ackstorm
Managed Cloud Solutions

Istio demo

La demo consiste de 3 deployments (blue, green y yellow). Todos sirven un cuadrado de su color, tal que así.

Green sirve en "/green", blue y yellow sirven en "/".

Yellow iba a ser el canary, recibiendo un 10% del tráfico. Blue, el 90%. Y el green, el 100% de "/green".

Nota: En Kubernetes, sin Istio, para conseguir esto, tendríamos que tener:
- 3 deployments.
- 2 servicios; de los cualaes 1 estaría redirigiendo el tráfico hacía el blue y yellow, y el otro hacía green.
- 1 Ingress. Para conseguir enviar el tráfico hacía el green (por "/green"), habría que crear un Ingress, ya que los servicios son de capa 4 y no tienen conocimiento de host o path.
- Para conseguir un 90-10, hay que tener el número de pods, que aritméticamente nos permita tener el 90-10. Es decir, tendría que crear 9 réplicas del blue y 1 réplica del yellow, y atacar con el mismo servicio.

Con Istio, se pueda hacer todo esto con:
- 3 deployments (1 réplica de cada deployment)
- 1 servicio
- 3 componentes de Istio (Gateway, VirtualService y DestinationRule).

- Gateway: Define un puerto de entrada (o varios), para un host espécifico (en nuestro caso es un "*") hacía el mesh. Tambien gestiona certificados TLS. El certificado se guarda en el pod de istio-ingressgateway, y obviamente se puede definir un certificado por cada host.
- VirtualService. Hay un par de formas de utilizar un VirtualService. En nuestro caso lo enganchamos al gateway, para que sepa que tiene que enrutar tráfico que provenga del gateway definido (el decir por el puerto definido y para el host especificado). También es aquí donde se define el peso que va a tener cada ruta (en este caso 90 para el blue y 10 para el yellow).
- DestinationRule: Los DestinationRules son reglas que se aplican a un paquete, una vez este ha sido redirigido hacía un servicio.

Nota: Si miráis los deployments, veréis que tienen 2 labels ("app: onboard" y "version: blue/green/yellow"), pero el servicio solo tiene un selector (app: onboard). Aquí, es gracias al DestinatioRule que el tráfico se redirige hacía el backend adecuedo. En nuestro caso hemos definido los sub-servicios que hay detrás dell servicio principal (los subsets).

Dicho todo esto, esto es lo que hay que hacer para la demo (asumo que tenéis istio instalado y sabéis como hacer "kube-inject" o tenéis un namespace con el label "istio-injection=enabled").

- Crear los 3 deployments, y el servicio (fichero app)
- Crear el gateway.
- Crear el virtual service y destination rule.

Nota: A veces pasa que si deployamos los archivos en orden equivocada, la configuración falla. Por que se configuran los proxies Envoy sobre la marcha. Esto lo podéis ver mirando los logs del servicio "istio-ingressgateway" mientras créais los objetos (kubectl logs istio-ingressgateway-XXX-XXX -n istio-system).
Para esto si lo dejáis un rato, suele funcionar a los 10 minutos o algo así, pero yo no tengo tanta paciencia, así que mato a pod del ingress-gateway y al recrearse, se recrea con la configuración correcta.

Y ya está. Ahora se puede o ir al navegador (el punto de entrada hacía el mesh/cluster es la IP del "istio-ingressgateway") o hacer un curl.

```
$ kubectl get svc -n istio-system | grep istio-ingressgateway
> istio-ingressgateway     LoadBalancer   10.3.245.9     35.204.3.92   80:31380/TCP,443:31390/TCP,31400:31400/TCP,15011:31361/TCP,8060:30372/TCP,853:30576/
TCP,15030:30558/TCP,15031:32147/TCP   1d


blue
blue
blue
blue
blue
blue
blue
blue
yellow
blue

$ for i in {1..10}; do curl -s 35.204.3.92/green | grep color | awk '{print substr($2, 1, length($2)-1)}'; sleep 0.2; done;
green
green
green
green
green
green
green
green
green
green
```