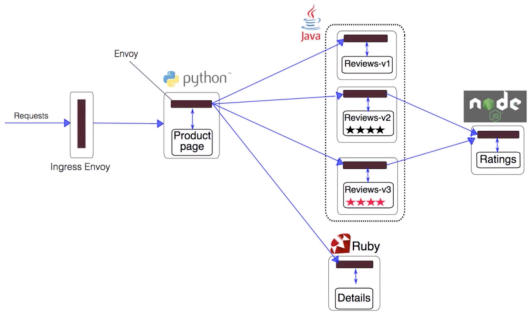
istio灰度发布测试

Bookinfo 是官方提供一个图书评测系统微服务项目示例，
分为四个微服务：

服务	说明	调用服务
productpage	主页	reviews、details
reviews	评论内容	ratings
details	详细内容	
ratings	评分	



1、创建命名空间并开启自动注入

kubectl create ns bookinfo

kubectl label namespace bookinfo istio-injection=enabled

2、部署应用YAML

cd istio-1.8.3/samples/bookinfo

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

kubectl get pod -n bookinfo

3、创建Ingress网关

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

4、确认网关和访问地址，访问应用页面

kubectl port-forward services/istio-ingressgateway 10002:80 -n istio-system

访问地址：http://127.0.0.1:10002/productpage

测试路由

任务：

1. 流量全部发送到reviews v1版本（不带五角星）

```
kubectl apply -f networking/virtual-service-all-v1.yaml

kubectl apply -f networking/destination-rule-all.yaml
```

2. 将80%的流量发送到reviews v1版本，另外20%的流量发送到reviews v2版本（5个黑色五角星），

最后完全切换到v2版本

```
kubectl apply -f networking/virtual-service-reviews-80-20.yaml
```

3. 将50%的流量发送到v2版本，另外50%的流量发送到v3版本（5个红色五角星）

```
kubectl apply -f networking/virtual-service-reviews-v2-v3.yaml
```

任务：将特定用户的请求发送到reviews v2版本（5个黑色五角星），其他用户则不受影响（v3） 这种配置为A/B测试，基于不同用户来进行灰度。

```
kubectl apply -f networking/virtual-service-reviews-jason-v2-
v3.yaml

#####
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user: # end-user是程序中带的请求头
          exact: jason #这个在网页右上角login输入用户为jason，密码随意就
能访问到v2,默认到v3
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
```

istio故障注入（ HTTP 延迟故障注入）

超时故障注入

我们将为用户 `jason` 在 `reviews:v2` 和 `ratings` 服务之间注入一个 7 秒的延迟。 这个测试将会发现一个故意引入 Bookinfo 应用程序中的 bug,他会显示一个无法加载到内容

`reviews:v2` 程序对 `ratings` 服务的调用具有 10 秒的硬编码连接超时。如果是我们线上环境的话，我们尽管引入了 7 秒的延迟，我们仍然在7秒后还能正常访问到评论内容，而不是现实无法加载。

再举个例子，有个程序要去连接数据库，如果在此期间服务器出现了网络问题，中断了10秒钟，他如果连接了一次没有连接到他就不工作了，没有写重试的机制，这个时候我们就可以通过故障注入检测到这个bug，因为这种bug可能正常测试是无法测试出来的，然后istio故障注入就完美的解决了这个问题，减少线上环境这些未知的风险。

故障注入对整个应用的健壮性进行测试起到了决定性作用，发现系统隐藏的bug，可以对各种预想场景进行测试，测试人员可以很方便地模拟微服务之间的各种通信故障，极大的减少线上无法预知的事故。

```
# kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      delay:
        percentage:
          value: 100.0 #延迟将被注入的请求的百分比，比如我每次访问都是否延迟7秒，如果改成10就是访问10次才出现一次。
        fixedDelay: 7s
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

abort故障

还有一种abort故障注入，如果我们在访问程序时候出现了500，502， 404，等错误的时候我们希望程序会调转到一个特定的错误页面，比如双11的时候我们淘宝下单的时候，人数太多的话程序就崩溃了，返回一个空白页，这个空白页并不是特别友好，我们想给用户返回一个友好的提示比如：商品太过火爆，请稍等！ 这个时候通过abort故障注入就能测试出这个程序是否会跳转到提示页面

执行下面文件：下面httpStatus中如果错误是500的时候，程序就会返回评论无法加载，如果改成200的话就会显示抱歉，该书目前无法 提供评论。

```
# kubectl apply -f samples/bookinfo/networking/virtual-service-
ratings-test-abort.yaml

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      abort:
        percentage:
          value: 100
        httpStatus: 500
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

istio请求超时配置

一般我们开发写程序的时候调用一些服务的时候，都会设置连接超时的部分,istio网格中也可以帮助我们做到。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      abort:
        percentage:
          value: 100
        httpStatus: 200
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
    timeout: 0.001s #在这里增加访问调用评分服务时，超时时间为0.001s秒，这样的话如果其他服务调用ratings服务的的话如果0.001s没有响应就会在页面上面就是显示Ratings无法访问（Ratings），线上环境开发根据程序来自行设置超时时间
```

istio熔断限流

一个程序服务在固定某个时间段会有大量访问量时候，也是就是pv 峰值即将超过服务能承载的量时，我们可能会考虑熔断限流来进行控制，防止服务器崩溃，如果有istio的话就免去了开发人员来写熔断限流功能了，一切由istio控制。

#在productpage 配置限制熔断限流规则，

```
#这个需要自己创建
#kubectl apply -f samples/bookinfo/networking/productpage-
rule.yaml


apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: productpage
spec:
  host: productpage
  subsets:
  - name: v1
    labels:
      version: v1
    trafficPolicy:  #流量策略字段 (限流功能)
      connectionPool:  #设置控制服务的连接池
        tcp:
          maxConnections: 1 #到后端的最大HTTP1 / TCP连接数
        http:
          http1MaxPendingRequests: 1 #到后端请求的最大排队数量
          maxRequestsPerConnection: 1 #到后端的每个连接的最大请求数
      outlierDetection: #用于控制从负载平衡池中驱逐不正常主机的设置 (熔断
功能)
        consecutiveErrors: 1 #当通过HTTP访问时，返回代码是502、503或
504则视为错误。当访问不透明的TCP连接时，连接超时和连接错误/失败也会都视为错
误。即将实例从负载均衡池中剔除，需要连续的错误（HTTP5XX或者TCP断开/超时）次
数。默认是5。
        interval: 1s #获取检查不正常主机的间隔,即在interval（1s）内连续
发生1个consecutiveErrors错误，则触发服务熔断，格式是1h/1m/1s/1ms，但必须
大于等于1ms。即分析是否需要剔除的频率，多久分析一次，默认10秒。
        baseEjectionTime: 3m #和maxEjectionPercent结合使用,代表着移除
的主机如果想要再次被访问到需要3分钟等待时间,默认30秒
        maxEjectionPercent: 100 #负载均衡池中可以移除的服务最大主机百分
比。默认为10%

---

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
```

```
  name: productpage
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1
        port:
          number: 9080
```

然后启动一个fortio容器（Fortio 是golang开发的一个服务端请求测试工具，模拟客户端请求,其可以控制连接数、并发数及发送 HTTP 请求的延迟。通过 Fortio 能够有效的触发前面在 `DestinationRule` 中设置的熔断策略）

```
#部署到图书馆命名空间
kubectl apply -f samples/httpbin/sample-client/fortio-deploy.yaml


#测试(发送并发数为 5 的连接（-c 2），请求 20 次（-n 20）：)
kubectl exec "$FORTIO_POD" -c fortio -- /usr/bin/fortio load -c 5
-qps 0 -n 20 -loglevel Warning http://productpage:9080

#现在，您将开始看到预期的熔断行为，只有 63.3% 的请求成功，其余的均被熔断器拦截：
Code 200 : 19 (63.3 %)
Code 503 : 11 (36.7 %)
```

```
#之后删除熔断规则
kubectl delete -f samples/bookinfo/networking/productpage-
rule.yaml

#再次测试
kubectl exec "$FORTIO_POD" -c fortio -- /usr/bin/fortio load -c 5
-qps 0 -n 20 -loglevel Warning http://productpage:9080

#现在，您将开始看到预期的熔断行为，请求就能全部成功了
Code 200 : 20 (100.0 %)
```

流量镜像

istio还有一个流量镜像功能，能将用户的请求复制一份，根据策略来处理这个请求，可以根据复制的流量进行线上文件的排查，用真实的流量验证功能是否正常，收集真实流量进行分析。这样呢，就不需要我们手动造一些假的流量进行分析了。

在samples下建立nginx-mirror文件夹创建3个文件：

```
#创建nginx v1配置文件
kubectl create deploy nginx --image=nginx --dry-run -o yaml >>
nginx.yaml
#然后在matchLabels下增加version: v1 和 template.metadata.labels下增加
version: v1

#之后cp nginx.yaml 为nginx2.yaml
#然后在matchLabels下增加version: v2 和 template.metadata.labels下增加
version: v2，然后name改名为nginx2

#生成2个deploy
kubectl apply -f .


#创建svc
 kubectl expose deploy nginx --port=80 --target-port=80 --dry-run
-o yaml >> nginx-svc.yaml
 #然后去除selector下的version: v1

#生成svc
 kubectl apply -f nginx-svc.yaml
```

```
#当前目录创建VirtualService.yaml文件
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
```

```
metadata:
  name: nginx
spec:
  host: nginx
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2

---


apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: nginx
spec:
  hosts:
  - nginx
  http:
  - route:
    - destination:
        host: nginx
        subset: v1
      weight: 100 #这里代表着所有流量都由nginx v1 接收
    mirror:  #这里代表着从nginx v1 接收的流量复制给nginx v2
      host: nginx
      subset: v2
    mirror_percent: 100 #这里代表着从nginx v1 接收的流量的百分之100都会进行复制给v2，如果是写10的话就是百分之10
```

```
#同时打开nginx和nginx2容器查看实时日志
#最后进行验证：   进入一个能用curl 命令的容器中执行（这里我们直接进入nginx容器）
 curl  nginx

#我们这个时候就可以在nginx和nginx2实时日志中看到同时会有刚才我们的请求
```

kiali使用

```
#来到istio自带的addons目录下，这个目录下有写好的istio可观测组件的配置
cd istio-1.8.3/samples/addons && ls
README.md       grafana.yaml    kiali.yaml
extras          jaeger.yaml     prometheus.yaml

#然后部署(因为里面有CRD类，如果报错找不到的话，多执行apply部署一下)
kubectl apply -f .


#查看kiali
kubectl port-forward services/kiali 10002:20001 -n istio-system
```

Kiali是一款Isito服务网格可视化工具,可以观测到当前istio配置规则，已经健康状态

Overview：显示着各个命名空间的状况。

Graph：显示着有使用这istio网格的调用图，这样可以看出哪些服务之间进行的调用关系。

Workloads：显示着istio网格中服务详情配置的情况。

Service: 显示着服务向关联着的后端pod及规则情况。

Istio Config: 显示着istio资源类的yaml配置文件，在上面可以直接更改。

#上面这些都是相互关联的，可以点击Graph上的图标调转到Workloads然后可以跳到Service然后可以跳到Istio Config，很简单。

prometheus查看

Prometheus用于收集Isito指标，通过Grafana可视化展示。

仪表盘：

• Istio Control Plane Dashboard：控制面板仪表盘

• Istio Mesh Dashboard：网格仪表盘，查看应用（服务）数据 （通过多次请求可以看到请求的成功率，如果低于百分百就证明服务出现bug了）

• Istio Performance Dashboard：查看Istio 自身（各组件）数据

• Istio Service Dashboard：服务仪表盘

• Istio Workload Dashboard：工作负载仪表盘

• Istio Wasm Extension Dashboard

jaeger使用

---

kubectl port-forward services/tracing 10002:80 -n istio-system

通过多次请求可以看到请求，productpage.istio-book就查询出来分析网络是否有问题，但是skywarking东西也是很全的，如果同学们追求简单方便的链路监控的话，也可以使用jaeger。