Mirroring of Live Traffic in Kubernetes using Istio Traffic Mirroring

Why is traffic mirroring needed?

Pavan Kumar
Jul 25, 2020 · 6 min read

In the previous article **Weighted routing in Kubernetes using Istio** we have learned how to migrate traffic from version:v1 to version:v2 by enabling 25% of the customers to version:v2. And when the customers are satisfied with the new version we migrated the total traffic from version:v1 to version:v2 with zero downtime. Let us assume that you have deployed an e-commerce website and it is Black Friday time. Your website had a greater number of customers than at the usual time. The customers using the version:v2 faced issues like UI Slowness, Improper images loading, etc. These issues were caused because the application was not tested with real-time traffic. In the pre-production environment the load testing had been made, but with historic traffic/artificial traffic. It would be really hard to predict how an application would be responding to real-time traffic. Thanks to Istio for its Traffic Mirroring feature. Traffic mirroring also called shadowing, is a powerful concept that allows feature teams to bring changes to production with as little risk as possible. With this Traffic mirroring feature live traffic of the actual application is sent to the mirrored service.

Advantages of Using Istio Traffic Mirroring :

1. Bring the changes in the Production environment with as little risk as possible.

2. Reduction of the cost and efforts of maintaining a Pre-Production environment.

3. Testing of the new version of the application with real-time traffic.

4. Identification of Bugs and Issues in the application would be faster.

You would be wondering now when the traffic is sent to both the actual service and the mirrored service there would be chaos in the response sent to the customer. The actual magic of Istio is seen here. When the live traffic is sent to the mirrored service the mirrored service doesn't respond to the requests. It is just a mere receiver of the realtime traffic.

Let's get into the action now.

> *Before you decide to enable the features of Istio make sure that you label your namespace with the following label* ***istio-injection=enabled.*** *When you label the namespace the injection webhook is enabled and any new pods created in that namespace will automatically have the sidecar proxy (envoy proxy) enabled.*

Now Deploy the Pod having the webpage serving the version:v1 contents

Pod spec file for version:v1

Expose your pod via ClusterIP service as we would be testing traffic mirroring using a curl command inside the cluster. You can expose the service as a LoadBalancer and test it in the browser as well. Since I have Installed my Kubernetes cluster in Oracle Virtual Box I would be exposing the pod via ClusterIP and use the curl command to test traffic mirroring.

Now the requests are only sent to version:v1 when the curl command is executed

```
[root@master istio-medium]# kubectl run curl-test — image=odise/busybox-curl — rm -it — /bin/sh -c “while true; do curl web-service; sleep 1; done”
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
<html> <body> <h1>This is version V1!</h1> </body></html>
```

Let us configure the Virtual service and Destination Rule to configure traffic mirroring

Let us try to understand what is the above file doing. When Istio is Installed in your Kubernetes Cluster a lot of CRD's are deployed. The CRD's that we would be focusing on now is Virtual Service and Destination Rule. A Virtual Service defines a set of traffic routing rules for Kubernetes service. And Destination Rules specify where the traffic should be routed once the routing rules are met. Let us examine the YAML file in detail.

Virtual Service:

a) name: Specifies the name of the Virtual Service

b) hosts: The host to which traffic is sent. Here the host is the DNS name of our Kubernetes Service

c) http: It is the list of routing rules for HTTP traffic

d) subset: The name of the subset that the traffic should be directed to which is defined in the corresponding Destination Rule

e) weight: The amount of traffic to be forwarded to the service version

f) mirror: The destination service where the traffic should be mirrored

g) mirror_percent: Percentage of the traffic to be mirrored by the mirror field

Destination Rule:

a) name: The name of the Destination Rule

b) hosts: The host to which traffic is sent. Here the host is the DNS name of our Kubernetes Service

c) subsets: Named set that represents the Individual version of the service

d) subsets.name: Name of the subset

e) labels: Map of the labels that are used to select the pods

Now deploy a new pod with the new version:v2. The pod would be picked by the web-service service selector.

```
[root@master istio-medium]# kubectl run curl-test — image=odise/busybox-curl -it — /bin/sh -c “while true; do date;curl web-service; sleep 1; done”
Fri Jul 24 18:03:12 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
```

```
Fri Jul 24 18:03:13 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:14 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:15 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:16 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:17 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:18 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:19 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:20 UTC 2020
<html> <body> <h1>This is version V1!</h1> </body></html>
Fri Jul 24 18:03:21 UTC 2020
```

Requests to version:v1

We can deduce that all the requests are being directed to version:v1, What is version:v2 doing. Let’s look at the logs of version:v2

```
[root@master ~]# kubectl logs -f httpd-v2 -c httpd

127.0.0.1 — — [24/Jul/2020:18:03:12 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:13 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:14 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:15 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:16 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:17 +0000] “GET / HTTP/1.1” 200 58
```

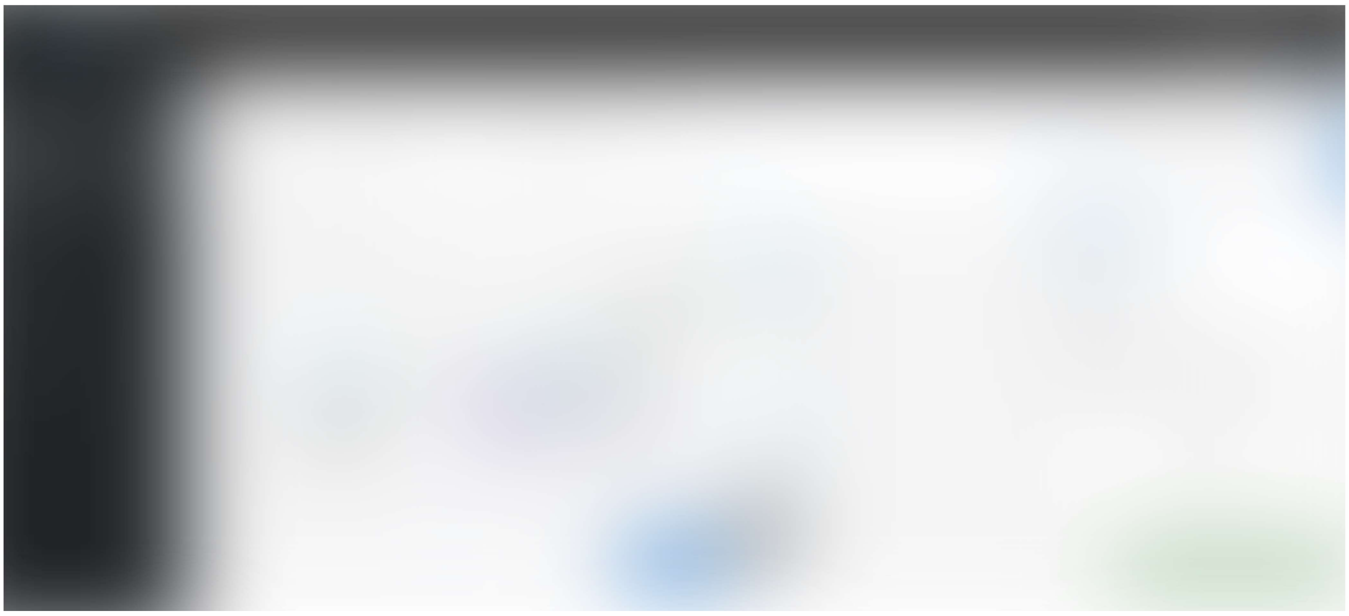
```
127.0.0.1 — — [24/Jul/2020:18:03:18 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:19 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:20 +0000] “GET / HTTP/1.1” 200 58
127.0.0.1 — — [24/Jul/2020:18:03:21 +0000] “GET / HTTP/1.1” 200 58
```

Did you see that?

Did you see that? Where is my version:v2 pod getting traffic from? Istio’s Data plane and Control plane behind the scenes are making this happen. The actual version of the application responds to the customer's requests. At the same time, the copy of the traffic is also being sent to the version:v2. And that is the reason we can see the traffic in version:v2

Let us visualize the traffic flow in Kiali. It is an addon given by Istio for better Visualization of your service mesh.

```
istioctl dashboard kiali
```



Kiali dashboard showing the flow of traffic

Kudos, We have successfully tested our application with real-time traffic. Now let us apply **Weighted routing** to slowly migrate traffic from version:v1 to version:v2

Prerequisites

Introduction to Istio Service Mesh

What is Istio Service Mesh?

medium.com

How to Install Istio using istioctl

As discussed in my earlier article we would be working on a few real-time scenarios where we would be able to…

medium.com

Recommended

Weighted routing in Kubernetes using Istio

Let us suppose version:v1 of your application is being used by your customers. A new version:v2 of an application is…

medium.com

MTLS in Istio

Setup Mutual TLS in Istio Service Mesh by plugging in existing CA Certificates

medium.com

Kubernetes Istio Istio Service Mesh Service Mesh Traffic