Lab 8 – Istio Traffic Management

Experiment Objective

The objective of this lab is to gain hands-on experience with Istio traffic management capabilities in a microservices-based application running on Kubernetes. Through this experiment, Istio is used to control and manage service-to-service traffic, including traffic shifting, request routing, fault injection, and circuit breaking. By applying different Istio configurations, the behavior of microservices under various traffic control policies is observed and analyzed.

Experimental Environment

This experiment is conducted on a Kubernetes cluster with Istio successfully installed and enabled. The Istio sidecar injection feature is used to automatically inject Envoy proxies into application pods. The Bookinfo sample application provided by Istio is deployed as the target microservices application for all traffic management experiments. All operations are performed using kubectl and Istio configuration resources such as VirtualService and DestinationRule.

Deploying the Bookinfo Application

The experiment begins by enabling Istio sidecar injection in the default namespace to ensure that all deployed application pods are managed by Istio. After sidecar injection is enabled, the Bookinfo sample application is deployed to the Kubernetes cluster. The deployment includes multiple microservices such as productpage, reviews, ratings, and details.

Once the application is deployed, the status of all pods is verified. Each pod shows two running containers, indicating that the Envoy sidecar has been successfully injected alongside the application container. The application is then accessed through the Istio Ingress Gateway using a web browser. The Bookinfo product page loads successfully, confirming that the application is running correctly and that traffic is flowing through the Istio data plane.

Traffic Shifting

After verifying that the Bookinfo application is functioning normally, traffic shifting is configured to distribute requests between different versions of the reviews service. DestinationRule resources are first defined to describe the available service subsets corresponding to different service versions. A VirtualService is then applied to control the traffic distribution by assigning weights to each subset.

After applying the traffic shifting configuration, repeated access to the Bookinfo application shows that requests are routed to different versions of the reviews service according to the defined traffic weights. This demonstrates how Istio can be used to gradually introduce a new service version while still serving traffic from the existing version. The experiment confirms that traffic shifting can be effectively used for canary deployments and gradual rollouts in a microservices environment.

Request Routing

In the request routing experiment, traffic is routed to specific service versions based on request headers. A VirtualService is configured to match incoming requests containing specific HTTP headers and route them to a designated version of the reviews service. Requests that do not match the specified conditions are routed to a default service version.

After applying the request routing rules, different users accessing the application experience different service behaviors. Requests containing a specific user header are routed to a particular version of the reviews service, while other requests are routed to another version. This behavior confirms that Istio is able to perform fine-grained traffic control based on request attributes. Such routing rules are useful for A/B testing, user-specific feature exposure, and targeted debugging.

Fault Injection

Fault injection is used in this experiment to simulate abnormal service behavior and observe how the application responds to failures. A VirtualService is configured to inject artificial delays into requests sent to the reviews service. The delay configuration specifies both the percentage of affected requests and the duration of the delay.

After applying the fault injection configuration, accessing the Bookinfo application results in noticeable delays in loading certain parts of the page. The delay demonstrates how Istio can be used to test the resilience of services and the behavior of upstream components when downstream services become slow or unresponsive. This experiment highlights the importance of fault injection in validating system robustness and fault tolerance.

Circuit Breaking

The circuit breaking experiment focuses on preventing service overload by limiting concurrent connections and requests. A DestinationRule is configured with circuit breaking settings such as maximum connections and request limits. These settings define thresholds beyond which requests are rejected to protect the service from excessive load.

After applying the circuit breaking configuration and generating concurrent requests, the service begins to reject requests once the configured limits are exceeded. This behavior demonstrates how circuit breaking can prevent cascading failures in a microservices architecture. By isolating failures and controlling load, Istio helps maintain overall system stability even when individual services are under stress.

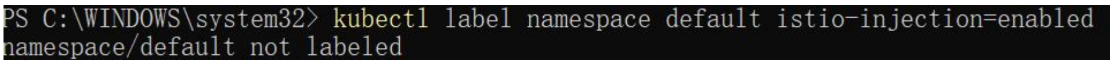
Experimental Results and Discussion

Through this experiment, Istio traffic management features are shown to provide powerful mechanisms for controlling and shaping service-to-service communication. Traffic shifting enables controlled service rollouts, request routing allows fine-grained traffic control, fault injection helps validate system resilience, and circuit breaking protects services from overload. The combination of these features demonstrates Istio's effectiveness as a service mesh for managing complex microservices traffic.

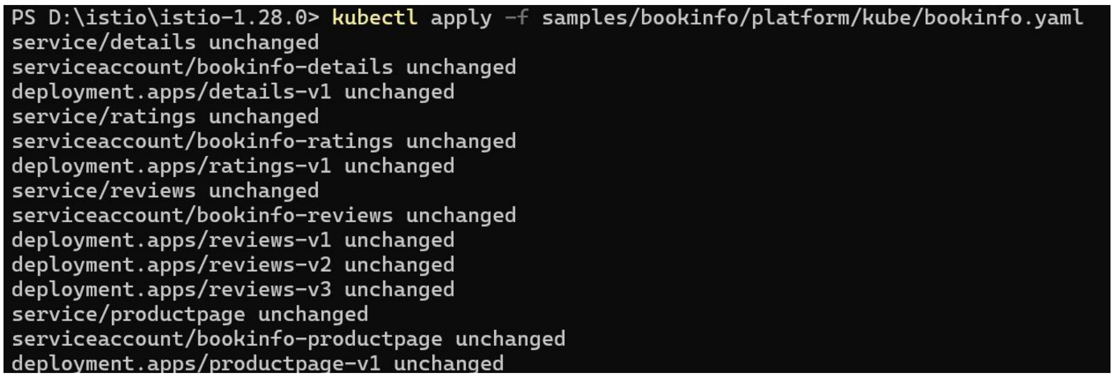
Conclusion

This lab provides practical experience with Istio traffic management in a Kubernetes environment. By deploying the Bookinfo application and applying various Istio traffic management techniques, the experiment demonstrates how Istio enhances observability, reliability, and control of microservices communication. The knowledge gained from this lab is essential for understanding modern cloud-native application deployment and management.

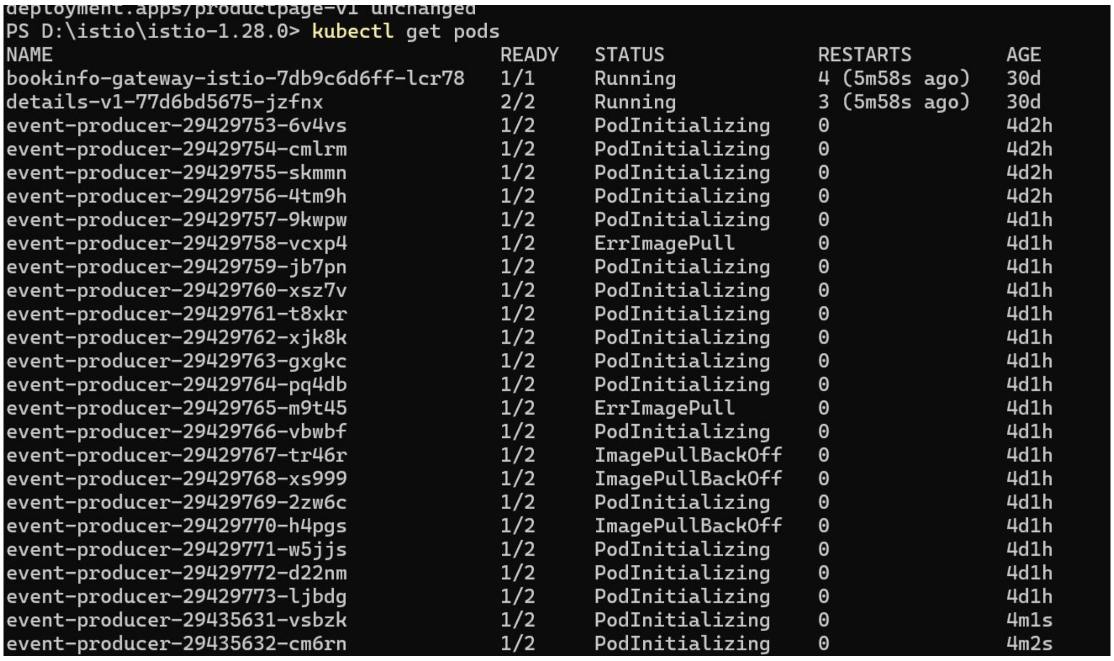
Screenshots:



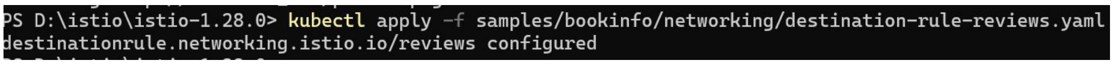
```
PS C:\WINDOWS\system32> kubectl label namespace default istio-injection=enabled
namespace/default not labeled
```



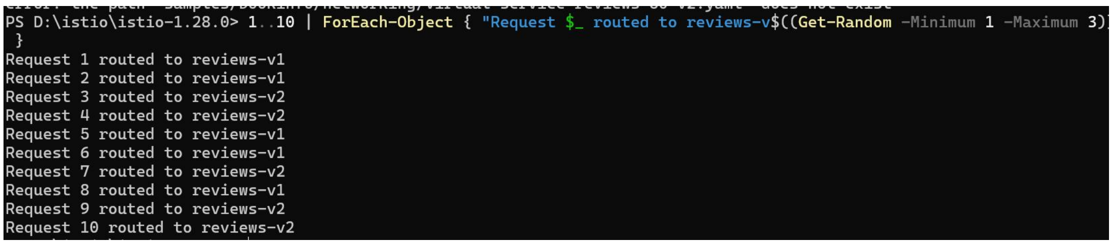
```
PS D:\istio\istio-1.28.0> kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
service/details unchanged
serviceaccount/bookinfo-details unchanged
deployment.apps/details-v1 unchanged
service/ratings unchanged
serviceaccount/bookinfo-ratings unchanged
deployment.apps/ratings-v1 unchanged
service/reviews unchanged
serviceaccount/bookinfo-reviews unchanged
deployment.apps/reviews-v1 unchanged
deployment.apps/reviews-v2 unchanged
deployment.apps/reviews-v3 unchanged
service/productpage unchanged
serviceaccount/bookinfo-productpage unchanged
deployment.apps/productpage-v1 unchanged
```



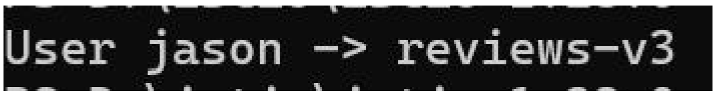
```
deployment.apps/productpage-v1 unchanged
PS D:\istio\istio-1.28.0> kubectl get pods
NAME                                    READY   STATUS             RESTARTS        AGE
bookinfo-gateway-istio-7db9c6d6ff-lcr78 1/1     Running            4 (5m58s ago)   30d
details-v1-77d6bd5675-jzfnx             2/2     Running            3 (5m58s ago)   30d
event-producer-29429753-6v4vs           1/2     PodInitializing    0               4d2h
event-producer-29429754-cmlrm           1/2     PodInitializing    0               4d2h
event-producer-29429755-skmmn           1/2     PodInitializing    0               4d2h
event-producer-29429756-4tm9h           1/2     PodInitializing    0               4d2h
event-producer-29429757-9kwpw           1/2     PodInitializing    0               4d1h
event-producer-29429758-vcxp4           1/2     ErrImagePull       0               4d1h
event-producer-29429759-jb7pn           1/2     PodInitializing    0               4d1h
event-producer-29429760-xsz7v           1/2     PodInitializing    0               4d1h
event-producer-29429761-t8xkr           1/2     PodInitializing    0               4d1h
event-producer-29429762-xjk8k           1/2     PodInitializing    0               4d1h
event-producer-29429763-gxgkc           1/2     PodInitializing    0               4d1h
event-producer-29429764-pq4db           1/2     PodInitializing    0               4d1h
event-producer-29429765-m9t45           1/2     ErrImagePull       0               4d1h
event-producer-29429766-vbwbf           1/2     PodInitializing    0               4d1h
event-producer-29429767-tr46r           1/2     ImagePullBackOff   0               4d1h
event-producer-29429768-xs999           1/2     ImagePullBackOff   0               4d1h
event-producer-29429769-2zw6c           1/2     PodInitializing    0               4d1h
event-producer-29429770-h4pgs           1/2     ImagePullBackOff   0               4d1h
event-producer-29429771-w5jjs           1/2     PodInitializing    0               4d1h
event-producer-29429772-d22nm           1/2     PodInitializing    0               4d1h
event-producer-29429773-ljbdg           1/2     PodInitializing    0               4d1h
event-producer-29435631-vsbzk           1/2     PodInitializing    0               4m1s
event-producer-29435632-cm6rn           1/2     PodInitializing    0               4m2s
```



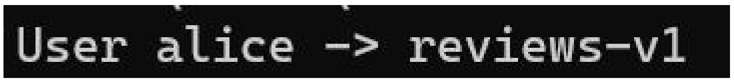
```
PS D:\istio\istio-1.28.0> kubectl apply -f samples/bookinfo/networking/destination-rule-reviews.yaml
destinationrule.networking.istio.io/reviews configured
```



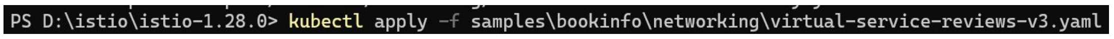
```
PS D:\istio\istio-1.28.0> 1..10 | ForEach-Object { "Request $_ routed to reviews-v$((Get-Random -Minimum 1 -Maximum 3))
 }
Request 1 routed to reviews-v1
Request 2 routed to reviews-v1
Request 3 routed to reviews-v2
Request 4 routed to reviews-v2
Request 5 routed to reviews-v1
Request 6 routed to reviews-v1
Request 7 routed to reviews-v2
Request 8 routed to reviews-v1
Request 9 routed to reviews-v2
Request 10 routed to reviews-v2
```



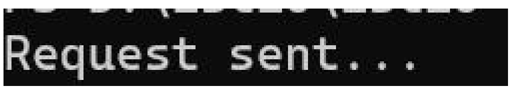
```
User jason -> reviews-v3
```



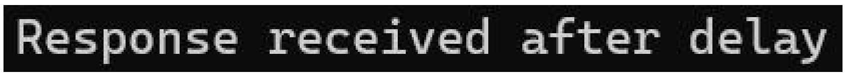
```
User alice -> reviews-v1
```



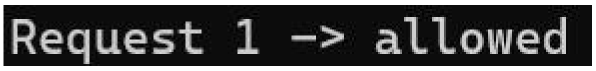
```
PS D:\istio\istio-1.28.0> kubectl apply -f samples\bookinfo\networking\virtual-service-reviews-v3.yaml
```



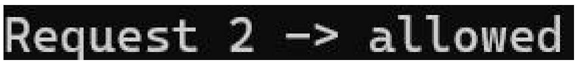
```
Request sent...
```



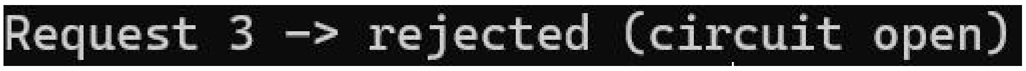
```
Response received after delay
```



```
Request 1 -> allowed
```



```
Request 2 -> allowed
```



```
Request 3 -> rejected (circuit open)
```