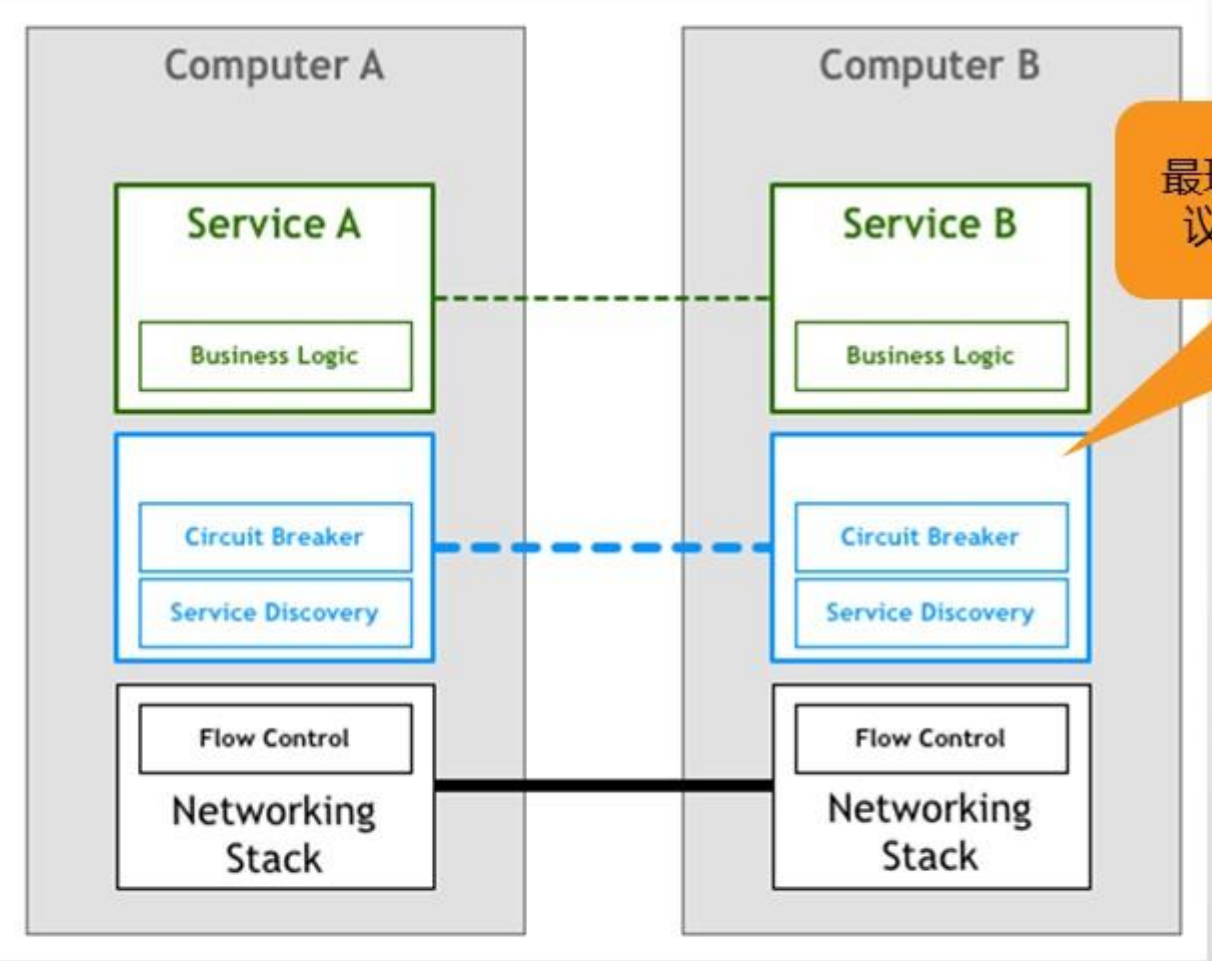
适用于所有的微服务

根源 目标 本质 普适

无论过程如何，请求从不更改

最艰巨的挑战，与服务本身无关，而是服务间的通讯

Service Mesh的演进3：技术栈下移



- **先驱者：使用代理解决部分问题**

使用nginx，HaProxy，Apache等反向代理，避免服务间产生直接连接。所有流量都经由代理，代理实现需要的特性如负载均衡。

但是：功能过于简陋

Service Mesh的演进4：Sidecar出现

- **Airbnb**

  2013年，Airbnb开发了Synapse和Nerve

- **Netflix**

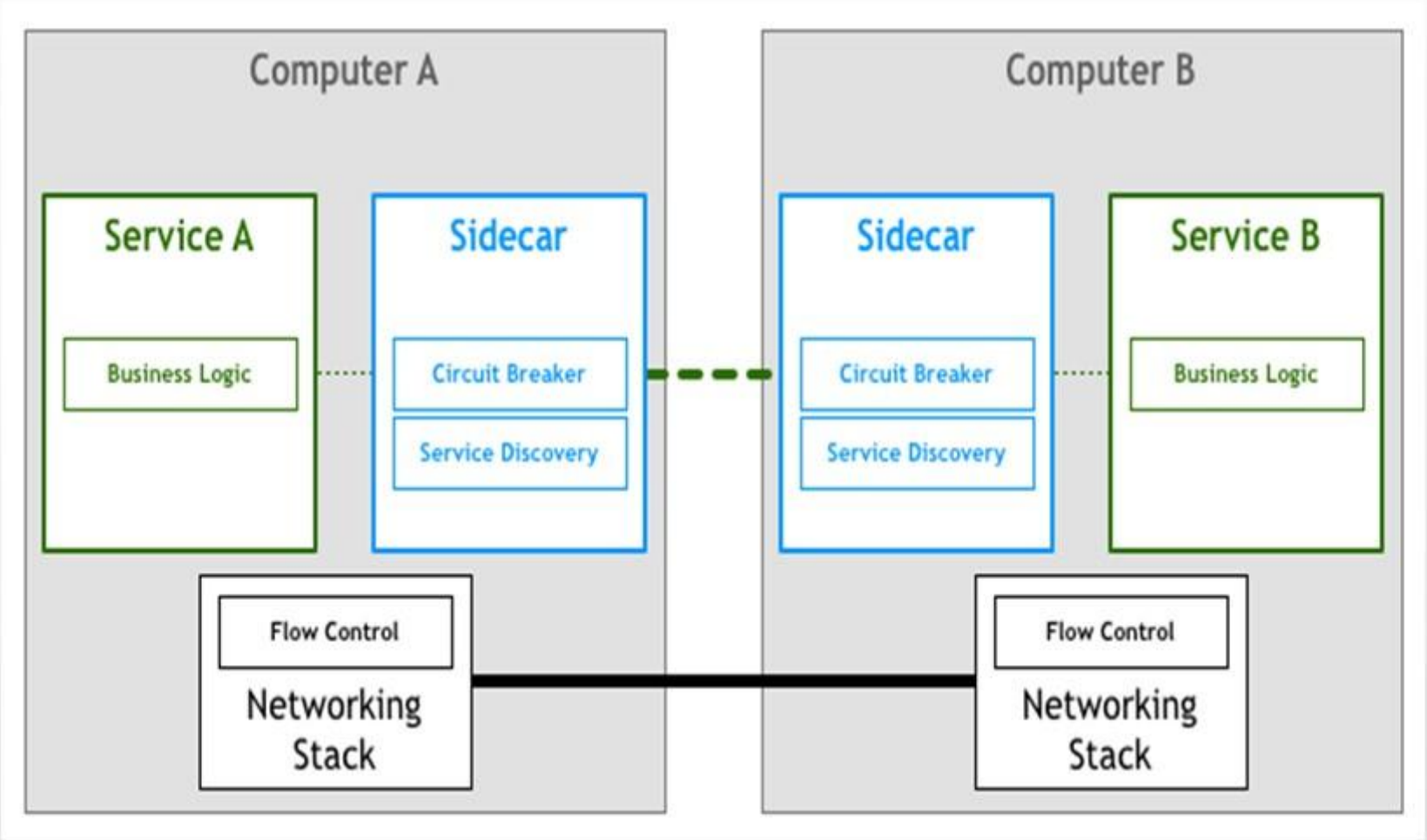
  2014年，Netflix发布了Prana

- **SoundCloud**

  据说也开发了一些sidecar

- **唯品会**

  2015年，唯品会的OSP服务化框架，加入名为local proxy的sidecar



Service Mesh的由来5：通用型的Service Mesh出现

Linkerd

- 来自Buoyant，Scala语言
- Service Mesh名词的创造者
- 2016年1月15日，0.0.7发布
- 2017年1月23日，加入CNCF
- 2017年4月25日，1.0版本发布

Envoy

- 来自Lyft，c++语言
- 2016年9月13日，1.0版本发布
- 2017年9月14日，加入CNCF

Service Mesh的由来6：Istio王者风范

- **Istio**
  - 来自Google，IBM和Lyft，Go语言
  - Service Mesh集大成者
  - 风头正劲的新一代Service Mesh
    - 2017年5月24日，0.1 release版本发布
    - 2017年10月4日，0.2 release版本发布
    - 2018年7月31日，1.0 release版本发布

Thank you!

Since Dec 7th, 2018

@rh01

http://www.shenhengheng.xyz

openstack™