**Exercise 4.1: Traffic Route**

| To test the Traffic Route policy we will create a new backend API to balance our traffic across. The backend-v0 will display the product page with no sale items. Backend-v1 will display the same product page with sale items so we can differentiate between the API versions we are using. |
| --- |

1. Additional backends are already deployed as part of the Marketplace Application (created in Lab 3). However, some of the additional deployments are created with 0 replicas so we will need to scale them up.

student@controlplane~: $ kubectl -n kuma-demo scale deployment kuma-demo-backend-v1 --replicas=1

| deployment.apps/kuma-demo-backend-v1 scaled |
| --- |

1. Confirm the new pod is now in a running state.

student@controlplane ~: $ kubectl get pods -n kuma-demo

| NAME READY STATUS RESTARTS AGE  kuma-demo-app-6787b4f7f5-l5lkp 2/2 Running 0 27m  kuma-demo-backend-v0-56db47c579-xbbhj 2/2 Running 0 27m  **kuma-demo-backend-v1-79584cdc57-t75lk 2/2 Running 0 61s**  postgres-master-645bc44fd-h86nb 2/2 Running 0 27m  redis-master-55fd8f6f54-nks5q 2/2 Running 0 27m |
| --- |

1. The control plane creates a default TrafficRoute every time a new Mesh is created. The default TrafficRoute enables the traffic between all the services in the mesh. Review the current policy:

student@controlplane bin: $ kubectl get trafficroutes.kuma.io route-all-default -o yaml

| apiVersion: kuma.io/v1alpha1  kind: TrafficRoute  mesh: default  metadata:  name: route-all-default  spec:  sources:  - match:  kuma.io/service: '\*'  destinations:  - match:  kuma.io/service: '\*'  conf:  loadBalancer:  roundRobin: {}  destination:  kuma.io/service: '\*' |
| --- |

Due to the order of policy enforcement, we will be adding a new policy to override the default behavior.

4. We will now create a new policy to balance traffic across the two backends. The valus for the service names and version tags are obtained from the Admin API or Mesh GUI from the data planes.

student@controlplane bin: $ kubectl apply -f trafficRoute.yaml

| trafficroute.kuma.io/balanced created |
| --- |

5. Refresh the Web GUI and approximately 1/10 should now show the sale on “Manufact Frugal Sun Dress - Size M”



**Exercise 4.2: Traffic Permission [See Exercise 3.4]**

**Exercise 4.3: Health Check**

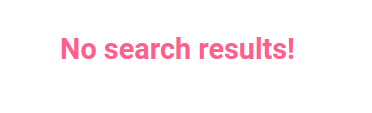
| To test the Health Check policy intentionally look for a TCP response from the database that does not exist causing a failure and stopping traffic from reaching it. |
| --- |

1. Apply the health check policy

student@controlplane bin: $ kubectl apply -f healthCheck.yaml

| healthcheck.kuma.io/backend-to-db-check created |
| --- |

1. Refresh the Web GUI for the Marketplace application and you should now see the below. **Note:** It may take several seconds for the changes to take effect and the health check to fail.



1. Removing the health check and refreshing the GUI will again display the results.

student@controlplane bin: $ kubectl delete -f healthCheck.yaml

| healthcheck.kuma.io "backend-to-db-check" deleted |
| --- |

**Exercise 4.4: Circuit Breaker [TODO]**

1. Enable Mutual TLS with a builtin CA backend.

student@controlplane bin: $ echo "apiVersion: kuma.io/v1alpha1

kind: Mesh

metadata:

name: default

spec:

mtls:

enabledBackend: ca-1

backends:

- name: ca-1

type: builtin" | kubectl apply -f -

| mesh.kuma.io/default configured |
| --- |

**Exercise 4.4: Fault Injection**

The fault injection policy supports aborting the connection, delaying the response or limiting the speed of responding to requests. We will test all 3 configurations to see how they behave.

1. Apply the abort Fault Injection policy. This policy is set to return a HTTP status code of 503 100% of the time. Communication from the frontend to the backend will be aborted.

student@controlplane bin: $ kubectl apply -f abortFaultInjection.yaml

| faultinjection.kuma.io/abort created |
| --- |

1. Reload the Marketplace Application by navigating to <http://localhost:8080/#>. Observe that the 503 status is returned.

**There are Product API issues:**

Error: Request failed with status code 503

1. Apply the delay Fault Injection policy. Delay defines a configuration of delaying a response from a destination. In this example we will delay the response for 20 percent of the requests by 5 seconds. So about half of our requests will show a loading symbol for approximately 5 seconds before loading the content.

student@controlplane bin: $ kubectl apply -f delayFaultInjection.yaml

| faultinjection.kuma.io/delay created |
| --- |

1. Reload the Marketplace Application several times to confirm the behavior. A small percentage of requests should sit loading for several seconds.



1. Apply the bandwidth Fault Injection policy. ResponseBandwidth defines a configuration to limit the speed of responding to the requests. The example will limit the response to 10 kbps about 50% of the time.

student@controlplane bin: $ kubectl apply -f bandwidthFaultInjection.yaml

| faultinjection.kuma.io/bandwidth configured |
| --- |

1. Reload the Marketplace Application several times to confirm the behavior. About half of the requests should sit loading for several seconds.

