Istio 服务网格应用实战

前言：

课程名称：Istio 服务网格应用实战

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：6GB 开启虚拟化

参考文献：

官网手册：https://istio.io/latest/zh/docs/

Github 地址：https://github.com/istio/istio/releases

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、快速入门 Istio

1.1、Istio 是什么？

Istio 是运行于分布式应用程序之上的非侵入式（无代码入侵）服务网格系统，它的主要目的是为了更好更轻松的解决服务治理问题（Istio 是一套非侵入式一站式服务治理解决方案）。

Istio 的实现原理是，为每个微服务部署一个 Sidecar，代理微服务之间的所有网络通信。

Istio 是一个与 Kubernetes 紧密结合的适用于云原生场景的 Service Mesh 形态的用于服务治理的开放平台。

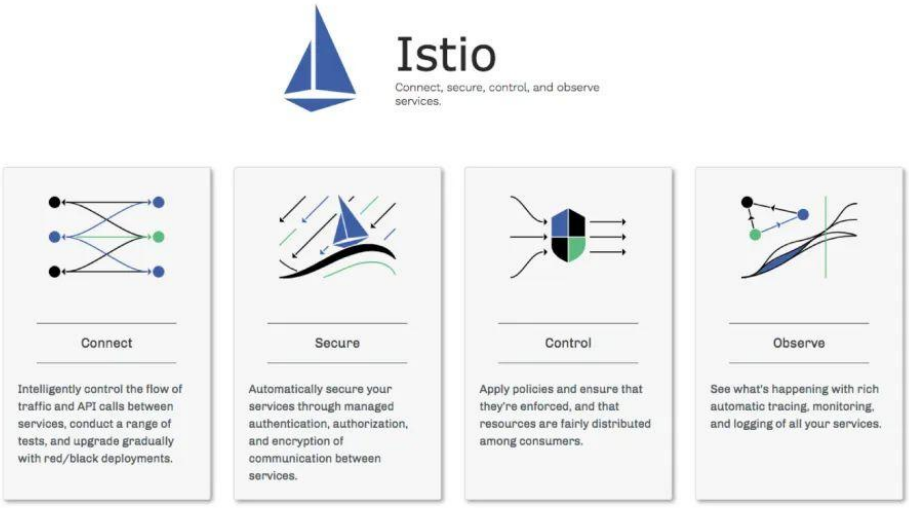
这里的关键字“治理”不限于“微服务治理”的范畴，任何服务只要服务间有访问，如果需要对服务间的访问做管理，就可以使用 Istio。根据 Istio 官方的介绍服务治理涉及到“连接”、“安全”、“控制”、“观察”，中间的四个动词就是 Istio 的主要功能。

官方浓缩的一句话：

An open platform to connect, secure, control and observe services.

翻译过来，就是“连接、安全加固、控制和观察服务的开放平台”。开放平台就是指它本身是开源的，服务对应的是微服务，也可以粗略地理解为单个应用。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



- 连接（Connect）：智能控制服务之间的调用流量，能够实现灰度升级、AB 测试和蓝绿部署等功能。
- 安全加固（Secure）：自动为服务之间的调用提供认证、授权和加密。
- 控制（Control）：应用用户定义的 policy，保证资源在消费者中公平分配。
- 观察（Observe）：查看服务运行期间的各种数据，比如日志、监控和 tracing（跟踪），了解服务的运行情况。

1.2、Istio 能做什么？

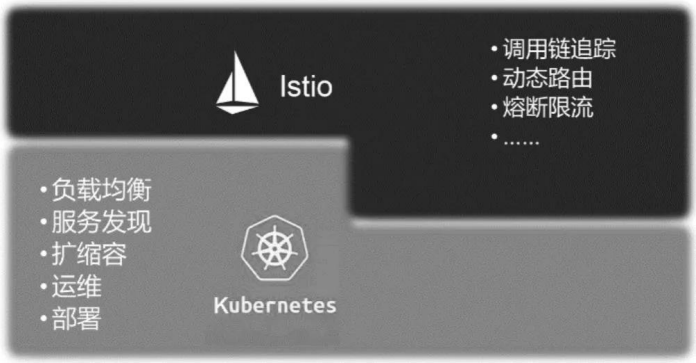
通过负载均衡、服务间的身份验证、监控等方法，Istio 可以轻松地创建一个已经部署了服务的网络，而服务的代码只需很少更改甚至无需更改。通过在整个环境中部署一个特殊的 sidecar 代理为服务添加 Istio 的支持，而代理会拦截微服务之间的所有网络通信，然后使用其控制平面的功能来配置和管理 Istio。Istio 的主要功能如下：

（1）服务发现（discovery）。
（2）负载均衡（load balancing）。
（3）故障恢复（failure recovery）。
（4）服务度量（metrics）。
（5）服务监控（monitoring）。
（6）A/B 测试（A/B testing）。
（7）灰度发布（canary rollouts）。
（8）限流限速（rate limiting）。
（9）访问控制（access control）。
（10）身份认证（end-to-end authentication）。

1.2.1、Istio 与 Kubernetes

从场景来看，Kubernetes 已经提供了非常强大的应用负载的部署、升级、扩容等运行管理能力。Kubernetes 中的 Service 机制也已经可以做服务注册、服务发现和负载均衡，支持通过服务名访问到服务实例。

从微服务的工具集观点来看，Kubernetes 本身是支持微服务的架构，在 Pod 中部署微服务很合适，也已经解决了微服务的互访互通问题，但对服务间访问的管理如服务的熔断、限流、动态路由、调用链追踪等都不在 Kubernetes 的能力范围内。那么，如何提供一套从底层的负载部署运行到上层的服务访问治理端到端的解决方案？目前，最完美的答案就是在 Kubernetes 上叠加 Istio。



Istio 最大化地利用了 Kubernetes 这个基础设施，与之叠加在一起形成了一个更强大的用于进行服务运行和治理的基础设施，并提供了更透明的用户体验。

1.2.2、什么是服务熔断？

服务熔断是一种在微服务架构中常见的容错技术。当某个服务的调用链路中的某个服务出现故障或者响应时间过长时，服务熔断机制会立即介入，直接返回一个错误响应，从而阻止对该服务的继续调用，以保护整个系统的稳定性和可用性。这样可以防止故障的扩散和级联效应，将影响范围限制在最小。

在 Istio 这样的服务网格中，服务熔断可以通过其内置的熔断器功能来实现。Istio 的熔断器可以根据服务的响应时间、错误率等指标来判断是否需要触发熔断。

举一个例子来说明服务熔断：

假设我们有一个在线购物系统，其中包含了用户服务、商品服务、订单服务等多个微服务。当用户想要购买商品时，系统会依次调用用户服务、商品服务和订单服务。如果商品服务由于某种原因（比如数据库连接失败、服务过载等）出现了故障，导致响应时间特别长，那么此时服务熔断机制就会被触发。 Istio 的熔断器可以检测到商品服务的响应时间超过了预设的阈值，于是会立即熔断该服务，直接返回一个错误响应给用户。这样，用户服务就不会继续等待商品服务的响应，而是立即返回给用户一个友好的错误信息，比如“商品服务暂时不可用，请稍后再试”。同时，Istio 还可以将熔断事件记录下来，供开发人员进行后续的故障排查和优化。

通过服务熔断机制，我们可以有效地防止故障的扩散，保证系统的整体稳定性和可用性。当然，在实际应用中，还需要结合其他容错技术（如超时控制、限流等）来综合提升系统的健壮性。

1.2.3、什么是服务降级？

服务降级是在分布式系统中应对故障或性能下降时采取的一种策略。当某个服务出现故障或响应性能下降，无法满足正常的业务需求时，服务降级会主动降低该服务的某些功能或质量，以保证整体系统的稳定性和可用性。通过服务降级，我们可以将有限的资源用于保证核心业务的正常运行，同时避免非关键业务对系统造成过大的压力。

在 Istio 这样的服务网格中，服务降级可以通过配置路由规则来实现。Istio 允许你根据服务的性能指标（如响应时间、错误率等）或自定义条件来动态调整路由规则，将流量引导到降级后的服务版本或备选服务上。

举一个例子来说明服务降级：

假设我们有一个在线支付系统，其中包含了用户服务、支付服务、订单服务等多个微服务。当用户发起支付请求时，系统会依次调用用户服务、支付服务和订单服务。如果支付服务由于某种原因（比如系统过载、第三方支付接口故障等）出现了性能下降，导致响应时间延长或错误率上升，那么此时可以考虑对支付服务进行降级处理。

Istio 可以配置路由规则，当检测到支付服务的性能指标超过预设的阈值时，将部分或全部流量引导到一个降级后的支付服务版本上。这个降级后的版本可能简化了某些功能（比如只支持固定金额支付），或者使用了备选的第三方支付接口。通过这样的降级处理，我们可以保证支付系统的核心功能仍然可用，同时减轻支付服务的压力，避免整个系统因为单个服务的故障而瘫痪。

服务降级是一种主动的策略，它允许我们在系统出现故障或性能下降时，通过调整服务的行为和质量来保持整体系统的稳定性和可用性。在 Istio 这样的服务网格中，通过灵活的路由规则和性能监控，我们可以轻松实现服务降级策略，提升系统的容错能力和韧性。

1.2.4、什么是服务超时？

在 Istio 中，服务超时是一种机制，用于控制服务之间调用的响应时间。当一个服务调用另一个服务时，如果超过了预设的超时时间，Istio 将会中断该调用，并返回一个错误响应。服务超时可以帮助防止因长时间等待无响应的服务而导致的系统阻塞和性能下降。

下面举一个关于服务超时的例子：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

假设我们有一个在线书店应用，其中包含了用户服务、书籍服务、订单服务等微服务。当用户想要购买一本书籍时，用户服务会调用书籍服务来获取书籍的详细信息，然后再调用订单服务来创建订单。

在这个场景中，我们可以为服务之间的调用设置超时时间。例如，我们可以设置用户服务调用书籍服务的超时时间为 2 秒，调用订单服务的超时时间为 3 秒。这意味着如果书籍服务在 2 秒内没有返回响应，或者订单服务在 3 秒内没有完成创建订单的操作，Istio 将会中断这些调用，并返回一个错误响应给用户。

服务超时的好处是可以避免系统陷入长时间的等待状态，提高系统的响应速度和用户体验。同时，通过设置合理的超时时间，我们还可以对服务的质量进行监控和评估，及时发现并处理性能问题或故障。

需要注意的是，服务超时时间的设置应该根据实际业务需求和系统性能来进行权衡。如果超时时间设置得过短，可能会导致不必要的失败和用户体验下降；而如果超时时间设置得过长，则可能会导致系统阻塞和性能下降。因此，在实际应用中，我们需要根据具体情况来合理设置服务超时时间。

1.2.5、什么是服务重试？

在 Istio 中，服务重试是一种容错机制，当服务之间的调用失败时，Istio 可以自动重试该调用，以提高服务的可用性和健壮性。服务重试可以减少因瞬时故障或网络波动导致的调用失败，提高系统的容错能力。

下面举一个关于服务重试的例子：

假设我们有一个在线零售应用，其中包含了用户服务、商品服务、库存服务等微服务。当用户尝试购买一个商品时，用户服务会先调用商品服务获取商品信息，然后再调用库存服务检查库存量。

在这个场景中，我们可以配置 Istio 的服务重试机制。例如，我们可以设置用户服务调用库存服务时，如果初次调用失败（可能是因为库存服务暂时不可用或网络波动），Istio 会自动重试该调用一次或多次。这样，即使库存服务在第一次调用时出现故障，用户服务仍然有机会通过重试获取到正确的库存信息，从而完成购买操作。

服务重试的好处是可以自动处理一些瞬时故障或网络波动，提高系统的可用性和用户体验。然而，需要注意的是，服务重试并不是万能的，对于一些持续性的故障或错误，重试可能只是浪费时间和资源。因此，在配置服务重试时，我们需要根据实际业务需求和系统性能来进行权衡，设置合理的重试次数和重试间隔。

此外，Istio 还支持基于条件的重试策略，例如只有在满足特定条件（如错误码、响应时间等）时才进行重试。这样可以更加灵活地控制重试行为，提高系统的容错能力和性能。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1.2.6、什么是动态路由？

在 Istio 中，动态路由是一种功能强大的机制，允许根据实时条件动态地改变服务之间的流量路由。通过动态路由，可以实现诸如 A/B 测试、灰度发布、故障转移等场景，提高系统的灵活性和可控性。

下面举一个关于动态路由的例子：

假设我们有一个在线书店应用，正在准备进行一次新功能的 A/B 测试。新功能涉及到用户界面的改进，我们想要将一部分用户流量引导到新版本的用户界面上，以收集用户反馈和性能数据。

在这个场景中，我们可以使用 Istio 的动态路由功能来实现：

首先，我们部署两个版本的用户服务：旧版本和新版本。旧版本的用户服务继续提供原有的功能，而新版本的用户服务则包含了改进后的用户界面。

接着，我们使用 Istio 的 Virtual Service 资源来定义路由规则。我们可以创建一个 Virtual Service，其中包含一个或多个路由规则，用于根据条件将流量路由到不同的用户服务版本上。

例如，我们可以设置一个路由规则，将 50%的用户流量路由到新版本的用户服务上，而将另外 50%的用户流量路由到旧版本的用户服务上。这样，我们就可以同时收集两个版本的用户反馈和性能数据，以便进行后续的优化和决策。

动态路由的好处是可以根据实际需求实时调整流量路由，实现灵活的 A/B 测试、灰度发布等场景。同时，通过动态路由还可以实现故障转移和容灾备份，提高系统的可靠性和稳定性。

需要注意的是，动态路由的配置和管理需要谨慎进行，以避免对系统造成不必要的影响。在实际应用中，我们需要根据具体需求和场景来设计和实施动态路由策略，确保系统的稳定性和性能。

1.4、Istio 架构说明

Istio 是一个开源的服务网格平台，专为微服务架构而设计，提供了一系列功能，如流量管理、服务安全、服务发现和可观察性等。Istio 的架构主要由两部分组成：数据平面（Data Plane）和控制平面（Control Plane）。

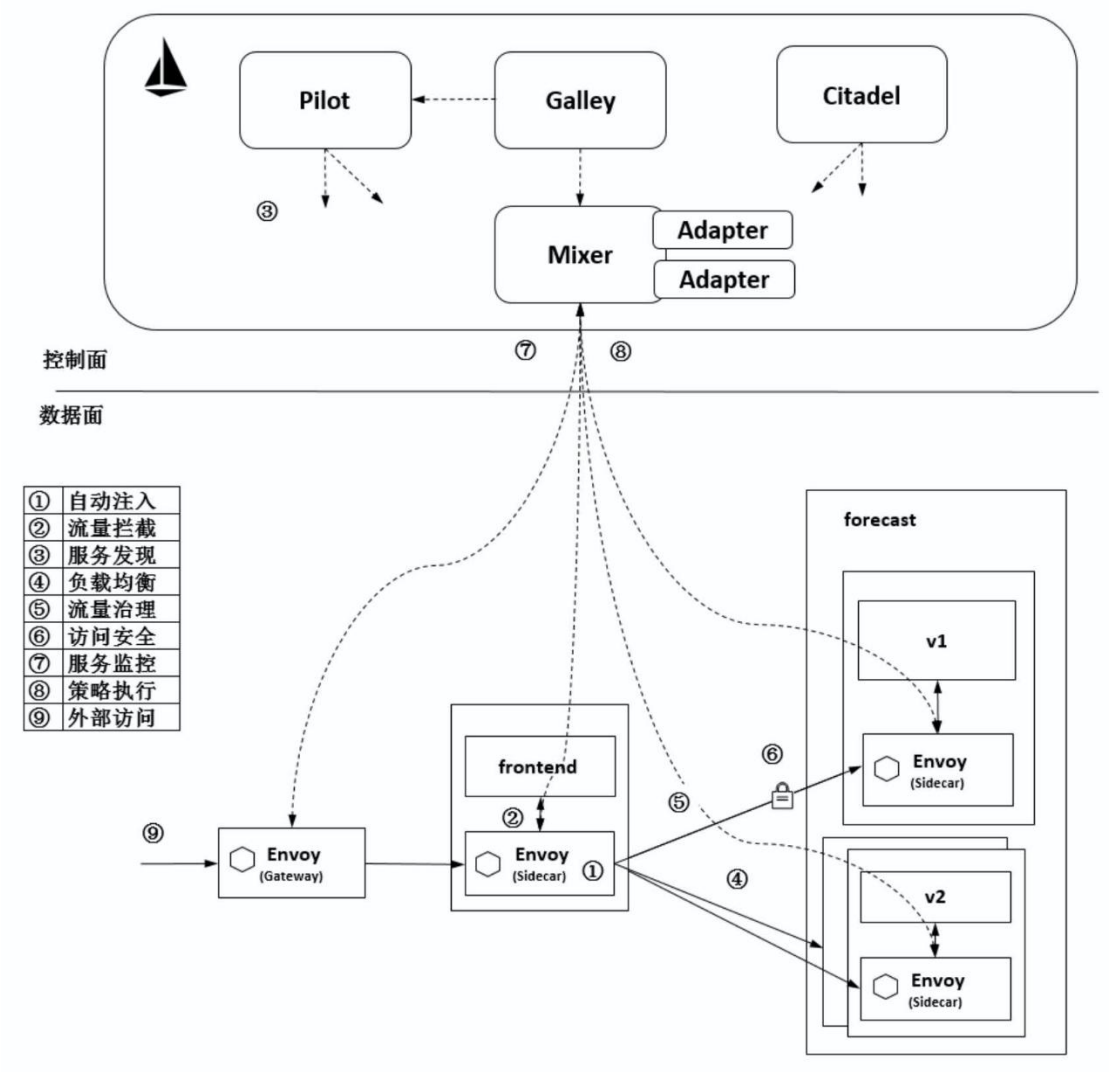
数据平面：由网格内的所有智能代理（Envoy）组成，这些代理被部署为 Sidecar，与应用程序一起运行。每个 Envoy 代理都会接管进入和离开服务的流量，并负责协调和控制微服务之间的所有网络通信。数据平面还负责收集和报告所有网格流量的遥测数据。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

控制平面：负责管理和配置数据平面中的 Envoy 代理。它负责接收用户配置，生成路由规则，并将这些规则分发到各个代理。此外，控制平面还负责服务发现、策略实施和遥测数据收集等功能。

在 Istio 中，Envoy 代理是唯一与数据平面流量交互的组件。Envoy 是一个高性能的代理，由 C++开发，具有动态服务发现、负载均衡、TLS 终端、HTTP/2 与 gRPC 代理、熔断器、健康检查以及基于百分比流量分割的分阶段发布等特性。

结合下图我们来理解 Istio 的各组件的功能及工作方式。



控制平面组件：

Pilot：Envoy sidecar 的服务发现组件，同时为智能路由（如 A/B 测试、金丝雀部署等）和弹性（如超时、重试、熔断器等）提供流量管理功能。

Citadel：提供访问控制和用户身份认证功能。

Galley 是 Istio 控制平面的一个组件，它并不直接处理数据平面的流量，而是作为其他控制平面组件的协调者。Galley 的主要职责是处理 Istio 的配置信息，包括验证、合并和分发配置更新。它确保控制平面的各个组件之间的配置

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

信息保持一致，并且符合 Istio 的规范和策略。通过 Galley 的协调，控制平面能够更高效地管理和配置数据平面的 Envoy 代理。

Mixer：负责执行访问控制和策略管理功能。Mixer 可以接收来自 Envoy 代理的流量统计数据，并执行限流、限额等策略。此外，Mixer 还负责收集代理观察到的服务之间的流量统计数据，即遥测数据（如监控、APM 等）。这些数据可以用于监控和调试服务网格中的流量行为，帮助运维人员了解系统的性能和健康状况。

下面说明每个步骤做的事情：

1、自动注入：在创建应用程序时自动注入 Sidecar 代理 Envoy 程序。在 Kubernetes 中创建 Pod 时，Kube-apiserver 调用控制面组件的 Sidecar-Injector 服务，自动修改应用程序的描述信息并注入 Sidecar。在真正创建 Pod 时，在创建业务容器的 Pod 中同时创建 Sidecar 容器。

2、流量拦截：在 Pod 初始化时设置 iptables 规则，基于配置的 iptables 规则拦截业务容器的 Inbound 流量和 Outbound 流量到 Sidecar 上。而应用程序感知不到 Sidecar 的存在，还以原本的方式进行互相访问。上图中，流出 frontend 服务的流量会被 frontend 服务侧的 Envoy 拦截，而当流量到达 forecast 容器时，Inbound 流量被 forecast 服务侧的 Envoy 拦截。

3、服务发现：服务发起方的 Envoy 调用控制面组件 Pilot 的服务发现接口获取目标服务的实例列表。上图中，frontend 服务侧的 Envoy 通过 Pilot 的服务发现接口得到 forecast 服务各个实例的地址。

4、负载均衡：服务发起方的 Envoy 根据配置的负载均衡策略选择服务实例，并连接对应的实例地址。上图中，数据面的各个 Envoy 从 Pilot 中获取 forecast 服务的负载均衡配置，并执行负载均衡动作。

5、流量治理：Envoy 从 Pilot 中获取配置的流量规则，在拦截到 Inbound 流量和 Outbound 流量时执行治理逻辑。上图中，frontend 服务侧的 Envoy 从 Pilot 中获取流量治理规则，并根据该流量治理规则将不同特征的流量分发到 forecast 服务的 v1 或 v2 版本。

6、访问安全：在服务间访问时通过双方的 Envoy 进行双向认证和通道加密，并基于服务的身份进行授权管理。上图中，Pilot 下发安全相关配置，在 frontend 服务和 forecast 服务的 Envoy 上自动加载证书和密钥来实现双向认证，其中的证书和密钥由另一个管理面组件 Citadel 维护。

7、服务监测：在服务间通信时，通信双方的 Envoy 都会连接管理面组件 Mixer 上报访问数据，并通过 Mixer 将数据转发给对应的监控后端。上图中，frontend 服务对 forecast 服务的访问监控指标、日志和调用链都可以通过这种方式收集到对应的监控后端。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

8、策略执行：在进行服务访问时，通过 Mixer 连接后端服务来控制服务间的访问，判断对访问是放行还是拒绝。上图中，Mixer 后端可以对接一个限流服务对从 frontend 服务到 forecast 服务的访问进行速率控制等操作。

9、外部访问：在网格的入口处有一个 Envoy 扮演入口网关的角色。上图中，外部服务通过 Gateway 访问入口服务 frontend，对 frontend 服务的负载均衡和一些流量治理策略都在这个 Gateway 上执行。

2、基于 Kubernetes 部署 Istio

Istio 安装有几种方式：

1、Helm 安装

2、Istioctl 安装

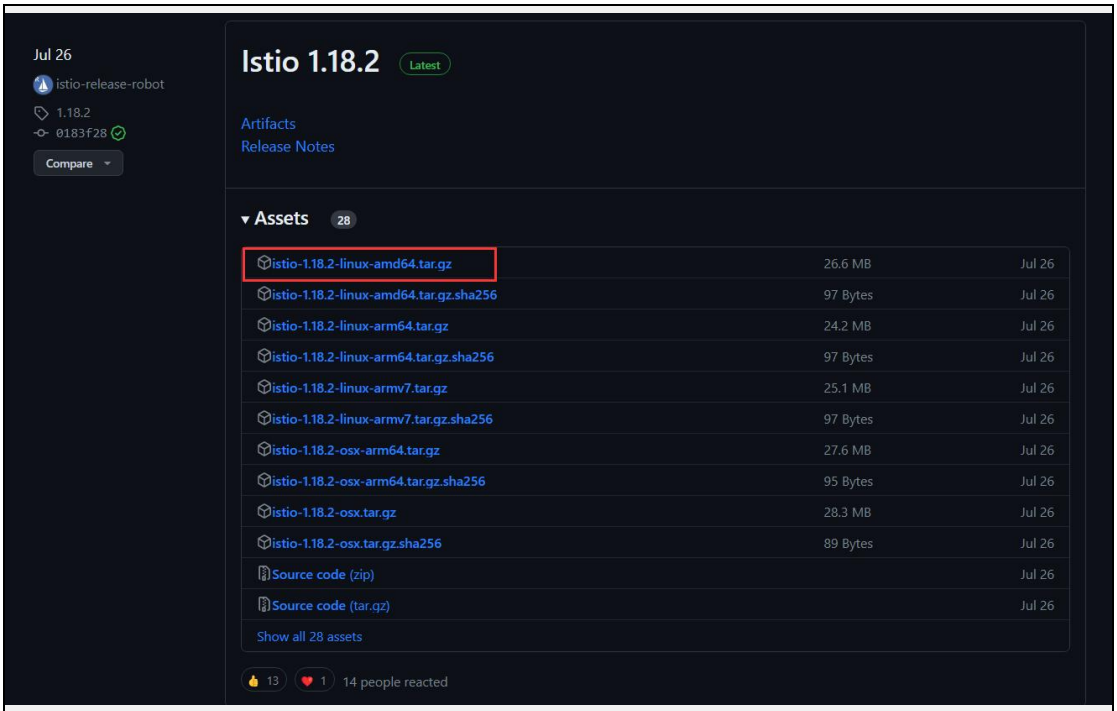
3、Istio Operator 安装

Istioctl 高安全性的简单、合格的安装和管理方法。这是社区推荐的安装方法。这里我们就以这种方式进行演示。

- 使用 Istioctl 安装

（1）下载 Istio

下载地址：https://github.com/istio/istio/releases



1、将下载好的软件包上传到服务器、解压

```
[root@k8s-master01 ~]# ll istio-1.18.2-linux-amd64.tar.gz
-rw-r--r-- 1 root root 27936557 Sep  5 06:06 istio-1.18.2-linux-amd64.tar.gz
[root@k8s-master01 ~]# tar xf istio-1.18.2-linux-amd64.tar.gz
```

2、查看目录结构

```
[root@k8s-master01 ~]# cd istio-1.18.2/
[root@k8s-master01 istio-1.18.2]# ll
total 28
drwxr-x---  2 root root    22 Jul 21 19:40 bin
-rw-r--r--  1 root root 11348 Jul 21 19:40 LICENSE
drwxr-xr-x  5 root root    52 Jul 21 19:40 manifests
-rw-r-----  1 root root   900 Jul 21 19:40 manifest.yaml
-rw-r--r--  1 root root  6595 Jul 21 19:40 README.md
drwxr-xr-x 24 root root  4096 Jul 21 19:40 samples
drwxr-xr-x  3 root root    57 Jul 21 19:40 tools
```

提示：

samples 目录包含一些示例应用程序。

bin 目录包含 istioctl 客户端二进制文件。

（2）安装 istio

对于本次安装，使用 demo 配置组合。选择它是因为它包含了一组专为测试准备的功能集合，另外还有用于生产或性能测试的配置组合。

可选组合：

	default	demo	minimal	remote	empty	preview	ambient
核心组件							
istio-egressgateway		✓					
istio-ingressgateway	✓	✓				✓	
istiod	✓	✓	✓			✓	✓
CNI							✓
Ztunnel							✓

1、安装 istio

```
[root@k8s-master01 istio-1.18.2]# ./bin/istioctl install --set profile=demo -y
```

```
[root@k8s-master01 istio-1.18.2]# ./bin/istioctl install --set profile=demo -y
✔ Istio core installed
✔ Istiod installed
✔ Ingress gateways installed
✔ Egress gateways installed
✔ Installation complete
Making this installation the default for injection and validation.
[root@k8s-master01 istio-1.18.2]#
```

2、检查

```
[root@k8s-master01 istio-1.18.2]# kubectl get pods -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
istio-egressgateway-6cf4f6489d-7jxnn    1/1     Running   0          2m42s
istio-ingressgateway-7b84b459d6-8vlcg   1/1     Running   0          2m42s
istiod-8498fbd896-mc56c                 1/1     Running   0          3m56s
[root@k8s-master01 istio-1.18.2]#
```

（3）卸载 istio 集群，暂时先不执行，记住这个命令即可

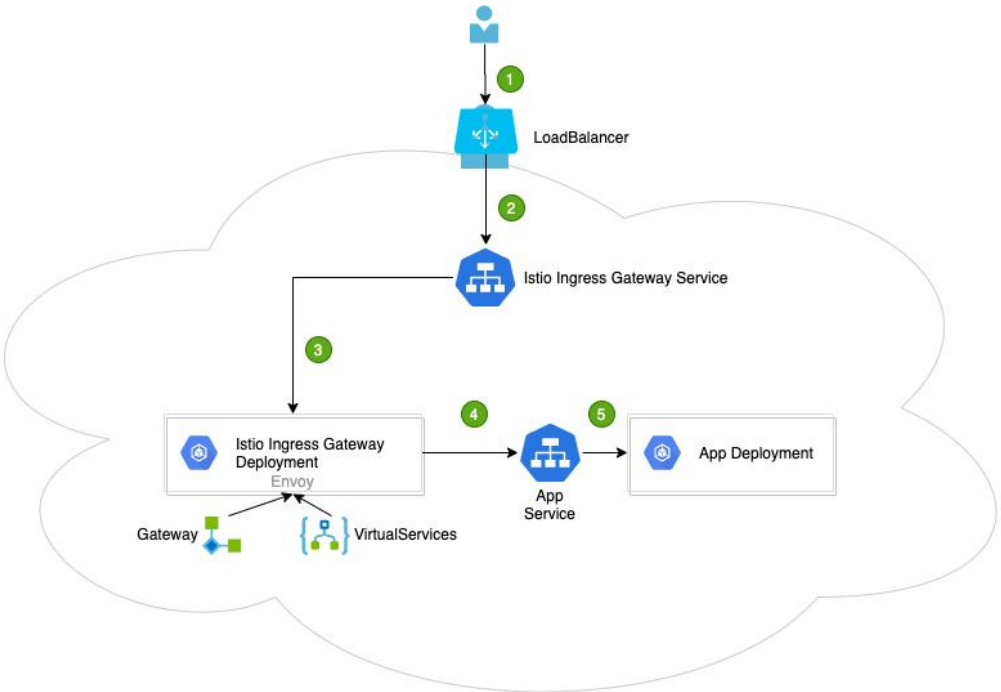
```
[root@k8s-master01 istio-1.18.2]# ./bin/istioctl uninstall --purge
[root@k8s-master01 istio-1.18.2]# kubectl delete ns istio-system
```

3、Istio 核心资源

说明：本小节不做操作，主要要了解 Istio 核心资源 Gateway、VirtualService、DestinationRule。

3.1、Gateway

在 Kubernetes 环境中，Ingress controller 用于管理进入集群的流量。在 Istio 服务网格中 Istio Ingress Gateway 承担相应的角色，它使用新的配置模型（Gateway 和 VirtualServices）完成流量管理的功能。通过下图做一个总的描述。



1、用户向某端口发出请求

2、负载均衡器监听端口，并将请求转发到集群中的某个节点上。Istio Ingress Gateway Service 会监听集群节点端口的请求。

3、Istio Ingress Gateway Service 将请求交给 Istio Ingress Gateway Pod 处理。IngressGateway Pod 通过 Gateway 和 VirtualService 配置规则处理请求。其中，Gateway 用来配置端口、协议和证书。VirtualService 用来配置一些路由信息（找到请求对应处理的服务 App Service）。

4、Istio Ingress Gateway Pod 将请求转给 App Service。

5、最终的请求会交给 App Service 关联的 App Deployment 处理。

示例：

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: myapp-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

说明：

网关是一个运行在网格边缘的负载均衡器，用于接收传入或传出的 HTTP/TCP 连接。主要工作是接受外部请求，把请求转发到内部服务。网格边缘的 Ingress 流量，会通过对应的 Istio IngressGateway Controller 进入到集群内部。

在上面这个 yaml 里我们配置了一个监听 80 端口的入口网关，它会将 80 端口的 http 流量导入到集群内对应的 Virtual Service 上。

注意：

```
hosts:
- "*"
```

*表示通配符，表示该服务器接受来自所有主机的请求。

3.2、VirtualService

VirtualService 是 Istio 流量治理的一个核心配置，可以说是 Istio 流量治理中最重要、最复杂的。VirtualService 在形式上表示一个虚拟服务，将满

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

足条件的流量都转发到对应的服务后端，这个服务后端可以是一个服务，也可以是在 DestinationRule 中定义的服务的子集。

示例：

```
apiVersion: networking.istio.io/v1beta3
kind: VirtualService
metadata:
  name: myapp
spec:
  hosts:
  - "*"
  gateways:
  - myapp-gateway
  http:
  - route:
    - destination:
        host: myapp.default.svc.cluster.local
        subset: v1
      weight: 90
    - destination:
        host: myapp.default.svc.cluster.local
        subset: v2
      weight: 10
```

说明：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 myapp 的虚拟服务（VirtualService）。虚拟服务用于定义网络流量规则，它指定了如何将流量路由到目标服务。

在 spec 部分，hosts 字段指定了该虚拟服务应用于哪些主机，这里使用*表示适用于所有主机。gateways 字段指定了该虚拟服务关联的网关，这里使用了一个名为 myapp-gateway 的网关。

然后，代码定义了 HTTP 路由规则。每条规则由一个或多个路由组成，每个路由指定一个目标服务。在这个例子中，两条规则将流量分别路由到 myapp 服务的 v1 和 v2 版本。

第一条规则将 90%的流量路由到 v1 版本。destination 字段指定了目标服务的地址和子集，这里使用了 Kubernetes 的服务 DNS 名称 myapp.default.svc.cluster.local 和子集名称 v1。weight 字段指定了该路由的权重，这里设置为 90，表示 90%的流量将通过该路由。

第二条规则将 10%的流量路由到 v2 版本，配置与第一条规则类似，只是子集名称改为 v2。

通过这样的配置，Istio 将根据规则将进入 myapp 服务的流量路由到相应的版本，实现服务版本的逐步迁移和灰度发布。

多路由规则：

```
apiVersion: networking.istio.io/v1alpha3
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
  kind: VirtualService
  metadata:
    name: bookinfo
  spec:
    hosts:
      - bookinfo.com
    http:
    - match:
      - uri:
          prefix: /reviews
      route:
      - destination:
          host: reviews
    - match:
      - uri:
          prefix: /ratings
      route:
      - destination:
          host: ratings
```

说明：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 bookinfo 的虚拟服务（VirtualService）。虚拟服务用于定义网络流量规则，它指定了如何将流量路由到目标服务。

在 spec 部分,hosts 字段指定了该虚拟服务应用于哪些主机,这里使用了 bookinfo.com。

然后，代码定义了 HTTP 路由规则。该虚拟服务中有两条规则，每条规则都根据 URI 的前缀来匹配流量，然后将流量路由到相应的目标服务。

第一条规则匹配以/reviews 开头的 URI，然后将流量路由到名为 reviews 的目标服务。

第二条规则匹配以/ratings 开头的 URI，然后将流量路由到名为 ratings 的目标服务。

通过这样的配置，Istio 可以根据 URI 的前缀将进入 bookinfo 服务的流量路由到相应的目标服务。

3.3、DestinationRule

destination rule 是 istio 流量路由功能的重要组成部分。一个 virtual service 可以看作是如何将流量分发给特定的目的地，然后调用 destination rule 来配置分发到该目的地的流量。destination rule 在 virtual service 的路由规则之后起作用（即在 virtual service 的 math -> route-destination 之后起作用，此时流量已经分发到真实的 service 上），应用于真实的目的地。

可以使用 destination rule 来指定命名的服务子集，例如根据版本对服务的实例进行分组，然后通过 virtual service 的路由规则中的服务子集将控制流量分发到不同服务的实例中。

示例：

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: myapp
spec:
  host: myapp.default.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

说明如下：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 myapp 的目标规则（DestinationRule）。目标规则用于定义目标服务的子集和标签。

在 spec 部分，host 字段指定了目标服务的地址，这里使用了 Kubernetes 的服务 DNS 名称 myapp.default.svc.cluster.local。

然后，代码定义了两个子集，分别命名为 v1 和 v2。每个子集都有一个标签，标签用于标识子集的版本。在这个例子中，v1 版本的标签为 version: v1，v2 版本的标签为 version: v2。

通过这样的配置，Istio 可以根据标签将流量路由到相应的子集。例如，如果一个请求的标签包含 version: v1，那么该请求将被路由到名为 v1 的子集。

4、Istio 快速入门（在线书店 bookinfo 项目部署）

4.1、在线书店功能介绍

在线书店：bookinfo

这个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：

1、productpage 这个微服务会调用 details 和 reviews 两个微服务，用来生成页面。

2、details 这个微服务中包含了书籍的信息。

3、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。

4、ratings 这个微服务中包含了由书籍评价组成的评级信息。

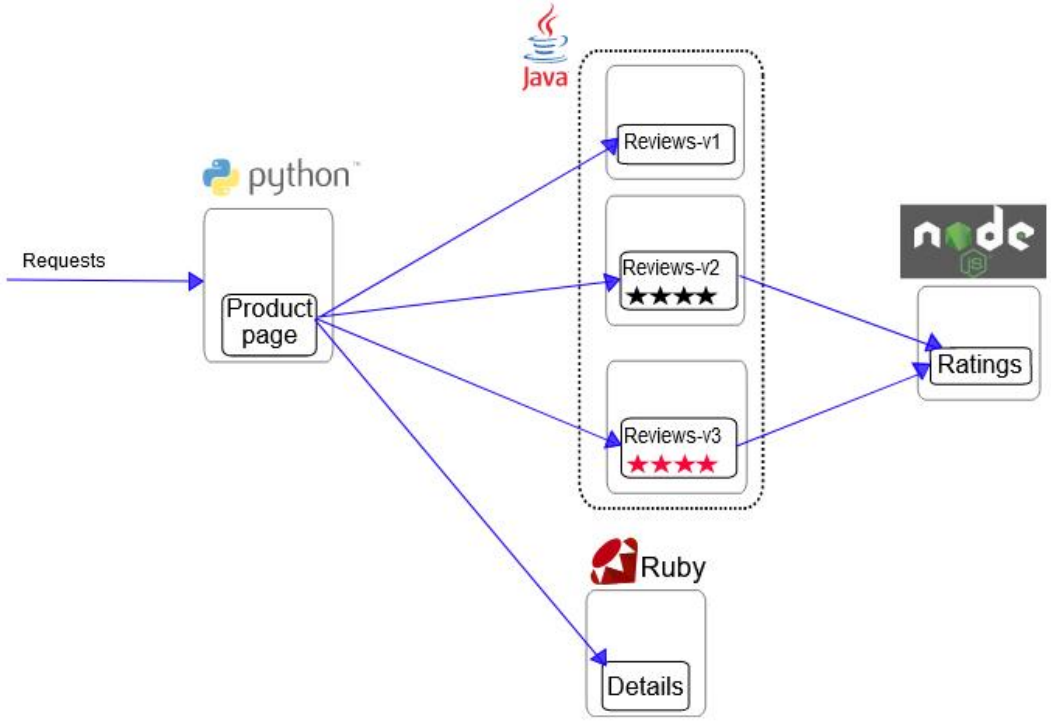
reviews 微服务有 3 个版本：

1、v1 版本不会调用 ratings 服务。

2、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。

3、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

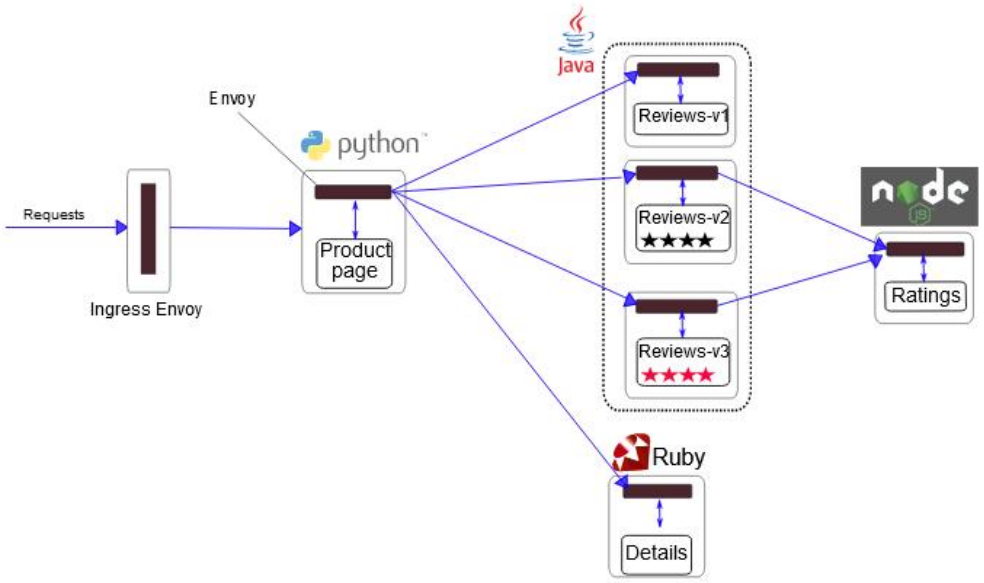


Bookinfo Application without Istio

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。

4.2、部署 Bookinfo 项目

要在 Istio 中运行这一应用，无需对应用自身做出任何改变。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：



Bookinfo Application

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 envoy 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。

（1）给命名空间添加标签，指示 Istio 在部署应用的时候，自动注入 Envoy 边车代理

```
[root@k8s-master01 ~]# kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

（2）部署 Bookinfo 示例应用

```
1、部署 Bookinfo 应用
[root@k8s-master01 ~]# cd istio-1.18.2/samples/bookinfo/platform/kube/
[root@k8s-master01 kube]# kubectl apply -f bookinfo.yaml
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
```

```
    deployment.apps/productpage-v1 created

2、查看 pod 状态
    [root@k8s-master01 ~]# kubectl get pods
    NAME                              READY   STATUS    RESTARTS        AGE
    details-v1-698b5d8c98-kdmj7       2/2     Running   2 (2m20s ago)   22m
    productpage-v1-bf4b489d8-r4689    2/2     Running   2 (2m20s ago)   22m
    ratings-v1-5967f59c58-wbnhl       2/2     Running   2 (2m20s ago)   22m
    reviews-v1-9c6bb6658-jfhsc        2/2     Running   2 (2m20s ago)   22m
    reviews-v2-8454bb78d8-jkgtd       2/2     Running   2 (2m20s ago)   22m
    reviews-v3-6dc9897554-t5clv       2/2     Running   2 (2m20s ago)   22m

3、查看 svc
    [root@k8s-master01 ~]# kubectl get services | egrep -v kubernetes
    NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
    details       ClusterIP   10.10.202.164   <none>        9080/TCP   44s
    productpage   ClusterIP   10.10.186.163   <none>        9080/TCP   44s
    ratings       ClusterIP   10.10.59.52     <none>        9080/TCP   44s
    reviews       ClusterIP   10.10.254.141   <none>        9080/TCP   44s

4、确认 Bookinfo 应用是否正在运行，在某个 Pod 中用 curl 命令对应用发送请求，例如 ratings：
    [root@k8s-master01 ~]# 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）确定 Ingress 的 IP 和端口

现在 Bookinfo 服务已经启动并运行，你需要使应用程序可以从 Kubernetes 集群外部访问，例如从浏览器访问，那可以用 Istio Gateway 来实现这个目标。

```
1、为应用程序定义 gateway 网关：
    [root@k8s-master01 ~]# cd istio-1.18.2/samples/bookinfo/networking/
    [root@k8s-master01 networking]# kubectl apply -f bookinfo-gateway.yaml
    gateway.networking.istio.io/bookinfo-gateway created
    virtualservice.networking.istio.io/bookinfo created

2、确认网关创建完成
    [root@k8s-master01 networking]# kubectl get gateway
    NAME               AGE
    bookinfo-gateway   67s

3、确定 ingress ip 和端口
```

```
[root@k8s-master01 networking]# kubectl get svc istio-ingressgateway -n istio-system
```

```
[root@k8s-master01 networking]# kubectl get svc istio-ingressgateway -n istio-system
NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
istio-ingressgateway   LoadBalancer   10.10.22.73   <pending>     15021:32294/TCP,80:32748/TCP,443:31146/TCP,31400:30757/TCP,15443:32300/TCP
[root@k8s-master01 networking]#
```

如果 EXTERNAL-IP 值已设置，说明环境正在使用外部负载均衡，可以用其为 ingress gateway 提供服务。如果 EXTERNAL-IP 值为<none>（或持续显示<pending>），说明环境没有提供外部负载均衡，无法使用 ingress gateway。在这种情况下，你可以使用服务的 NodePort 访问网关。

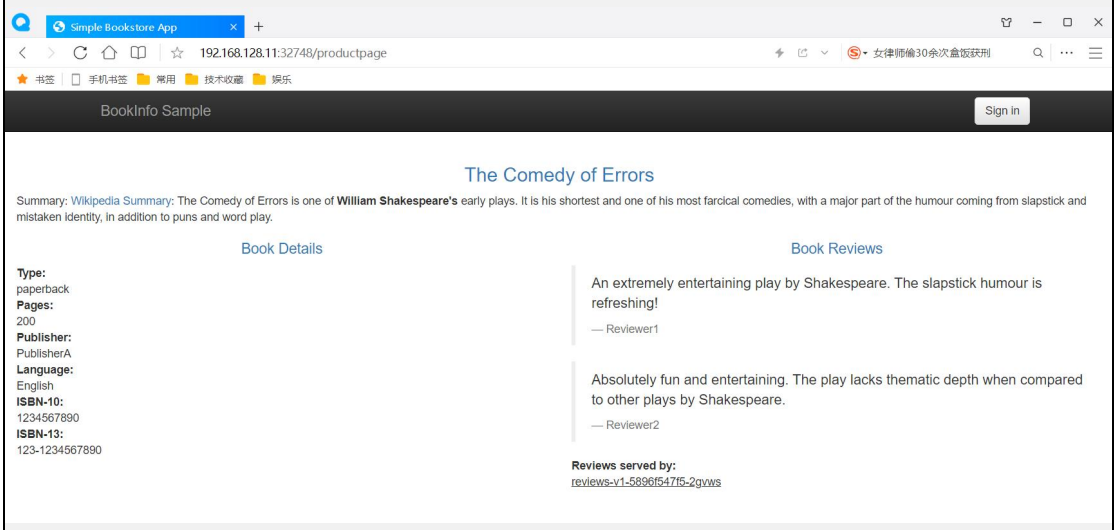
（4）访问 bookinfo

1、若自身环境未使用外部负载均衡器，可以通过 nodeport 访问。可以通过以下命令获取 Istio Gateway 的地址：

```
[root@k8s-master01 networking]# curl -s http://192.168.128.11:32748/productpage | grep -o "<title>.*</title>"
<title>Simple Bookstore App</title>
```

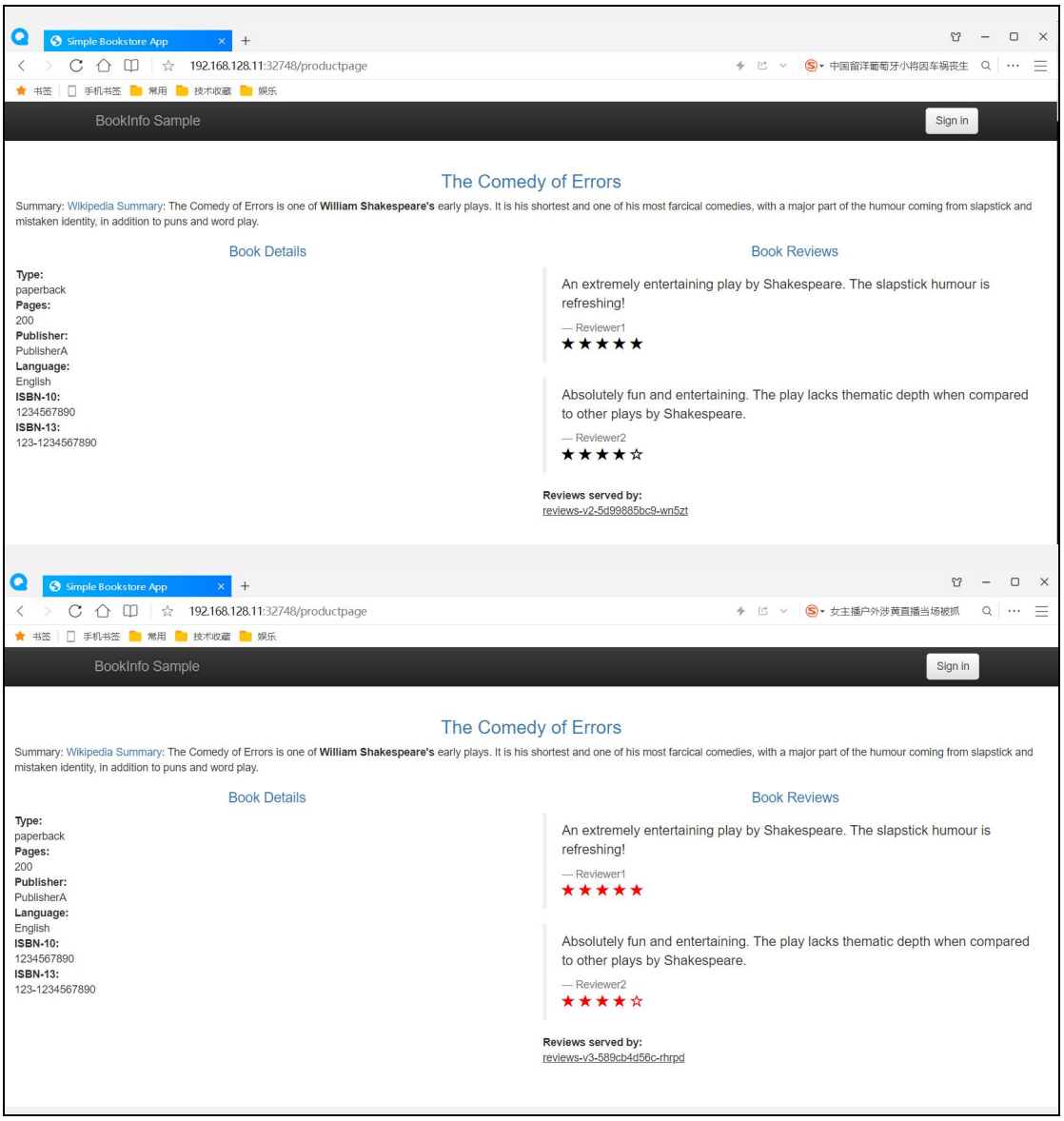
2、使用浏览器通过 nodeport 进行访问

浏览器输入：“http://192.168.128.12:32748/productpage”访问结果如下：



3、多刷新几次会看到 v2 和 v3 版本的 ratings 服务：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



4.3、卸载 bookinfo 项目

实验做完没问题之后，可以使用下面的命令来完成应用的删除和清理：

1、删除路由规则，并销毁应用的 Pod

```
[root@k8s-master01 ~]# cd istio-1.18.2/samples/bookinfo/platform/kube/
[root@k8s-master01 kube]# sh cleanup.sh
namespace ? [default] default   #输入 pod 部署在的名称空间
```

4.4、扩展：Bookinfo 项目部署报错

当执行完成 kubectl apply -f bookinfo.yaml 之后发现启动失败，如下：

```
[root@k8s-master01 kube]# kubectl get pods
```

```
[root@k8s-master01 kube]# kubectl get pods
NAME                          READY   STATUS                  RESTARTS     AGE
details-v1-65fbc6fbcf-nrlfd   0/2     Init:CrashLoopBackOff   1 (3s ago)   5s
productpage-v1-78d88f8f58-wxdlq   0/2     Init:CrashLoopBackOff   1 (3s ago)   4s
ratings-v1-7fcc55c79f-lwltd   0/2     Init:CrashLoopBackOff   1 (3s ago)   5s
reviews-v1-66c7d54957-mmdrn   0/2     Init:CrashLoopBackOff   1 (3s ago)   4s
reviews-v2-844777b87c-jxw2t   0/2     Init:CrashLoopBackOff   1 (3s ago)   4s
reviews-v3-b48665d5c-fnsbz    0/2     Init:CrashLoopBackOff   1 (3s ago)   4s
[root@k8s-master01 kube]#
```

所有的 pod 的状态都为 Init:CrashLoopBackOff，查看 pod init container 日志如下：

```
[root@k8s-master01 kube]# kubectl logs details-v1-65fbc6fbcf-nrlfd istio-init
-A ISTIO_OUTPUT -o lo -m owner ! --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -d 127.0.0.1/32 -j RETURN
-A ISTIO_OUTPUT -j ISTIO_REDIRECT
COMMIT
2024-02-19T13:55:51.087731Z     info    Running command (with wait lock): iptables-restore --noflush --wait=30
2024-02-19T13:55:51.088885Z     error   Command error output: xtables parameter problem: iptables-restore: unable to initializ
e table 'nat'

Error occurred at line: 1
Try `iptables-restore -h' or 'iptables-restore --help' for more information.
2024-02-19T13:55:51.088917Z     error   Failed to execute: iptables-restore --noflush, exit status 2
[root@k8s-master01 kube]#
```

问题原因：iptables 模块未被加载，所以我们可以尝试在所有 k8s 集群节点上加载 iptables 模块

修复：

```
# 在所有工作节点执行“直接加载模块”
[root@k8s-node01 ~]# modprobe ip_tables
[root@k8s-node01 ~]# modprobe iptable_filter
[root@k8s-node01 ~]# lsmod |grep -E "ip_tables|iptable_filter"
iptable_filter         16384  0
ip_tables              28672  1 iptable_filter
```

验证：

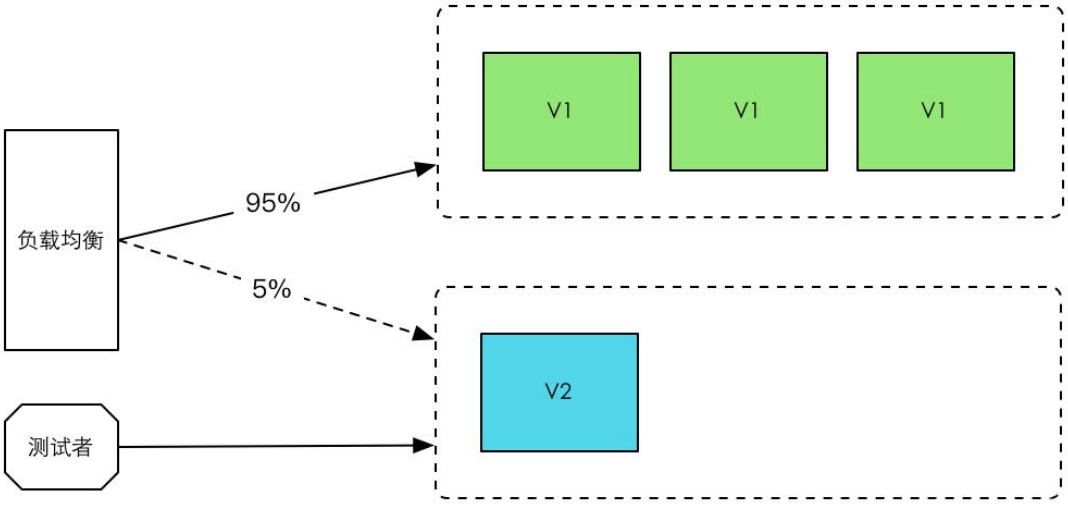
```
# 再次查看初始化容器日志
[root@k8s-master01 kube]# kubectl logs details-v1-65fbc6fbcf-nrlfd istio-init
-A ISTIO_IN_REDIRECT -p tcp -j REDIRECT --to-ports 15006
-A ISTIO_OUTPUT -s 127.0.0.6/32 -o lo -j RETURN
-A ISTIO_OUTPUT ! -d 127.0.0.1/32 -o lo -p tcp -m tcp ! --dport 15008 -m owner --uid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT ! -d 127.0.0.1/32 -o lo -p tcp -m tcp ! --dport 15008 -m owner --gid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -d 127.0.0.1/32 -j RETURN
-A ISTIO_OUTPUT -j ISTIO_REDIRECT
-A ISTIO_REDIRECT -p tcp -j REDIRECT --to-ports 15001
COMMIT
# Completed on Mon Feb 19 14:00:24 2024

[root@k8s-master01 kube]#
```

5、通过 Istio 实现灰度发布

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

金丝雀发布也叫灰度发布，起源是，矿井工人发现，金丝雀对瓦斯气体很敏感，矿工会在下井之前，先放一只金丝雀到井中，如果金丝雀不叫了，就代表瓦斯浓度高。



在灰度发布开始后，先启动一个新版本应用，但是并不直接将流量切过来，而是测试人员对新版本进行线上测试，启动的这个新版本应用，就是我们的金丝雀。如果没有问题，那么可以将少量的用户流量导入到新版本上，然后再对新版本做运行状态观察，收集各种运行时数据，如果此时对新旧版本做各种数据对比，就是所谓的 A/B 测试。

当确认新版本运行良好后，再逐步将更多的流量导入到新版本上，在此期间，还可以不断地调整新旧两个版本的运行的服务器副本数量，以使得新版本能够承受越来越大的流量压力。直到将 100%的流量都切换到新版本上，最后关闭剩下的老版本服务，完成灰度发布。

如果在灰度发布过程中（灰度期）发现了新版本有问题，就应该立即将流量切回老版本上，这样，就会将负面影响控制在最小范围内。

● Istio 灰度发布实战

（1）创建 v1 版本的应用

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi appv1.yaml
    ---
    kind: ConfigMap
    apiVersion: v1
    metadata:
      name: appv1
    data:
      index.html: |-
        Welcome to the appv1 test page
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appv1
spec:
  replicas: 1
  selector:
   matchLabels:
    version: v1
    app: web
  template:
   metadata:
    labels:
     version: v1
     app: web
   spec:
    containers:
    - name: web
      image: nginx:latest
      imagePullPolicy: IfNotPresent
      volumeMounts:
      - name: index-html
        mountPath: /usr/share/nginx/html/index.html
        subPath: index.html
    volumes:
      - name: index-html
        configMap:
          name: appv1
          items:
          - key: index.html
            path: index.html
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f appv1.yaml
configmap/appv1 created
deployment.apps/appv1 created
```

（2）创建 svc

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi app-svc.yaml
apiVersion: v1
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    kind: Service
    metadata:
      name: myapp
    spec:
      selector:
        app: web
      ports:
        - protocol: TCP
          port: 80
          targetPort: 80

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f app-svc.yaml
    service/myapp created

3、查看 svc
    [root@k8s-master01 ~]# kubectl get svc myapp
    NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
    myapp   ClusterIP   10.10.9.44   <none>        80/TCP     88s

4、测试访问
    [root@k8s-master01 test]# curl 10.10.9.44
    Welcome to the appv1 test page
```

（3）创建网关

```
1、创建资源清单文件
    [root@k8s-master01 test]# vi gateway.yaml
    apiVersion: networking.istio.io/v1beta1
    kind: Gateway
    metadata:
      name: myapp-gateway
    spec:
      selector:
        istio: ingressgateway
      servers:
      - port:
          number: 80
          name: http
          protocol: HTTP
        hosts:
        - "*"
```

```
2、更新资源清单文件
    [root@k8s-master01 test]# kubectl apply -f gateway.yaml
    gateway.networking.istio.io/myapp-gateway created

3、查看创建的资源
    [root@k8s-master01 test]# kubectl get gateway
    NAME            AGE
    myapp-gateway   12s
```

对上面的 yaml 文件说明如下：

这段代码定义了一个 Istio 网关，名称为 myapp-gateway。该网关使用了 Istio 的默认入口网关（istio: ingressgateway），这意味着这个网关将使用 Istio 控制的所有流量。

在 servers 部分，代码定义了一个 HTTP 服务器，监听 80 端口。该服务器接受来自所有主机的请求，因为 hosts 字段包含*。

通过定义这个网关，外部流量将被引导至服务网格中的应用程序。可以根据需要添加更多的服务器来监听不同的端口或协议。

（4）创建 VirtualService

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi virtualservice.yaml
    apiVersion: networking.istio.io/v1alpha3
    kind: VirtualService
    metadata:
      name: myapp
    spec:
      hosts:
        - "*"
      gateways:
      - myapp-gateway
      http:
      - route:
        - destination:
            host: myapp.default.svc.cluster.local

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f virtualservice.yaml
    virtualservice.networking.istio.io/myapp created
```

对上面的 yaml 文件说明如下：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 myapp 的虚拟服务（VirtualService）。虚拟服务用于定义网络流量规则，它指定了如何将流量路由到目标服务。

在 spec 部分，hosts 字段指定了该虚拟服务应用于哪些主机，这里使用了一个通配符*表示适用于所有主机。

然后，代码定义了一个网关（gateway）名为 myapp-gateway。网关用于将流量引导至服务网格中的应用程序。

接下来，代码定义了 HTTP 路由规则。该虚拟服务中有一条规则，将所有流量路由到名为 myapp 的目标服务，该服务位于 Kubernetes 集群的默认命名空间中。

通过这样的配置，Istio 可以将进入 myapp 服务的流量路由到相应的目标服务。

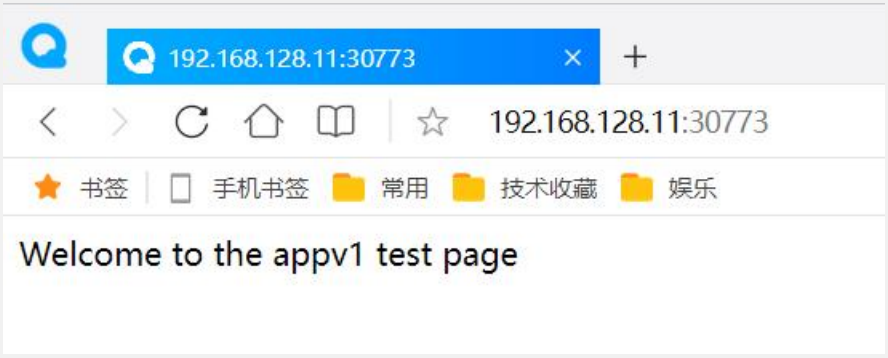
（5）测试访问

1、获取 istio ingress port

```
[root@k8s-master01 ~]# kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}'
30773
```

2、测试访问

浏览器访问：“http://192.168.128.11:30773/”，结果如下图所示



（6）部署金丝雀

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi appv2.yaml
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: appv2
data:
  index.html: |-
    Welcome to the appv2 test page

---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
    name: appv2
  spec:
    replicas: 1
    selector:
     matchLabels:
      version: v2
      app: web
    template:
     metadata:
      labels:
       version: v2
       app: web
     spec:
      containers:
      - name: web
        image: nginx:latest
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - name: index-html
          mountPath: /usr/share/nginx/html/index.html
          subPath: index.html
      volumes:
        - name: index-html
          configMap:
            name: appv2
            items:
            - key: index.html
              path: index.html
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f appv2.yaml
configmap/appv2 created
deployment.apps/appv2 created
```

3、测试访问 100 次，统计一下 v1 和 v2 版本访问次数

```
[root@k8s-master01 ~]# for i in `seq 1 100`;do curl 192.168.128.11:30773 &&
echo ;done > 1.txt
[root@k8s-master01 ~]# cat 1.txt | grep v1 | wc -l
43
[root@k8s-master01 ~]# cat 1.txt | grep v2 | wc -l
57
```

这时候我们觉得，新版本刚上线，存在太多不稳定性，不能够让太多用户去访问。那么这个时候我们需要去配置 VirtualService 权重。

（7）修改 VirtualService

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi virtualservice.yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp
spec:
  hosts:
    - "*"
  gateways:
  - myapp-gateway
  http:
  - route:
    - destination:
        host: myapp.default.svc.cluster.local
        subset: v1
      weight: 95
    - destination:
        host: myapp.default.svc.cluster.local
        subset: v2
      weight: 5
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: myapp
spec:
  host: myapp.default.svc.cluster.local
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

2、更新资源清单文件

```
[root@k8s-master01 test]# kubectl apply -f virtualservice.yaml
```

```
    virtualservice.networking.istio.io/myapp configured
    destinationrule.networking.istio.io/myapp created
```

对上面的清单文件说明：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 myapp 的虚拟服务（VirtualService）。虚拟服务用于定义网络流量规则，它指定了如何将流量路由到目标服务。

在 spec 部分，hosts 字段指定了该虚拟服务应用于哪些主机，这里使用了一个通配符*表示适用于所有主机。

然后，代码定义了一个网关（gateway）名为 myapp-gateway。网关用于将流量引导至服务网格中的应用程序。

接下来，代码定义了 HTTP 路由规则。该虚拟服务中有两条规则，第一条规则将 95%的流量路由到名为 v1 的子集，第二条规则将 5%的流量路由到名为 v2 的子集。这两个子集分别对应着不同的服务版本。

接下来，代码定义了一个目标规则（DestinationRule）。目标规则用于定义目标服务的子集和标签。

在 spec 部分，host 字段指定了目标服务的地址，这里使用了 Kubernetes 的服务 DNS 名称 myapp.default.svc.cluster.local。

然后，代码定义了两个子集，分别命名为 v1 和 v2。每个子集都有一个标签，标签用于标识子集的版本。在这个例子中，v1 版本的标签为 version: v1，v2 版本的标签为 version: v2。

通过这样的配置，Istio 可以根据标签将进入 myapp 服务的流量路由到相应的子集。例如，v1 子集的请求会转发到拥有 version: v1 标签的 pod 上。

（8）测试访问 100 次，统计一下 v1 和 v2 版本访问次数

```
    [root@k8s-master01 ~]# for i in `seq 1 100`;do curl 192.168.128.11:30773 && echo ;done > 1.txt
    [root@k8s-master01 test]# cat 1.txt | grep v1 | wc -l
    93
    [root@k8s-master01 test]# cat 1.txt | grep v2 | wc -l
    7
```

（9）清空环境

```
    kubectl delete -f appv1.yaml
    kubectl delete -f app-svc.yaml
    kubectl delete -f gateway.yaml
    kubectl delete -f virtualservice.yaml
    kubectl delete -f appv2.yaml
```

6、Istio 实现服务熔断

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

熔断器是一个容错管理组件，为服务中出现的延迟和故障提供强大的容错能力。

● Istio 服务熔断实战

（1）给命名空间添加标签，指示 Istio 在部署应用的时候，自动注入 Envoy 边车代理

```
[root@k8s-master01 ~]# kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

（2）创建后端服务

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    server: nginx
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      server: nginx
      app: web
  template:
    metadata:
      name: nginx
      labels:
        server: nginx
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        imagePullPolicy: IfNotPresent
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nginx.yaml
deployment.apps/nginx created
```

3、查看创建的 pod

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl get pods -l server=nginx
NAME                     READY   STATUS    RESTARTS   AGE
nginx-66f4b775cc-5tqvj   2/2     Running   0          5m2s
# 发现 Nginx 容器个数已经是 2 个，就说明注入成功了。
```

（3）创建熔断器

1、创建资源清单文件

```
[root@k8s-master01 ~]# cat destination.yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: http-web
spec:
  host: nginx-svc
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutiveGatewayErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f destination.yaml
destinationrule.networking.istio.io/http-web created
```

对上面的 yaml 文件说明如下：

这段代码是使用 Istio 进行服务路由和负载均衡的配置。

首先，代码定义了一个名为 http-web 的目标规则（DestinationRule）。目标规则用于定义目标服务的子集和标签，并可以配置一些流量控制和路由规则。

在 spec 部分，host 字段指定了目标服务的地址，这里使用了 nginx-svc 作为目标服务的名称。

在 trafficPolicy 部分，配置了一些流量控制规则。

在 connectionPool 字段中，配置了 TCP 和 HTTP 连接池的参数。TCP 连接池的 maxConnections 参数设置为 1，表示限制最大并发 TCP 连接数为 1。HTTP 连接池的

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

http1MaxPendingRequests 参数设置为 1，表示限制最大未响应的 HTTP 1.1 请求数为 1，maxRequestsPerConnection 参数设置为 1，表示限制每个 TCP 连接的最大 HTTP 请求数为 1。

在 outlierDetection 字段中，配置了异常检测的参数。consecutiveGatewayErrors 参数设置为 1，表示当连续出现 1 次错误时触发异常检测。interval 参数设置为 1 秒，表示每隔 1 秒检测一次。baseEjectionTime 参数设置为 3 分钟，表示从触发异常检测开始，需要等待 3 分钟才会开始进行流量重定向。maxEjectionPercent 参数设置为 100，表示最大驱逐的百分比。

通过这样的配置，Istio 可以控制进入 nginx-svc 服务的流量，并进行异常检测和流量重定向。

（4）创建 svc

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi web-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    app: web
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f web-svc.yaml
service/nginx-svc created
```

（5）创建客户端访问 nginx 服务

Istio 二进制包中提供了测试工具 Fortio。Fortio 可以控制连接数，并发数和 HTTP 调用延迟。使用此客户端来“跳闸”在 DestinationRule 中设置的断路器策略。

1、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f
istio-1.18.2/samples/httpbin/sample-client/fortio-deploy.yaml
service/fortio created
deployment.apps/fortio-deploy created
```

2、查看创建的 pod 和 svc

```
    [root@k8s-master01 ~]# kubectl get pods -l app=fortio
    NAME                              READY   STATUS    RESTARTS   AGE
    fortio-deploy-956754996-q9c46     2/2     Running   0          112s

    [root@k8s-master01 ~]# kubectl get svc -l app=fortio
    NAME      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)     AGE
    fortio    ClusterIP   10.10.62.74    <none>        8080/TCP    116s

3、查看 nginx svc 地址
    [root@k8s-master01 ~]# kubectl get svc nginx-svc
    NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
    nginx-svc    ClusterIP   10.10.64.21    <none>        80/TCP     6m24s

4、测试通过 fortio 容器访问 nginx-svc 地址
    [root@k8s-master01 ~]# kubectl exec fortio-deploy-956754996-q9c46 -c fortio --
/usr/bin/fortio curl http://10.10.64.21
    HTTP/1.1 200 OK
    server: envoy
    date: Wed, 06 Sep 2023 10:08:10 GMT
    content-type: text/html
    content-length: 615
    last-modified: Tue, 28 Dec 2021 15:28:38 GMT
    etag: "61cb2d26-267"
    accept-ranges: bytes
    x-envoy-upstream-service-time: 1

    <!DOCTYPE html>
    <html>
    <head>
    <title>Welcome to nginx!</title>
    <style>
    html { color-scheme: light dark; }
    body { width: 35em; margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif; }
    </style>
    </head>
    <body>
    <h1>Welcome to nginx!</h1>
    <p>If you see this page, the nginx web server is successfully installed and
    working. Further configuration is required.</p>

    <p>For online documentation and support please refer to
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    <a href="http://nginx.org/">nginx.org</a>.<br/>
    Commercial support is available at
    <a href="http://nginx.com/">nginx.com</a>.</p>

    <p><em>Thank you for using nginx.</em></p>
    </body>
    </html>
```

（6）触发断路器

1、以两个并发连接（-c 2）和发送 20 个请求（-n 20）调用服务

```
    [root@k8s-master01 ~]# kubectl exec fortio-deploy-956754996-q9c46 -c fortio --
/usr/bin/fortio load -c 2 -qps 0 -n 20 -loglevel Warning http://nginx-svc
```

```
[root@k8s-master01 ~]# kubectl exec fortio-deploy-956754996-q9c46 -c fortio -- /usr/bin/fortio load -c 2 -qps 0 -n 20 -loglevel Warning http://nginx-svc
10:10:30 I logger.go:127> Log level is now 3 Warning (was 2 Info)
Fortio 1.17.1 running at 0 queries per second, 6->6 procs, for 20 calls: http://nginx-svc
Starting at max qps with 2 thread(s) [gomax 6] for exactly 20 calls (10 per thread + 0)
10:10:30 W http_client.go:806> [1] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [1] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [1] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [1] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [1] Non ok http code 503 (HTTP/1.1 503)
10:10:30 W http_client.go:806> [0] Non ok http code 503 (HTTP/1.1 503)
Ended after 26.848957ms : 20 calls. qps=744.91
Aggregated Function Time : count 20 avg 0.0023010804 +/- 0.002023 min 0.000294471 max 0.006518547 sum 0.046021609
# range, mid point, percentile, count
>= 0.000294471 <= 0.001 , 0.000647235 , 45.00, 9
> 0.001 <= 0.002 , 0.0015 , 55.00, 2
> 0.002 <= 0.003 , 0.0025 , 60.00, 1
> 0.003 <= 0.004 , 0.0035 , 85.00, 5
> 0.005 <= 0.006 , 0.0055 , 95.00, 2
> 0.006 <= 0.00651855 , 0.00625927 , 100.00, 1
# target 50% 0.0015
# target 75% 0.0036
# target 90% 0.0055
# target 99% 0.00641484
# target 99.9% 0.00650818
Sockets used: 12 (for perfect keepalive, would be 2)
Jitter: false
Code 200 : 9 (45.0 %)
Code 503 : 11 (55.0 %)
Response Header Sizes : count 20 avg 108.45 +/- 119.9 min 0 max 241 sum 2169
Response Body/Total Sizes : count 20 avg 517.75 +/- 306 min 241 max 856 sum 10355
All done 20 calls (plus 0 warmup) 2.301 ms avg, 744.9 qps
[root@k8s-master01 ~]#
```

访问成功的数量，断开的数量

从执行结果来看，有 11 个报错。 说明断路器生效。

（7）清空环境

```
    kubectl delete -f nginx.yaml
    kubectl delete -f destination.yaml
    kubectl delete -f web-svc.yaml
    kubectl delete -f
istio-1.18.2/samples/httpbin/sample-client/fortio-deploy.yaml
```

7、Istio 实现服务延迟和超时

在 Istio 中，延迟和超时可以通过虚拟服务和目标规则进行配置。虚拟服务定义了从应用程序到 Istio 代理的流量规则，而目标规则定义了从 Istio 代理到服务实例的流量规则。

虚拟服务中的超时配置定义了在将请求从应用程序路由到服务实例之前，Istio 代理需要等待的最长时间。如果超过这个时间，Istio 将终止请求并返回

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

错误响应。这个设置可以避免应用程序长时间等待无响应的服务，从而节约资源并避免级联错误。

目标规则中的超时配置定义了在将请求从 Istio 代理发送到服务实例之前，Istio 代理需要等待的最长时间。如果超过这个时间，Istio 代理将关闭与服务实例的连接并返回错误响应。这个设置也可以避免服务实例长时间无响应，从而节约资源并避免级联错误。

通过设置不同的超时时间，可以观察延迟对服务的影响，理解超时的原理，以及了解如果单个服务发生异常可能对系统整体造成的影响。

- Istio 服务延迟和操作实战

在生产环境中经常会遇到调用方等待下游响应时间过长，堆积大量的请求阻塞了自身服务，造成服务雪崩的情况。为了增强服务的可用性，可以使用 istio 进行超时配置。

下面模拟客户端调用 nginx，nginx 将请求转发给 tomcat。nginx 服务设置了超时时间为 2 秒，如果超出这个时间就不在等待，返回超时错误。tomcat 服务设置了响应时间延迟 10 秒，任何请求都需要等待 10 秒后才能返回。client 通过访问 nginx 服务去反向代理 tomcat 服务，由于 tomcat 服务需要 10 秒后才能返回，但 nginx 服务只等待 2 秒，所以客户端会提示超时错误。

（1）创建 pod

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi nginx-tomcat-deployment.yaml
    ---
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: nginx
      labels:
        server: nginx
        app: web
    spec:
      replicas: 1
      selector:
        matchLabels:
          server: nginx
          app: web
      template:
        metadata:
          name: nginx
          labels:
            server: nginx
```

```
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        imagePullPolicy: IfNotPresent
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat
  labels:
    server: tomcat
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      server: tomcat
      app: web
  template:
    metadata:
      name: tomcat
      labels:
        server: tomcat
        app: web
    spec:
      containers:
      - name: tomcat
        image: kubeguide/tomcat-app:v1
        imagePullPolicy: IfNotPresent
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nginx-tomcat-deployment.yaml
deployment.apps/nginx created
deployment.apps/tomcat created
```

3、查看创建的 pod

```
[root@k8s-master01 ~]# kubectl get pods -l app=web
NAME                      READY   STATUS    RESTARTS   AGE
nginx-66f4b775cc-ddl7q    2/2     Running   0          27s
tomcat-68855ff4d9-jvp28   2/2     Running   0          27s
```

（2）创建 svc

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi nginx-tomcat-svc.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    server: nginx
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: tomcat-svc
spec:
  selector:
    server: tomcat
  ports:
  - name: http
    port: 8080
    targetPort: 8080
    protocol: TCP
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nginx-tomcat-svc.yaml
service/nginx-svc created
service/tomcat-svc created
```

3、查看创建的 svc

```
[root@k8s-master01 ~]# kubectl get svc nginx-svc && kubectl get svc tomcat-svc
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
nginx-svc   ClusterIP   10.10.70.38    <none>        80/TCP     44s
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
tomcat-svc   ClusterIP   10.10.43.108    <none>        8080/TCP    44s
```

（3）创建 VirtualService 资源，配置延迟和超时

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi virtual-nginx-tomcat.yaml
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: nginx-vs
spec:
  hosts:
  - nginx-svc
  http:
  - route:
    - destination:
        host: nginx-svc
    timeout: 2s
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: tomcat-vs
spec:
  hosts:
  - tomcat-svc
  http:
  - fault:
      delay:
        percentage:
          value: 100
        fixedDelay: 10s
    route:
    - destination:
        host: tomcat-svc
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f virtual-nginx-tomcat.yaml
virtualservice.networking.istio.io/nginx-vs created
virtualservice.networking.istio.io/tomcat-vs created
```

3、查看创建的 VirtualService 资源

```
[root@k8s-master01 ~]# kubectl get VirtualService
```

```
NAME        GATEWAYS   HOSTS            AGE
nginx-vs                ["nginx-svc"]    12m
tomcat-vs               ["tomcat-svc"]   12m
```

对上面的资源清单文件解释如下：

1、超时配置：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: nginx-vs
spec:
  hosts:
  - nginx-svc
  http:
  - route:
    - destination:
        host: nginx-svc
    timeout: 2s
```

解释如下：

这段代码是使用 Kubernetes 和 Istio 的配置定义了一个名为 nginx-vs 的虚拟服务。虚拟服务是一种定义服务路由的 Kubernetes 资源，它可以将流量引导到不同的服务目标。

以下是这段代码的主要部分的详细解释：

- apiVersion: networking.istio.io/v1beta1 指定了使用 Istio 的虚拟服务 API 的版本。
- kind: VirtualService 指定了此资源类型为虚拟服务。
- metadata: {name: nginx-vs} 定义了此虚拟服务的名字为 nginx-vs。
- spec: {hosts: [nginx-svc]} 定义了此虚拟服务的主机列表，这里只定义了一个主机名，即 nginx-svc。
- http: [{route: [{destination: {host: nginx-svc}}], timeout: 2s}] 定义了 HTTP 路由规则。这里只有一个路由规则，它将所有 HTTP 请求都导向到 nginx-svc 这个服务。timeout: 2s 定义了在将请求发送到目标服务之前的超时时间，单位为秒。

总的来说，这段代码创建了一个虚拟服务，将所有对 nginx-svc 的 HTTP 请求都导向到该服务，并设置了 2 秒的超时时间。

2、延迟配置：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: tomcat-vs
spec:
```

```
    hosts:
    - tomcat-svc
    http:
    - fault:
        delay:
          percentage:
            value: 100
          fixedDelay: 10s
      route:
      - destination:
          host: tomcat-svc
```

解释如下：

这段代码是使用 Kubernetes 和 Istio 的配置定义了一个名为 tomcat-vs 的虚拟服务。虚拟服务是一种定义服务路由的 Kubernetes 资源，它可以将流量引导到不同的服务目标。

以下是这段代码的主要部分的详细解释：

- apiVersion: networking.istio.io/v1beta1 指定了使用 Istio 的虚拟服务 API 的版本。
- kind: VirtualService 指定了此资源类型为虚拟服务。
- metadata: {name: tomcat-vs} 定义了此虚拟服务的名字为 tomcat-vs。
- spec:{hosts:[tomcat-svc]} 定义了此虚拟服务的主机列表，这里只定义了一个主机名，即 tomcat-svc。
- http:[{fault:{delay:{percentage:{value:100}, fixedDelay:10s}}, route:[{destination: {host: tomcat-svc}}]}] 定义了 HTTP 的故障注入和路由规则。首先，故障注入部分定义了一个固定的延迟，有 100%的概率发生，延迟时间为 10 秒。然后，路由规则部分将所有 HTTP 请求都导向到 tomcat-svc 这个服务。

总的来说，这段代码创建了一个虚拟服务，将所有对 tomcat-svc 的 HTTP 请求都导向到该服务，并在 100%的概率下添加了一个 10 秒的固定延迟。这是一种用于故障测试或模拟网络不稳定性的方法。

（4）配置 nginx 代理转发

```
1、进入 nginx pod 内，更新源，安装 Vim 工具
    [root@k8s-master01 ~]# kubectl exec -it nginx-66f4b775cc-ddl7q -- /bin/bash
    root@nginx-66f4b775cc-ddl7q:/# apt-get update
    root@nginx-66f4b775cc-ddl7q:/# apt-get install vim -y

2、配置 nginx 转发
    root@nginx-66f4b775cc-ddl7q:/# vim /etc/nginx/conf.d/default.conf
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #access_log  /var/log/nginx/host.access.log  mai

    location / {
        # root   /usr/share/nginx/html;
        # index  index.html index.htm;
        proxy_pass http://tomcat-svc:8080;
        proxy_http_version 1.1;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page
    #
```

```
    location / {
        # root   /usr/share/nginx/html;
        # index  index.html index.htm;
        proxy_pass http://tomcat-svc:8080;
        proxy_http_version 1.1;
    }
```

3、验证配置是否正确

```
root@nginx-66f4b775cc-ddl7q:/# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

4、重新加载 nginx 配置

```
root@nginx-66f4b775cc-ddl7q:/# nginx -s reload
2023/09/07 02:19:28 [notice] 436#436: signal process started
```

（5）验证延迟和超时

1、启动一个 client

```
[root@k8s-master01 ~]# kubectl run busybox --image busybox:1.28
--restart=Never --rm -it busybox -- sh
```

2、访问一次 nginx-svc，验证超时配置。

```
/ # time wget -q -O - http://nginx-svc
wget: server returned error: HTTP/1.1 504 Gateway Timeout
Command exited with non-zero status 1
real    0m 2.00s
user    0m 0.00s
sys 0m 0.00s
# 可以看到报 504 网关超时
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
3、循环访问 nginx-svc，验证超时配置。
    / # while true; do wget -q -O - http://nginx-svc; done
    wget: server returned error: HTTP/1.1 504 Gateway Timeout
    wget: server returned error: HTTP/1.1 504 Gateway Timeout
    wget: server returned error: HTTP/1.1 504 Gateway Timeout
    # 可以看到每隔 2 秒，由于 nginx 服务的超时时间到了而 tomcat 没有响应，则提示返回超时错误。

4、访问 tomcat-svc，验证延迟配置。
    / # time wget -q -O - http://tomcat-svc:8080
    <!DOCTYPE html>
    ...部分内容省略...
    real    0m 10.01s
    user    0m 0.00s
    sys 0m 0.00s
    # 可以看到是能够看到 tomcat 界面结果的，只不过消耗时间比较久，延迟了 10s，响应时间仅有 0.01s。
```

（6）清空环境

```
    kubectl delete -f nginx-tomcat-deployment.yaml
    kubectl delete -f nginx-tomcat-svc.yaml
    kubectl delete -f virtual-nginx-tomcat.yaml
```

8、Istio 实现服务故障注入和重试

Istio 故障注入和重试的功能可以帮助开发测试人员评估应用程序在面对各种故障情况时的健壮性。

● 故障注入

Istio 可以注入故障，模拟上游服务请求的异常行为。这种功能是通过配置 VirtualService 中的规则来实现的。在接收到用户请求后，Istio 可以根据配置，向请求中注入错误，例如 HTTP 请求错误。具体来说，可以模拟上游服务处理时长异常、服务异常状态、自定义响应状态码等故障信息。

在 Istio 中，故障注入有两种类型：延迟（Delay）和中止（Abort）：

延迟故障：模拟出高负载时系统响应很慢的一种现象。通过定义延迟故障，可以模拟出系统处理请求的时间延长，从而观察系统在负载情况下的表现。

中止故障：模拟出系统崩溃，直接返回 HTTP 错误代码或者 TCP 连接失败的现象。通过定义中止故障，可以模拟出上游服务发生错误的情况，从而观察系统在错误情况下的表现。

● 重试

在微服务架构中，网络请求可能会经过多个服务节点，如果某个节点发生故障或者响应超时，就需要进行重试。然而，简单的重试可能导致更多的错误，甚至引发重复操作的问题。

Istio 通过配置 Envoy 的重试机制，提供了一种可靠的重试机制。在 VirtualService 中，可以通过配置重试的次数和间隔时间，控制重试的行为。例如，可以配置重试次数为 3 次，每次重试的间隔时间为 2 秒。

需要注意的是，重试可能会导致更多的网络延迟，因此在进行重试配置时，需要平衡延迟和正确性的需求。

● Istio 服务故障注入和重试实战

（1）重建 tomcat 和 nginx 服务

```
1、重建服务
    [root@k8s-master01 ~]# kubectl apply -f nginx-tomcat-deployment.yaml
    [root@k8s-master01 ~]# kubectl apply -f nginx-tomcat-svc.yaml
```

（2）创建 VirtualService 资源，配置故障注入和重试

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi virtualservice.yaml
    ---
    apiVersion: networking.istio.io/v1beta1
    kind: VirtualService
    metadata:
      name: nginx-vs
    spec:
      hosts:
      - nginx-svc
      http:
      - route:
        - destination:
            host: nginx-svc
        retries:
          attempts: 3
          perTryTimeout: 2s
    ---
    apiVersion: networking.istio.io/v1beta1
    kind: VirtualService
    metadata:
      name: tomcat-vs
    spec:
      hosts:
      - tomcat-svc
```

```
    http:
    - fault:
        abort:
          percentage:
            value: 100
          httpStatus: 503
      route:
      - destination:
          host: tomcat-svc
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f virtual-retry.yaml
virtualservice.networking.istio.io/nginx-vs created
virtualservice.networking.istio.io/tomcat-vs created
```

3、查看创建的 VirtualService 资源

```
[root@k8s-master01 ~]# kubectl get VirtualService
NAME        GATEWAYS   HOSTS            AGE
nginx-vs                ["nginx-svc"]    11s
tomcat-vs               ["tomcat-svc"]   11s
```

对上面 yaml 文件说明如下：

1、重试配置

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: nginx-vs
spec:
  hosts:
  - nginx-svc
  http:
  - route:
    - destination:
        host: nginx-svc
    retries:
      attempts: 3
      perTryTimeout: 2s
```

解释如下：

这段代码是使用 Kubernetes 和 Istio 的配置定义了一个名为 nginx-vs 的虚拟服务。虚拟服务是一种定义服务路由的 Kubernetes 资源，它可以将流量引导到不同的服务目标。

以下是这段代码的主要部分的详细解释：

- apiVersion: networking.istio.io/v1beta1 指定了使用 Istio 的虚拟服务 API 的版本。
- kind: VirtualService 指定了此资源类型为虚拟服务。
- metadata: {name: nginx-vs} 定义了此虚拟服务的名字为 nginx-vs。
- spec: {hosts: [nginx-svc]} 定义了此虚拟服务的主机列表，这里只定义了一个主机名，即 nginx-svc。
- http: [{route: [{destination: {host: nginx-svc}}]}, {retries: {attempts: 3, perTryTimeout: 2s}}] 定义了 HTTP 的路由规则和重试机制。首先，路由规则部分将所有 HTTP 请求都导向到 nginx-svc 这个服务。然后，重试机制部分定义了在发送请求时，如果有错误发生，将尝试重新发送最多 3 次，每次尝试的超时时间为 2 秒。

总的来说，这段代码创建了一个虚拟服务，将所有对 nginx-svc 的 HTTP 请求都导向到该服务，并设置了 3 次重试和每次 2 秒的超时时间。这是一种用于提高请求成功率的机制。

2、故障注入配置

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: tomcat-vs
spec:
  hosts:
  - tomcat-svc
  http:
  - fault:
      abort:
        percentage:
          value: 100
        httpStatus: 503
    route:
    - destination:
        host: tomcat-svc
```

解释如下：

这段代码是使用 Kubernetes 和 Istio 的配置定义了一个名为 tomcat-vs 的虚拟服务。虚拟服务是一种定义服务路由的 Kubernetes 资源，它可以将流量引导到不同的服务目标。

以下是这段代码的主要部分的详细解释：

- apiVersion: networking.istio.io/v1beta1 指定了使用 Istio 的虚拟服务 API 的版本。
- kind: VirtualService 指定了此资源类型为虚拟服务。
- metadata: {name: tomcat-vs} 定义了此虚拟服务的名字为 tomcat-vs。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

- spec: {hosts: [tomcat-svc]} 定义了此虚拟服务的主机列表，这里只定义了一个主机名，即 tomcat-svc。
- http: [{fault: {abort: {percentage: {value: 100}, httpStatus: 503}}, route: [{destination: {host: tomcat-svc}}]}] 定义了 HTTP 的故障中断和路由规则。首先，故障中断部分定义了一个百分百的概率，并且中断的 HTTP 状态码为 503，这将导致请求失败。然后，路由规则部分将所有 HTTP 请求都导向到 tomcat-svc 这个服务。

总的来说，这段代码创建了一个虚拟服务，将所有对 tomcat-svc 的 HTTP 请求都导向到该服务，并在百分百的概率下中断请求，返回 HTTP 状态码 503。这是一种用于测试或模拟故障情况的机制。

（3）配置 nginx 代理转发

1、进入 nginx pod 内，更新源，安装 Vim 工具

```
[root@k8s-master01 ~]# kubectl exec -it nginx-66f4b775cc-ddl7q -- /bin/bash
root@nginx-66f4b775cc-ddl7q:/# apt-get update
root@nginx-66f4b775cc-ddl7q:/# apt-get install vim -y
```

2、配置 nginx 转发

```
root@nginx-66f4b775cc-ddl7q:/# vim /etc/nginx/conf.d/default.conf
```

```
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #access_log  /var/log/nginx/host.access.log  mai

    location / {
        # root   /usr/share/nginx/html;
        # index  index.html index.htm;
        proxy_pass http://tomcat-svc:8080;
        proxy_http_version 1.1;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page
```

```
    location / {
        # root   /usr/share/nginx/html;
        # index  index.html index.htm;
        proxy_pass http://tomcat-svc:8080;
        proxy_http_version 1.1;
    }
```

3、验证配置是否正确

```
root@nginx-66f4b775cc-ddl7q:/# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
```

```
    nginx: configuration file /etc/nginx/nginx.conf test is successful

4、重新加载 nginx 配置
    root@nginx-66f4b775cc-ddl7q:/# nginx -s reload
    2023/09/07 02:19:28 [notice] 436#436: signal process started
```

（4）验证重试和故障注入

```
1、启动一个 client
    [root@k8s-master01 ~]# kubectl run busybox --image busybox:1.28
--restart=Never --rm -it busybox -- sh

2、测试访问
    / # wget -q -O - http://nginx-svc
    wget: server returned error: HTTP/1.1 503 Service Unavailable
    # 可以看到访问 nginx-svc，将请求转发给了 tomcat-svc，因为配置了故障注入，并在百分百的概率下中断请求，返回 HTTP 状态码 503。

3、查看 nginx pod 的 istio-proxy 日志
    [root@k8s-master01 ~]# kubectl logs -f nginx-66f4b775cc-8x6g4 -c istio-proxy
```

由上图可知，重试和故障注入设置生效。

（5）清空环境

```
    kubectl delete -f nginx-tomcat-deployment.yaml
    kubectl delete -f nginx-tomcat-svc.yaml
    kubectl delete -f virtualservice.yaml
```

9、手动注入 SidCar（对指定 pod 进行注入）

对于某些场景，我们不想让整个命名空间下的 pod 都注入 sidcar，只想对某个 pod 注入 sidcar，那该如何操作呢？

步骤如下：

```
1、创建测试命名空间
    [root@k8s-master01 ~]# kubectl create ns test
    namespace/test created

2、准备资源清单文件
```

```
[root@k8s-master01 ~]# vi test-sidcar.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: test
spec:
  replicas: 1
  selector:
   matchLabels:
    app: nginx
  template:
   metadata:
    labels:
     app: nginx
   spec:
    containers:
    - name: nginx
      image: nginx:latest
      imagePullPolicy: IfNotPresent
      ports:
      - containerPort: 80

3、更新资源清单文件，注入 sidcar
[root@k8s-master01 ~]# ./istio-1.18.7/bin/istioctl kube-inject -f
test-sidcar.yaml | kubectl apply -f -
deployment.apps/nginx created

4、检查
[root@k8s-master01 ~]# kubectl get pods -n test
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7f5958b4b9-d9r5z   2/2     Running   0          7s
```

10、istio 可视化工具 kiali

Kiali 为我们提供了查看相关服务与配置提供了统一化的可视化界面，并且能在其中展示他们的关联；

同时他还提供了界面让我们可以很方便的验证 istio 配置与错误提示。

通过 Kiala 可以帮助我们了解服务网格的结构，显示网格的拓扑结构以及分析网格的运行状况。

步骤如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、部署 kiali

```
[root@k8s-master01 ~]# kubectl apply -f istio-1.18.7/samples/addons/
```

2、检查 kiali 运行情况

```
[root@k8s-master01 ~]# kubectl get pods -n istio-system
```

```
[root@k8s-master01 ~]# kubectl get pods -n istio-system
NAME                                    READY   STATUS    RESTARTS      AGE
grafana-97fbd954-z2vv8                  1/1     Running   0             62s
istio-egressgateway-58fdc85767-4vpxp    1/1     Running   1 (47m ago)   10h
istio-ingressgateway-84bb566b49-89t6k   1/1     Running   1 (47m ago)   10h
istiod-66ddb6b5d5-mb7s8                 1/1     Running   1 (47m ago)   10h
jaeger-7cf8c7c56d-j4fpn                 1/1     Running   0             62s
kiali-84766d44dd-sdbc7                  1/1     Running   0             61s
prometheus-db8b4588f-m4ttr              2/2     Running   0             60s
[root@k8s-master01 ~]#
```

3、修改 kiali svc 为 nodeport 类型

```
[root@k8s-master01 ~]# kubectl edit svc kiali -n istio-system
  type: NodePort
```

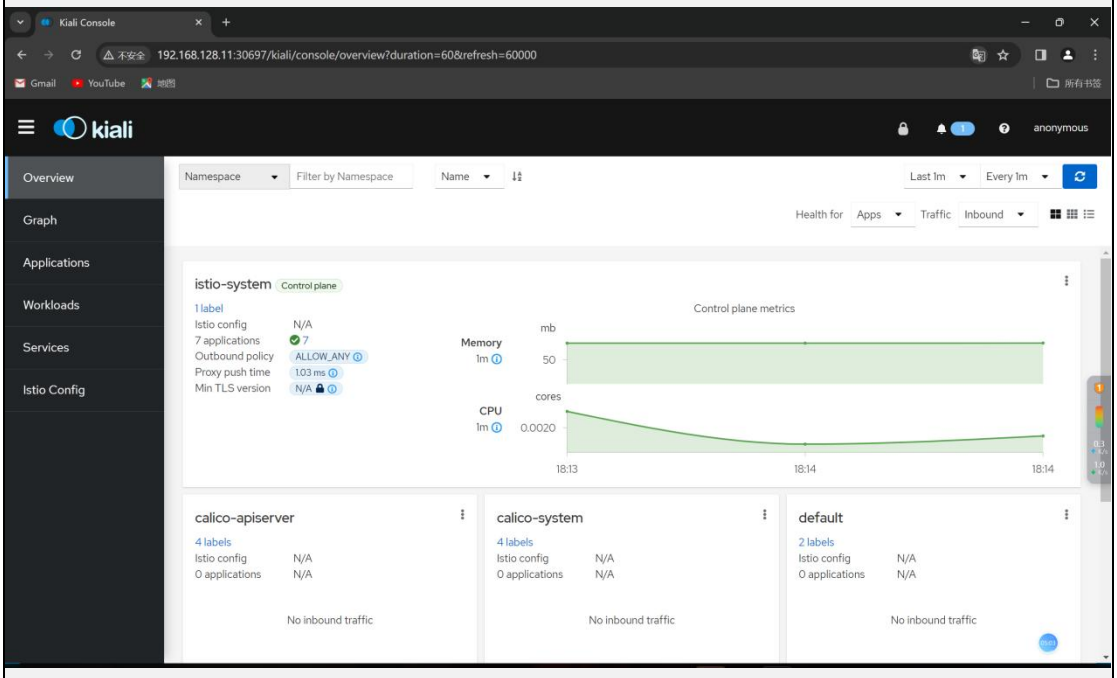
4、查看 kiali svc 映射到的宿主机端口

```
[root@k8s-master01 ~]# kubectl get svc kiali -n istio-system
```

```
[root@k8s-master01 ~]# kubectl get svc kiali -n istio-system
NAME    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)                          AGE
kiali   NodePort   10.10.5.136    <none>        20001:30697/TCP,9090:32402/TCP   2m40s
[root@k8s-master01 ~]#
```

5、浏览器访问 kiali

浏览器输入：“http://192.168.128.11:30697/”



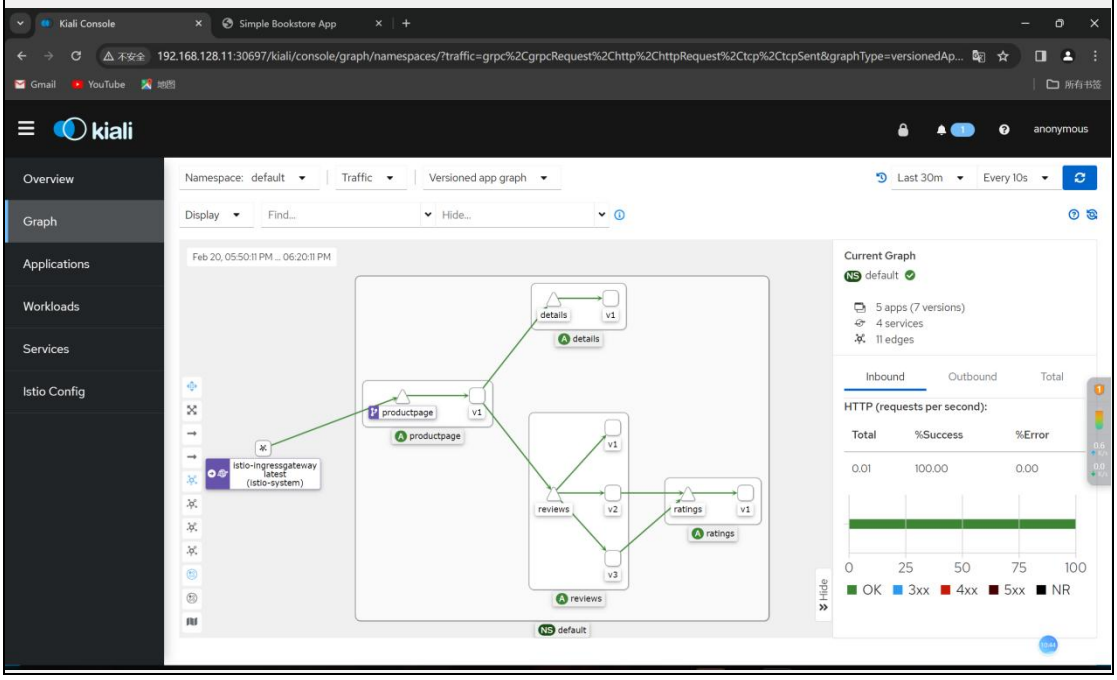
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

6、部署测试应用

```
    [root@k8s-master01 ~]# kubectl apply -f
istio-1.18.7/samples/bookinfo/networking/bookinfo-gateway.yaml
    [root@k8s-master01 ~]# kubectl apply -f
istio-1.18.7/samples/bookinfo/platform/kube/bookinfo.yaml
```

7、测试访问几次应用

8、kiali 查看服务网格结构



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。