实验环境：

Istio 安装在已经存在的 k8s 集群上即可

k8s 集群：　　k8s 的控制节点

ip：192.168.40.180

主机名：xianchaomaster1

配置：6vCPU/6Gi 内存

k8s 的工作节点：

ip：192.168.40.181

主机名：xianchaonode1

配置：12vCPU/8Gi 内存

1.Istio 介绍？

官方文档：https://istio.io/docs/concepts/what-is-istio/

中文官方文档：https://istio.io/zh/docs/concepts/what-is-istio/

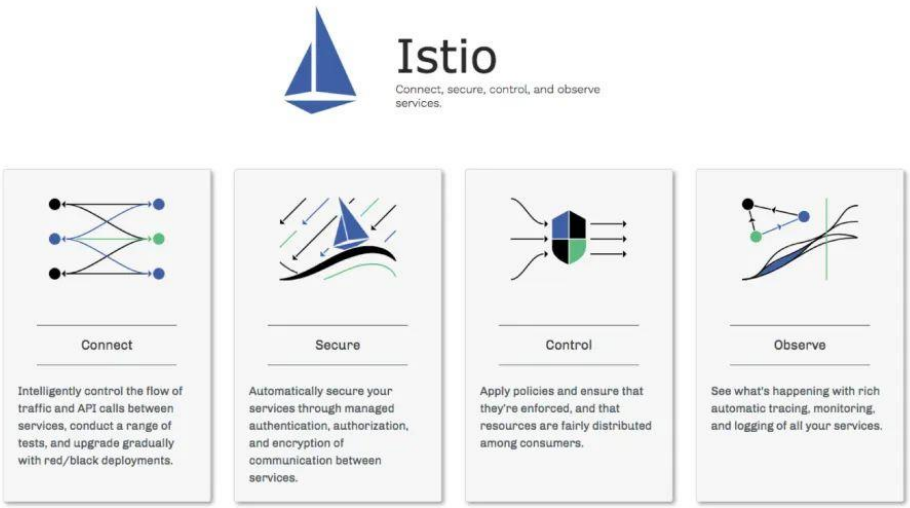
Github 地址：https://github.com/istio/istio/releases

1.1 Istio 是什么？

官当解释：

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。



1、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

2、安全加固（Secure）：自动为服务之间的调用提供认证、授权和加密。

3、控制（Control）：应用用户定义的 policy，保证资源在消费者中公平分配。

4、观察（Observe）：查看服务运行期间的各种数据，比如日志、监控和 tracing，了解服务的运行情况。

Istio 是 ServiceMesh 的产品化落地，可以通过在现有的服务器新增部署边车代理(sidecar proxy)，应用程序不用改代码，或者只需要改很少的代码，就能实现如下基础功能：

1、帮助微服务之间建立连接，帮助研发团队更好的管理与监控微服务，并使得系统架构更加安全；

2、帮助微服务分层解耦，解耦后的 proxy 层能够更加专注于提供基础架构能力，例如：

（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)；

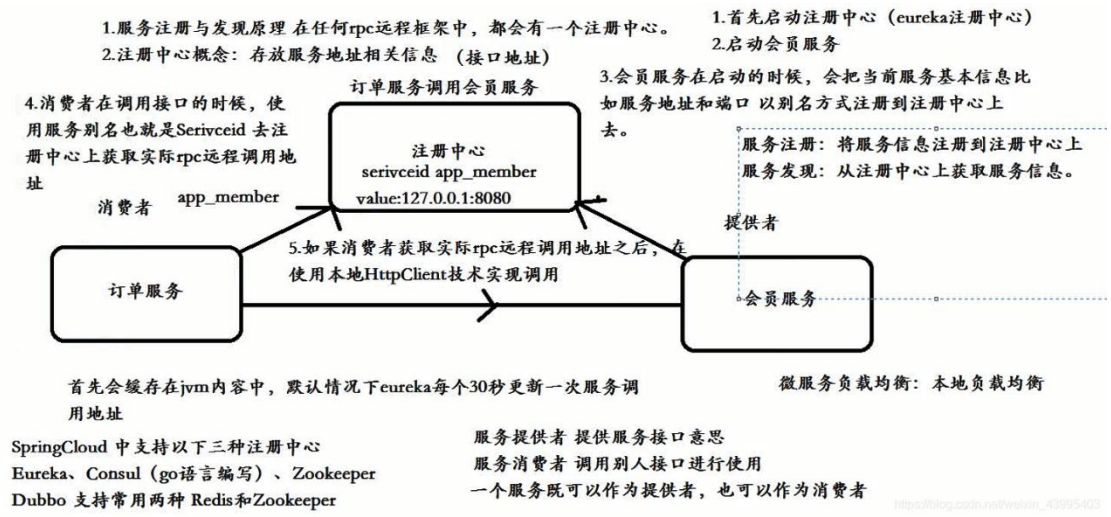
（7）灰度发布(canary rollouts)；

（8）限流限速(rate limiting)；

（9）访问控制(access control)；

（10）身份认证(end-to-end authentication)。

1.1.1 服务注册和发现



RPC：RPC（Remote Procedure Call）远程过程调用，简单的理解是一个节点请求另一个节点提供的服务

1.1.2 负载均衡？

把前端的请求分发到后台多个服务器

1.1.3 故障恢复

出现故障具备自恢复的能力

1.1.4 服务度量

对于 HTTP，HTTP/2 和 GRPC 流量，Istio 生成以下指标：

1、请求计数（istio_requests_total）：这是一个用于累加每个由 Istio 代理所处理请求的 COUNTER 指标。

2、请求持续时间（istio_request_duration_seconds）：这是一个用于测量请求的持续时间的 DISTRIBUTION 指标。

3、请求大小（istio_request_bytes）：这是一个用于测量 HTTP 请求 body 大小的 DISTRIBUTION 指标。

4、响应大小（istio_response_bytes）：这是一个用于测量 HTTP 响应 body 大小的 DISTRIBUTION 指标。

对于 TCP 流量，Istio 生成以下指标：

1、Tcp 发送字节数（istio_tcp_sent_bytes_total）：这是一个用于测量在 TCP 连接下响应期间发送的总字节数的 COUNTER 指标。

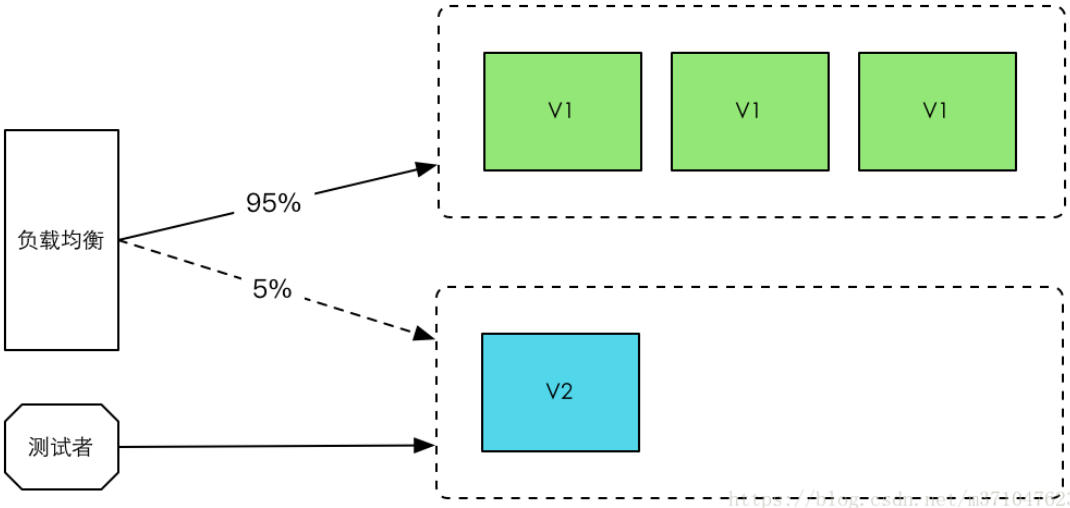
2、Tcp 接收字节数（istio_tcp_received_bytes_total）：这是一个用于测量在 TCP 连接下请求期间接收的总字节数的 COUNTER 指标。

3、Tcp 打开连接数（istio_tcp_connections_opened_total）：这是一个用于累加每个打开连接的 COUNTER 指标。

4、Tcp 关闭连接数（istio_tcp_connections_closed_total）：这是一个用于累加每个关闭连接的 COUNTER 指标。

1.1.5 灰度发布

灰度发布也叫金丝雀发布，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。

1.2 Istio 核心特性

1、流控(traffic management)

断路器(circuit breakers)、超时、重试、多路由规则、AB 测试、灰度发布、按照百分比分配流量等。

2、安全(security)

加密、身份认证、服务到服务的权限控制、K8S 里容器到容器的权限控制等。

3、可观察(observability)

追踪、监控、数据收集，通过控制后台全面了解上行下行流量，服务链路情况，服务运行情况，系统性能情况，国内微服务架构体系，这一块做得比较缺乏。

4、平台无关系(platform support)

K8s，物理机，自己的虚机都没问题。

5、集成与定制(integration and customization)

可定制化扩展功能。

1.2.1 断路器

互动 1：举个生活中的例子解释断路器

当电路发生故障或异常时，伴随着电流不断升高，并且升高的电流有可能能损坏电路中的某些重要器件，也有可能烧毁电路甚至造成火灾。若电路中正确地安置了保险丝，那么保险丝就会在电流异常升高到一定的高度和热度的时候，自身熔断切断电流，从而起到保护电路安全运行的作用。

很多技术都是来源生活的，随着社会进步，科技发展

断路器也称为服务熔断，在多个服务调用的时候，服务 A 依赖服务 B，服务 B 依赖服务 C，如果服务 C 响应时间过长或者不可用，则会让服务 B 占用太多系统资源，而服务 A 也依赖服 B，同时也在占用大量的系统资源，造成系统雪崩的情况出现。 Istio 断路器通过网格中的边车对流量进行拦截判断处理，避免了在代码中侵入控制逻辑，非常方便的就实服务熔断的能力。

在微服务架构中，在高并发情况下，如果请求数量达到一定极限（可以自己设置阈值），超出了设

置的阈值，断路器会自动开启服务保护功能，然后通过服务降级的方式返回一个友好的提示给客户端。假设当 10 个请求中，有 10%失败时，熔断器就会打开，此时再调用此服务，将会直接返回失败，不再调远程服务。直到 10s 钟之后，重新检测该触发条件，判断是否把熔断器关闭，或者继续打开。

互动 2：服务降级（提高用户体验效果）

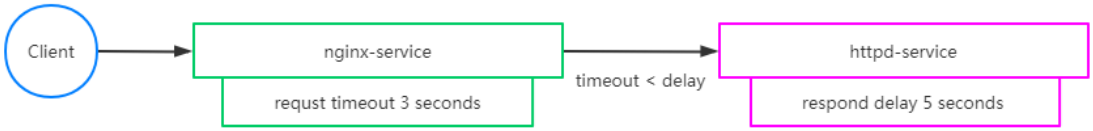
比如电商平台，在针对 618、双 11 的时候会有一些秒杀场景，秒杀的时候请求量大，可能会返回报错标志“当前请求人数多，请稍后重试”等，如果使用服务降级，无法提供服务的时候，消费者会调用降级的操作，返回服务不可用等信息，或者返回提前准备好的静态页面写好的信息。

1.2.2 超时

什么时候需要用到超时？

在生产环境中经常会碰到由于调用方等待下游的响应过长，堆积大量的请求阻塞了自身服务，造成雪崩的情况，通过超时处理来避免由于无限期等待造成的故障，进而增强服务的可用性。

通过例子来理解



nginx 服务设置了超时时间为 3 秒，如果超出这个时间就不在等待，返回超时错误

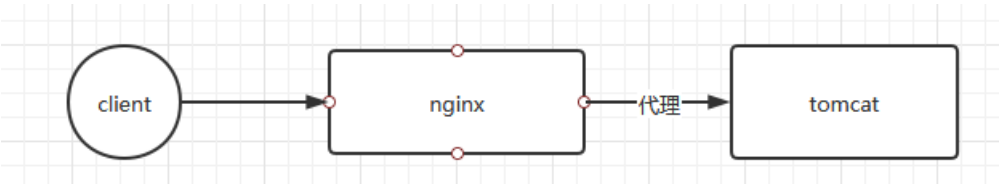
httpd 服务设置了响应时间延迟 5 秒，任何请求都需要等待 5 秒后才能返回

client 通过访问 nginx 服务去反向代理 httpd 服务，由于 httpd 服务需要 5 秒后才能返回，但 nginx 服务只等待 3 秒，所以客户端会提示超时错误。

1.2.3 重试

istio 重试机制就是如果调用服务失败，Envoy 代理尝试连接服务的最大次数。而默认情况下，Envoy 代理在失败后并不会尝试重新连接服务。

举个例子：



客户端调用 nginx，nginx 将请求转发给 tomcat。tomcat 通过故障注入而中止对外服务，nginx 设置如果访问 tomcat 失败则会重试 3 次。

1.2.4 多路由规则

1、HTTP 重定向（HTTPRedirect）

2、HTTP 重写（HTTPRewrite）

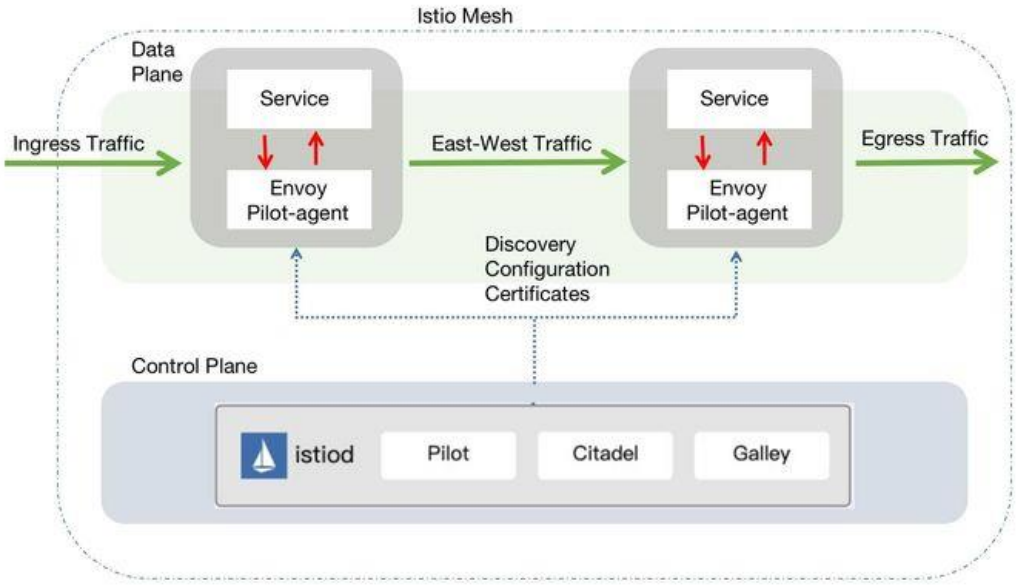
3、HTTP 重试（HTTPRetry）

4、HTTP 故障注入（HTTPFaultInjection）

5、HTTP 跨域资源共享（CorsPolicy）

2. Istio 架构

istio 服务网格从逻辑上分为数据平面和控制平面。



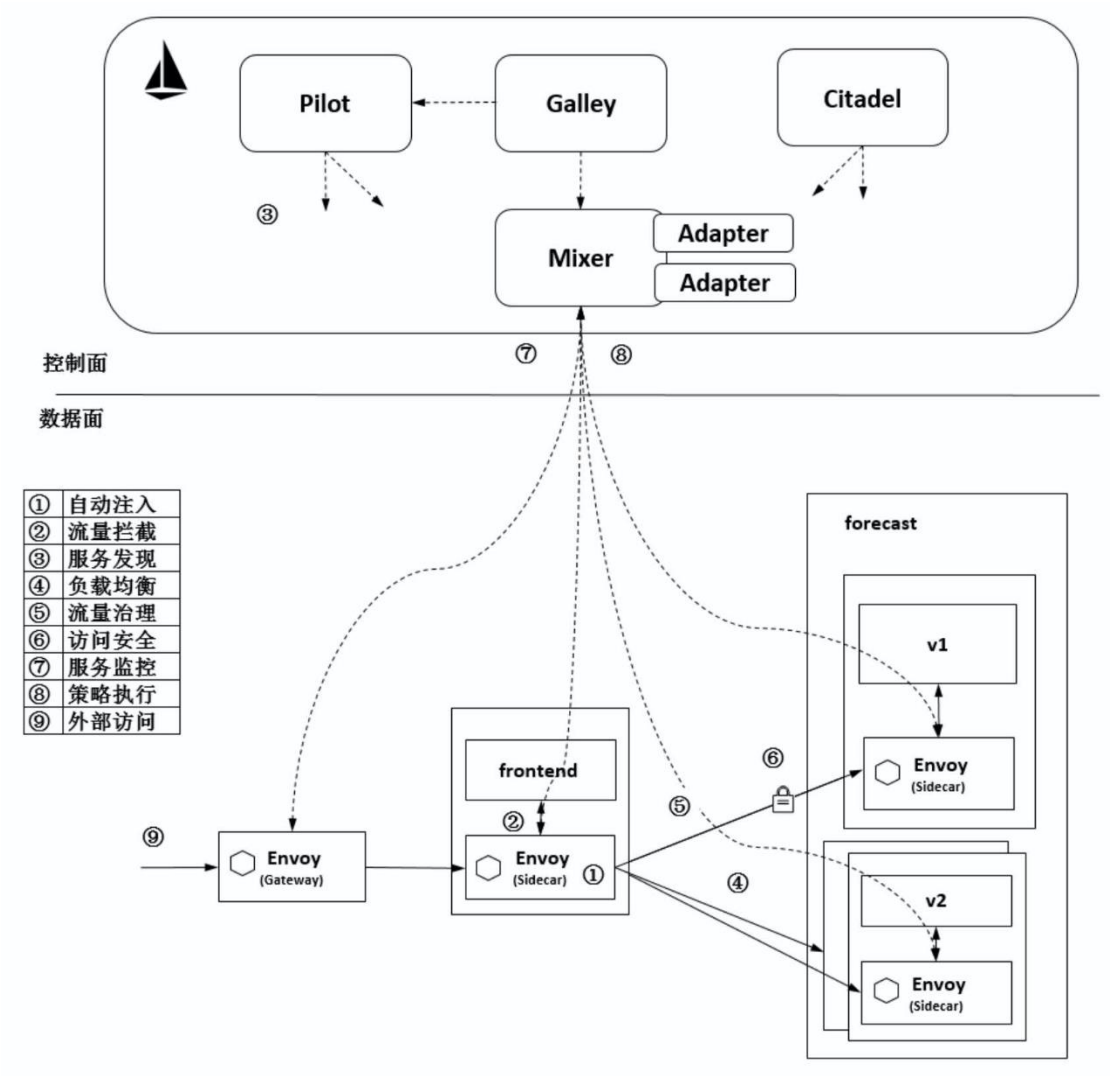
1、数据平面由一组以 Sidecar 方式部署的智能代理(Envoy+Polit-agent)组成。这些代理承载并控制微服务之间的所有网络通信，管理入口和出口流量，类似于一线员工。 Sidecar 一般和业务容器绑定在一起（在 Kubernets 中以自动注入的方式注入到到业务 pod 中），来劫持业务应用容器的流量，并接受控制面组件的控制，同时会向控制面输出日志、跟踪及监控数据。

Envoy 和 pilot-agent 打在同一个镜像中，即 sidecar Proxy。

2、控制平面负责管理和配置代理来路由流量。
istiol.5+中使用了一个全新的部署模式，重建了控制平面，将原有的多个组件整合为一个单体结构 istiod，这个组件是控制平面的核心，管理 Istio 的所有功能，主要包括 Pilot、Mixer、Citadel 等服务组件。

istiod 是新版本中最大的变化，以一个单体组件替代了原有的架构，降低了复杂度和维护难度，但原有的多组件并不是被完全移除，而是在重构后以模块的形式整合在一起组成了 istiod。

结合下图我们来理解 Istio 的各组件的功能及相互之间的协作方式。



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。

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。

问题 1：为什么代理会叫 sidecar proxy?

看了上图就容易懂了，sidecar 和 proxy 相生相伴，就像摩托车(motor)与旁边的车厢(sidecar)。未来，sidecar 和 proxy 就指微服务进程解耦成两个进程之后，提供基础能力的那个代理进程。

3. istio 组件详解

Istio 服务组件有很多，从上面的流程中基本能看出每个组件如何协作的，下面具体讲解每个组件的具体用途和功能。

```
[root@xianchaomaster1 ~]# kubectl get svc -n istio-system |awk '{print $1}'
istio-egressgateway
```

istio-ingressgateway

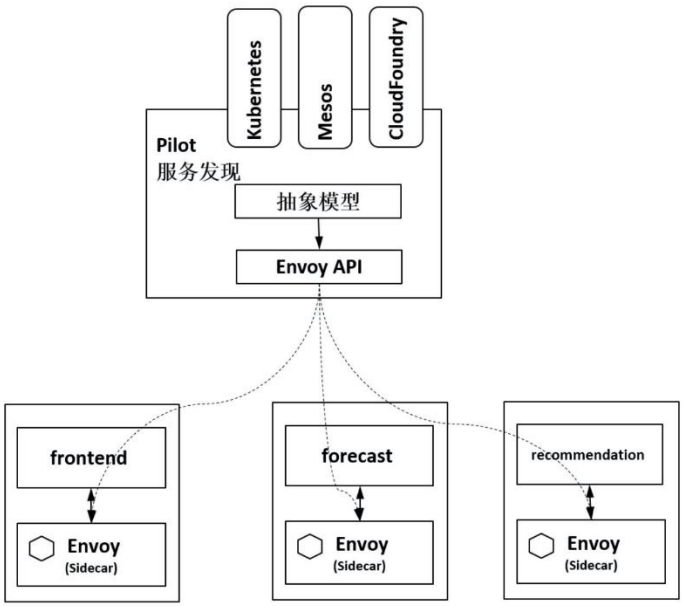
istiod

3.1 Pilot

Pilot 是 Istio 的主要控制组件，下发指令控制客户端。在整个系统中，Pilot 完成以下任务：

1、从 Kubernetes 或者其他平台的注册中心获取服务信息，完成服务发现过程。

2、读取 Istio 的各项控制配置，在进行转换之后，将其发给数据面进行实施。



Pilot 将配置内容下发给数据面的 Envoy，Envoy 根据 Pilot 指令，将路由、服务、监听、集群等定义信息转换为本地配置，完成控制行为的落地。

1）Pilot 为 Envoy 提供服务发现

2）提供流量管理功能（例如，A/B 测试、金丝雀发布等）以及弹性功能（超时、重试、熔断器等）；

3）生成 envoy 配置

4）启动 envoy

5）监控并管理 envoy 的运行状况，比如 envoy 出错时 pilot-agent 负责重启 envoy，或者 envoy 配置变更后 reload envoy

3.2 Envoy

Envoy 是什么？

Envoy 是用 C++ 开发的高性能代理，用于协调服务网格中所有服务的入站和出站流量。

Envoy 有许多强大的功能，例如:

动态服务发现

负载均衡

TLS 终端

HTTP/2 与 gRPC 代理

断路器

健康检查

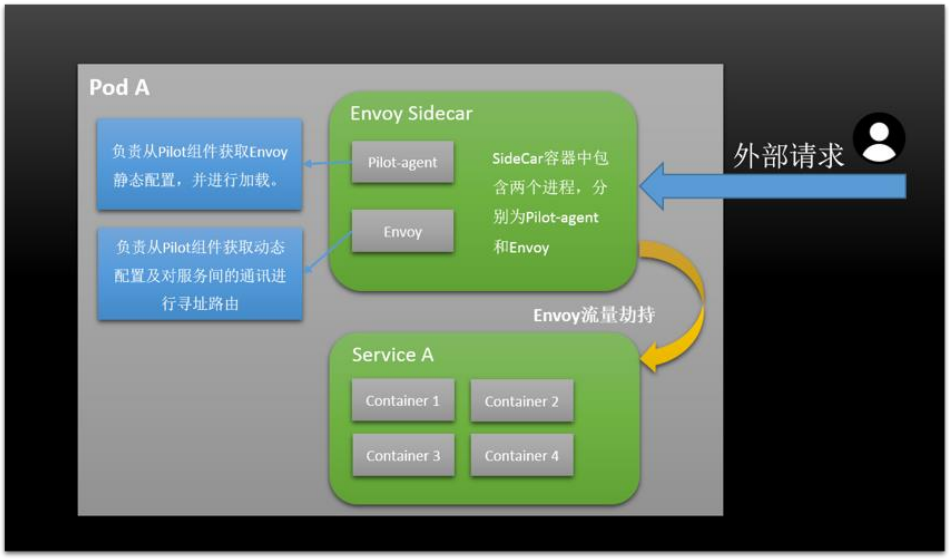
流量拆分

灰度发布

故障注入

Istio 中 Envoy 与服务什么关系？

为了便于理解 Istio 中 Envoy 与服务的关系，下图为 Envoy 的拓扑图，如图所示：



Envoy 和 Service A 同属于一个 Pod，共享网络和命名空间，Envoy 代理进出 Pod A 的流量，并将流量按照外部请求的规则作用于 Service A 中。

Pilot-agent 是什么？

Envoy 不直接跟 k8s 交互，通过 pilot-agent 管理的

Pilot-agent 进程根据 K8S APIserver 中的配置信息生成 Envoy 的配置文件，并负责启动 Envoy 进程。

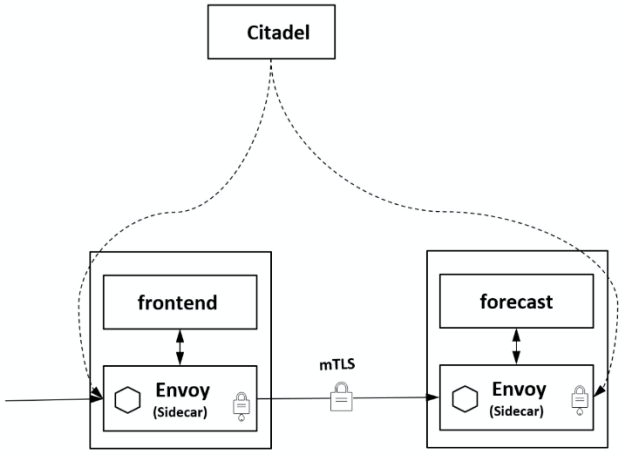
Envoy 由 Pilot-agent 进程启动，启动后，Envoy 读取 Pilot-agent 为它生成的配置文件，然后根据该文件的配置获取到 Pilot 的地址，通过数据面从 pilot 拉取动态配置信息，包括路由（route），监听器（listener），服务集群（cluster）和服务端点（endpoint）。

3.3 Citadel

负责处理系统上不同服务之间的 TLS 通信。 Citadel 充当证书颁发机构(CA)，并生成证书以允许在数据平面中进行安全的 mTLS 通信。

Citadel 是 Istio 的核心安全组件，提供了自动生 成、分发、轮换与撤销密钥和证书功能。

Citadel 一直监听 Kube- apiserver，以 Secret 的形式为每个服务都生成证书密钥，并在 Pod 创建时挂载到 Pod 上，代理容器使用这些文件来做服务身份认证，进而代 理两端服务实现双向 TLS 认证、通道加密、访问授权等安全功能。如图 所示，frontend 服 务对 forecast 服务的访问用到了 HTTP 方式，通过配置即可对服务增加认证功能，双方的 Envoy 会建立双向认证的 TLS 通道，从而在服务间启用双向认证的 HTTPS。



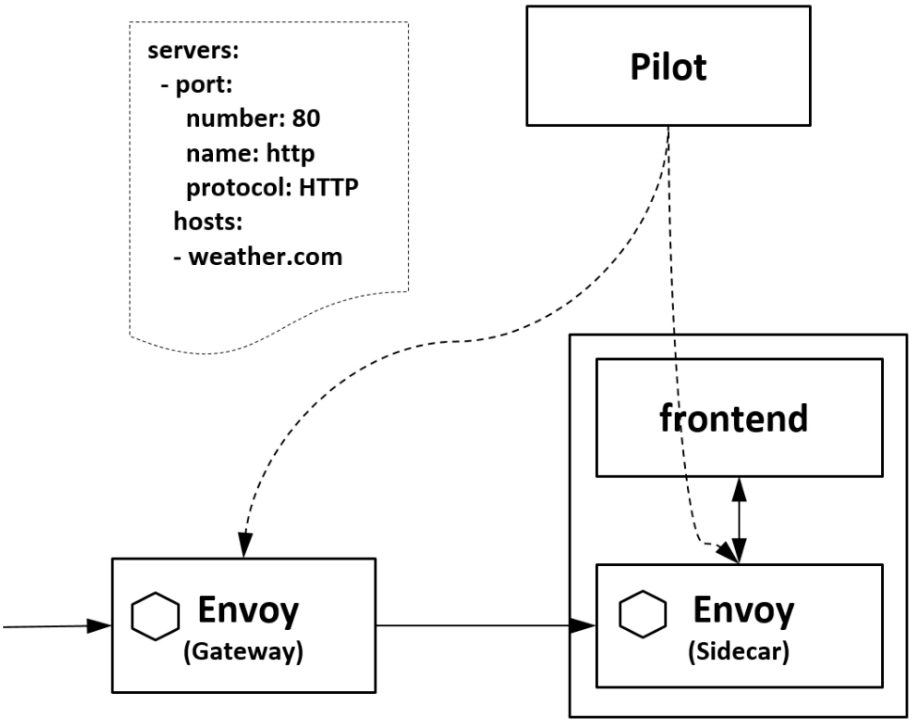
3.4 Galley

Galley 是 istio 的配置验证、提取、处理和分发的组件。Galley 是提供配置管理的服务。实现原理是通过 k8s 提供的 ValidatingWebhook 对配置进行验证。

Galley 使 Istio 可以与 Kubernetes 之外的其他环境一起工作，因为它可以将不同的配置数据转换为 Istio 可以理解的通用格式。

3.5 Ingressgateway

Ingressgateway 就是入口处的 Gateway，从网格外访问网格内的服务就是通过这个 Gateway 进行的。istio-ingressgateway 是一个 Loadbalancer 类型的 Service，不同于其他服务组件只有一两个端 口，istio-ingressgateway 开放了一组端口，这些就是网格内服务的外部访问端口。如下图所示，网格入口网关 istio-ingressgateway 的负载和网格内的 Sidecar 是同样的执行流程，也和网格内的其他 Sidecar 一样从 Pilot 处接收流量规则并执行。



3.6 Sidecar-injector

Sidecar-injector 是负责自动注入的组件，只要开启了自动注 入，在 Pod 创建时就会自动调用 istio-sidecar-injector 向 Pod 中注入 Sidecar 容器。

在 Kubernetes 环境下，根据自动注入配置，Kube-apiserver 在拦截到 Pod 创建的请求时，会调用自动注入服务 istio-sidecar-injector 生成 Sidecar 容器的描述并将其插入原 Pod 的定义中，这样，在创建的 Pod 内除了包括业务容器，还包括 Sidecar 容器，这个注入过程对用户透明。

3.7 其他组件

除了以“istio”为前缀的 Istio 自有组件，在集群中一般还安装 Jaeger-agent、Jaeger-collector、Jaeger-query、Kiali、Prometheus、Grafana、 Tracing、Zipkin 等组件，这些组件提供了 Istio 的调用链、监控等功能，可以选择安装来完成完整的服务监控管理功能。

5、k8s1.23 及以上版本安装 istio 新版微服务

11.1 准备安装 Istio 是要的压缩包

官网下载地址：

https://github.com/istio/istio/

官方访问相对较慢，我在课件提供了压缩包，大家最好用我的压缩包，这样做实验才不会出问题

1、把压缩包上传到 k8s 的控制节点 xianchaomaster1。手动解压：

[root@xianchaomaster1 ~]# tar zxvf istio-1.13.1.tar.gz

2、切换到 istio 包所在目录下。tar zxvf istio-1.13.1.tar.gz 解压的软件包包名是 istio-1.13.1，则：

cd istio-1.13.1

安装目录包含如下内容：

2）samples/目录下，有示例应用程序

3）bin/目录下，包含 istioctl 的客户端文件。istioctl 工具用于手动注入 Envoy sidecar 代理。

3、将 istioctl 客户端路径增加到 path 环境变量中，macOS 或 Linux 系统的增加方式如下：

export PATH=$PWD/bin:$PATH

4、把 istioctl 这个可执行文件拷贝到/usr/bin/目录

cd /root/istio-1.13.1/bin/

cp -ar istioctl /usr/bin/

11.2 安装 istio

1.下载镜像：

安装 istio 需要的镜像默认从官网拉取，但是官网的镜像我们拉取会有问题，可以从课件下载镜像，然后上传到自己 k8s 集群的各个节点，通过 docker load -i 手动解压镜像：

docker load -i examples-bookinfo-details.tar.gz
docker load -i examples-bookinfo-reviews-v1.tar.gz
docker load -i examples-bookinfo-productpage.tar.gz
docker load -i examples-bookinfo-reviews-v2.tar.gz
docker load -i examples-bookinfo-ratings.tar.gz
docker load -i examples-bookinfo-reviews-v3.tar.gz
docker load -i pilot.tar.gz
docker load -i proxyv2.tar.gz
docker load -i httpbin.tar.gz

2.安装

在 k8s 的控制节点 xianchaomaster1 操作

istioctl install --set profile=demo -y

看到如下，说明 istio 初始化完成：

```
Detected that your cluster does not support third party JWT authentication. Falling
back to less secure first party JWT. See https://istio.io/docs/ops/best-
practices/security/#configure-third-party-service-account-tokens for details.
- Applying manifest for component Base...
✔ Finished applying manifest for component Base.
- Applying manifest for component Pilot...
✔ Finished applying manifest for component Pilot.
  Waiting for resources to become ready...
  Waiting for resources to become ready...
- Applying manifest for component EgressGateways...
- Applying manifest for component IngressGateways...
- Applying manifest for component AddonComponents...
✔ Finished applying manifest for component EgressGateways.
✔ Finished applying manifest for component IngressGateways.
✔ Finished applying manifest for component AddonComponents.
```

✔ Installation complete

3.验证 istio 是否部署成功

kubectl get pods -n istio-system

显示如下，说明部署成功

```
istio-egressgateway-d84f95b69-5gtdc      1/1     Running   0          15h
istio-ingressgateway-75f6d79f48-fhxjj    1/1     Running   0          15h
istiod-c9f6864c4-nrm82                   1/1     Running   0          15h
```

4.卸载 istio 集群，暂时不执行，记住这个命令即可

istioctl manifest generate --set profile=demo | kubectl delete -f -

6、通过 Istio 最新版部署在线书店 bookinfo

12.1 在线书店功能介绍

在线书店-bookinfo

该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。

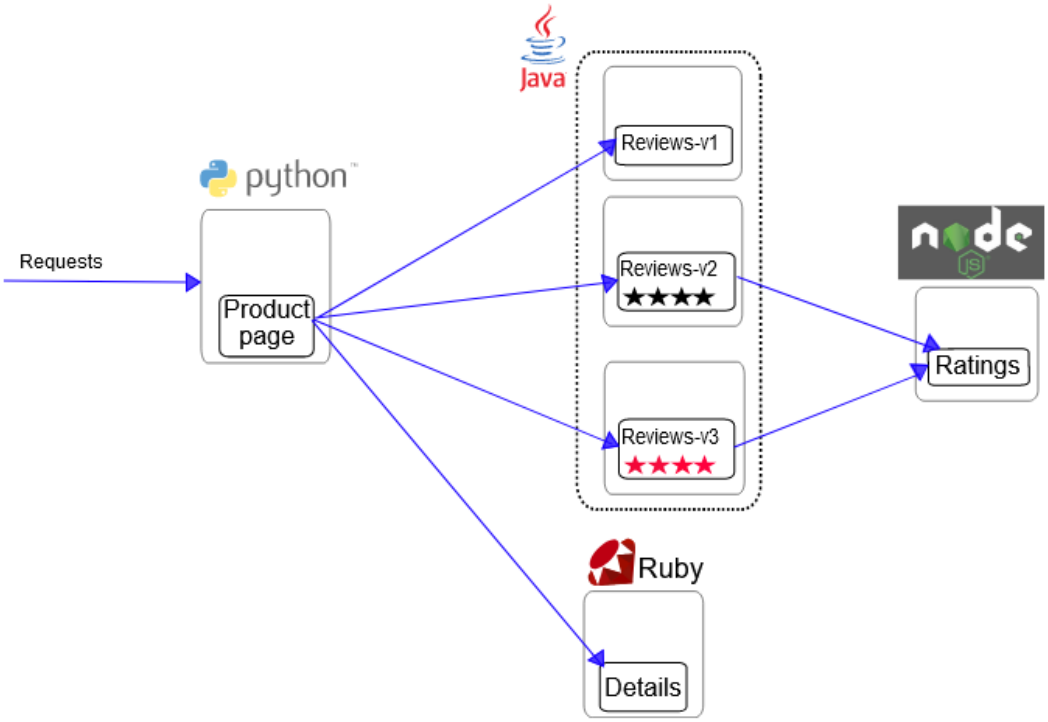
reviews 微服务有 3 个版本

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

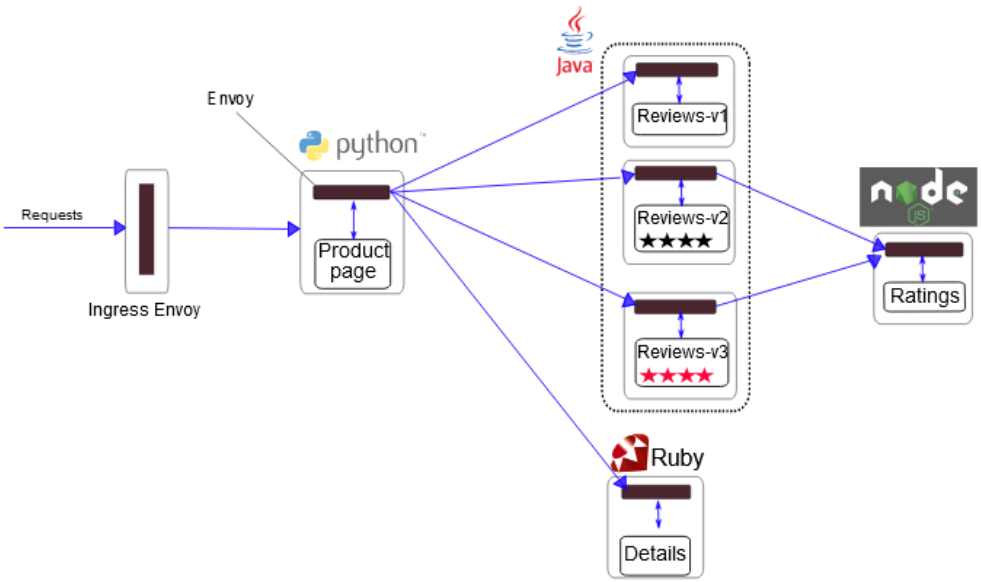


Bookinfo Application without Istio

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。

12.2 部署应用

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。

12.3 启动应用服务

1.进入 istio 安装目录。

2.istio 默认自动注入 sidecar，需要为 default 命名空间打上标签 istio-injection=enabled

kubectl label namespace default istio-injection=enabled

3.使用 kubectl 部署应用

cd istio-1.13.1

vim samples/bookinfo/platform/kube/bookinfo.yaml

修改镜像：

```
spec:
  serviceAccountName: bookinfo-productpage
  containers:
  - name: productpage
    image: istio/examples-bookinfo-productpage-v1:1.16.2
```

istio/examples-bookinfo-productpage-v1:1.16.2

kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml

上面的命令会启动全部的四个服务，其中也包括了 reviews 服务的三个版本（v1、v2 以及 v3）。

4.确认所有的服务和 Pod 都已经正确的定义和启动：

kubectl get services

显示如下

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
details	ClusterIP	10.109.124.202	<none>	9080/TCP
productpage	ClusterIP	10.102.89.129	<none>	9080/TCP
ratings	ClusterIP	10.101.97.75	<none>	9080/TCP
reviews	ClusterIP	10.100.105.33	<none>	9080/TCP

kubectl get pods

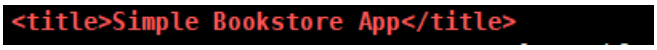
显示如下

NAME	READY	STATUS	RESTARTS	AGE
details-v1-78d78fbddf-qssjb	2/2	Running	0	73m
productpage-v1-85b9bf9cd7-r699f	2/2	Running	0	73m
ratings-v1-6c9dbf6b45-77kv7	2/2	Running	0	73m
reviews-v1-564b97f875-2jtxq	2/2	Running	0	73m
reviews-v2-568c7c9d8f-f5css	2/2	Running	0	73m
reviews-v3-67b4988599-fxfzx	2/2	Running	0	73m
tomcat-deploy-59664bcb6f-5z4nn	1/1	Running	0	22h
tomcat-deploy-59664bcb6f-cgjbn	1/1	Running	0	22h
tomcat-deploy-59664bcb6f-n4tqq	1/1	Running	0	22h

5.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：

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>"

显示如下：



6.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：

kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml

2）确认网关创建完成：

kubectl get gateway

显示如下：

NAME AGE

bookinfo-gateway 2m18s

3）确定 ingress ip 和端口

执行如下指令，明确自身 Kubernetes 集群环境支持外部负载均衡：

kubectl get svc istio-ingressgateway -n istio-system

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。

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

export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')

export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')

4）设置 GATEWAY_URL

INGRESS_HOST=192.168.40.180

#192.168.40.180 是安装 istio 的机器，即 k8s 控制节点 xianchaomaster1 的 ip

export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT

echo $GATEWAY_URL 显示如下：

192.168.40.180:30871

确认可以从集群外部访问应用

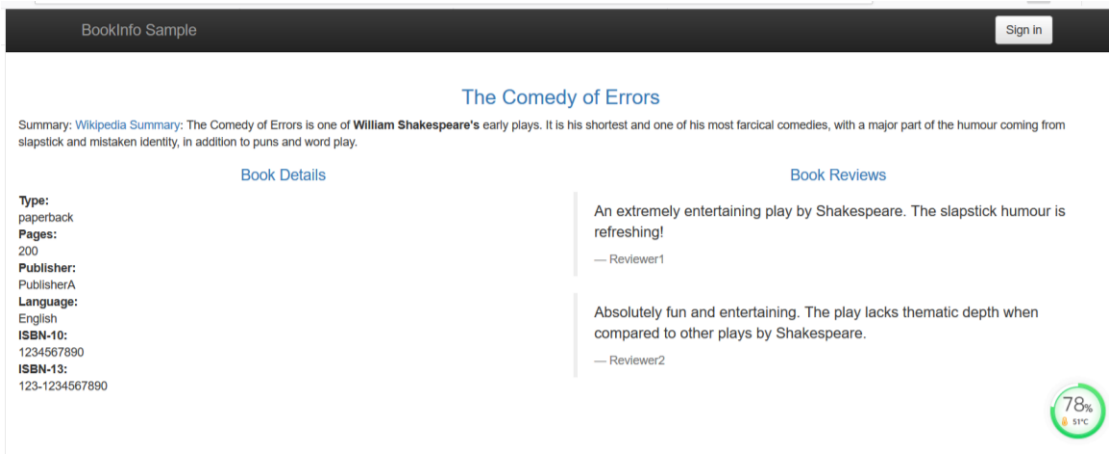
可以用 curl 命令来确认是否能够从集群外部访问 Bookinfo 应用程序：

curl -s http://${GATEWAY_URL}/productpage | grep -o "<title>.*</title>"

显示如下：

```
[root@k8s-master istio-1.5.1]# curl -s http://${GATEWAY_URL}/productpage | grep -o "<title>.
*</title>"
<title>Simple Bookstore App</title>
```

还可以用浏览器打开网址 http://$GATEWAY_URL/productpage，也就是 192.168.40.180:30871/productpage 来浏览应用的 Web 页面。如果刷新几次应用的页面，就会看到 productpage 页面中会随机展示 reviews 服务的不同版本的效果（红色、黑色的星形或者没有显示）。



通过 istio 的 ingressgateway 访问，官网：

https://istio.io/docs/examples/bookinfo/#determine-the-ingress-ip-and-port