科技报告

报告名称:	陈家镇国际生态社区能源管理中心建设关键技术
支持渠道:	科技部等中央单位与上海市共同推进重大任务科研专项
报告类型:	最终报告
编制单位:	上海陈家镇建设发展有限公司
编制时间:	2020 年 11 月 9 日

摘 要

请在这里输入摘要内容.

版 权 声 明

目 录

1 基于 Istio 的协议实现

传统的应用程序将所有功能都打包成一个独立的单元，因此也被称为单体应用。单体应用架构简单，在开发、测试和部署等方面都比较方便。但随着软件应用的发展，单体应用的弊端开始显现：不够灵活，对应用程序做任何细微的修改都需要将整个应用程序重新构建、重新部署，妨碍软件应用的持续交付；技术栈受限，开发团队的所有成员通常都必须使用相同的开发语言。因此，微服务架构应运而生。微服务架构将大型的单体应用拆分成若干个更小的服务，使得每个服务都可以独立地进行部署、升级、扩展和替换，服务间使用轻量级的通信协议进行通信 (例如，同步的 REST，异步的 AMQP 等)，降低服务间耦合度的同时满足了软件程序对于快速持续集成和持续交付的需求。

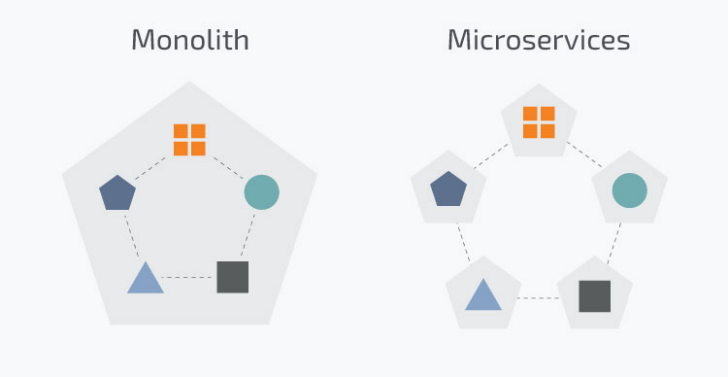

图 1: 单体应用和微服务应用

与单体应用相比，微服务架构将系统应用的复杂度从单体应用内部的测试、部署、维护等转变到了微服务的连接调用、管理部署和监控等方面，因此微服务架构会极大地增加运维工作量，开发人员需要投入更多的精力来保证远程调用的可靠性与数据一致性。为了简化开发，开发人员通过典型的类库和框架 (如 Netflix OSS 套件、Spring Cloud 框架)，编写较少的代码和注解就可以完成微服务间的服务发现、负载均衡、熔断、重试等功能。但此种办法缺点在于，开发人员需要掌握并熟练使用的内容较多，而且服务治理的功能不够齐全，很多功能需要自己进行拓展，编程语言也有所受限。这样所开发出来的微服务中，关键的业务逻辑代码和其它的用于管理服务间关系的非功能性代码混杂在一起。此时，若考虑为每一个微服务实例部署一个 Sidecar，将所有前述的非功能性代码移到 Sidecar 中，由 Sidecar 来负责提供服务发现等辅助功能，而开发人员只需专注于微服务的业务逻辑，从而有效地进行了解耦。当为大型的微服务系统部署 Sidecar 时，微服务之间的服务调用关系便形成了服务网格 (Service Mesh)。

当前最主流的 Service Mesh 方案，是 Google 和 IBM 两个公司联合 Lyft 合作的开源项目 Istio。在 Istio 的架构中，直接在 Lyft 开发的 Envoy 之上进行了拓展，然后将其作为 Sidecar 代理进行部署。Envoy Proxy 作为数据面，除了提供服务发现、负载均衡、限流熔断等功能，还可以协调微服务的所有出入站流量，收集相关的性能指标，与控制面进行交互。而有了数据面的支持，控制面一方面通过 Pilot 组件下发配置信息到相应的 Envoy Proxy 中，负责流量管理，另一方面通过 Mixer 组件收集遥测数据，从而实现了对整个微服务系统多方面的掌管与监控。

基于 Istio，部署前述的示例系统，具体架构变为4。用户将访问请求发往 Ingress Gateway，由 Ingress Gateway 依据相应的路由规则，再将请求转发至具体的后端服务中，然后收

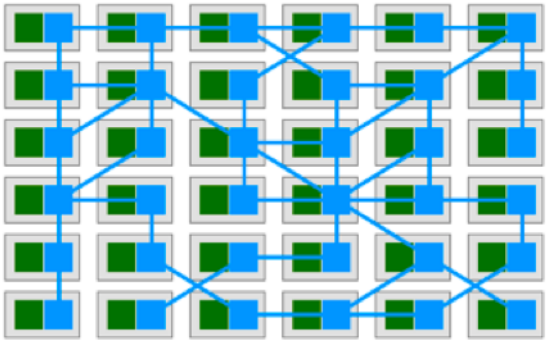

图 2: 服务网格 (Service Mesh)

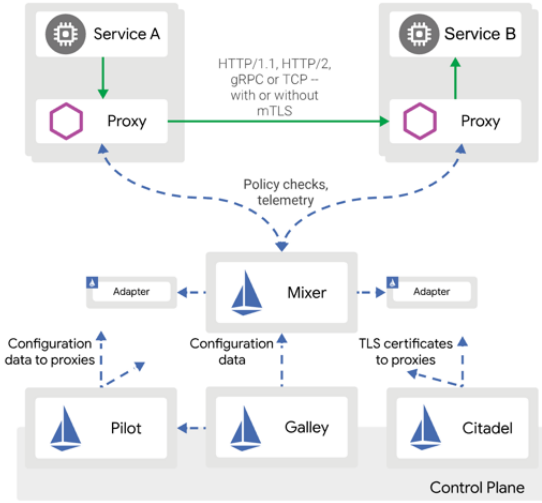

图 3: Istio 架构图（v1.4）

集运行结果并进行返回。每一次用户的请求都将产生一个分布式事务，其中可能会涉及多个服务的远程调用，但可以使用请求头中的 x-b3-traceid 字段进行唯一标识。Ingress Gaeway 一方面将对应的服务接口暴露出来供用户进行访问，另一方面为用户屏蔽了后端服务的具体实现和路由转发等服务治理功能。

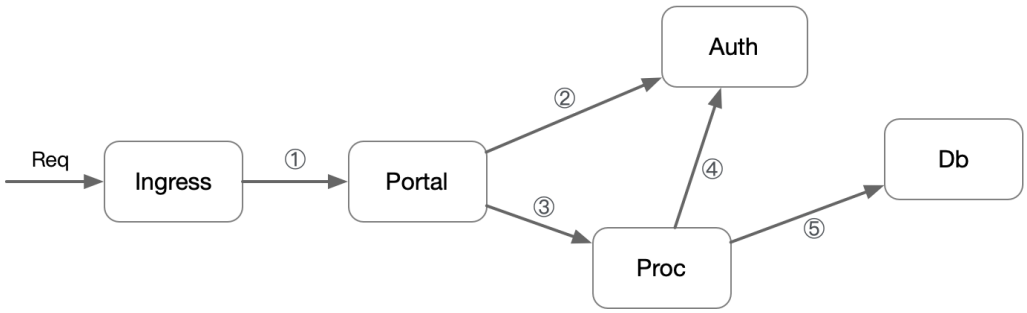

图 4: Istio 中的示例系统

以示例系统中的 auth-service 服务为例，具体的实现和更新流程为：

0. 插入 Envoy 容器和 TraceManager 容器：当每个服务实例部署时，我们将为每个服务实例对应部署 Envoy 容器和 TraceManager 容器。其中，Envoy 容器负责从 Istio 控制面中接收用户的配置信息，然后将配置信息转换成对应的规则，对所有的出入站

流量进行管理；与此同时，当存在具体的入站流量访问对应的服务时，Envoy 会将此次分布式调用的唯一标识 (x-b3-traceid) 和相关信息发送至 TraceManager 中。而 TraceManager 容器则负责接收从 Envoy 容器中发送过来的信息，对其进行记录管理，为后续的更新过程提供支持。插入的 Envoy 和 TraceManager 容器分别作为客户端和服务端，通过 Unix Domain Socket 的方式进行通信，从而最大程度地减少了消息传送对 Envoy 快速处理用户请求的影响。

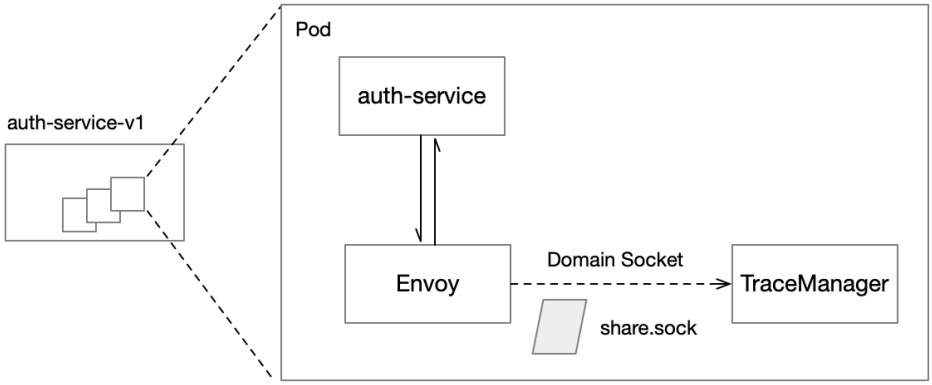

图 5: 插入容器

1. 创建初始的路由规则和版本信息：当整个系统部署完成时，为每个服务创建初始的路由规则和版本信息。图10中定义了路由规则，要求流量调用到 auth-service 时，所有流量都发往 v1 版本，且在返回的头信息中会带上自定义的信息，后续将利用此头信息进行流量的转发；图11中定义了版本信息，说明带有 version: v1 标签的所有 auth-service 的服务实例都属于 v1 版本；此时系统的运行状态对应显示为图8

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: auth-service
spec:
  hosts:
  - auth-service
  http:
  - route:
    - destination:
        host: auth-service
        subset: v1
      headers:
        response:
          add:
            x-version: auth-service-v1
```

所有流量发往v1版本，
返回头中添加自定义信息

图 6: v1 路由规则

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: auth-service
spec:
  host: auth-service
  subsets:
  - name: v1
    labels:
      version: v1
```

定义v1版本对应的标签

图 7: v1 版本规则

在系统的运行过程中，当存在入站流量调用到 auth-service 时，Envoy 容器执行拦截操作：一方面将流量转发往负责的服务实例，另一方面将此次调用所对应的 traceid 信息从头信息中提取出来并发送往 TraceManager 容器，TraceManager 容器则将对应的 traceid 存到对应的 TraceidSet 中，该集合代表了当前所有调用过 auth-service 的用户请求；同时，当本次用户请求处理完成并返回 Ingress Gateway 时，Ingress Gateway

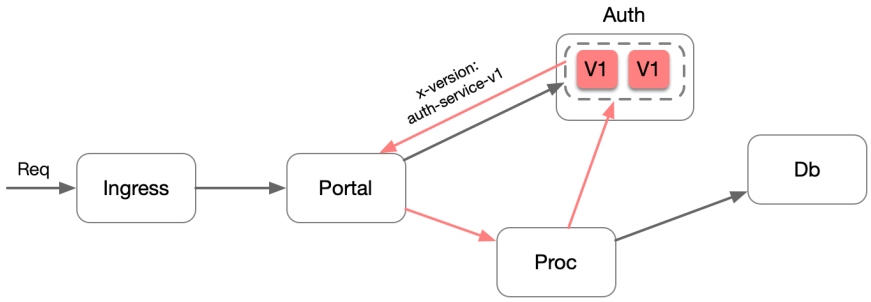

图 8: 流量全部发往 v1

实例内部会执行同样的添加操作，这里的 TraceidSet **则代表了当前已完成并返回的用户请求集合。**

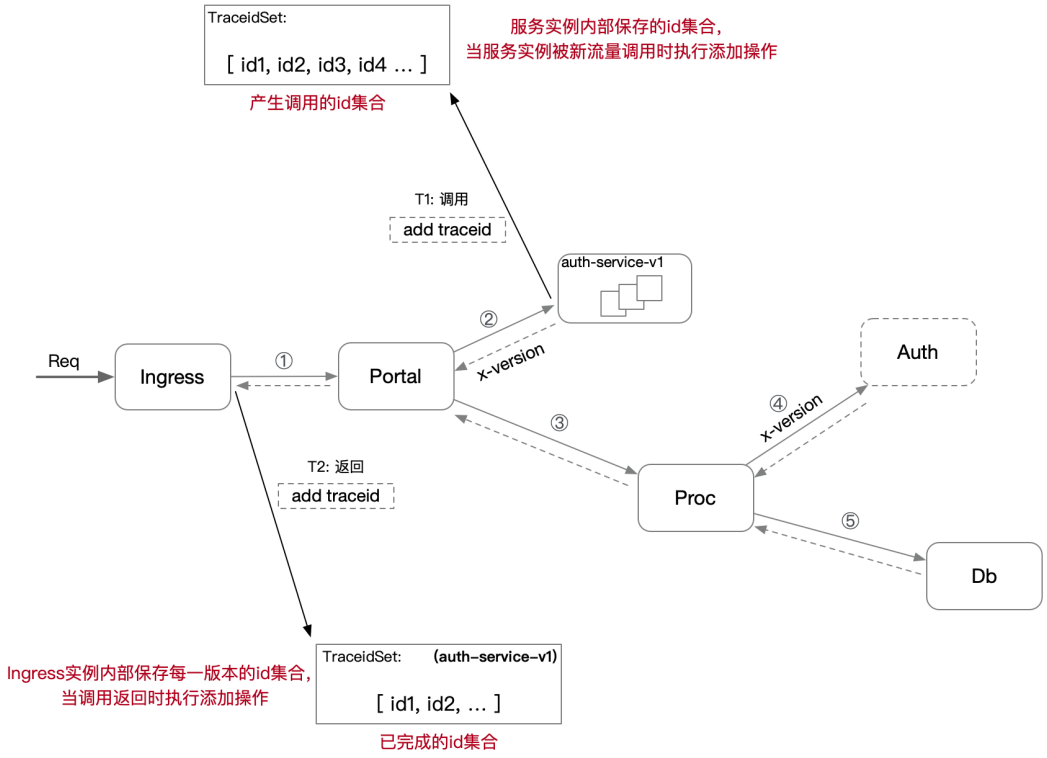

图 9: 事务记录

2. **新版本上线当我们需要对** auth-service **进行更新时，首先需要将新版本的服务进行上线部署。与此同时，相关的路由规则和版本信息修改为**:

3. **配置更新**

4. **旧版本撤销请求**

5.

«««< HEAD

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: auth-service
spec:
  hosts:
  - auth-service
  http:
  - route:
    - destination:
        host: auth-service
        subset: v1
      headers:
        response:
          add:
            x-version: auth-service-v1
```

所有流量发往v1版本，
返回头中添加自定义信息

图 10: v1 路由规则

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: auth-service
spec:
  host: auth-service
  subsets:
  - name: v1
    labels:
      version: v1
```

定义v1版本对应的标签

图 11: v1 版本规则

2 人机物融合平台架构与技术集成

互联网的快速发展催生新技术与新业务的出现。软件从单纯的信息处理环节逐步成为了应用价值观的主要载体，并形成将人、计算机以及物理世界相连接的人机物融合软件新形态。人机物融合平台旨在人机物三元融合开放多变的环境下，灵活地协同管理各种自治的资源，支撑人机物融合应用系统的运行。

2.1 总体的设计和结构

人机物融合平台的运行环境基于当前最主流的容器化编排系统 Kubernetes，在其上制定了一套人机物资源的描述规范，设计了一种消息驱动的人机物资源协作模型，构建了一套面向人机物融合的资源服务治理框架。

2.2 人机物资源的描述与编排

实际的物理资源都要抽象成一种数据结构存储在系统中，系统同时要更新和维护它们。本软件平台通过 Kubernetes CRD 在系统中表示抽象资源，存储在 etcd 中，Kubernetes Controller 负责维护它们，同步状态。

2.2.1 人机物资源描述规范

CRD 是 CustomResourceDefinition 的简写，资源是 kubernetes 中的抽象概念，CRD 是自定义的资源。

所有物理资源都要在系统中注册，且在系统中表达为 Resource CRD，需要使用一些参数描述资源。这些参数告诉了 Kubernetes 平台如何去访问资源并维护资源的状态。首先，资源需要一个唯一的标识。其次，因为需要使用探针去采集资源的各项信息，需要包含探针程序的镜像地址和启动命令以及启动参数。

2.2.2 人机物资源的运行时编排（WHG）

Resource CRD 由 Resource Controller 管理，控制器监控资源的状态，当监测资源的增删改事件时就会生成一个包含相关资源名称的 request 塞入工作队列，Reconcile 方法每隔很短的一段时间被调用一次，从工作队列中取出一个 request 作为参数进行处理。

Reconcile 首先根据 request 中包裹的资源名称和命名空间找到 Resource 的实例，如果是删除事件什么都不需要做，增改事件则会根据 ProbeEnabled 字段判断是否需要新建、更新、删除探针 Deployment 资源。

2.3 消息驱动的人机物资源协作模型

2.3.1 资源状态的同步

2.3.2 资源代理服务（TC）

2.3.3 消息队列（TC）

2.4 面向人机物融合的资源服务治理框架

2.4.1 人机物融合场景下的服务路由（TC）

2.4.2 资源服务的动态更新（WDY）

=======

3 动态更新系统实现

3.1 基本设计概念和结构

3.2 代码管理器

3.2.1 模块描述

3.2.2 功能

3.3 构建管理器

3.4 更新控制器

»»»> 95a231c4f30ca839e205329625997b652a539503