1. Traffic Shifting

(1) Download and install Istio

**Command:**

curl -L https://istio.io/downloadIstio | sh -

cd istio-1.27.1

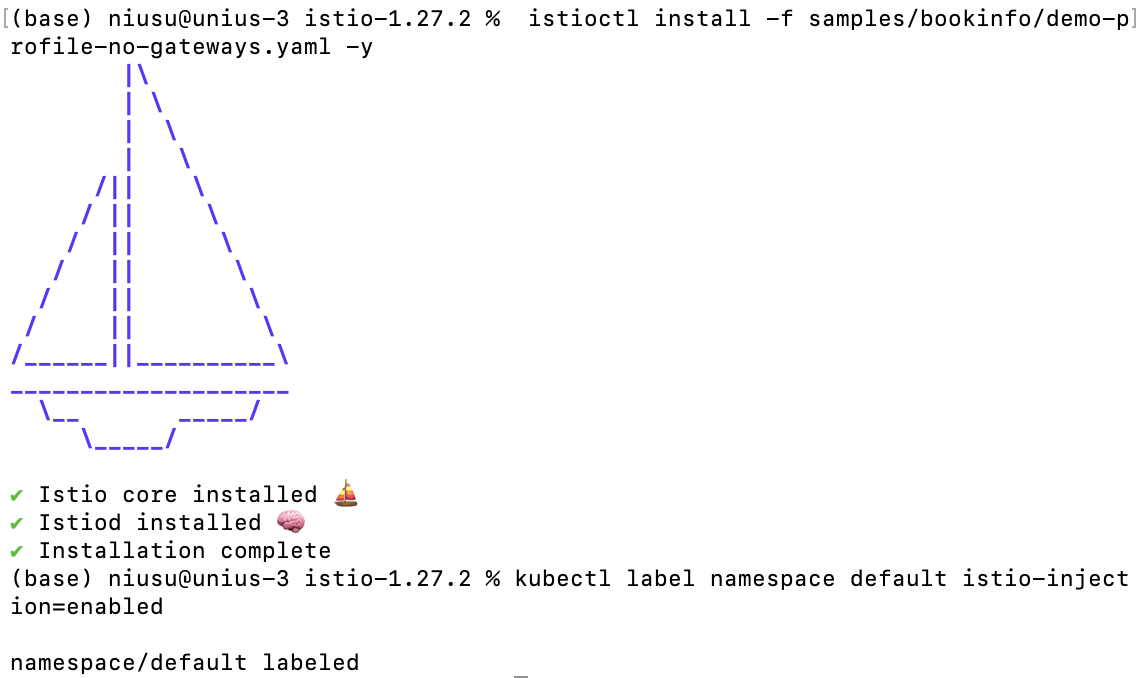
export PATH=$PWD/bin:$PATH

istioctl install -f samples/bookinfo/demo-profile-no-gateways.yaml -y

kubectl label namespace default istio-injection=enabled

**Note:**

Downloads Istio and adds the istioctl binary to your PATH so you can use Istio commands.



***Figure1: Install Istio***

(2) Deploy the sample application

**Command:**

cd ~

kubectl label -f experimental-install.yaml

kubectl get crd gateways.gateway.networking.k8s.io

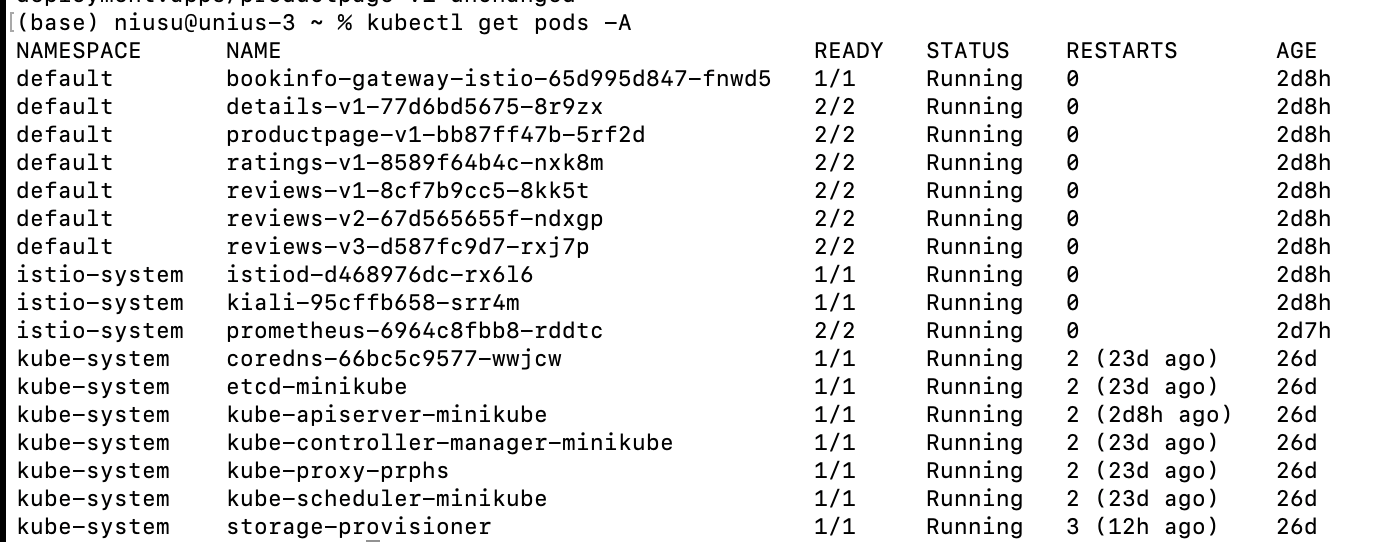
kubectl apply -f ~/istio-1.28.0/samples/bookinfo/platform/kube/bookinfo.yaml

kubectl get pods- A

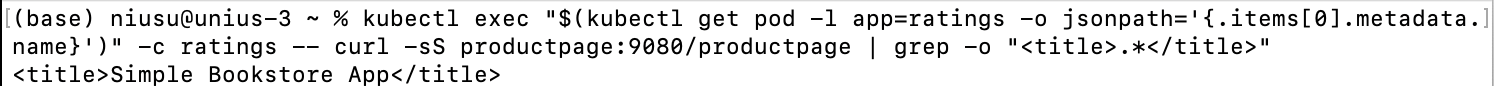
kubectl exec "$(kubectl get pod -l app=ratings -o jsonpath='{.items[0].metadata.name}')" -c ratings -- curl -sS productpage:9080/productpage | grep -o "<title>.\*</title>"

**Note:**

Deploys the Bookinfo sample application and checks that all services and pods are running. Confirms the application is running by checking the page title in the response.



***Figure1: Pods status***



***Figure2: Validate the app***

(3) Configure Gateway and Destination Rules

**Command:**

kubectl apply -f ~/istio-1.28.0/samples/bookinfo/gateway-api/bookinfo-gateway.yaml

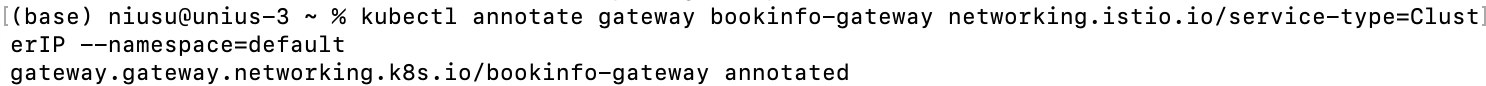
kubectl annotate gateway bookinfo-gateway networking.istio.io/service-type=ClusterIP --namespace=default

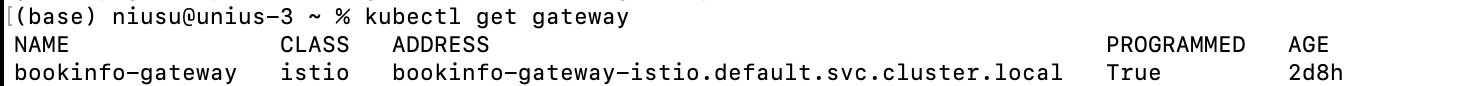
cd istio-1.28.0/samples/bookinfo/networking

kubectl apply -f destination-rule-all.yaml

**Note:**

Creates the Bookinfo Gateway, configures its service type, and applies destination rules.  
Destination rules define subsets required for advanced traffic routing such as canary or version-based routing.



******

***Figure1: Configure Gateway and Destination Rules***

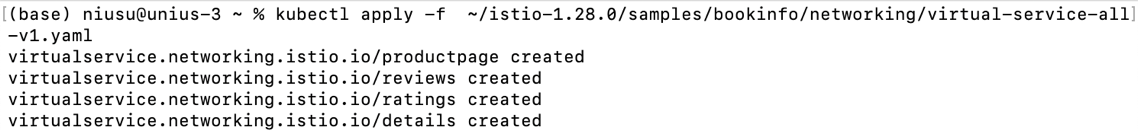
(4) Traffic to v1 only

**Command:**

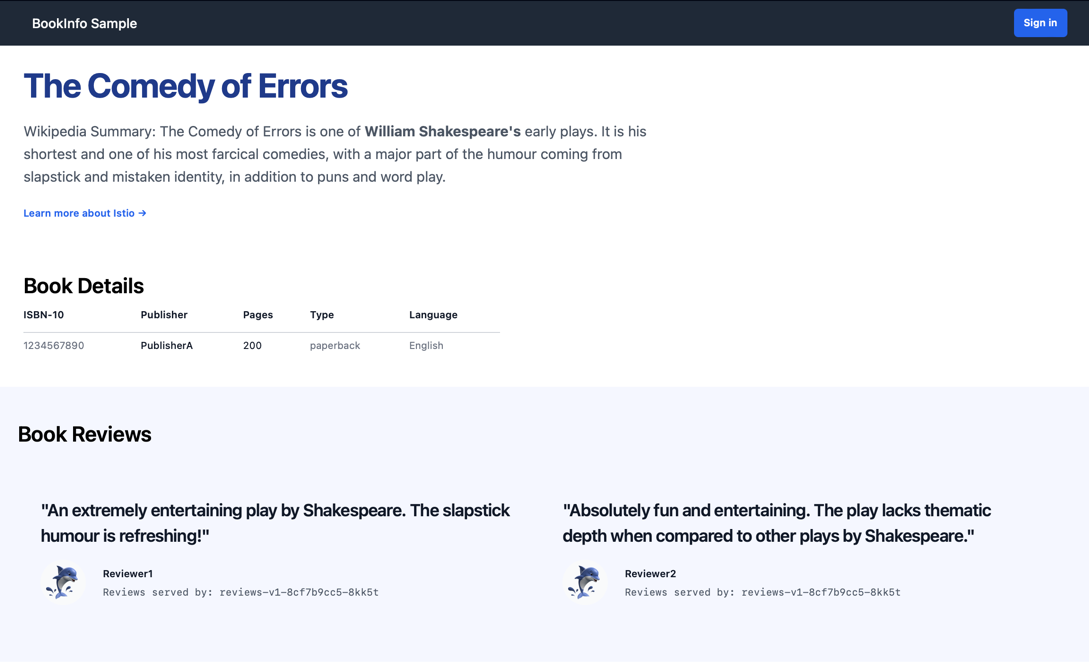
kubectl apply -f ~/istio-1.28.0/samples/bookinfo/networking/virtual-service-all-v1.yaml

**Note:**

The VirtualService configuration routes all incoming traffic to reviews:v1.



***Figure1: Configure the yaml file***



***Figure2: Display with no rating stars only***

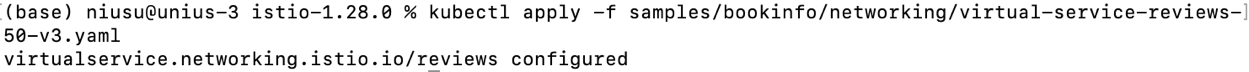
(5) Traffic split between v1-v3

**Command:**

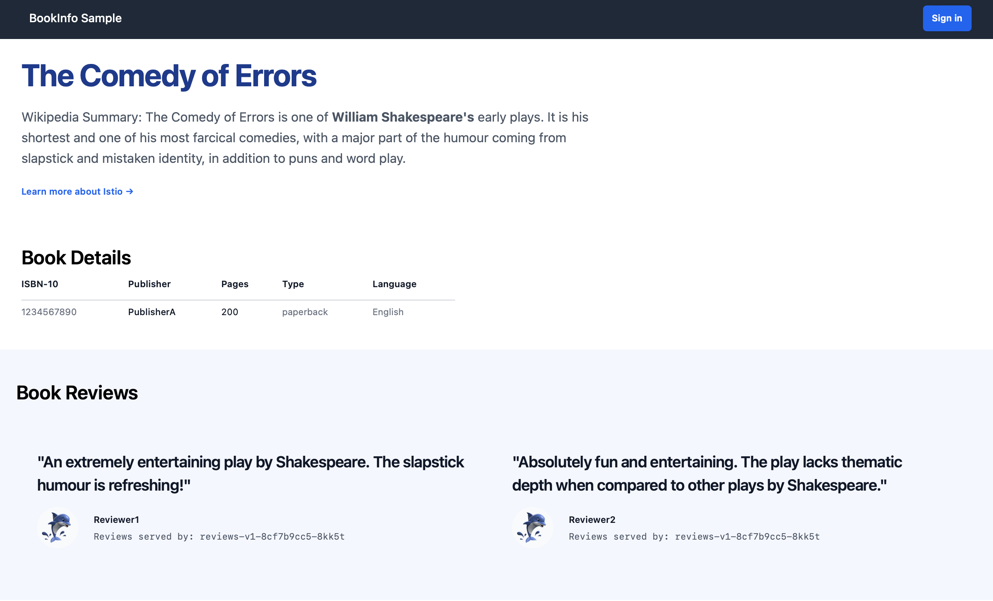
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml

**Note:**

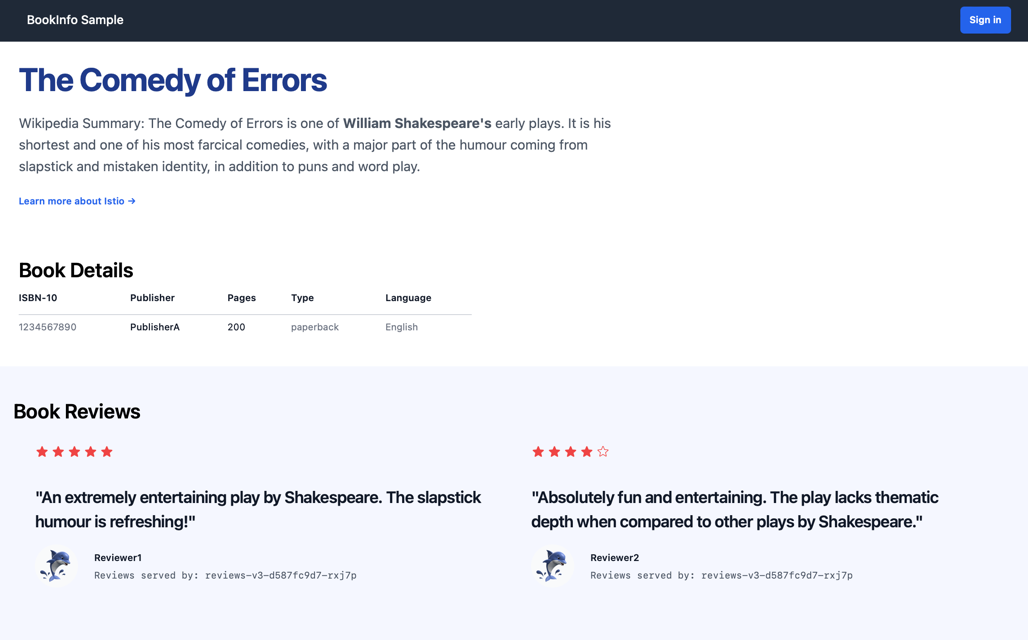
This configuration introduces traffic shifting, sending 50% of requests to reviews:v3 while the rest go to the default version.



***Figure1: Configure the yaml file***



***Figure2: No star ratings approximately 50% of the time***



***Figure3: Red colored star ratings approximately 50% of the time***

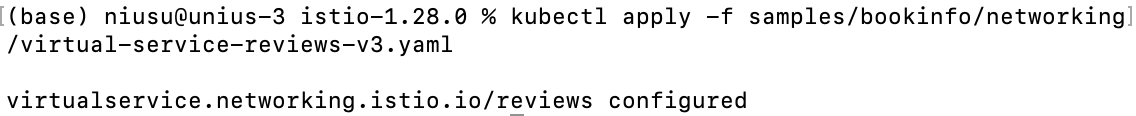
(6) Traffic to v3 only

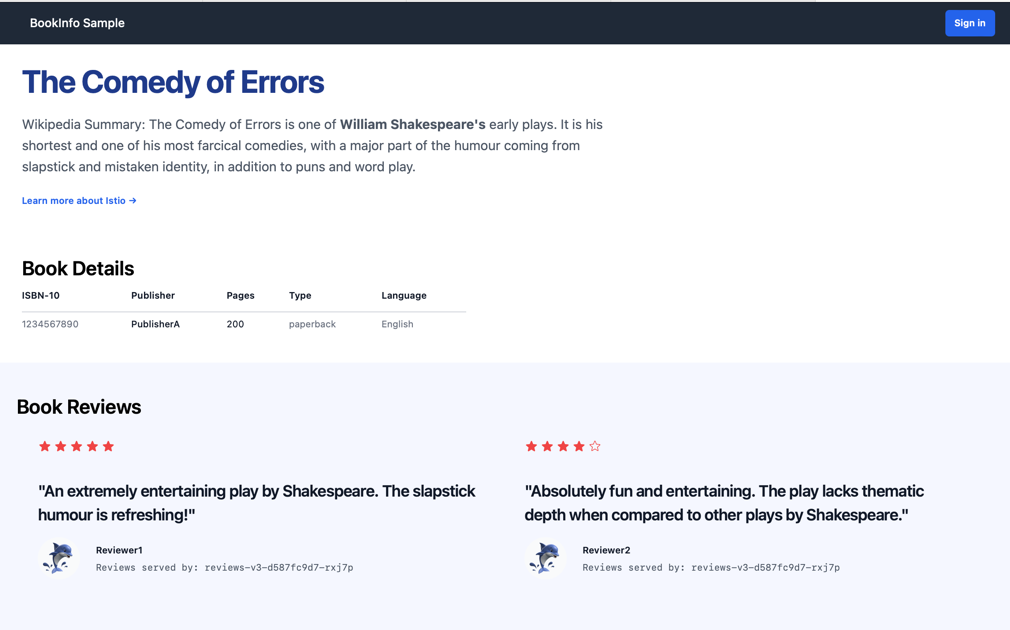
**Command:**

kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-v3.yaml

**Note:**

This step sends all traffic exclusively to reviews:v3.

 ***Figure1: Configure the yaml file***

******

***Figure2: Display red colored star rating only***

2. Request Routing

(1) Route all request to v1

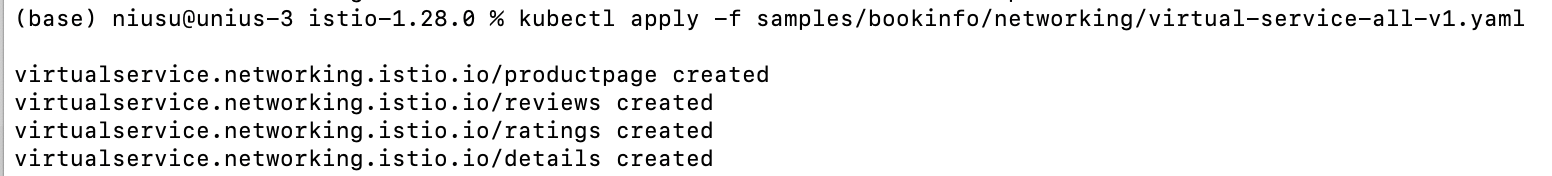
**Command:**

kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml

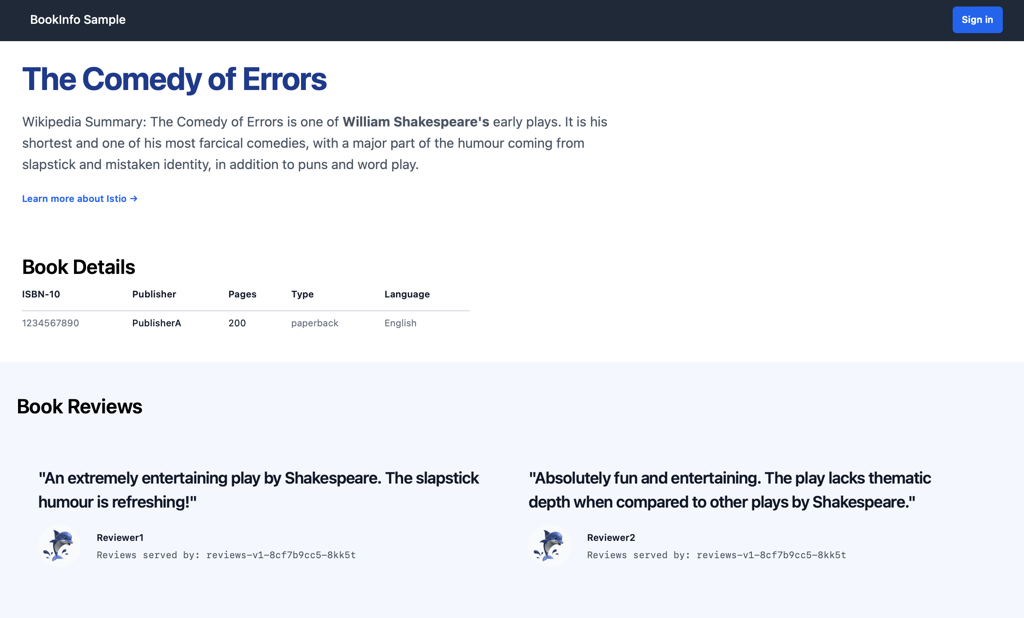
kubectl get virtualservice -o yaml

**Note:**

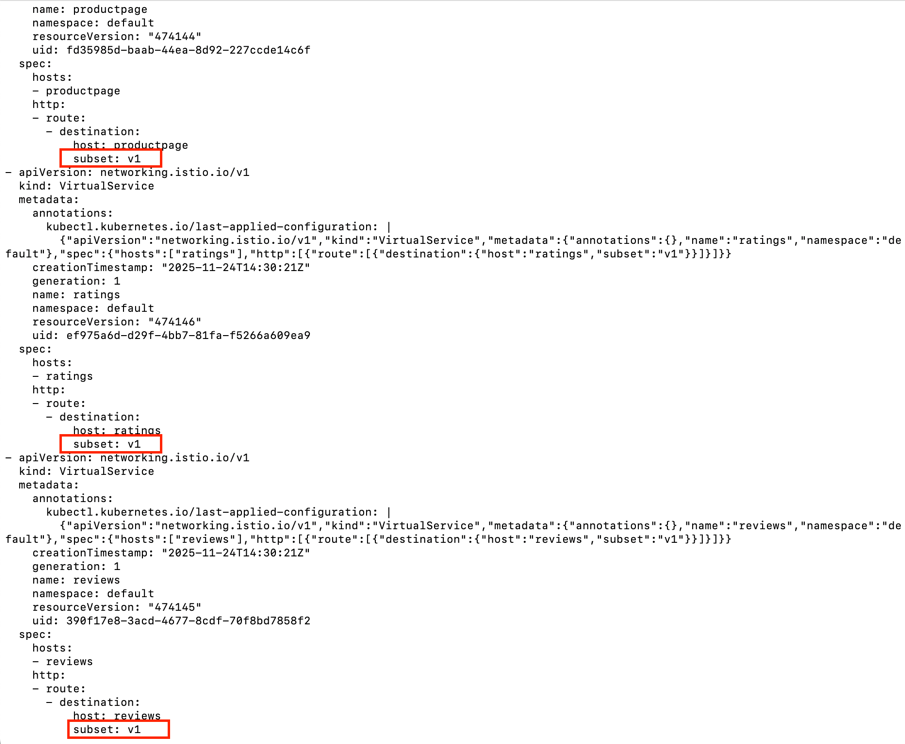
This VirtualService configuration forces all user traffic to be routed to reviews:v1.



***Figure1: Configure the yaml file***



***Figure2: Display with no rating stars only***

******

***Figure3: subset route to the v1 version rules***

(2) Route based on user identity

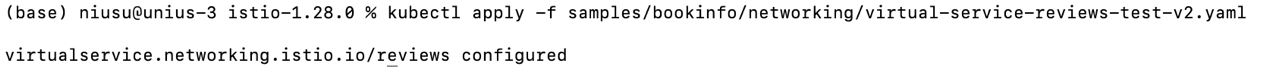
**Command:**

kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml

kubectl get virtualservice -o yaml

**Note:**

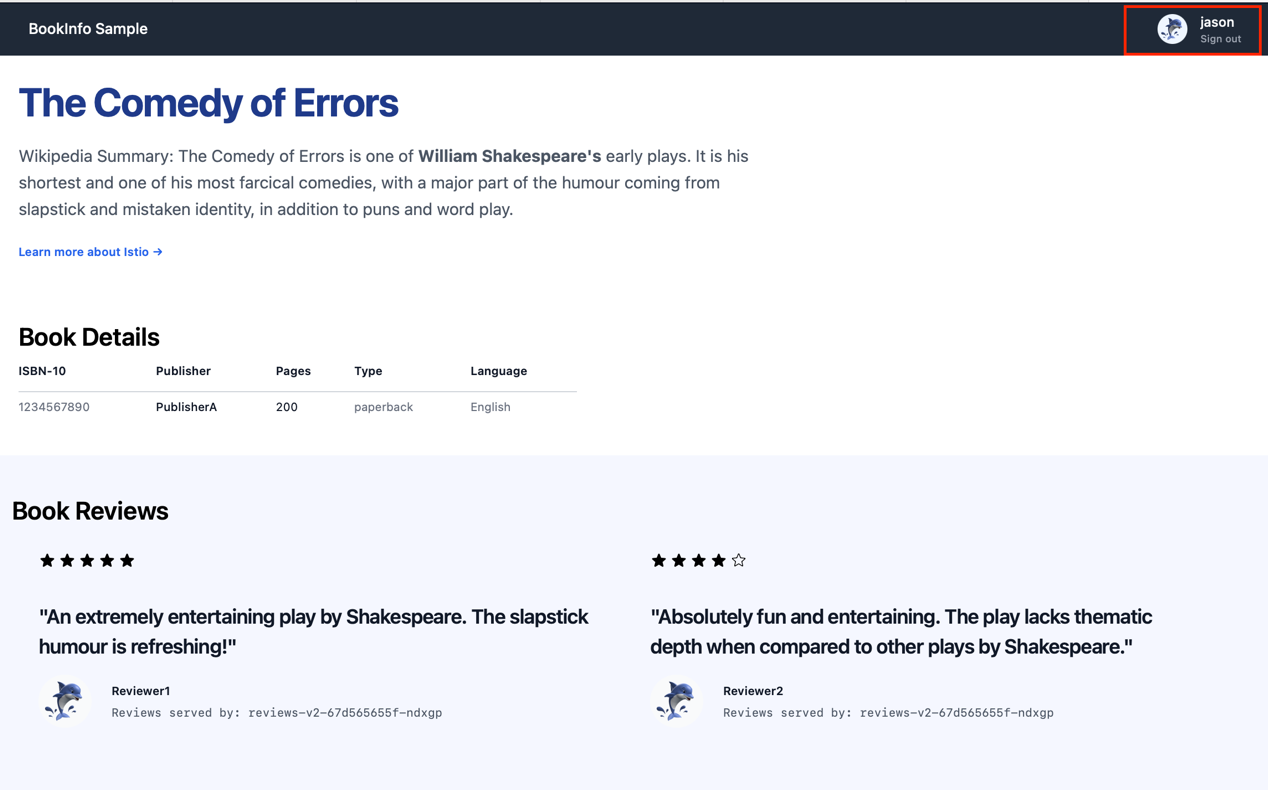
This configuration routes traffic for the specific user "Jason" to reviews:v2, while all other users continue receiving the default version.



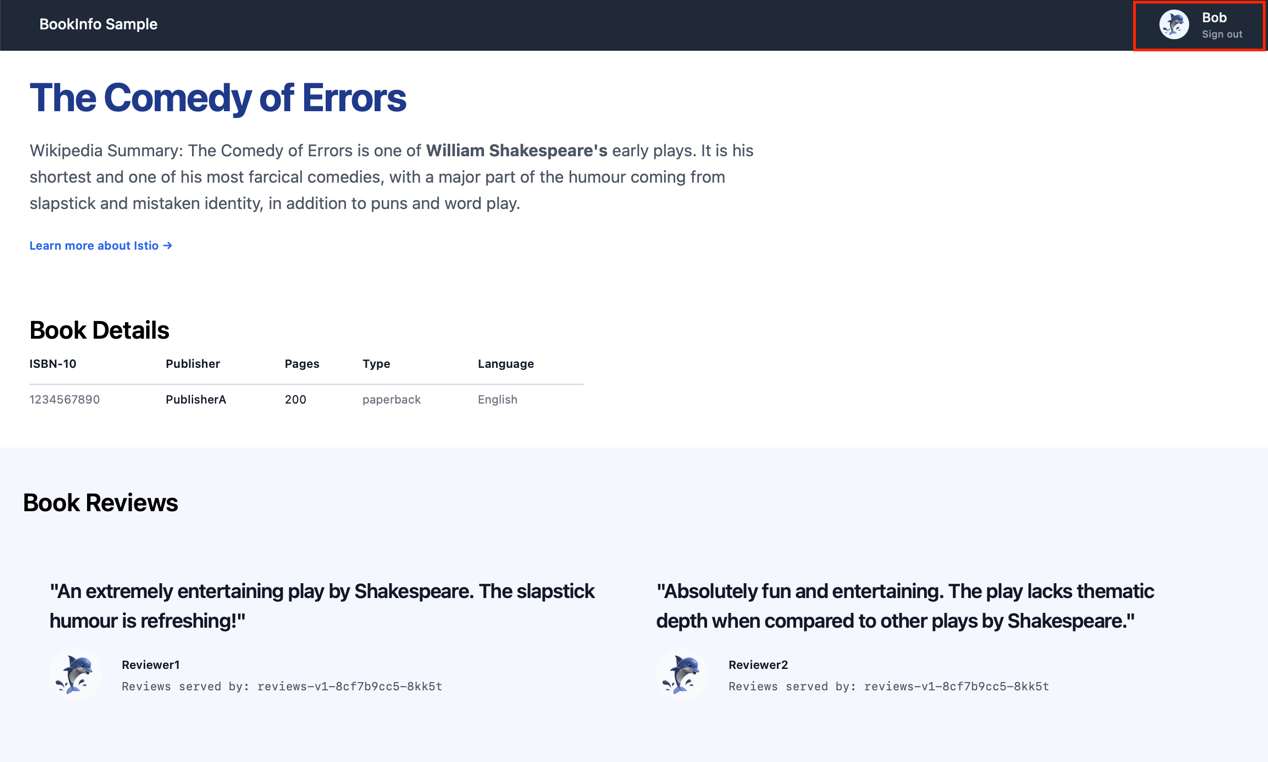
***Figure1: Configure the yaml file***

******

***Figure2: Confirm the rules***

******

***Figure3: User jason display the rating stars***

******

***Figure4: Other users display with no rating stars***

(3) Clean up

**Command:**

kubectl delete -f samples/bookinfo/networking/virtual-service-all-v1.yaml

3. Fault Injection

(1) Injecting an HTTP delay fault

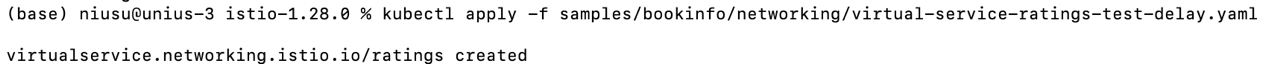
**Command:**

kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml

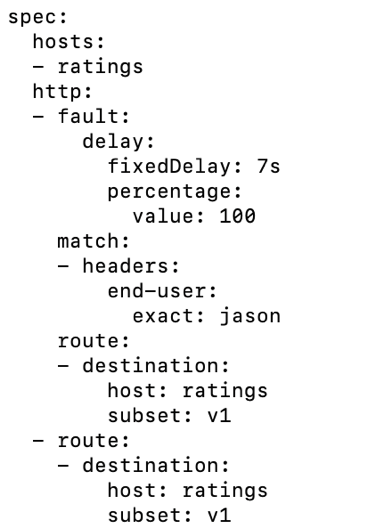
kubectl get virtualservice ratings -o yaml

**Note:**

This configuration injects an artificial HTTP delay into the ratings service.  
The VirtualService forces responses to be delayed before returning to the caller, allowing testing of system behavior under slow downstream dependencies.

******

***Figure1:*** ***Configure the yaml file***

******

***Figure2:*** ***Confirm the rules***

(2) Testing delay fault

**Note:**

When accessing the product page, the injected delay causes the reviews service to time out.



***Figure1:*** ***Reviews display "Sorry, product reviews are currently unavailable..."***

(3) Injecting an HTTP abort fault

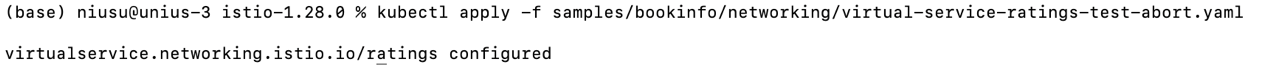
**Command:**

kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-abort.yaml

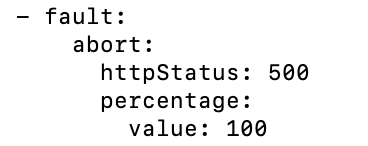
kubectl get virtualservice ratings -o yaml

**Note:**

This step configures the ratings service to return an HTTP error response (abort) instead of normal data. The abort fault simulates service failures and helps evaluate resilience and error-handling mechanisms in upstream services.

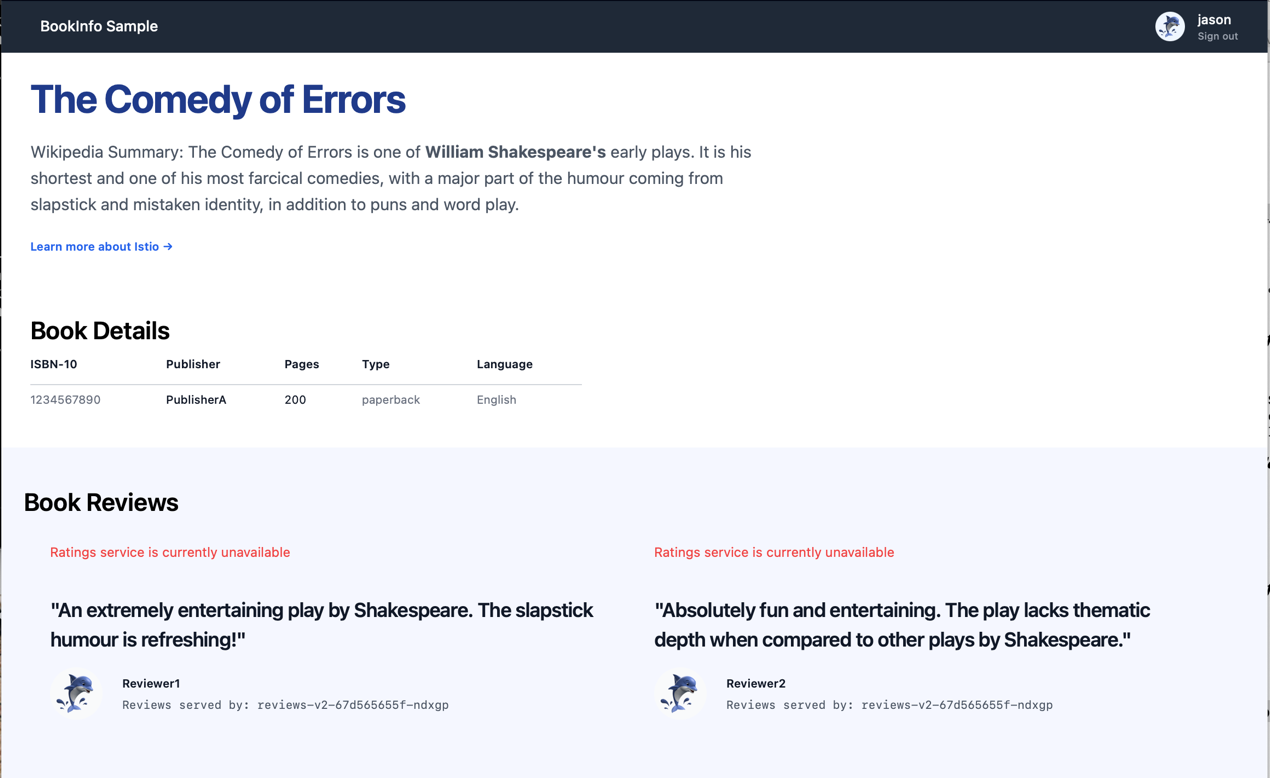


***Figure1:*** ***Configure the yaml file***

******

***Figure2:*** ***Confirm the rules***

(4) Testing the abort configuration



***Figure2:*** ***Rating display "Rating service is currently unavailable"***

(5) Clean up

**Command:**

kubectl delete -f samples/bookinfo/networking/virtual-service-all-v1.yaml

4. Circuit Breaking

(1) Deploy the httpbin service

**Command:**

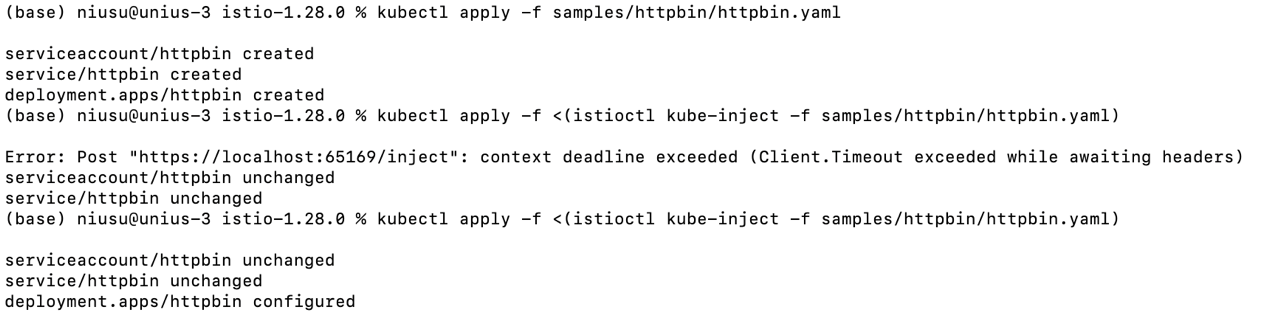
kubectl apply -f samples/httpbin/httpbin.yamlkubectl get services

kubectl get pods -l app=httpbin

**Note:**

This step deploys the httpbin sample service and verifies that it is running correctly.

The istioctl kube-inject command ensures the workload includes the Istio sidecar proxy, enabling traffic management features such as routing, telemetry, and fault injection.

******

***Figure1: Deploy the httpbin service***

(2) Configuring the circuit breaker

**Command:**

kubectl apply -f - <<EOF

apiVersion: networking.istio.io/v1

kind: DestinationRule

metadata:

name: httpbin

spec:

host: httpbin

trafficPolicy:

connectionPool:

tcp:

maxConnections: 1

http:

http1MaxPendingRequests: 1

maxRequestsPerConnection: 1

outlierDetection:

consecutive5xxErrors: 1

interval: 1s

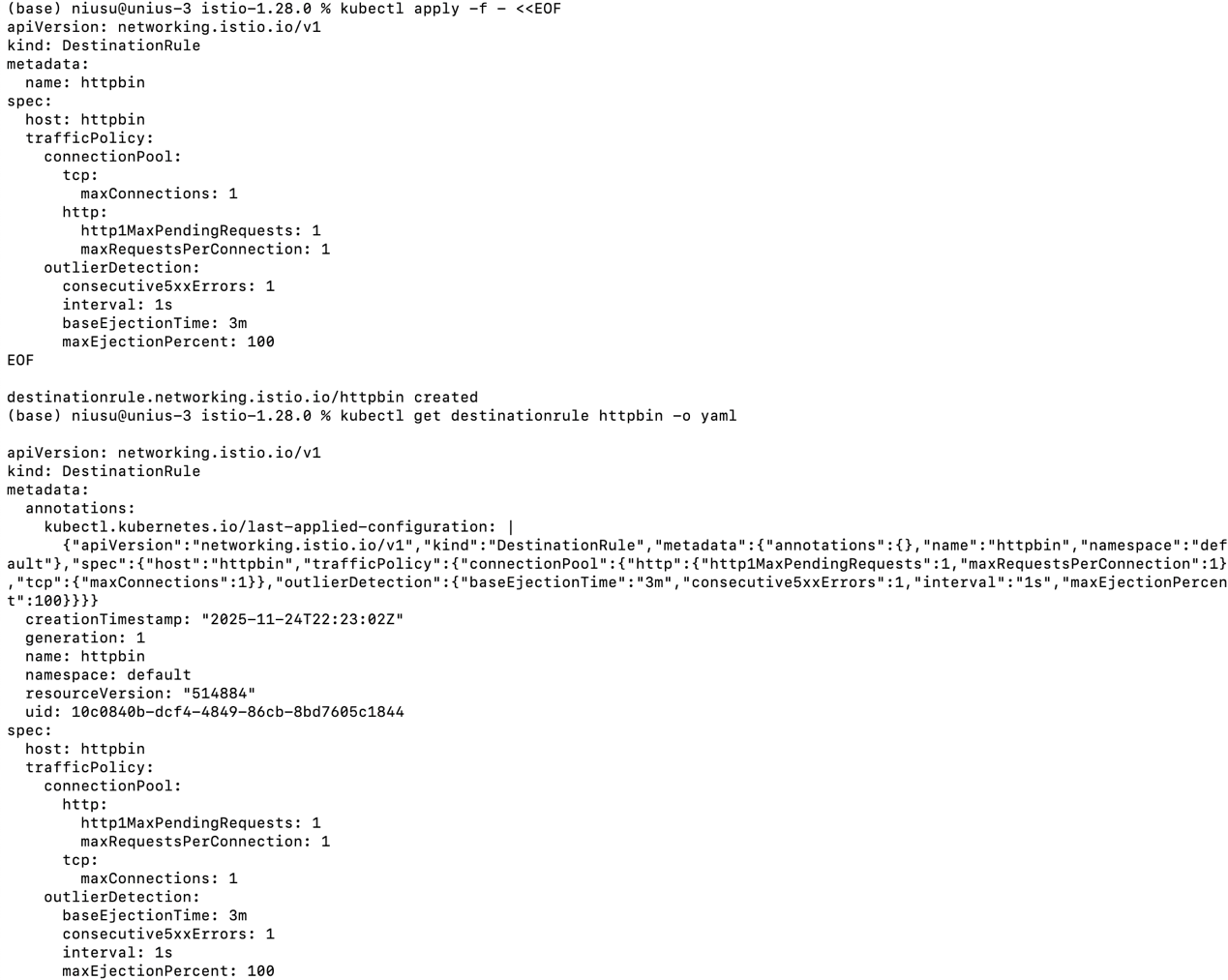
baseEjectionTime: 3m

maxEjectionPercent: 100

EOF

kubectl get destinationrule httpbin -o yaml

**Note:**



***Figure1: Configure & Confirm the destination rule***

(3) Adding a client

**Command:**

kubectl apply -f samples/httpbin/sample-client/fortio-deploy.yaml

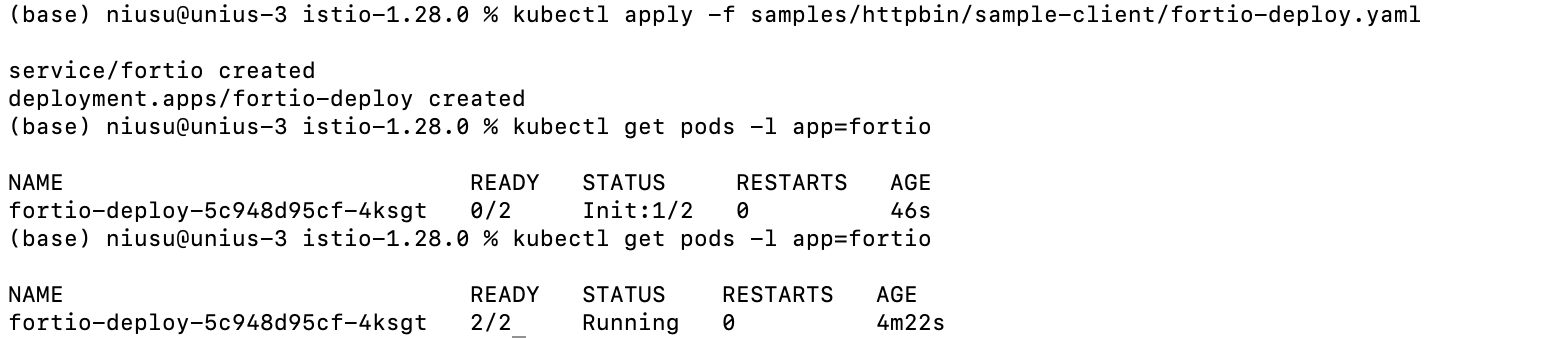
kubectl get pods -l app=fortio

kubectl apply -f <(istioctl kube-inject -f samples/httpbin/sample-client/fortio-deploy.yaml)

export FORTIO\_POD=$(kubectl get pods -l app=fortio -o 'jsonpath={.items[0].metadata.name}')

kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio curl -quiet http://httpbin:8000/get

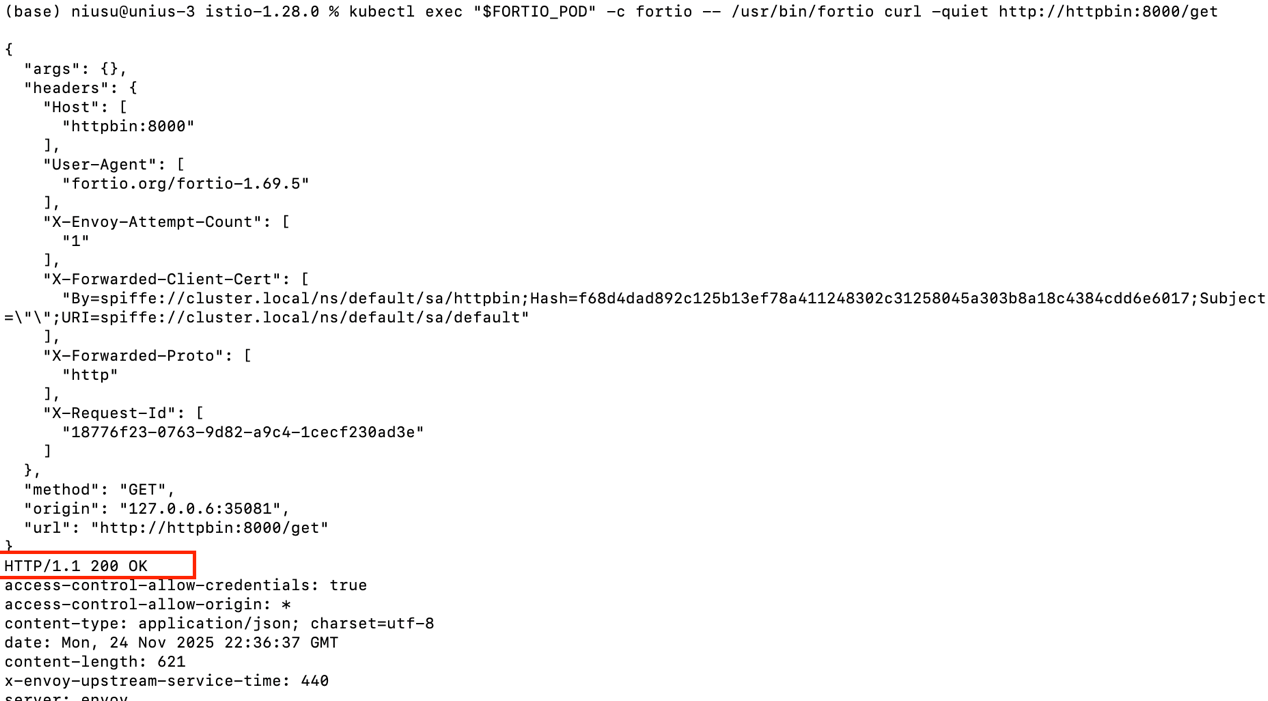
**Note:**



***Figure1: Deploy the Fortio***



***Figure2: Get Fortio Pod name***



***Figure3: Call httpbin - HTTP/1.1 200 OK***

(4) Tripping the circuit breaker

**Command:**

kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio load -c 2 -qps 0 -n 20 -loglevel Warning http://httpbin:8000/get

kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio load -c 3 -qps 0 -n 30 -loglevel Warning http://httpbin:8000/get

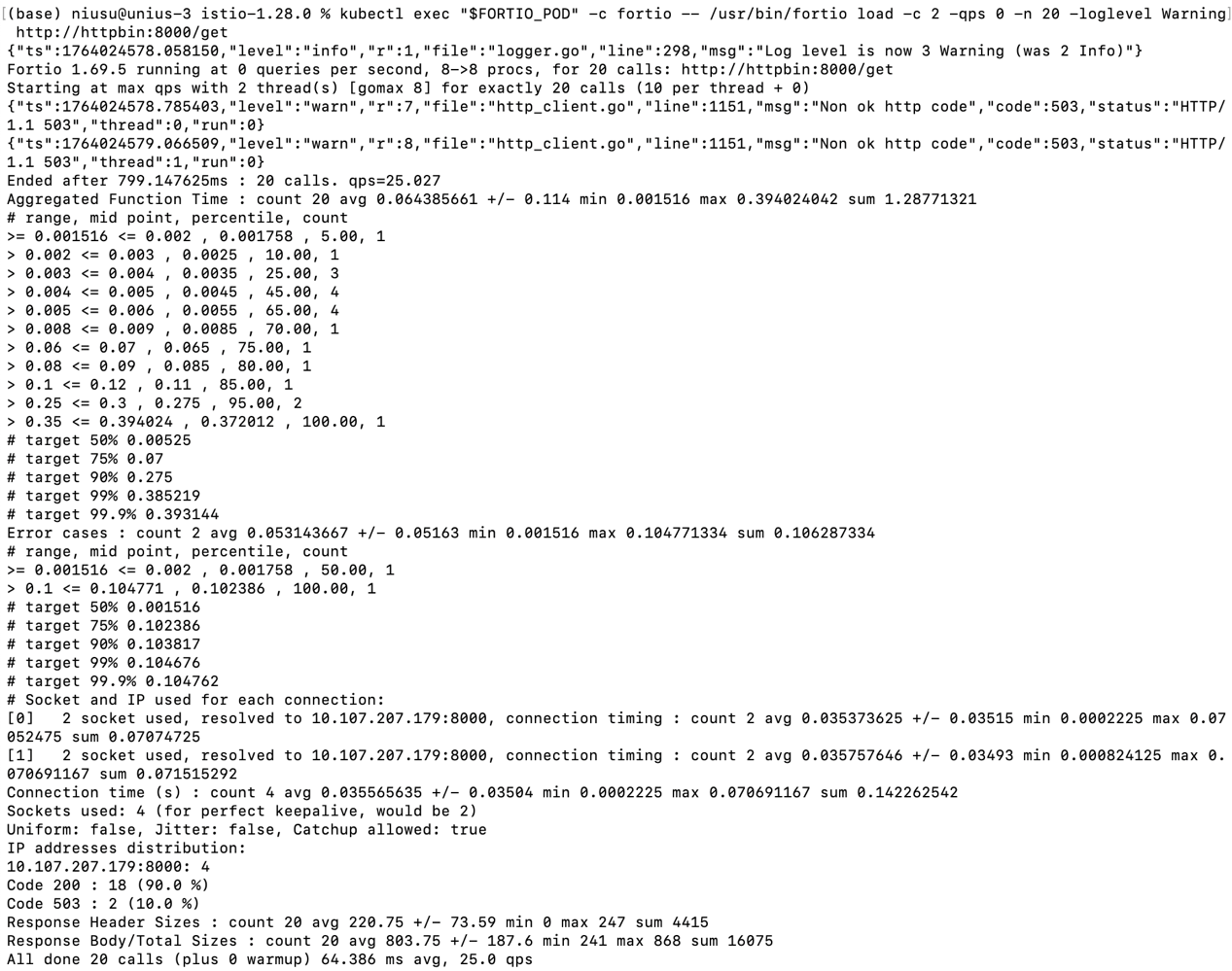
kubectl exec "$FORTIO\_POD" -c istio-proxy -- pilot-agent request GET stats | grep httpbin | grep pending

**Note:**

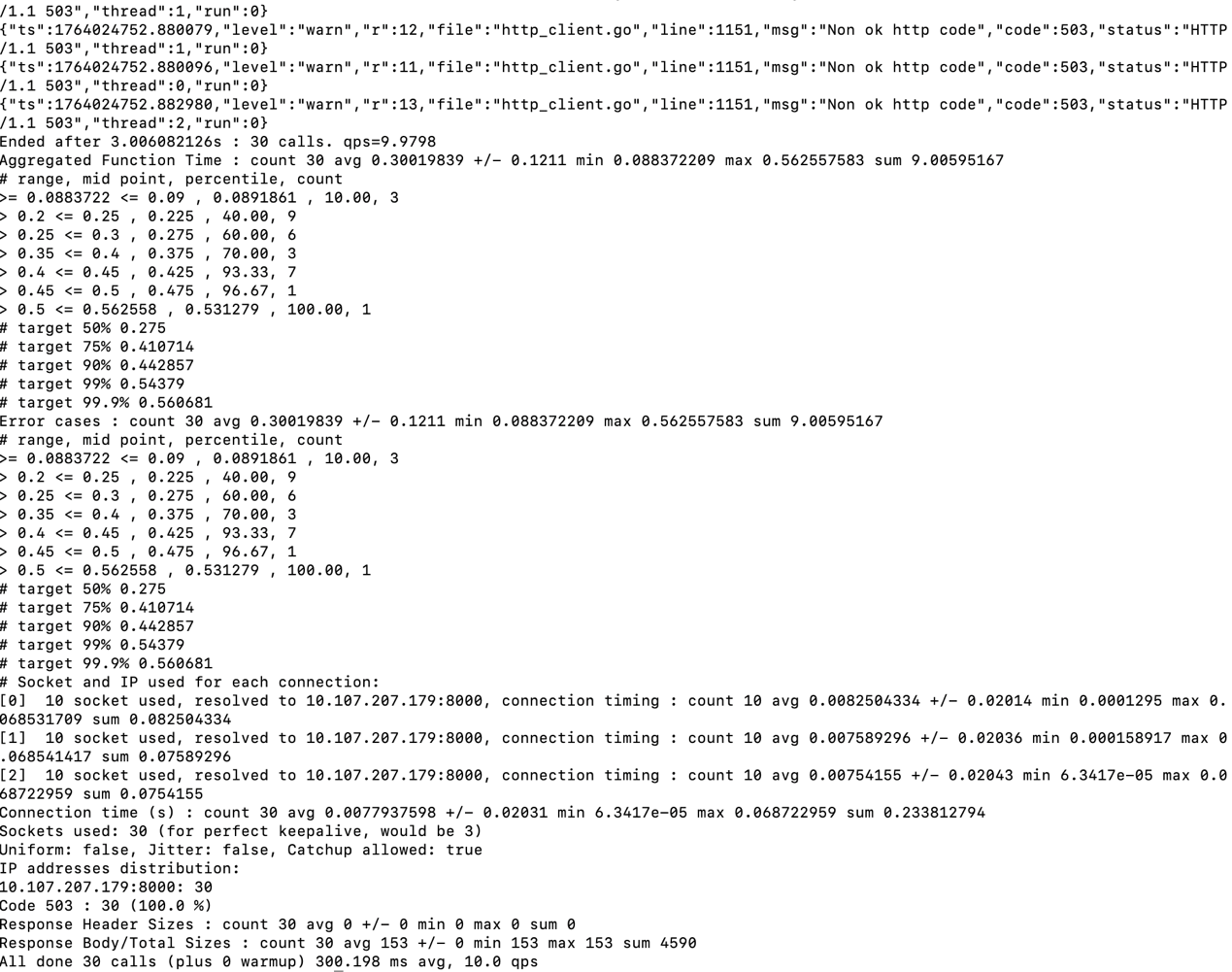
This step uses Fortio to generate increasing concurrent traffic to the httpbin service in order to trigger Istio’s circuit-breaker rules.

At low concurrency, only partial failures occur as the circuit breaker begins limiting requests.  
When concurrency increases further, the circuit breaker fully trips, causing all excess requests to be rejected with 503responses.

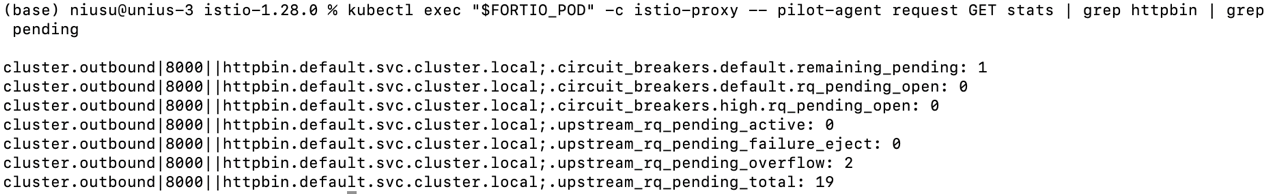
The final statistics output confirms the behavior by showing pending and overflow counters from the Envoy sidecar.

****

***Figure1: Slight Circuit Breaker***

****

***Figure2: Strong Circuit Breaker***

****

***Figure3: Circuit Breaker Statistics***

(5) Clean up

**Command:**

kubectl delete destinationrule httpbin

kubectl delete -f samples/httpbin/sample-client/fortio-deploy.yaml

kubectl delete -f samples/httpbin/httpbin.yaml