**DOCKER & KUBERNETES - ISTIO ON EKS**

**Pre-Requisites:**

* Install EKS
* Docker

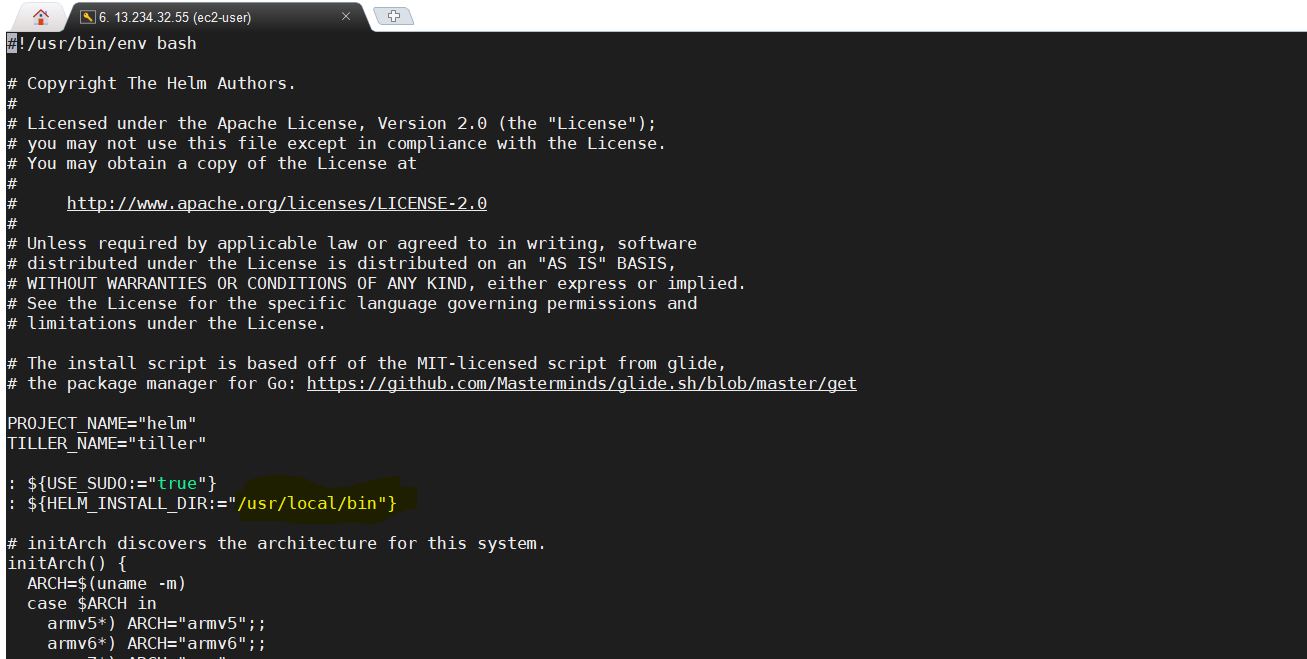
**Install Helm:**

Here we can install Helm by using script:

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get\_helm.sh

Open get\_helm.sh file and change below step:

vi get\_helm.sh



Repalce /usr/local/bin/helm with /usr/bin/helm.., Please check below image where we need to change exactly

Give Execution permission for file

chmod +x get\_helm.sh

Run script by using below command

./get\_helm.sh



Note: Once we install helm, the command will prompt us to run 'helm init'. Do not run 'helm init'. Follow the instructions to configure helm using Kubernetes RBAC and then install tiller as specified below If accidentally run 'helm init, we can safely uninstall tiller by running 'helm reset –force'

Configure Helm access with RBAC

Helm relies on a service called tiller that requires special permission on the kubernetes cluster, so we need to build a Service Account for tiller to use. We'll then apply this to the cluster. We'll do this while we install istio to EKS cluster.

Install Istio:

Download the Istio chart

https://github.com/istio/istio/releases/download/1.1.2/istio-1.1.2-linux.tar.gz

tar xvzf istio-1.1.2-linux.tar.gz

Helm relies on tiller that requires special permission on the kubernetes cluster, so we need to build a Service Account for tiller to use. We'll then apply this to the cluster.

We'll be using the following service account manifest (./install/kubernetes/helm/helm-service-account.yaml):

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: tiller

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: tiller

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: tiller

namespace: kube-system

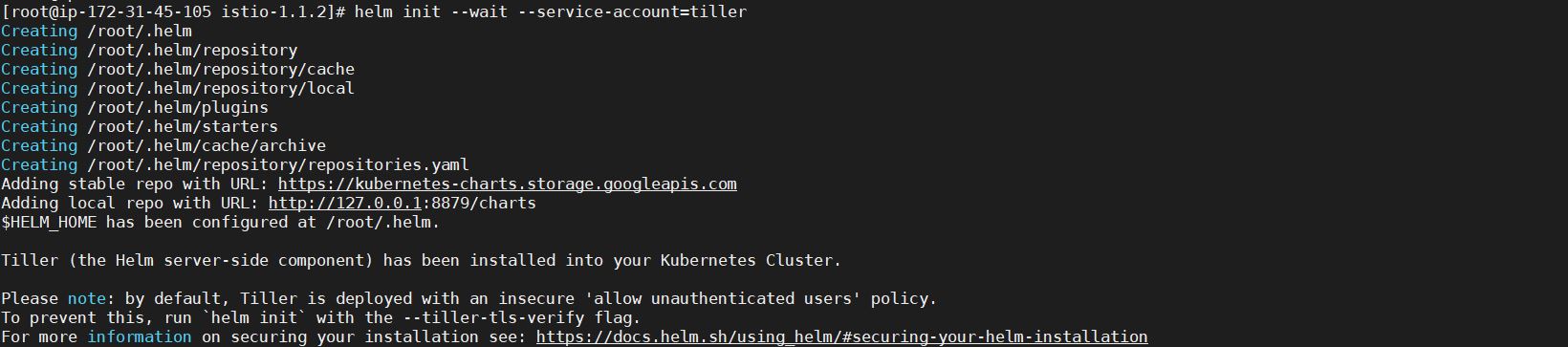
Let's install Tiller on the EKS cluster using the manifest. We want to make sure we have a service account with the cluster-admin role defined for Tiller.

cd istio-1.1.2/

kubectl create --filename=./install/kubernetes/helm/helm-service-account.yaml



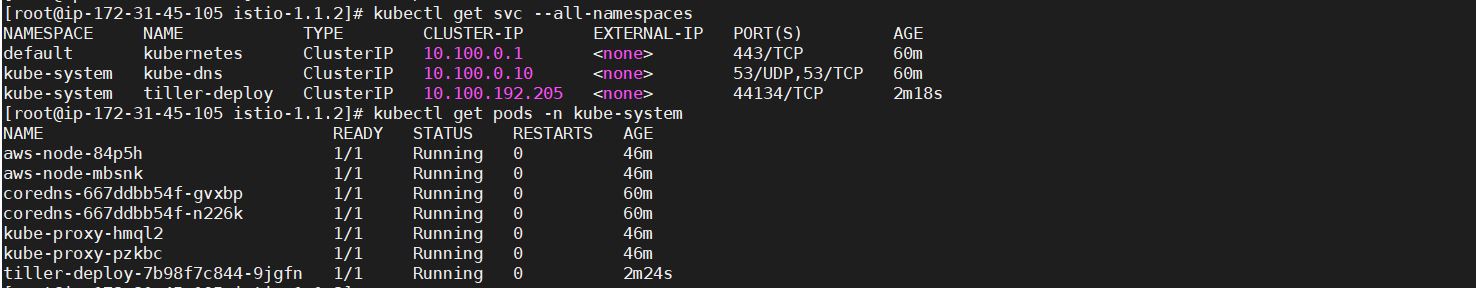
helm init --wait --service-account=tiller



Let's see what we have now:

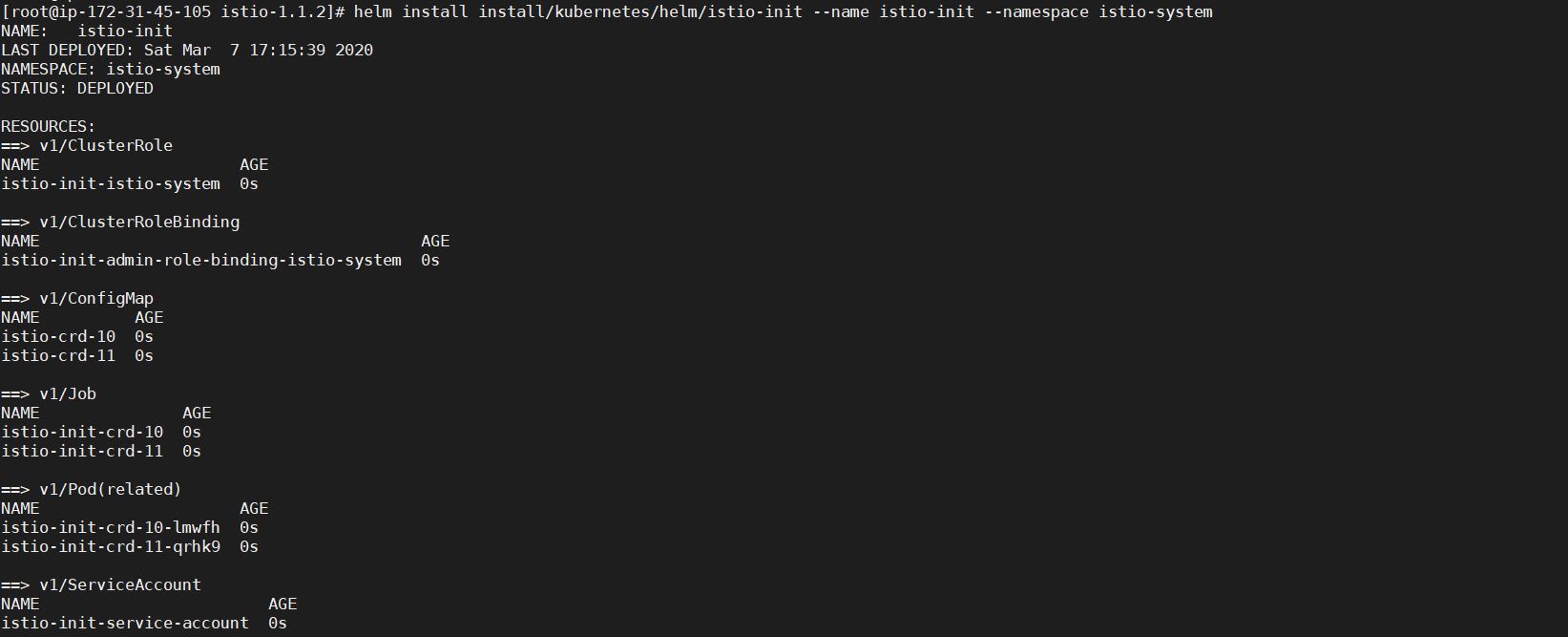
kubectl get svc --all-namespaces

kubectl get pods -n kube-system



Install the istio-init chart to bootstrap all the Istio's CRDs:

helm install install/kubernetes/helm/istio-init --name istio-init --namespace istio-system



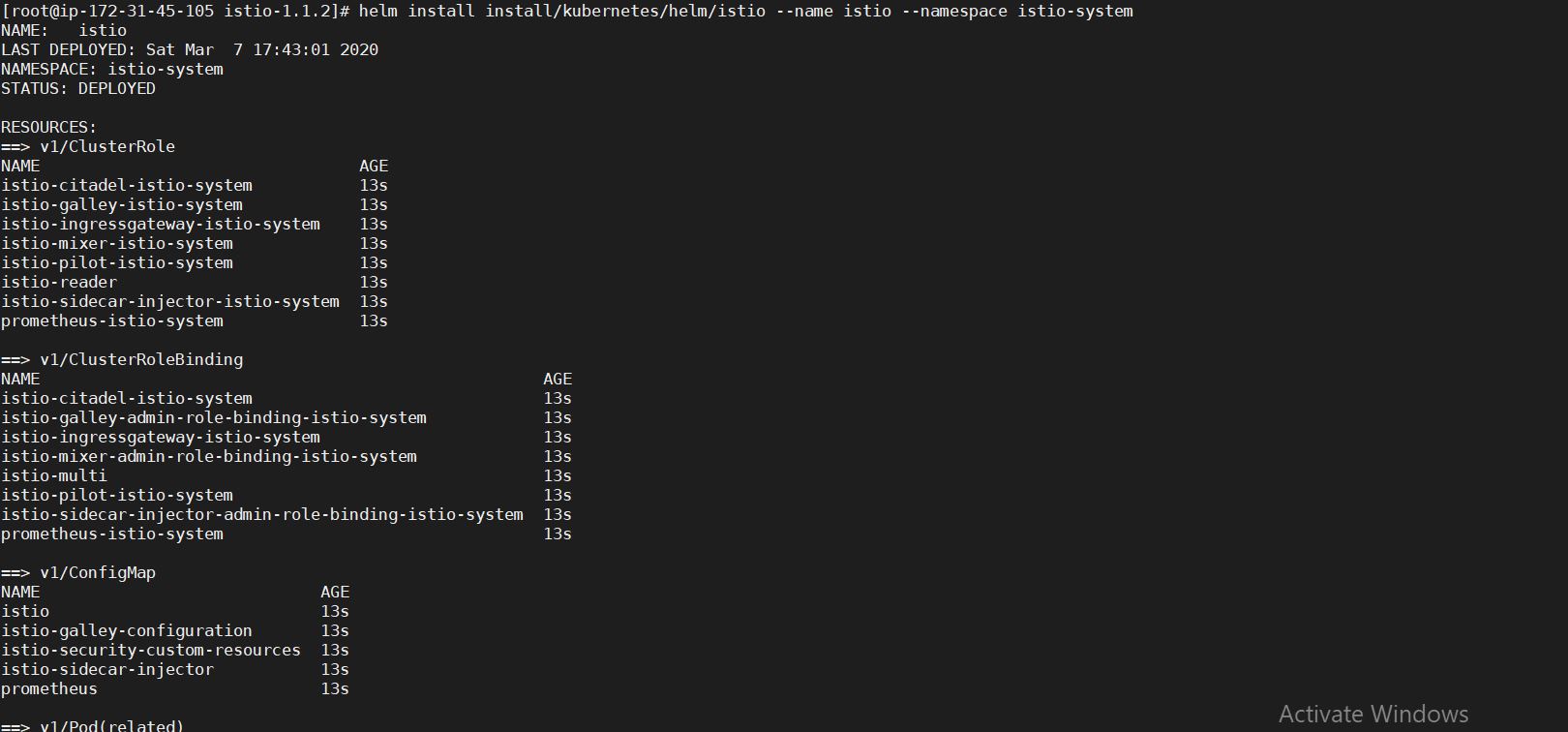
Verify that all 53 Istio CRDs were committed to the Kubernetes api-server using the following command:

kubectl get crds | grep 'istio.io\|certmanager.k8s.io' | wc -l

D:\Naresh\snipping\Capture.JPG

Install the istio chart:

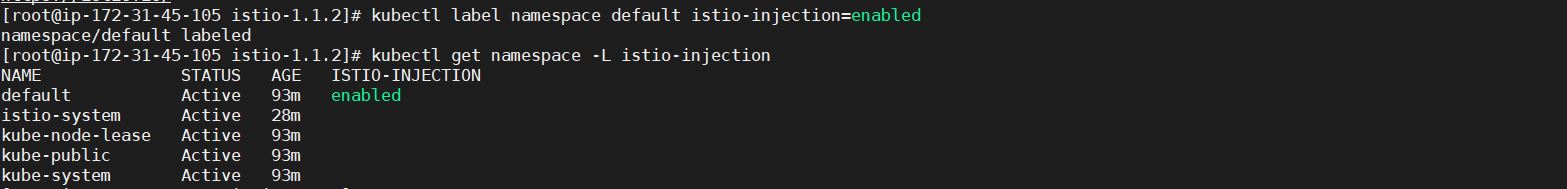
helm install install/kubernetes/helm/istio --name istio --namespace istio-system



To get started running application with Istio, we need to label namespace that application object will be deployed to by the following command (take default namespace as an example):

kubectl label namespace default istio-injection=enabled

kubectl get namespace -L istio-injection

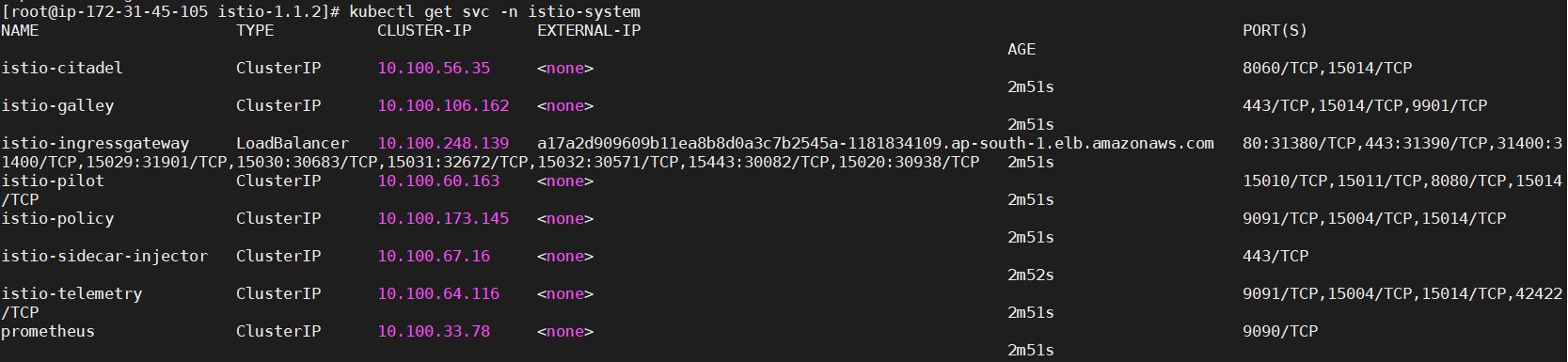


We can check the installation by running:

kubectl get crds | grep 'istio.io'

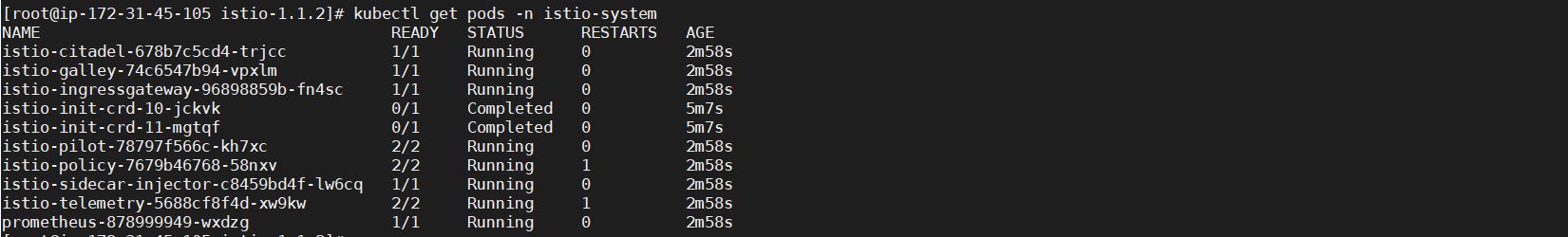
We can verify that the services have been deployed using:

kubectl get svc -n istio-system



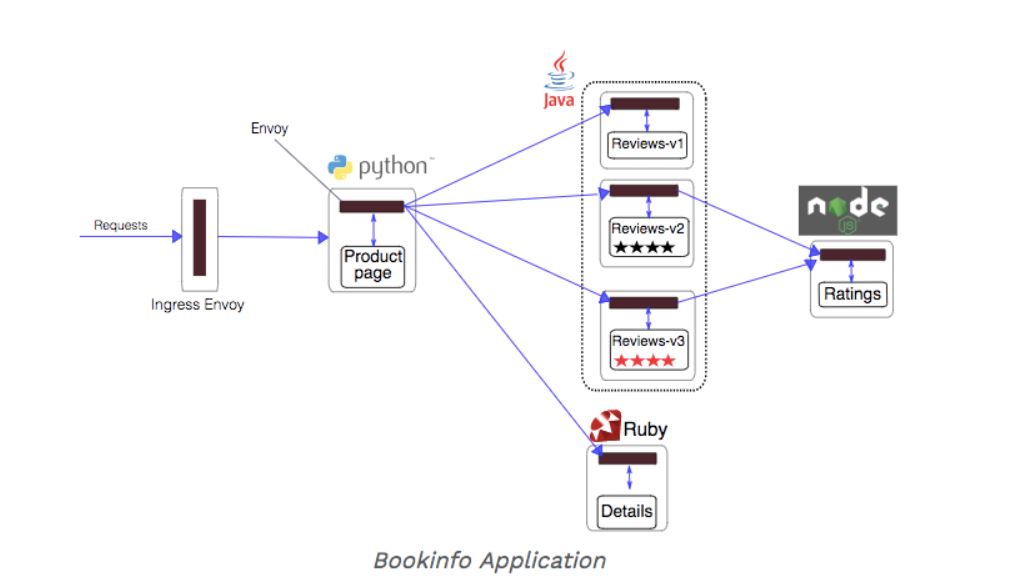
and check the corresponding pods with:

kubectl get pods -n istio-system



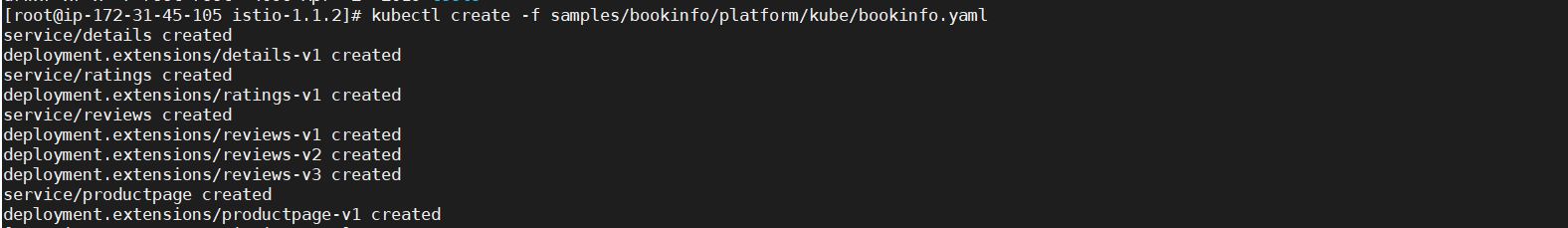
Deploy sample app:

Now that we have all the resources installed for Istio, we will use sample application called "BookInfo" to review key capabilities of the service mesh such as intelligent routing, and review telemetry data using Prometheus & Grafana.



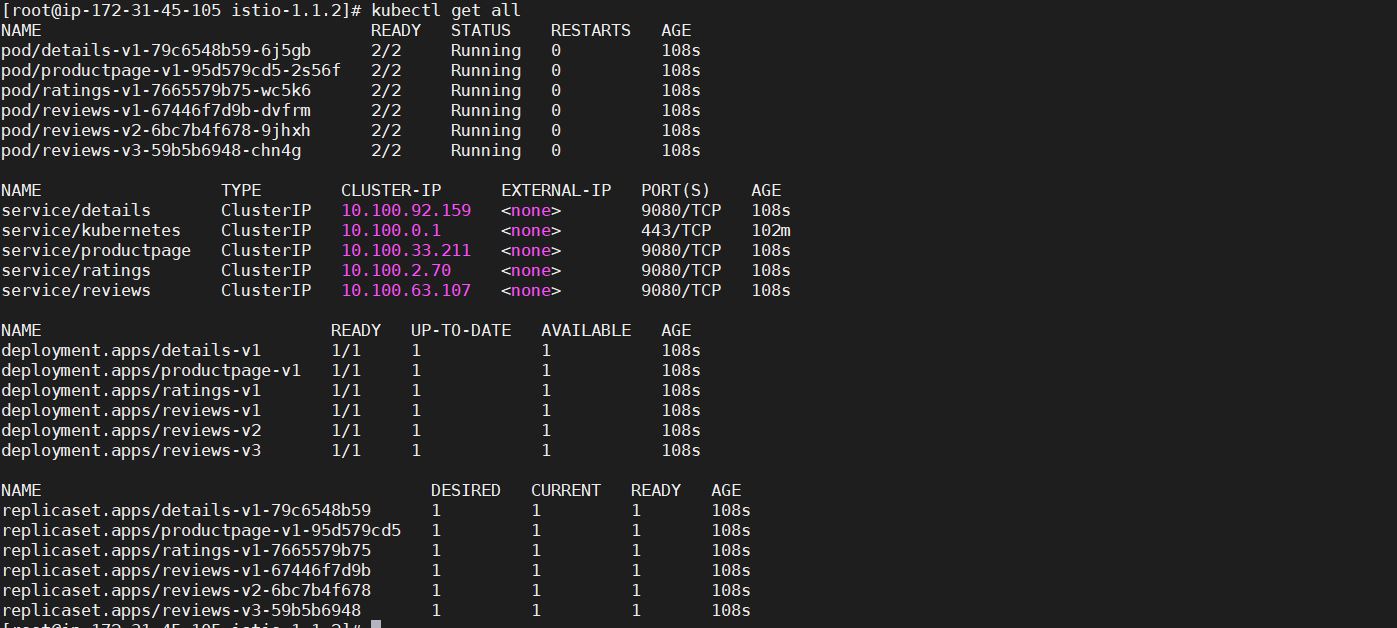
The default Istio installation uses automatic sidecar injection. Label the namespace that will host the application with istio-injection=enabled:

kubectl create -f samples/bookinfo/platform/kube/bookinfo.yaml



Here we can check all Deployments, Pods and Services

kubectl get all



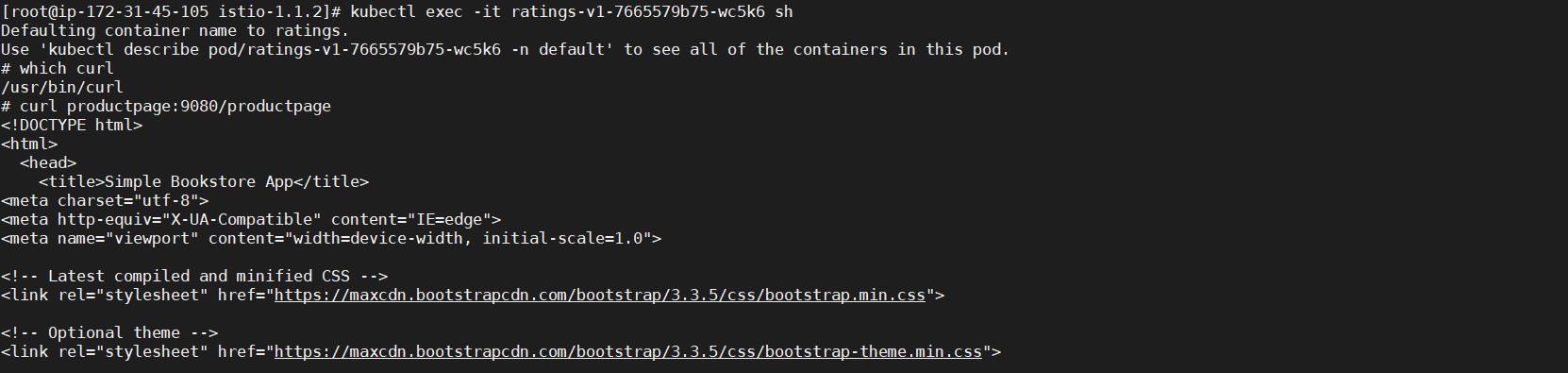
Check whether we get service response from pods: ( Here I taken ratings-v1-7665579b75-wc5k6 pod)

kubectl exec -it ratings-v1-7665579b75-wc5k6 sh

which curl

curl productpage:9080/productpage

exit



It actually went into a ratings pod and then curled to productpage with port 9080 and output the page. So, our app is not still accessible from outside yet.

kubectl get gateway

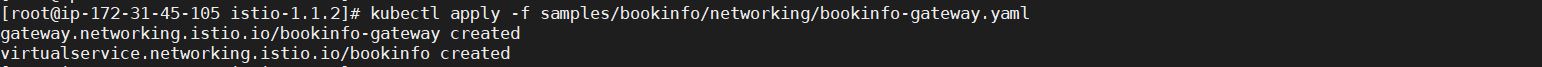
D:\Naresh\snipping\Capture.JPG

**Determining the ingress IP and port - Setup Istio Gateway:**

Now that the bookinfo services are up and running, we need to make the application accessible from outside of our Kubernetes cluster, e.g., from a browser. An Istio Gateway is used for this purpose.

Define the ingress gateway for the application:

kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml



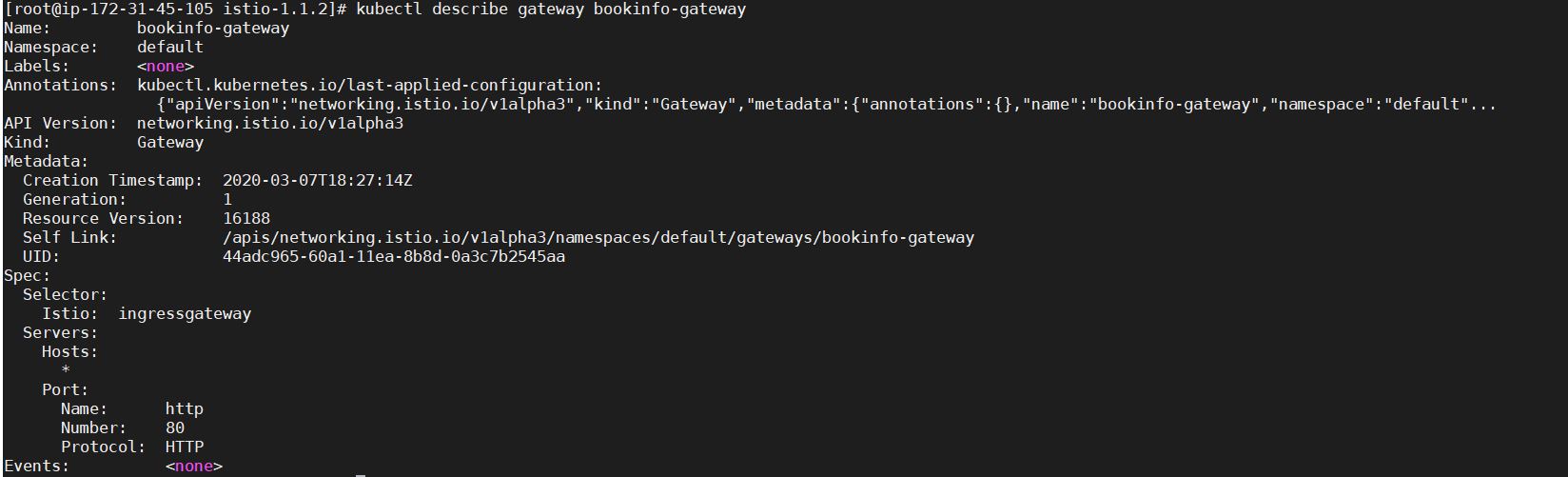
Confirm the gateway has been created:

kubectl get gateway

D:\Naresh\snipping\Capture.JPG

Here we can check gateway description:

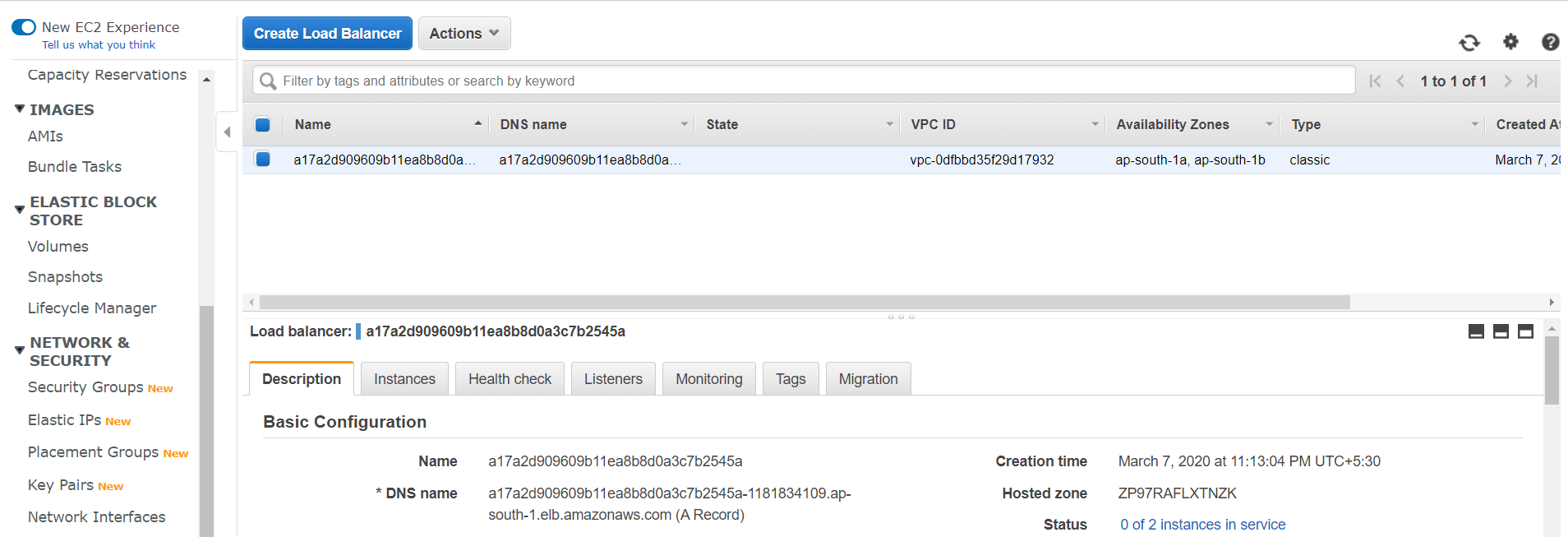
kubectl describe gateway bookinfo-gateway



Check whether LoadBalancer created or not in AWS:

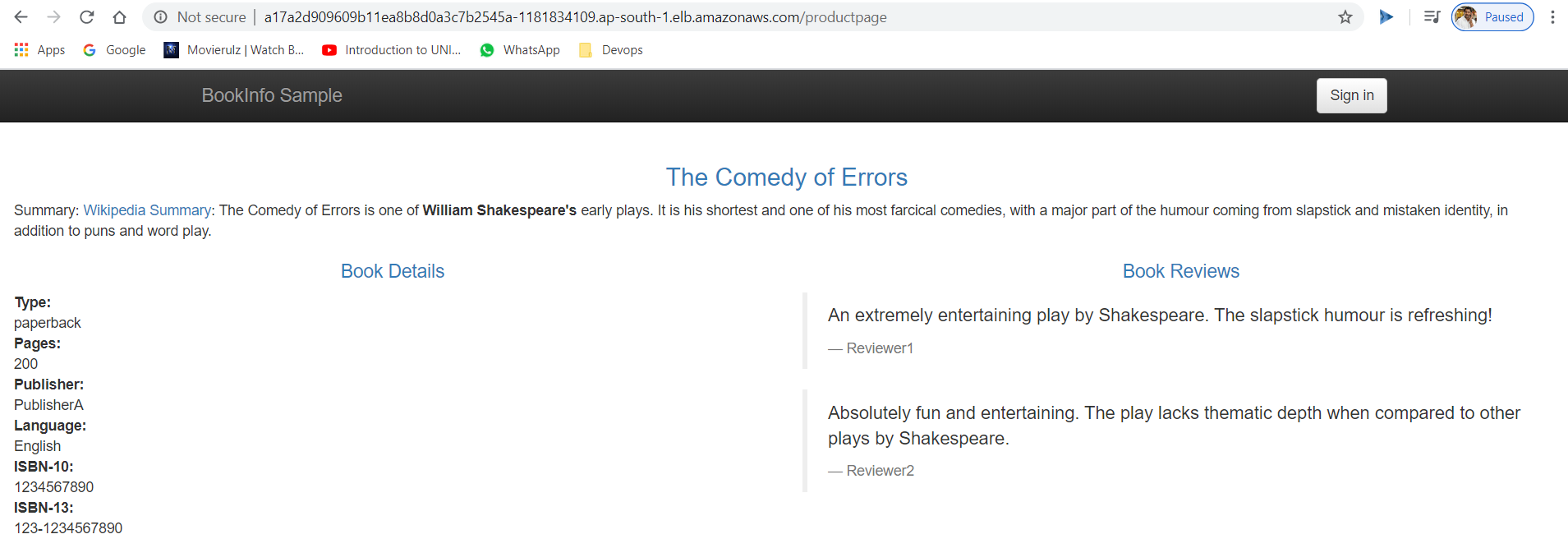
Goto EC2 service 🡪 LoadBalancer

Copy DNS name:

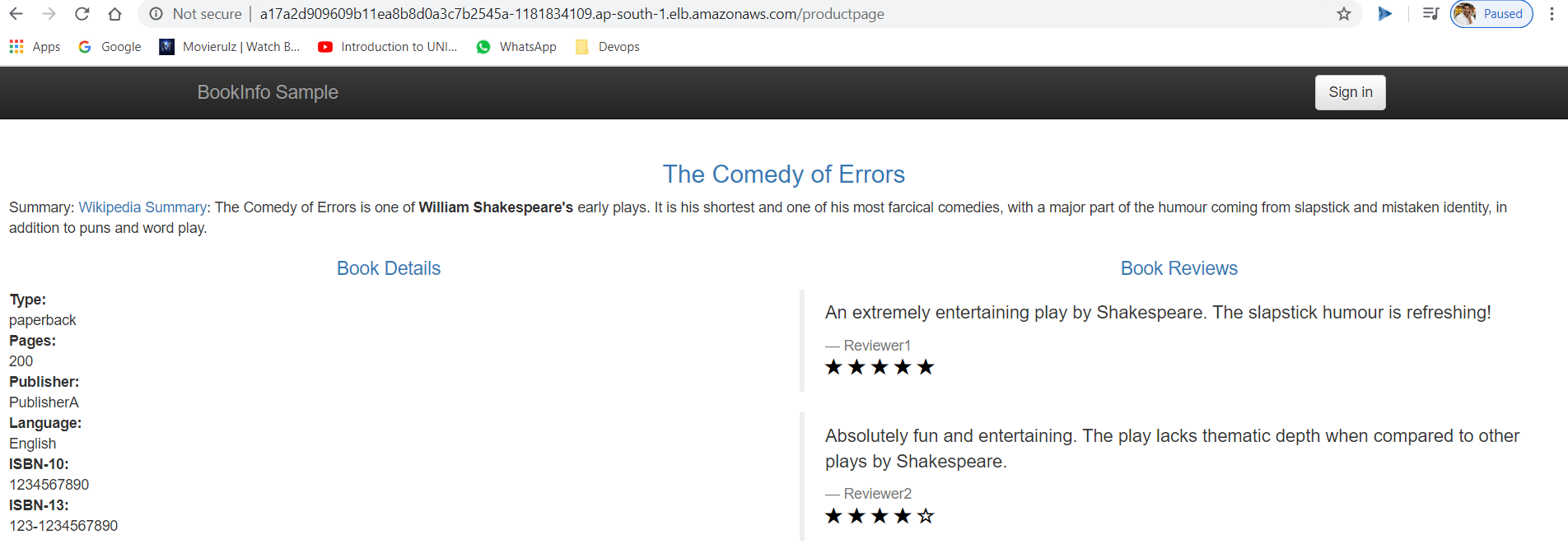


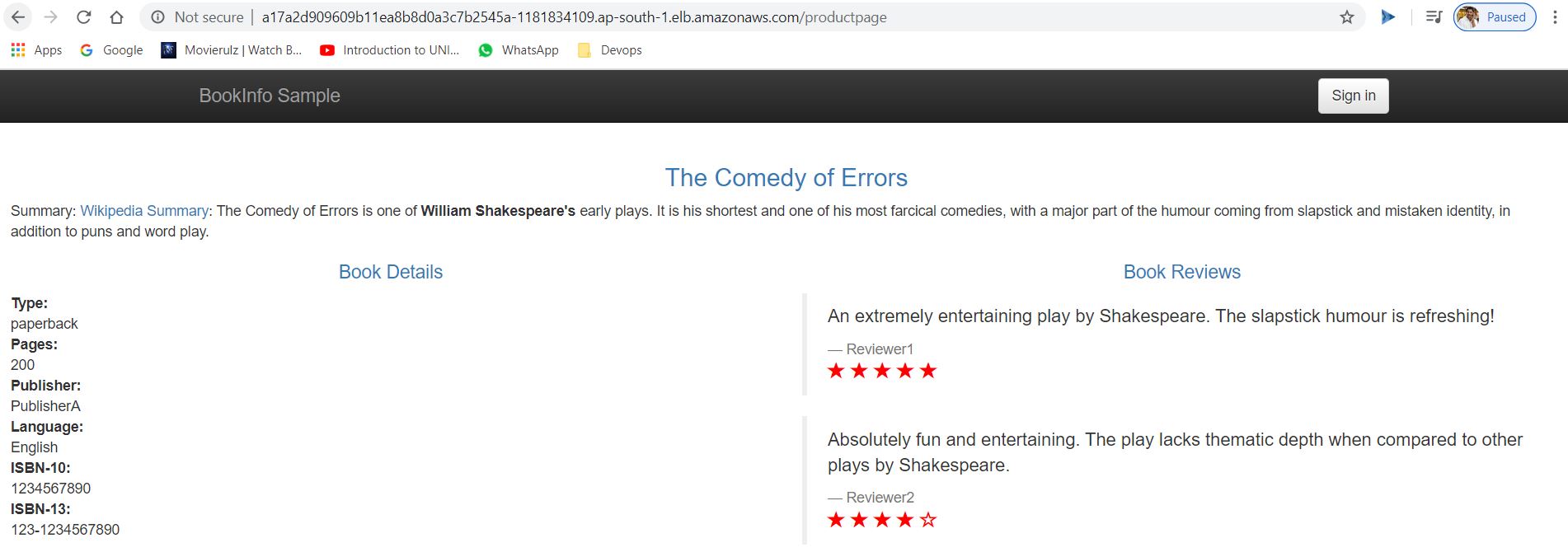
Now check from UI:

<http://a17a2d909609b11ea8b8d0a3c7b2545a-1181834109.ap-south-1.elb.amazonaws.com/productpage>



If we refresh the page several times, we should see different versions of reviews shown in productpage, presented in a round robin style (red stars, black stars, no stars), since we haven't yet used Istio to control the version routing.





**Intelligent routing:**

Before we can use Istio to control the Bookinfo version routing, we'll need to define the available versions, called subsets, in destination rules.

The following DestinationRule configures policies and subsets for the reviews service:

apiVersion: networking.istio.io/v1alpha3

kind: DestinationRule

metadata:

name: reviews

spec:

host: reviews

trafficPolicy:

loadBalancer:

simple: RANDOM

subsets:

- name: v1

labels:

version: v1

- name: v2

labels:

version: v2

trafficPolicy:

loadBalancer:

simple: ROUND\_ROBIN

- name: v3

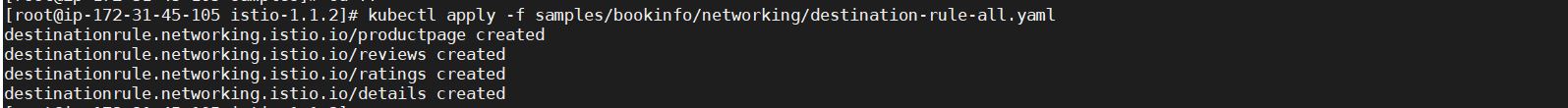
labels:

version: v3

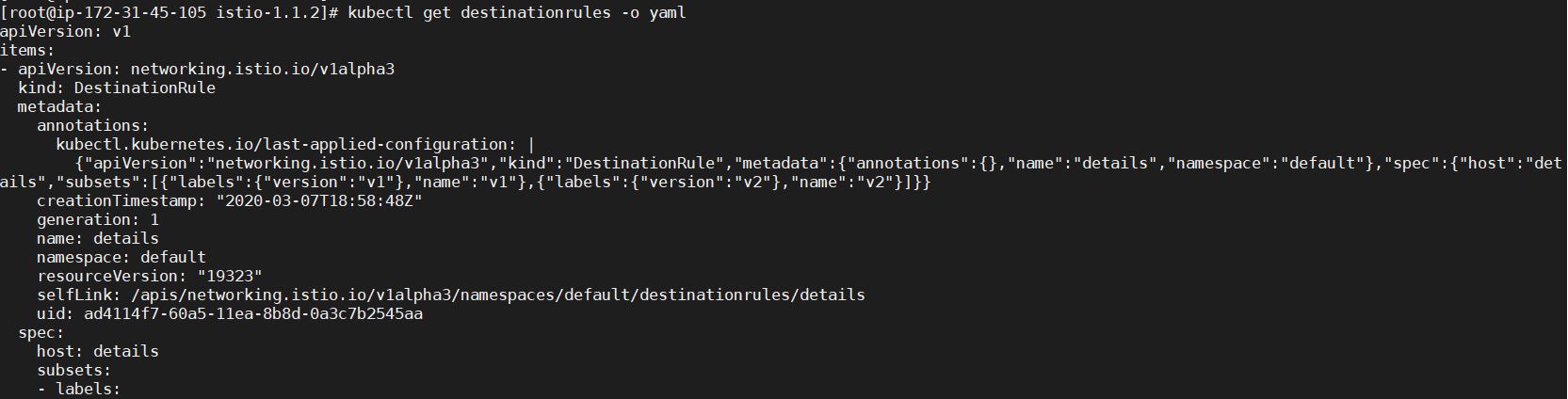
Notice that multiple policies, default (RANDOM) and v2-specific (ROUND\_ROBIN) in this example, can be specified in a single DestinationRule configuration.

To route to all:

kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml

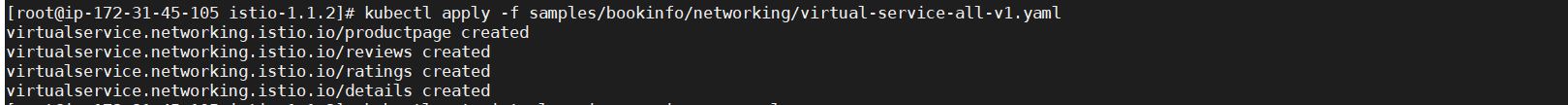


kubectl get destinationrules -o yaml

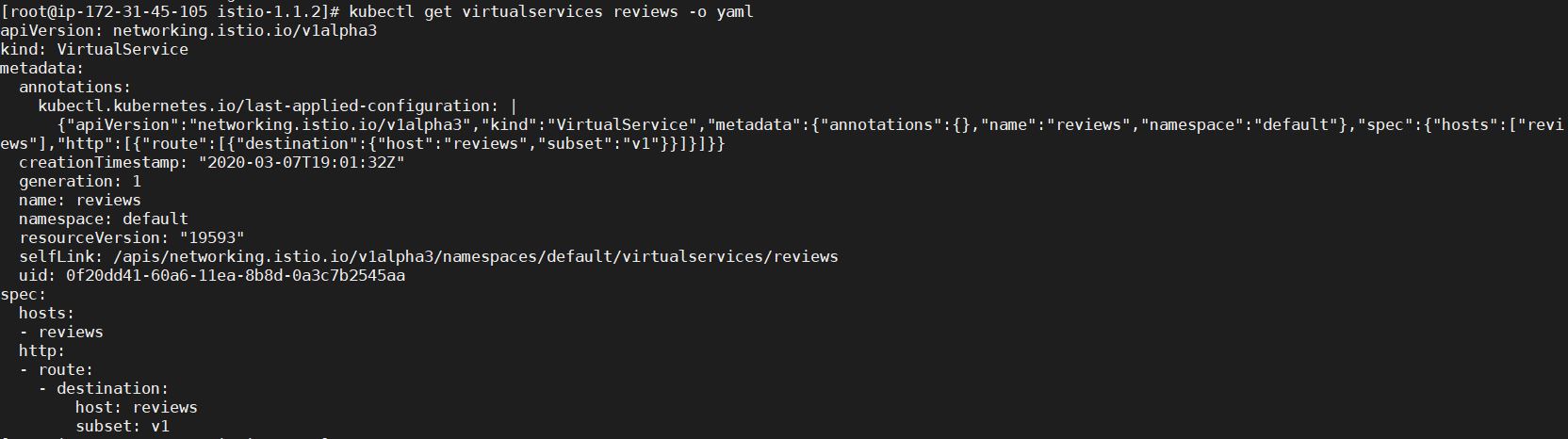


To route to one version only, we apply virtual services that set the default version for the microservices. In this case, the virtual services will route all traffic to reviews:v1 of microservice:

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

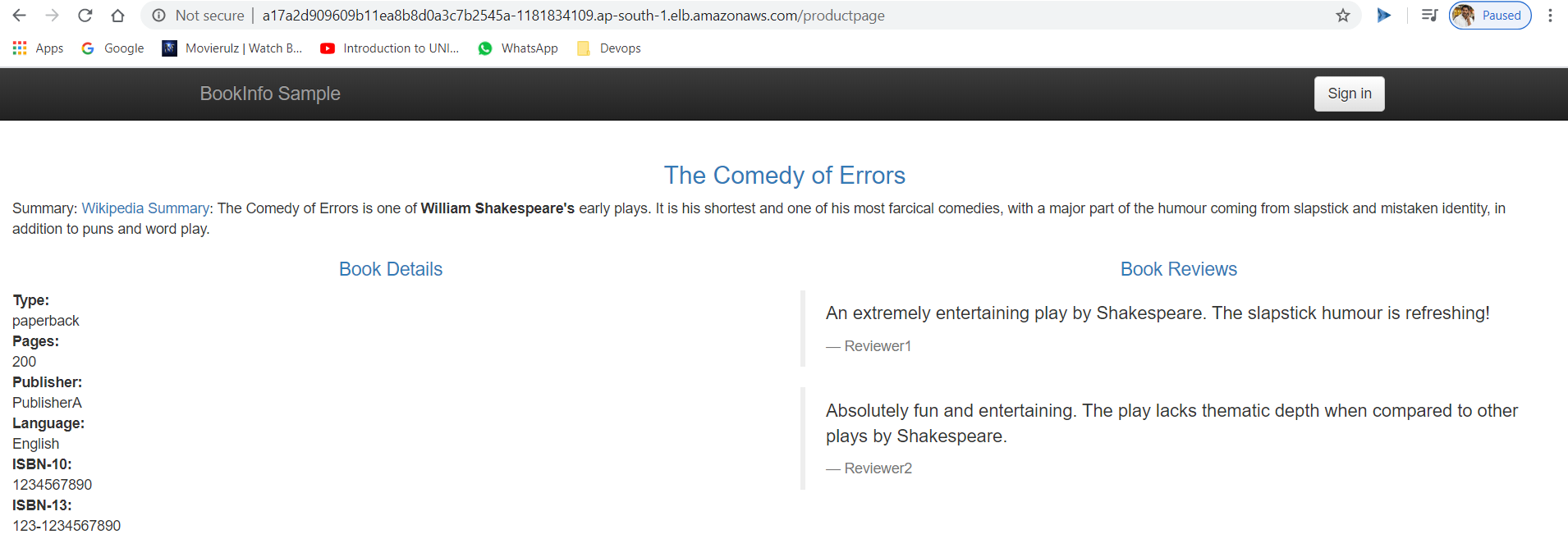


kubectl get virtualservices reviews -o yaml



Try now to reload the page multiple times, and note how only version 1 of reviews is displayed each time.

<http://a17a2d909609b11ea8b8d0a3c7b2545a-1181834109.ap-south-1.elb.amazonaws.com/productpage>

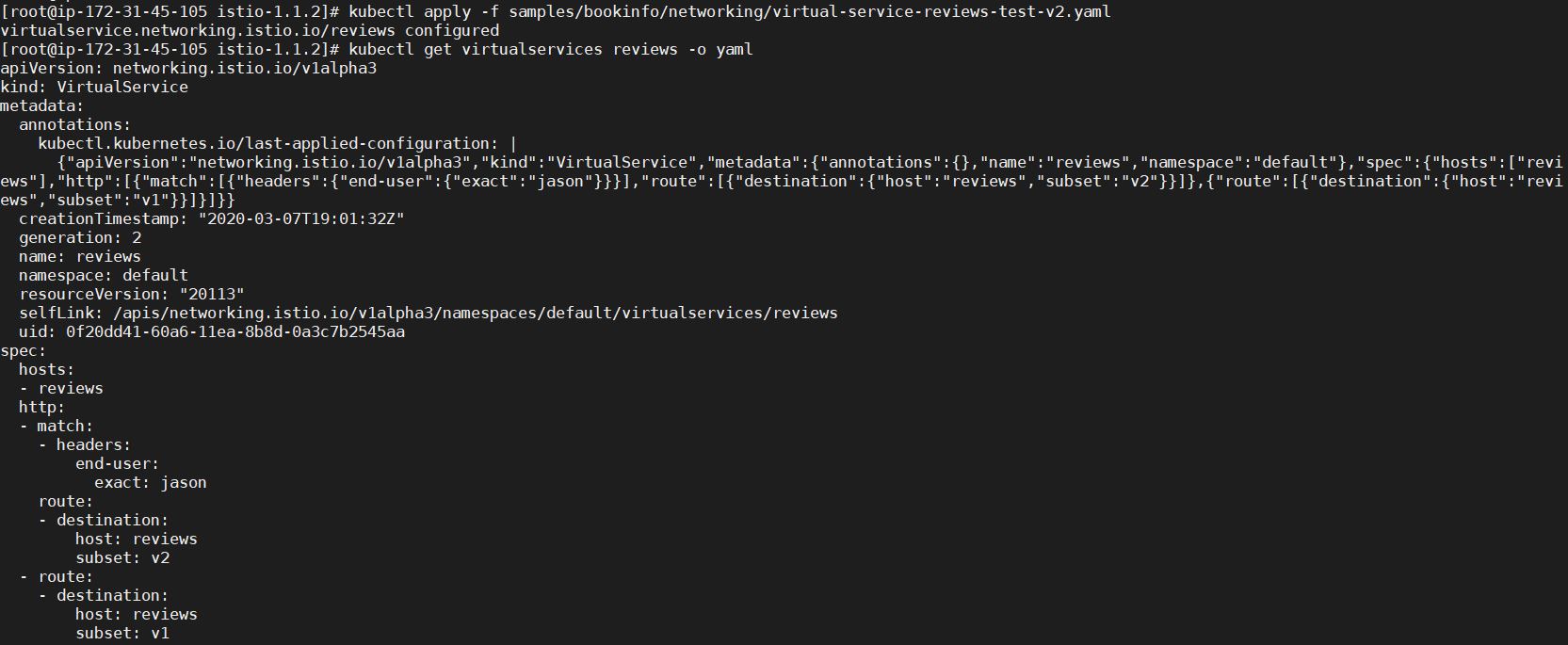


Next, we'll change the route configuration so that all traffic from a specific user is routed to a specific service version. In this case, all traffic from a user named 'jason' will be routed to the service reviews:v2

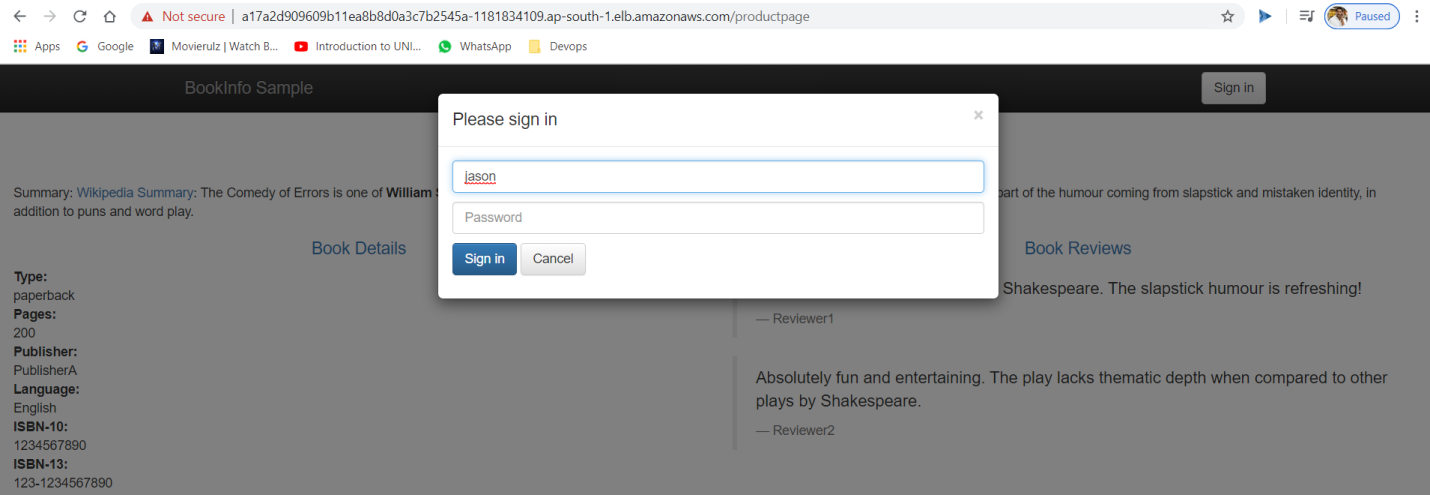
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml

D:\Naresh\snipping\Capture.JPG

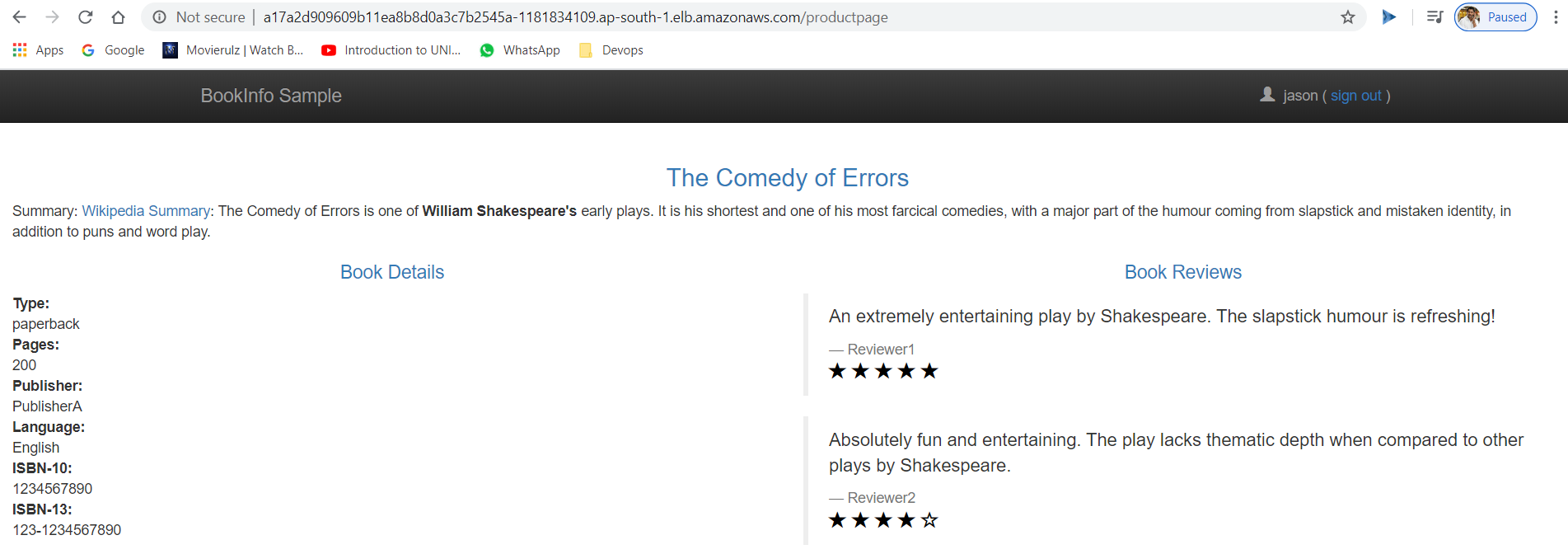
kubectl get virtualservices reviews -o yaml



To test, click Sign in from the top right corner of the page, and login using 'jason' as user name with a blank password.

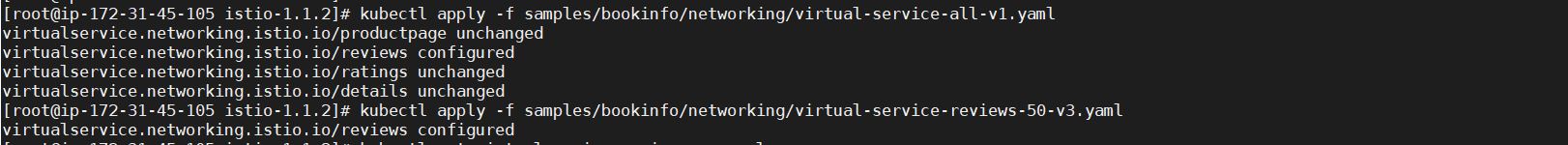


We will only see reviews:v2 all the time. Others will see reviews:v1.



Next, we'll demonstrate how to gradually migrate traffic from one version of a microservice to another. In our example, we'll send 50% of traffic to reviews:v1 and 50% to reviews:v3.

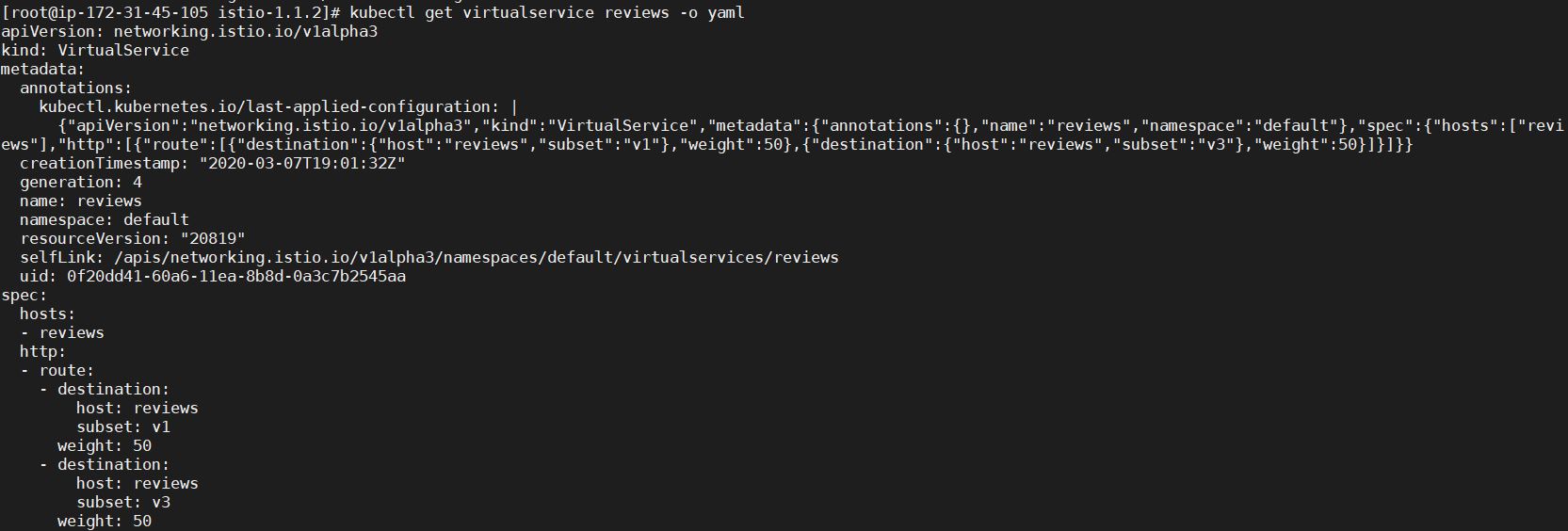
kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml



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

D:\Naresh\snipping\Capture.JPG

kubectl get virtualservice reviews -o yaml



To test it, refresh our browser over and over, and we'll see only reviews:v1 and reviews:v3.

<http://a17a2d909609b11ea8b8d0a3c7b2545a-1181834109.ap-south-1.elb.amazonaws.com/productpage>

