Download and Install Istio from here: <https://istio.io/latest/docs/setup/getting-started/>

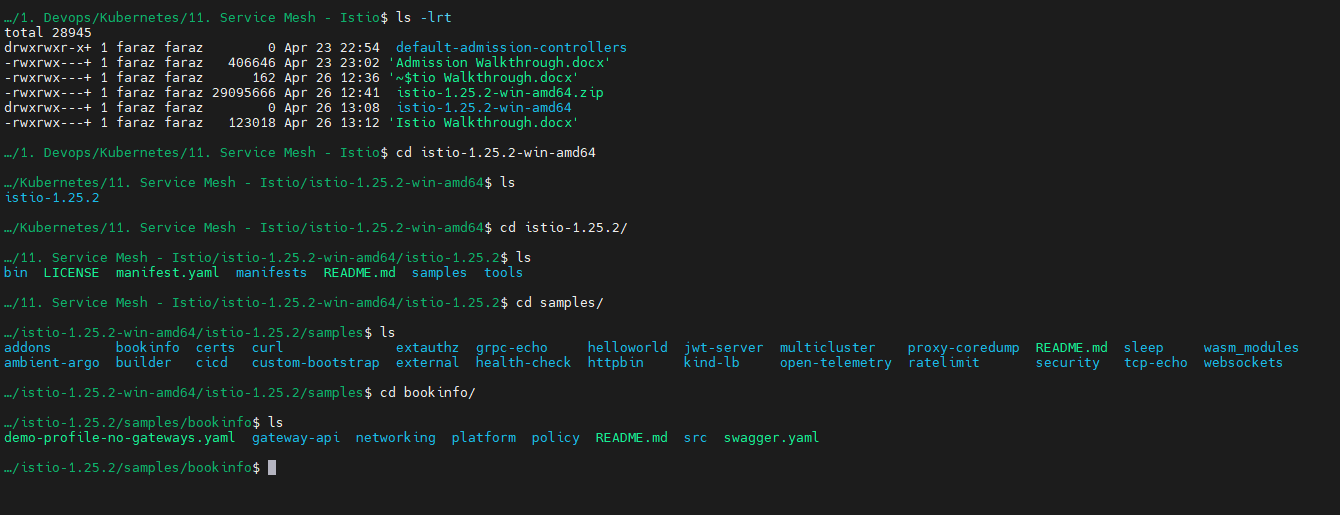
If you can’t curl it then manually download the file from <https://github.com/istio/istio/releases/tag/1.26.0-beta.0>



Unzip the file:



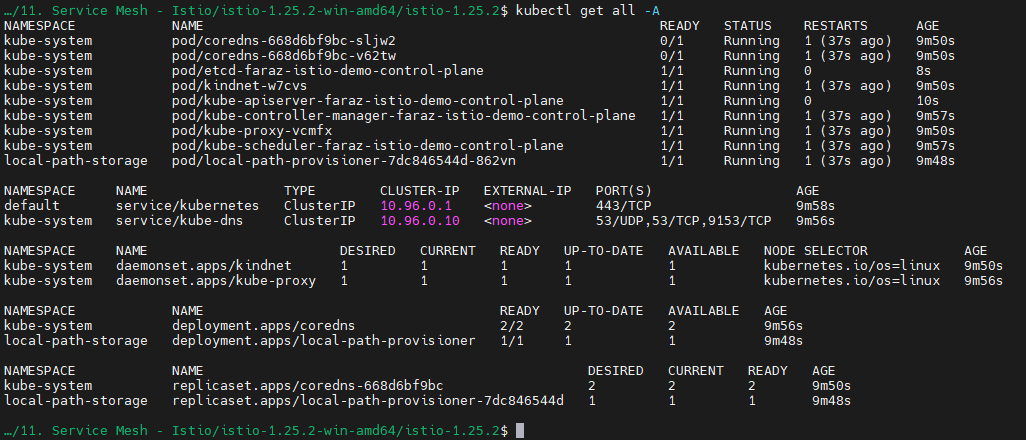
Navigate to the unzipped folder:



Create your cluster:

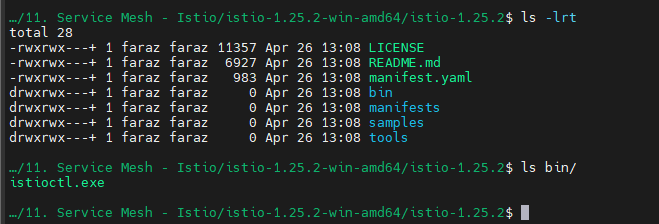
* kind create cluster --name faraz-istio-demo

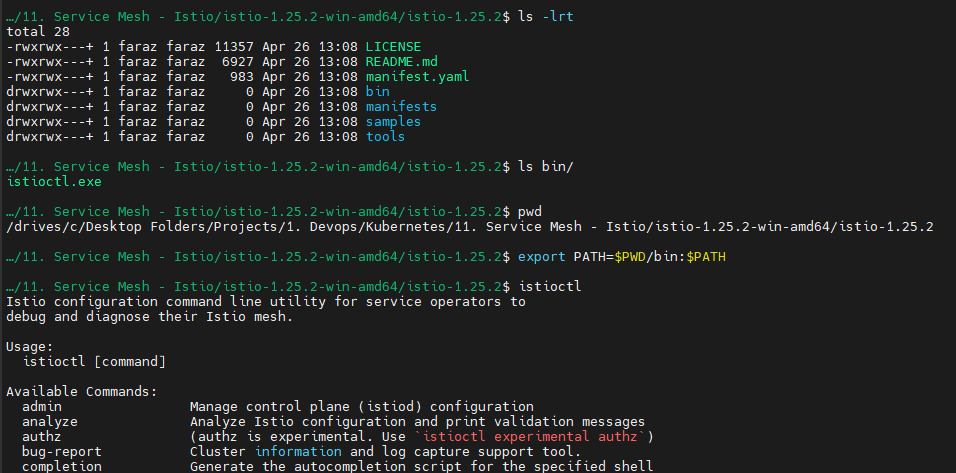




Now we will set the istioctl path:

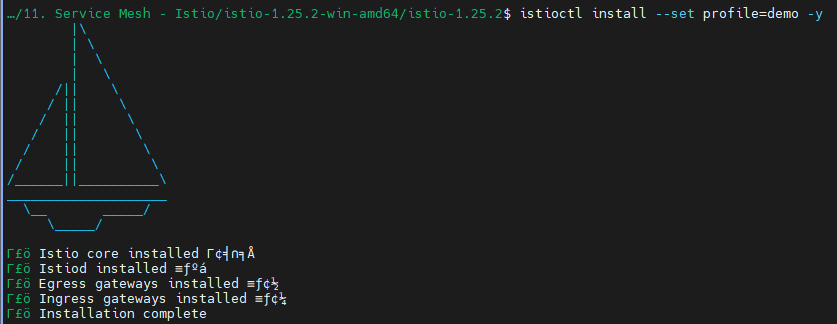
* export PATH=$PWD/bin:$PATH





Apply the demo profile:

* istioctl install --set profile=demo -y

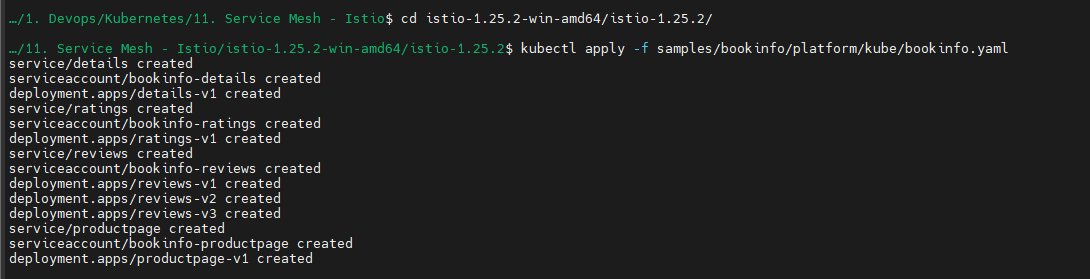


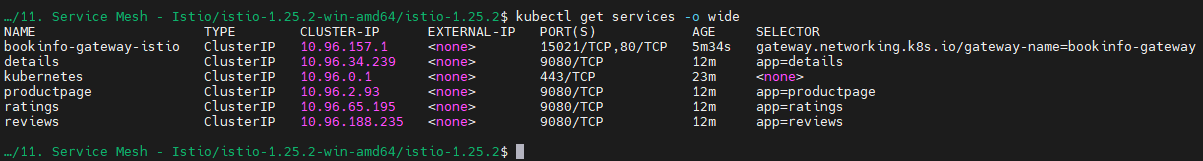
Enable istio injection in default namespace:

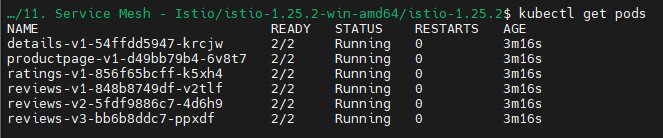
* kubectl label namespace default istio-injection=enabled



* kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml



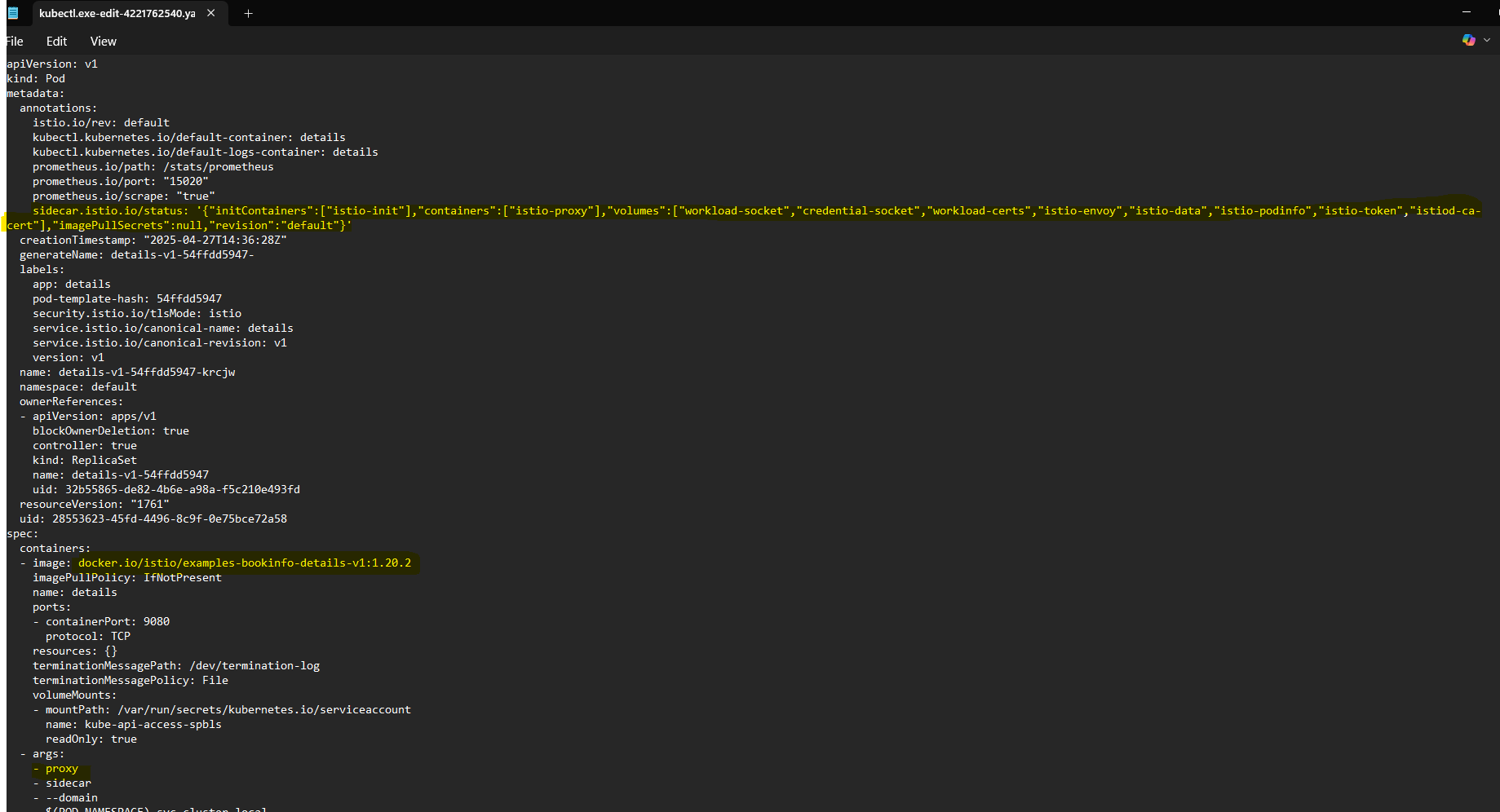




* kubectl edit pod details-v1-54ffdd5947-krcjw



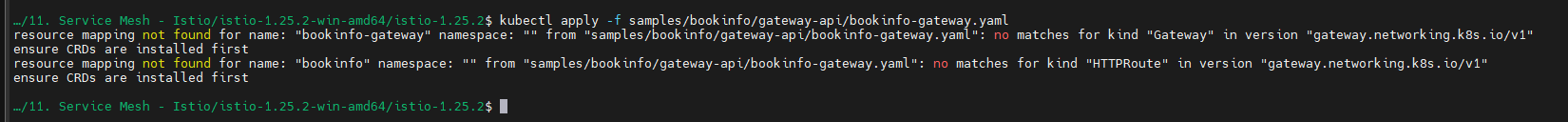
You will see the main container and sidecar container:



Now configure gateway-api:

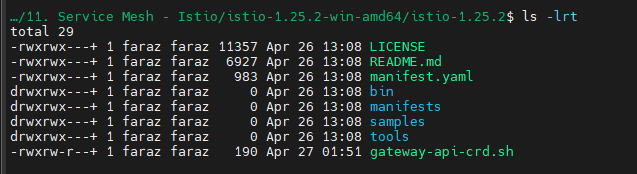
* kubectl apply -f samples/bookinfo/gateway-api/bookinfo-gateway.yaml

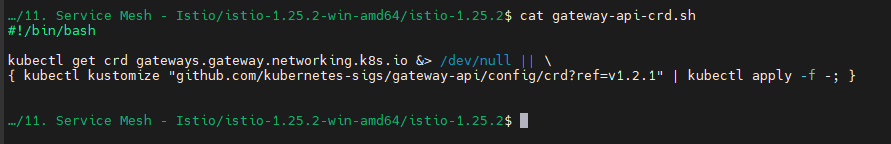
We may get the following error:



To solve this error:

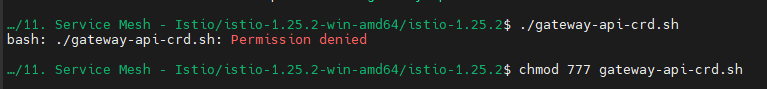


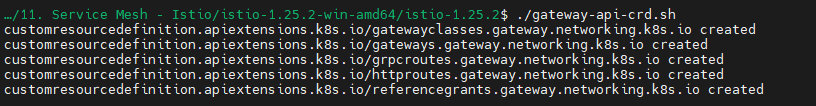




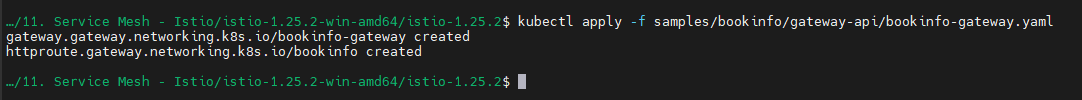
kubectl get crd gateways.gateway.networking.k8s.io &> /dev/null || \

{ kubectl kustomize "github.com/kubernetes-sigs/gateway-api/config/crd?ref=v1.2.1" | kubectl apply -f -; }



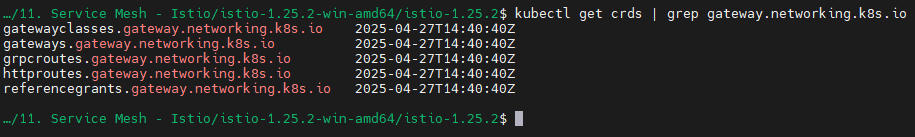


The error will be now solved:

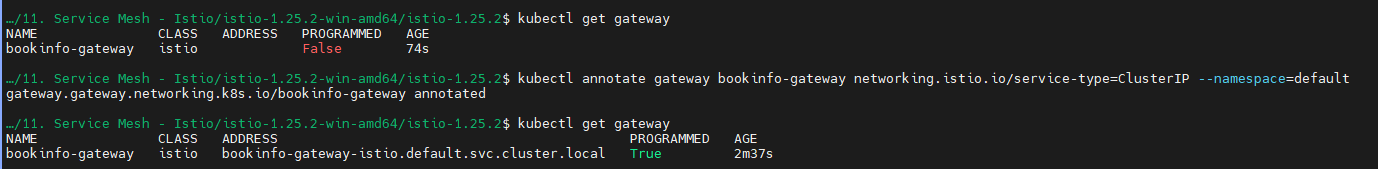


Ensure that CRD is defined:

* kubectl get crds | grep gateway.networking.k8s.io

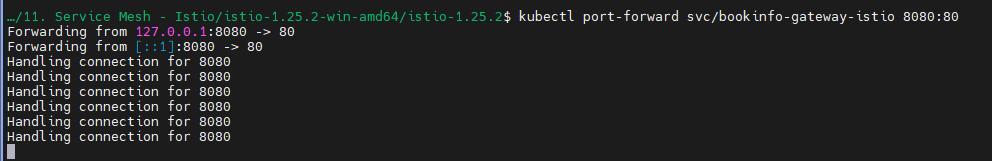


* kubectl annotate gateway bookinfo-gateway networking.istio.io/service-type=ClusterIP --namespace=default
* kubectl get gateway

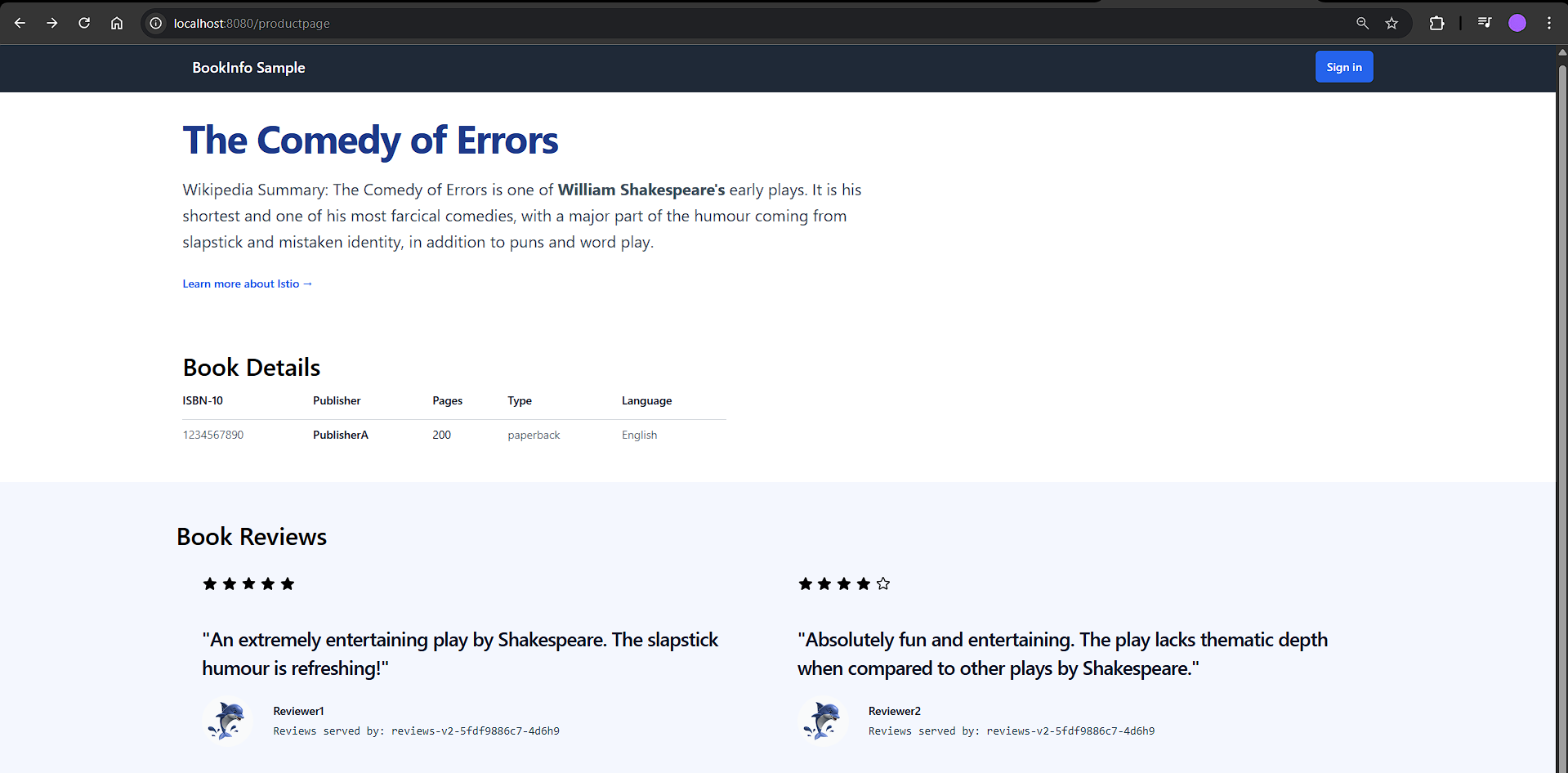


Forward the port:

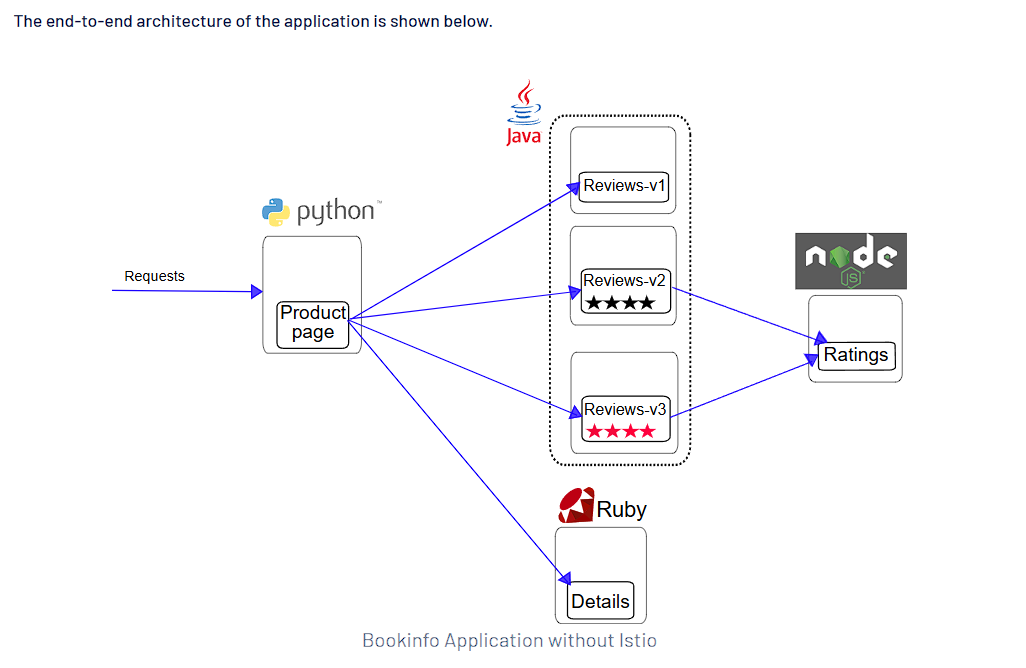
* kubectl port-forward svc/bookinfo-gateway-istio 8080:80



Now open: <http://localhost:8080/productpage>



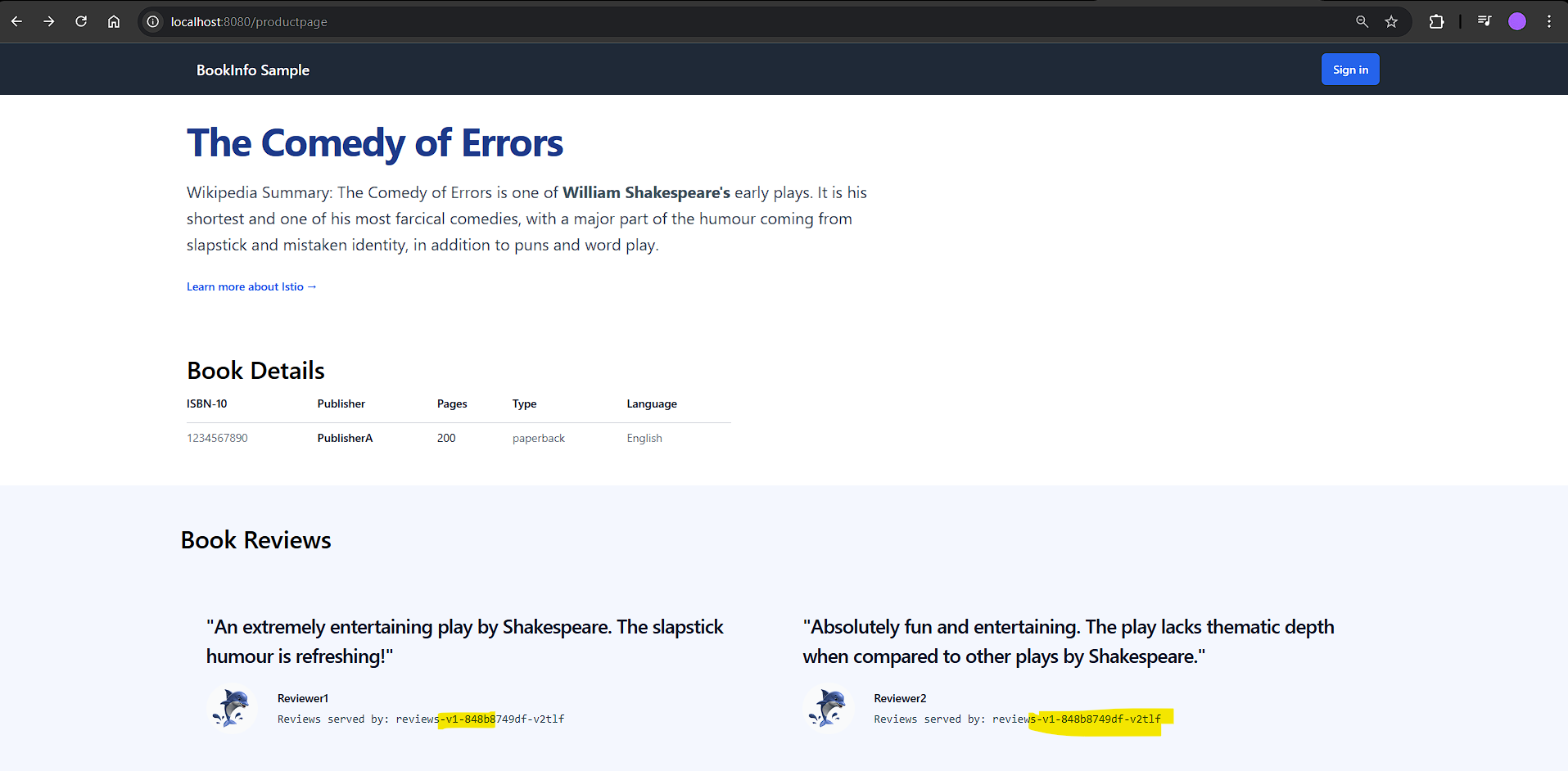
You will be able to see your book application



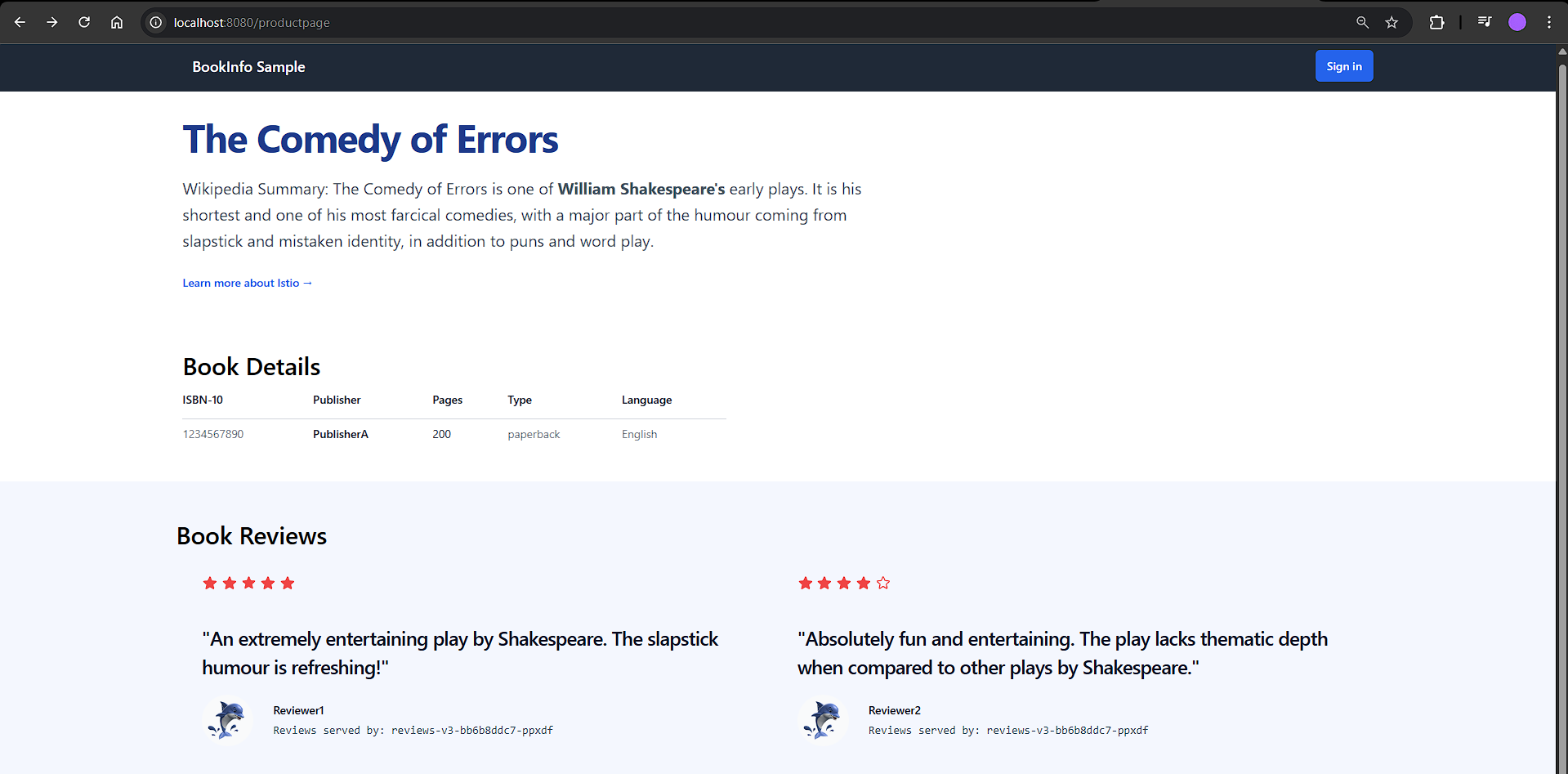
If you keep refreshing, you will see different versions of the application:

V1:

Reviews V1 is not connected to the ratings microservice



V3:

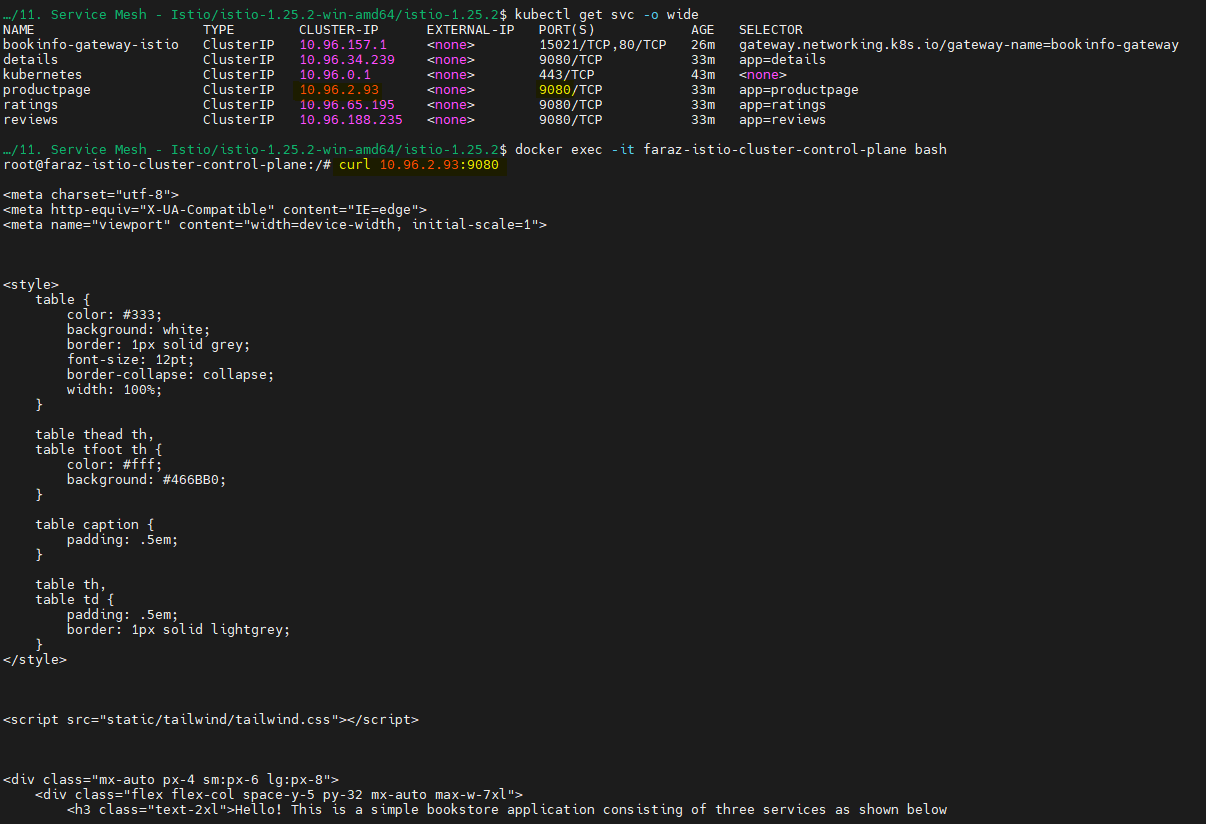


Now lets’ explore Istio features:

1. mTLS:

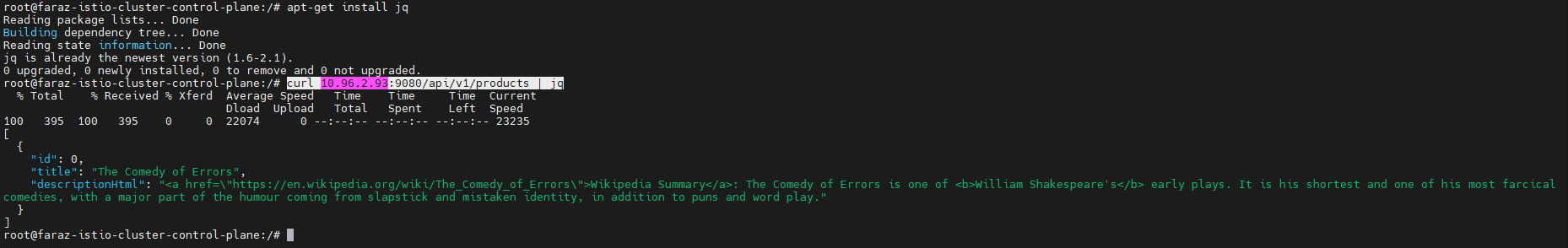
Since it currently we have the services in clusterIP mode, we will have to login to the node and curl on the cluster-ip:port

- docker exec -it faraz-istio-cluster-control-plane bash



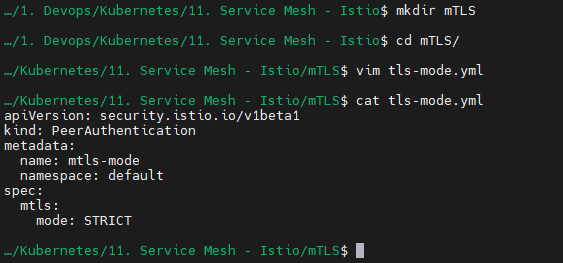
You can also hit the apis:

* For formatting the output: apt-get install jq
* curl 10.96.2.93:9080/api/v1/products | jq



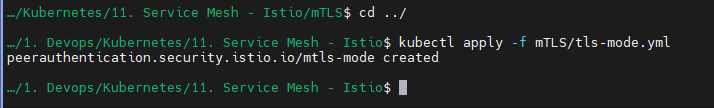
We are able to access the services in istio as by default istio runs in permissive mode.

To enable Strict mode:

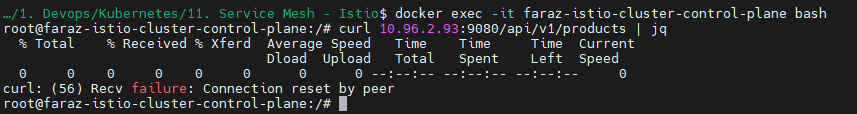


We will then apply the TLS mode:

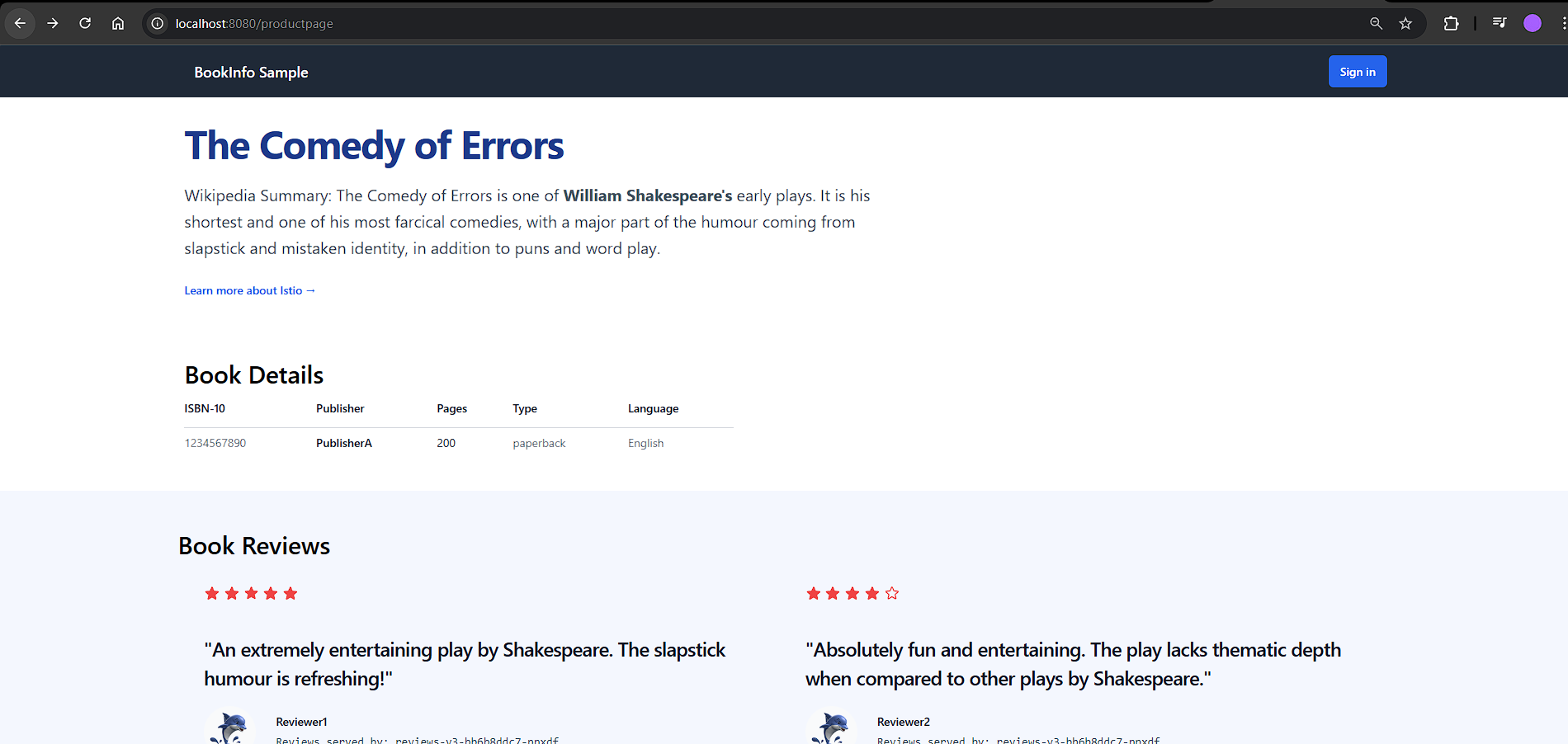
* kubectl apply -f mTLS/tls-mode.yml



Now you will see failure in the connection as we tried the connect without a certificate:



But my services are able to communicate with each other internally, as my application is working as expected:

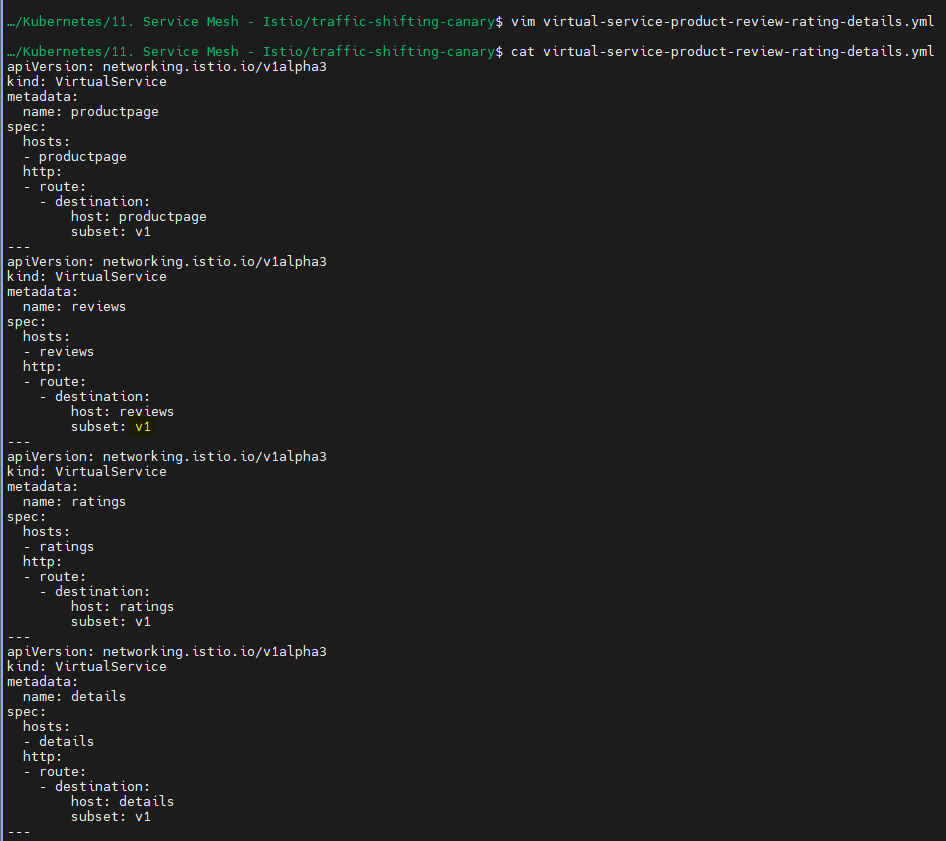


2. Virtual Services/Destination Rules to implement Canary:

* A Virtual Service in Istio is like giving a GPS to your traffic. It tells the requests where to go, based on rules like the URL path, headers, or other conditions. For example, you can say, "Send 80% of traffic to version 1 of my app and 20% to version 2."
* A Destination Rule is more like setting the ground rules for how the traffic behaves once it gets there. It manages things like how the traffic is balanced between app instances, whether secure communication (like mTLS) is used, or if unhealthy instances should be avoided.

They work together to make sure traffic goes to the right place and behaves the way you want. Like, "Route traffic to the right version of my app (Virtual Service), and make sure it's securely connected and balanced (Destination Rule)."

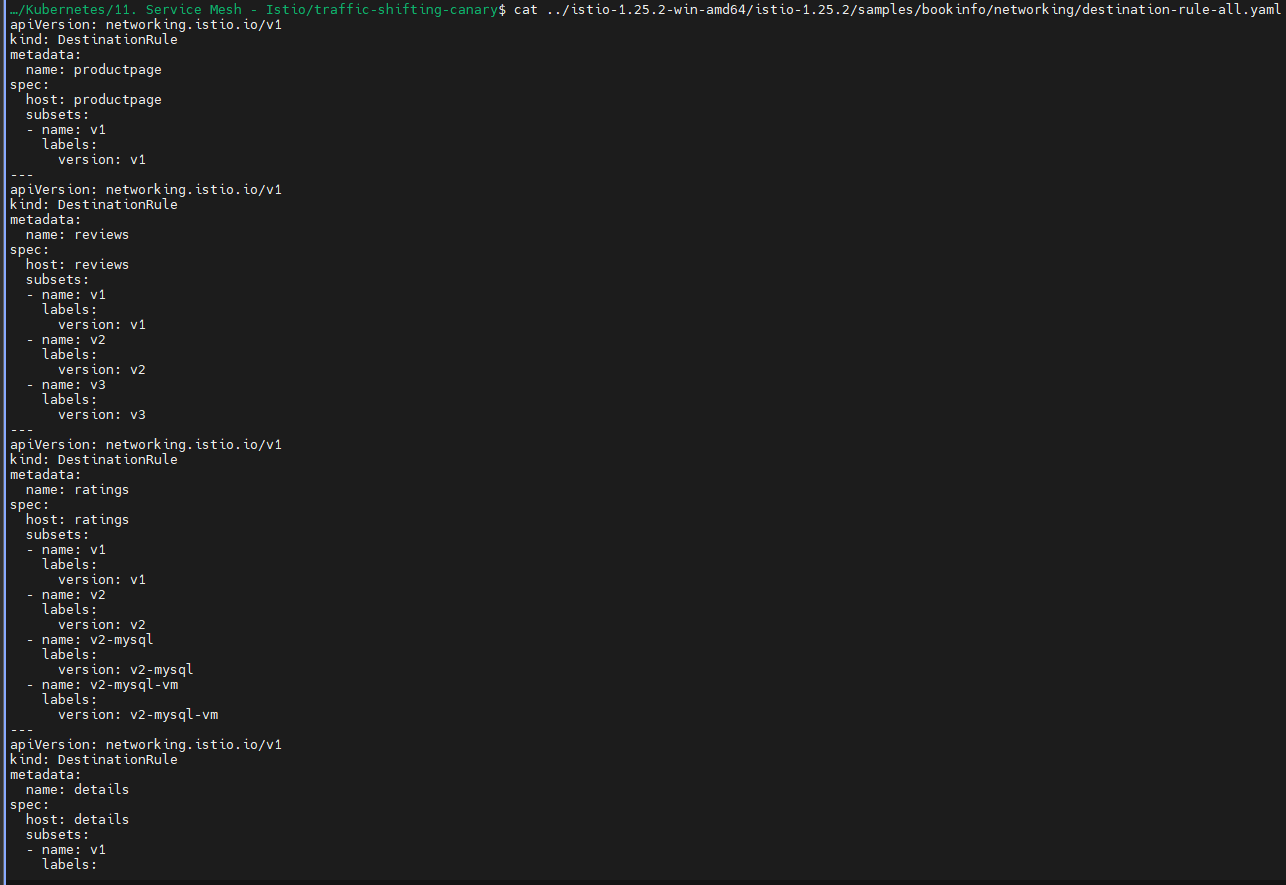
- Create virtual service:



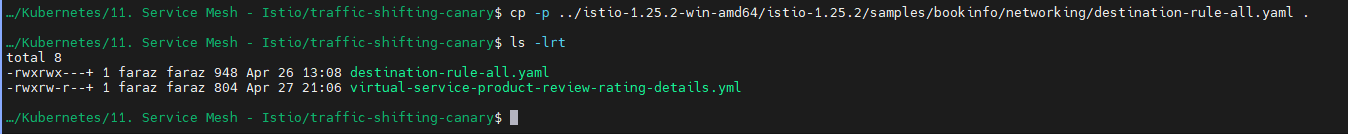
Here we are defining that the request must only go to reviews v1(which is not connected to ratings)

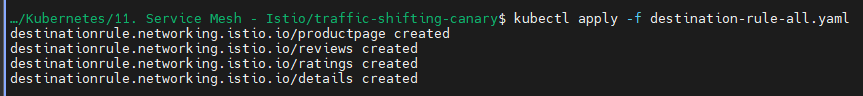
Apply the destination rule:

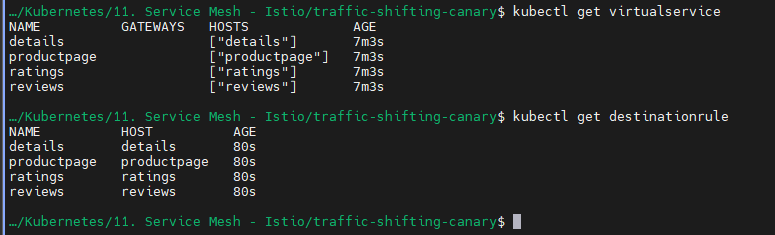
* cp -p ../istio-1.25.2-win-amd64/istio-1.25.2/samples/bookinfo/networking/destination-rule-all.yaml .



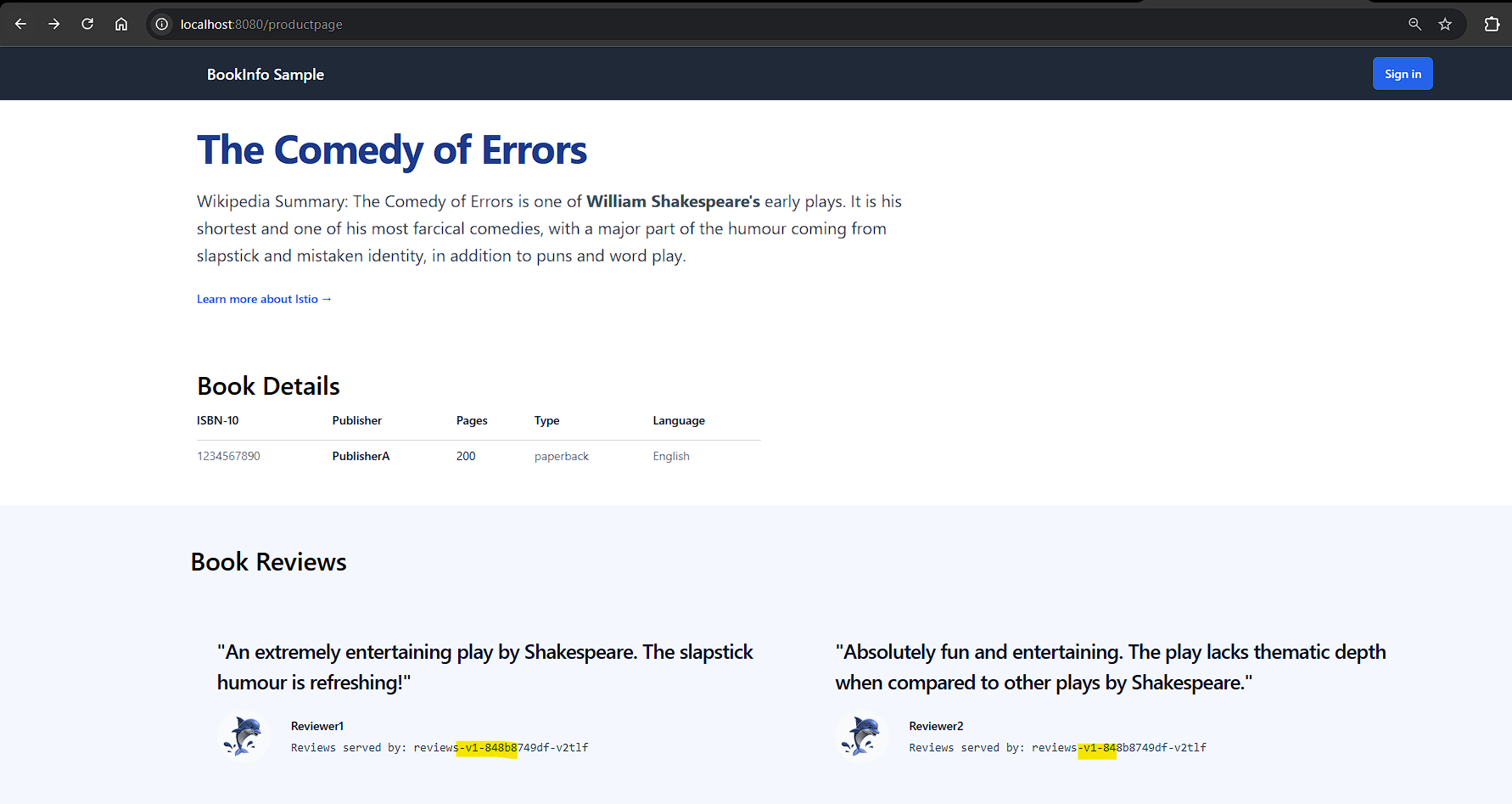
* kubectl apply -f destination-rule-all.yaml





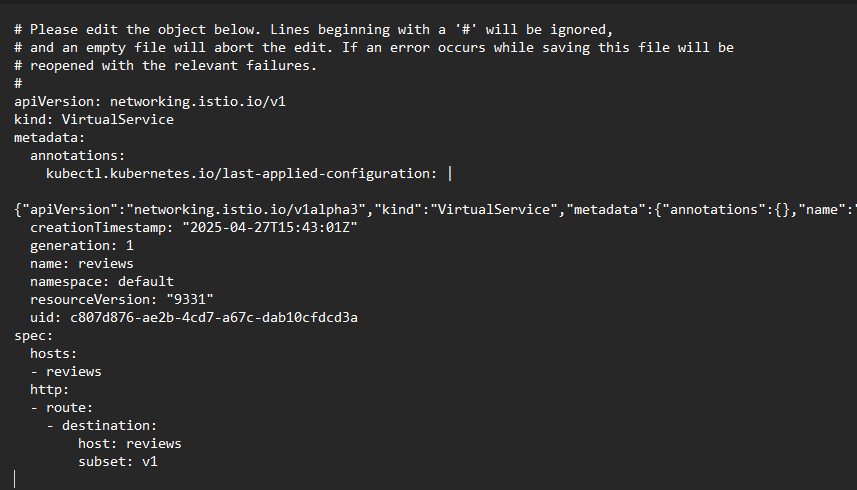


Since in Virtual service we mentioned V1, even on refreshing 1000s of time you will get V1:

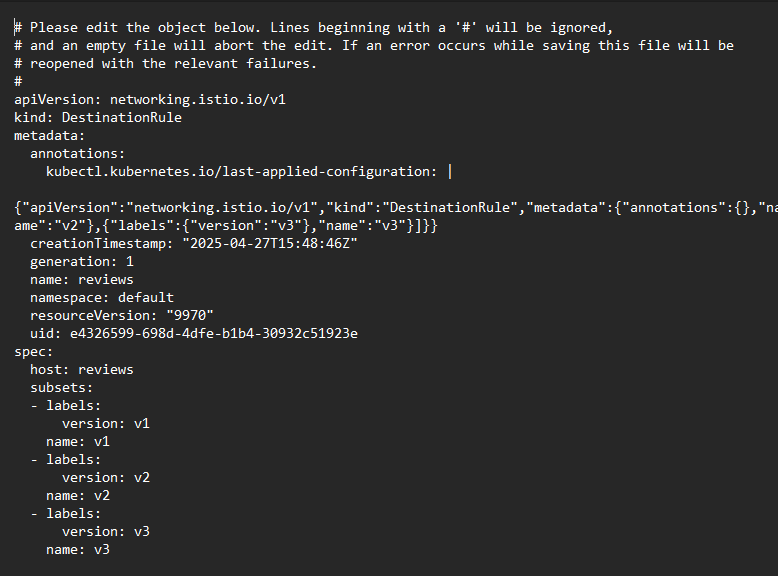


You can also view the virtual service for reviews:

* kubectl edit vs reviews

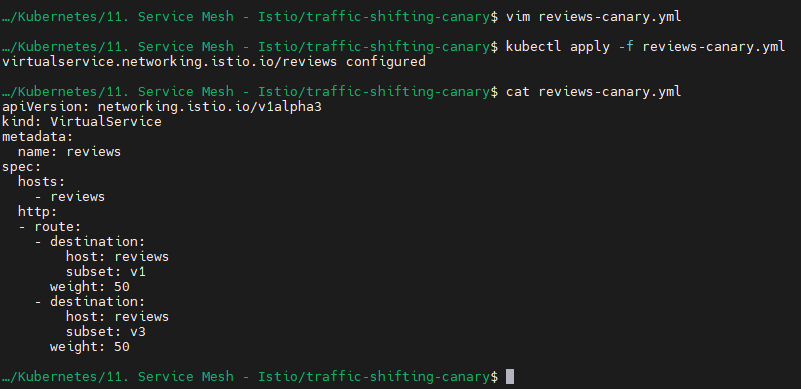


* kubectl edit dr reviews

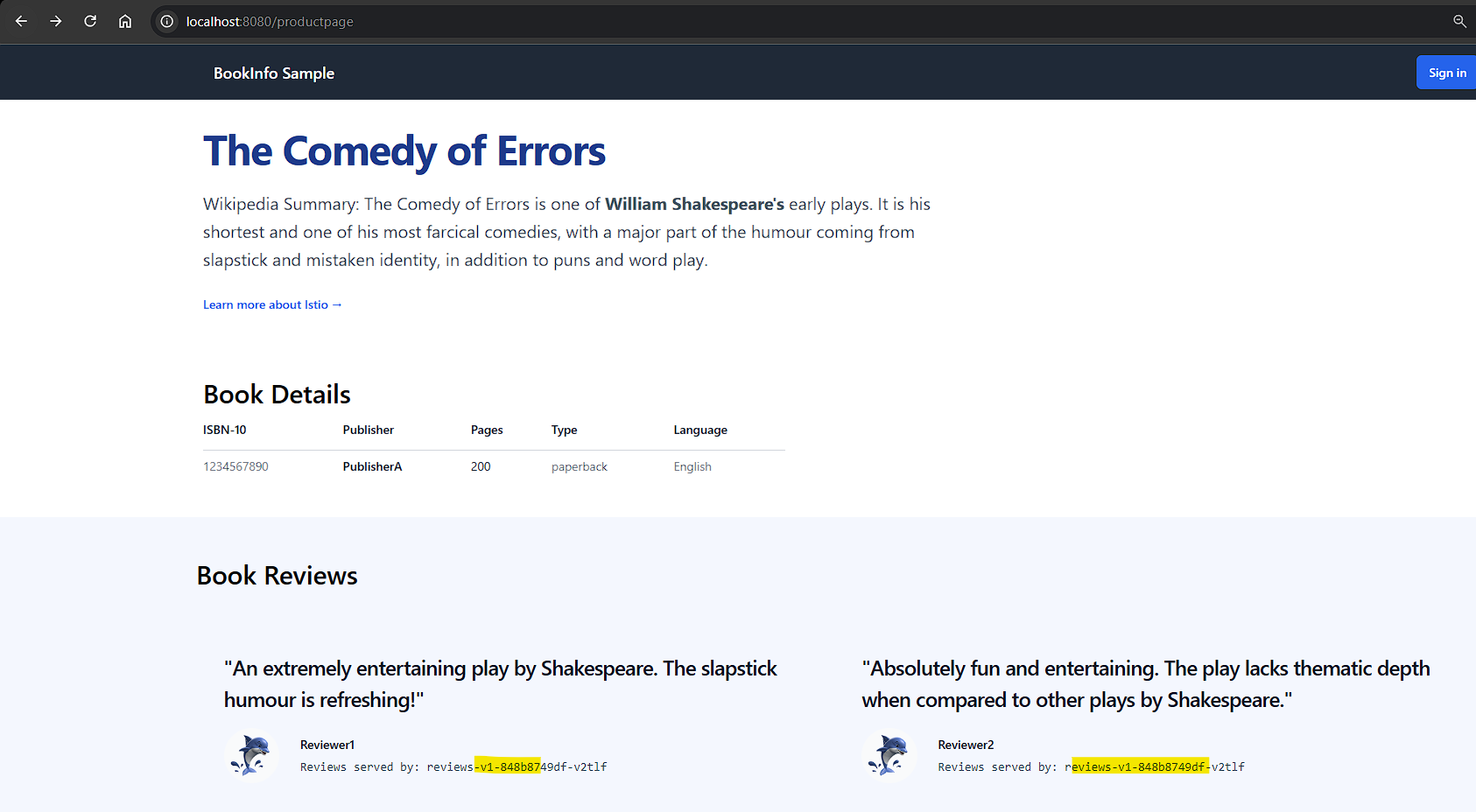


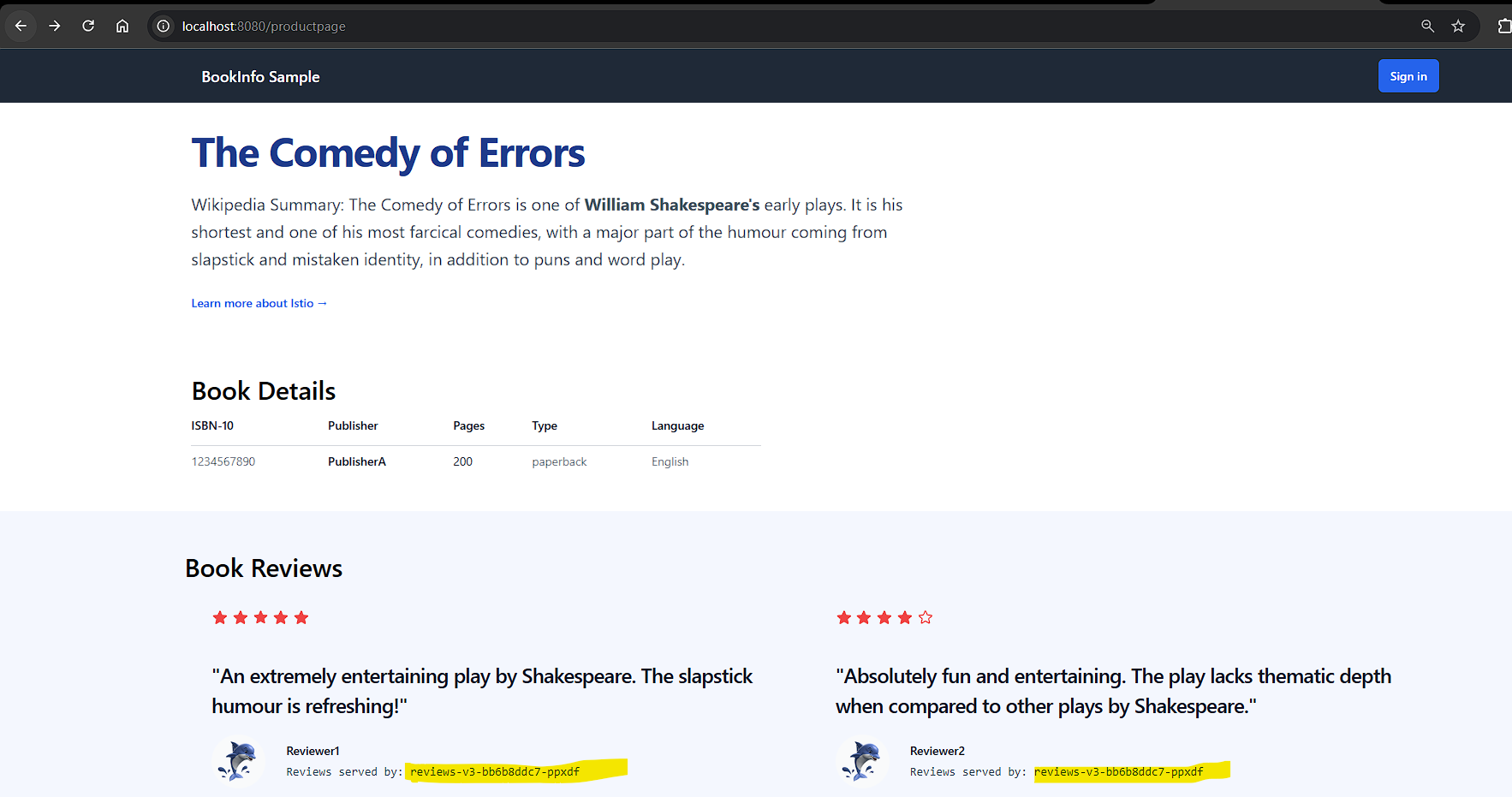
Now if you want to implement traffic shifting/canary:

Here we will direct 50%(We want to test if v3 works well) of request v1 and v3:

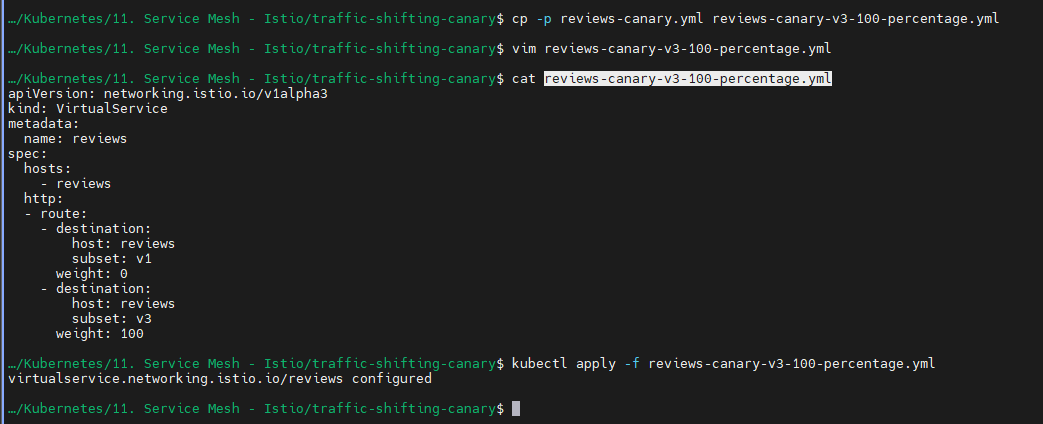


After applying, you can test it:





Once we are confident that v3 is as per the expectation, we can roll out 100% traffic to v3:



Now the 100% traffic will be redirected to V3:

