**演示示例**

在向您展示Istio流量管理魔术之前，我们将介绍第一个非常简单的Spring Boot MVC应用程序，我们将在两个版本中部署到kubernetes。

版本1:

@RestController
public class ControllerV1 {
@GetMapping(path = "/service")
public String getResult() {
return "Hello I'm V1!";
}
}

下面是这个版本的k8s的部署，标记mvc-service开始部分：v1-deploy.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
name: mvc-service
spec:
replicas: 1
selector:
matchLabels:
app: mvc-service
template:
metadata:
labels:
app: mvc-service
version: v1
spec:
containers:
- name: mvc-service
image: 192.168.18.25/k8s/istio-demo:v1
imagePullPolicy: IfNotPresent
ports:
- containerPort: 8080

版本2：

@RestController
public class ControllerV2 {
@GetMapping(path = "/service")
public String getResult() {
return "Hello i'm V2!";
}
}

v2-deploy.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
name: mvc-service-v2
spec:
replicas: 1
selector:
matchLabels:
app: mvc-service
template:
metadata:
labels:
app: mvc-service
version: v2
spec:
containers:
- name: mvc-service
image: 192.168.18.25/k8s/istio-demo:v2
imagePullPolicy: IfNotPresent
ports:
- containerPort: 8080

我们在标有“mvc-service”字符串的pod中有两个版本的应用程序。因此kubernetes服务还应该使用标签’mvc-service’来定位pod后端：

v1-deploy.yaml

apiVersion: v1
kind: Service
metadata:
name: mvc-service
labels:
app: mvc-service
spec:
type: NodePort
ports:
- port: 8080
name: http
selector:
app: mvc-service

**使用Istio支持部署到Kubernetes**

好的，我们已经准备好了kubernetes清单，但还没有部署任何东西！为了能够使用Istio流量管理，您需要向您的pod 注入sidecar代理。如果没有istio边车，你就不会形成服务网格。你有两个选择：

通过istioctl 手动注入边车

自动边车注射

由于安装 istio 的时候我们已经选择采用自动边车注入方式，所以这里不需要做改动。

部署程序：

kubectl apply -f v1-deploy.yaml
kubectl apply -f v1-deploy.yaml

查看部署后的输出（确保READY为 2/2，每个POD都有两个容器，因为它运行app容器和istio代理边车容器。）：

kubectl get pod

**Istio形成服务网格**

DestinationRule

destination-rule-all.yaml

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
name: mvc-service
spec:
host: mvc-service
subsets:
- name: v1
labels:
version: v1
- name: v2
labels:
version: v2
---

通过kubectl部署它

kubectl apply -f istio-destionation-rule.yaml

VirtualService

VirtualService定义了一组要在主机被寻址时应用的流量路由规则。每个路由规则定义特定协议的流量的匹配标准。如果流量匹配，则将其发送到注册表中定义的命名目标服务（或其子集 /版本）。

换句话说，您首先部署K8s部署和服务。然后，通过Istio DestionRule定义微服务的网络，然后通过VirtualService设置HTTP路由规则。

**用Istio进行金丝雀测试**

因此，假设我们的演示SpringMVC应用程序的第二个版本（在服务网格中，子集v2）不够稳定，无法处理满负载，因此我们只将20％的流量路由到它。

# 用Istio进行金丝雀测试
# 假设我们的演示SpringMVC应用程序的第二个版本（在服务网格中，子集v2）不够稳定，无法处理满负载，因此我们只将20％的流量路由到它。
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
name: service-gateway
spec:
hosts:
- "\*"
gateways:
- service-gateway
http:
- match:
- uri:
exact: /service
route:
- destination:
host: mvc-service
subset: v1
weight: 80
- destination:
host: mvc-service
subset: v2
weight: 20

作为最后一步，我们需要公开服务网关，即Istio-Ingress网关 ，它从服务网格外部获取流量并将其转发到该网关。

apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
name: service-gateway
spec:
selector:
istio: ingressgateway # use istio default controller
servers:
- port:
number: 80
name: http
protocol: HTTP
hosts:
- "\*"
---

部署：

kubectl apply -f mvc-service-gateway.yaml

测试：

kubectl get service istio-ingressgateway -n istio-system

10个点击中：2个进入V2版本，8个点击进入V1。

curl http://192.168.99.110:31380/service

**使用Istio进行蓝/绿部署**

现在让我们说我们的应用程序的V2版本足够稳定，我们可以将100％的流量路由到它。要使用Istio实现这一点，我们将更改 VirtualService 中的规则：

将 mvc-service-gateway.yaml 文件中的 VirtualService 修改为：

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
name: service-gateway
spec:
hosts:
- "\*"
gateways:
- service-gateway
http:
- match:
- uri:
exact: /service
route:
- destination:
host: mvc-service
subset: v2

重新部署：

kubectl apply -f mvc-service-gateway.yaml

测试：

curl http://192.168.99.110:31380/service

现在V2版本应该成为我们的生产版本并处理100％的流量。