**Istio Traffic Management Lab Documentation**

**Introduction**

This lab exercise provides hands-on experience with Istio's core traffic management capabilities. You will learn how to control service-to-service communication in a microservices architecture through four key tasks: **traffic shifting** (for gradual version migrations), **request routing** (for targeted traffic distribution), **fault injection** (for resilience testing), and **circuit breaking** (for preventing cascading failures). The lab uses Istio's Bookinfo sample application as the test environment.

**Prerequisites:** A running Kubernetes cluster with Istio installed (demo profile recommended), the Bookinfo application deployed, and Istio sidecar injection enabled in the default namespace. Refer to Istio's [Getting Started](https://istio.io/latest/docs/setup/getting-started/) guide if you need to set up the environment first.

**Task 1: Traffic Shifting**

Traffic shifting allows you to gradually route traffic from one service version to another, enabling safe canary deployments or blue-green migrations. In this task, you will shift traffic from reviews:v1 to reviews:v2 and then to reviews:v3.

**1.1 Initial Setup: Route All Traffic to v1**

First, ensure all traffic to the reviews service is routed to v1 (no star ratings):  
 kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yamlVerify the configuration:  
 kubectl get virtualservice reviews -o yamlThe output should show a route section with only reviews:v1 and weight: 100.

Access the Bookinfo product page (http://$GATEWAY\_URL/productpage)—you will see "Reviews: There are no reviews yet."

**1.2 Shift 50% Traffic to v2**

Update the VirtualService to split traffic 50-50 between v1 and v2 (v2 shows black stars):  
 kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml**Note:** The official YAML filename uses "v3" but configures 50% v1 and 50% v2. Check the file content if needed.

Verify the traffic split:  
 kubectl get virtualservice reviews -o jsonpath='{.spec.http[0].route}'The output should display two route entries with weight: 50 for both v1 and v2.

**1.3 Test the Traffic Split**

Refresh the Bookinfo product page multiple times. Approximately half the time, you will see black stars (v2), and half the time, no stars (v1). This confirms the traffic is being split as configured.

**1.4 Shift 100% Traffic to v3**

Once v2 is validated, shift all traffic to v3 (shows red stars):  
 kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-v3.yamlRefresh the product page—you will now only see red stars, indicating 100% traffic is routed to v3.

**Task 2: Request Routing**

Request routing enables you to route traffic based on specific criteria such as HTTP headers, query parameters, or client identities. In this task, you will route traffic from a logged-in user (jason) to reviews:v2 while sending all other traffic to reviews:v1.

**2.1 Configure Request Routing**

Apply the VirtualService that routes based on the end-user header:  
 kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yamlExamine the configuration:  
 kubectl get virtualservice reviews -o yamlThe YAML includes a match section that checks for headers.end-user.exact: jason, routing matching requests to v2. All other requests go to v1.

**2.2 Test the Routing**

* **Anonymous user:** Access the product page without logging in. You will see no stars (v1).
* **Logged-in user (jason):** Click "Sign in" on the product page, enter username jason (no password), and log in. Refresh the page—you will see black stars (v2).

**Use Case:** This is useful for A/B testing (routing specific user groups to new features) or internal testing (routing team traffic to pre-production versions).

**Task 3: Fault Injection**

Fault injection allows you to simulate service failures (e.g., delays, errors) to test the resilience of your application. In this task, you will inject a 7-second delay into requests from jason to the reviews service, and then inject a 503 error.

**3.1 Inject a Delay Fault**

Apply the VirtualService with a 7-second delay for user jason:  
 kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yamlExamine the fault configuration:  
 kubectl get virtualservice ratings -o yamlThe fault section specifies delay: {percent: 100, fixedDelay: 7s} for requests from user jason.

**3.2 Test the Delay Fault**

Log in as jason and access the product page. The page will take ~7 seconds to load, and you may see a "Error fetching product reviews" message if the productpage service has a timeout (default is 6 seconds). This simulates a slow downstream service.

**3.3 Inject an Error Fault**

Now inject a 503 (Service Unavailable) error for user jason:  
 kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-abort.yamlTest by logging in as jason and refreshing the product page. You will see an error message indicating the ratings service is unavailable.

**3.4 Clean Up Fault Injection**

Revert to the default routing to remove the fault:  
 kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml

**Task 4: Circuit Breaking**

Circuit breaking prevents cascading failures by stopping requests to a failing service after a certain threshold is reached. In this task, you will configure a circuit breaker for the ratings service and test it by overwhelming the service with requests.

**4.1 Configure Destination Rule with Circuit Breaking**

First, create a Destination Rule that defines circuit breaking settings for ratings:v1:  
 kubectl apply -f samples/bookinfo/networking/destination-rule-ratings.yamlExamine the circuit breaking configuration:  
 kubectl get destinationrule ratings -o yamlKey settings:  
 maxConnections: 1: Maximum number of concurrent connections to the service.http1MaxPendingRequests: 1: Maximum number of pending requests.maxRequestsPerConnection: 1: Maximum requests per connection.

**4.2 Route Traffic to ratings:v1**

Ensure all traffic to ratings goes to v1:  
 kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml

**4.3 Test the Circuit Breaker**

Use fortio (a load testing tool included with Istio) to send concurrent requests to the ratings service:  
 # Deploy fortio  
kubectl apply -f samples/bookinfo/networking/fortio-deploy.yaml  
  
# Exec into the fortio pod  
kubectl exec -it fortio-deploy-xxxxxxxxx-xxxxx -c fortio -- /usr/bin/fortio load -c 2 -qps 0 -t 20s http://ratings:9080/ratings/0**Command Explanation:**-c 2 = 2 concurrent connections; -qps 0 = unlimited QPS; -t 20s = run for 20 seconds.

The output will show that some requests succeed (200 OK) but others fail (503 Service Unavailable). This is because the circuit breaker limits concurrent connections to 1—excess requests are rejected, preventing the service from being overwhelmed.

**4.4 Visualize Circuit Breaking in Grafana**

Open the Grafana dashboard to see circuit breaking metrics:  
 istioctl dashboard grafanaNavigate to "Istio Service Dashboard > ratings" and look for metrics like "Circuit Breaker Open" or "Rejected Requests" to confirm the circuit breaker is active.

**Lab Cleanup**

After completing the lab, remove the Bookinfo application and Istio resources (if needed):  
 # Clean up Bookinfo  
samples/bookinfo/platform/kube/cleanup.sh  
  
# Remove Istio  
istioctl uninstall -y --purge  
kubectl delete namespace istio-system

**Key Takeaways**

* **Traffic Shifting:** Enables gradual, low-risk version migrations by controlling traffic distribution between service versions.
* **Request Routing:** Routes traffic based on custom criteria (user identity, headers) for targeted testing or feature rollouts.
* **Fault Injection:** Tests application resilience by simulating delays or errors in downstream services.
* **Circuit Breaking:** Prevents cascading failures by limiting requests to unhealthy services, improving overall system stability.

**Additional Resources**

For more details on Istio traffic management, refer to the official documentation:  
 [Istio Traffic Management Concepts](https://istio.io/latest/docs/concepts/traffic-management/)[VirtualService Configuration Reference](https://istio.io/latest/docs/reference/config/networking/virtual-service/)[DestinationRule Configuration Reference](https://istio.io/latest/docs/reference/config/networking/destination-rule/)