Username: [student-00-1750c8507dcf@qwiklabs.net](mailto:student-00-1750c8507dcf@qwiklabs.net)

Password: jx2r95XJB7

GCP Project ID: qwiklabs-gcp-02-f5985ee7c21a

[Istio](http://istio.io/) is an open source framework for connecting, securing, and managing microservices. It can be used with any services, including but not limited to services that are hosted in a Kubernetes cluster. Istio lets you create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without requiring any changes in service code.

You add Istio support to services by deploying a special Envoy sidecar proxy to each of your application's pods in your environment. Sidecar means that it gets deployed alongside your application, and your application interacts with the outside world, both ingress and egress, through the Envoy Proxy. Developers of applications can take advantage of the communication and networking enhancements provided by Envoy - like client-side load balancing, circuit breakers, logging, mTLS, etc. - without additional coding, and without finding the libraries in the language of choice

Because of the proximity of the sidecar to the application, there's no significant latency when communicating between them.

In this lab, you will learn how to perform the following tasks:

* Provision a cluster on Google Kubernetes Engine (GKE).
* Install and configure the Istio on GKE Add-On, which includes the Istio control-plane and a method to deploy Envoy proxies as sidecars.
* Deploy Bookinfo, an Istio-enabled multi-service application.
* Enable external access using an Istio Ingress Gateway.
* Use the Bookinfo application

You'll use a Google Kubernetes Engine (GKE) cluster named central, in the us-central1 region.

In this task, you:

* Deploy the central cluster on GKE using **Cloud Shell**
* Deploy a service mesh using the Istio on GKE add-on.

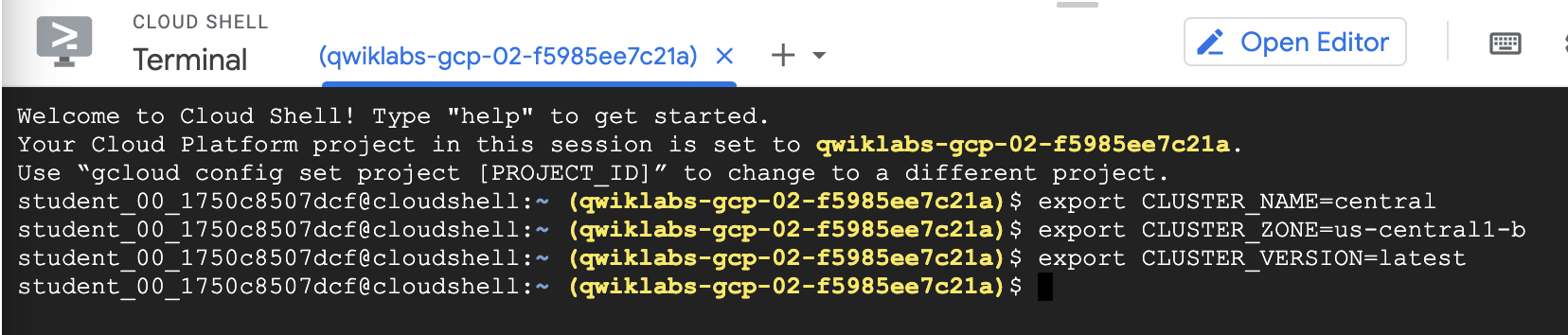
### **Use Cloud Shell to deploy your GKE cluster with Istio**

1. If you haven't already, open a new Cloud Shell session and run the following to set environment variables for the zone and cluster name:

export CLUSTER\_NAME=central

export CLUSTER\_ZONE=us-central1-b

export CLUSTER\_VERSION=latest



1. Now run the following command to create your GKE cluster using the Istio on GKE add-on:

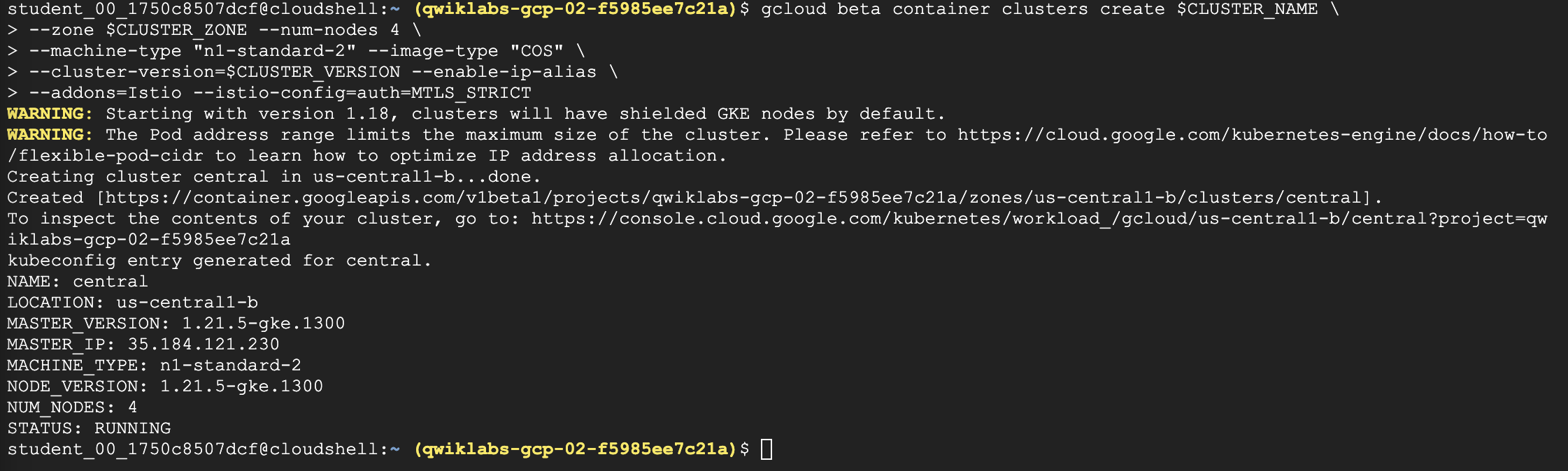
gcloud beta container clusters create $CLUSTER\_NAME \

--zone $CLUSTER\_ZONE --num-nodes 4 \

--machine-type "n1-standard-2" --image-type "COS" \

--cluster-version=$CLUSTER\_VERSION --enable-ip-alias \

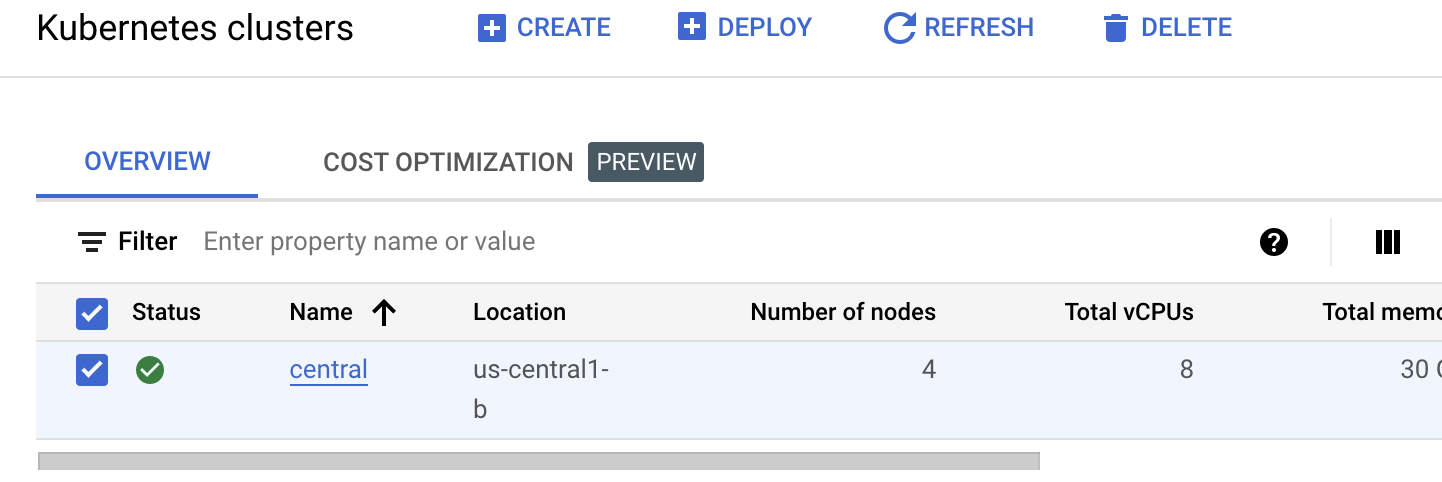
--addons=Istio --istio-config=auth=MTLS\_STRICT



This command creates a cluster in a single zone, with 4 nodes.

The nodes are in the default VPC network, and the cluster is operated in VPC-native (alias IP) mode. System and workload logging and monitoring are enabled as well by default.

The new cluster (central) takes several minutes to deploy. You can review your cluster, when deployment is complete, from the  **Navigation menu** > **Kubernetes Engine** > **Clusters**.

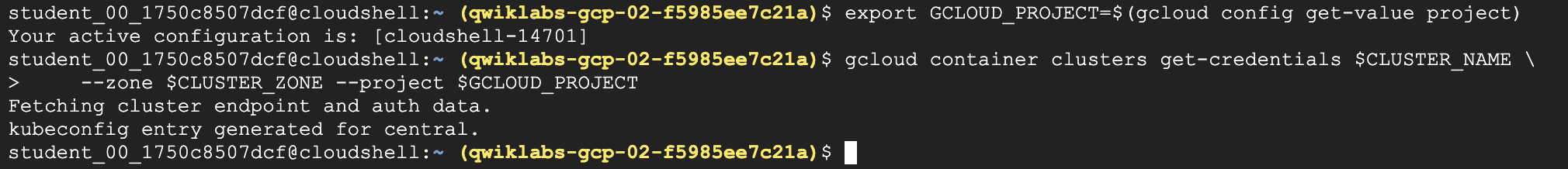


1. Once your cluster has been created, configure kubectl command line access by running the following:

export GCLOUD\_PROJECT=$(gcloud config get-value project)

gcloud container clusters get-credentials $CLUSTER\_NAME \

--zone $CLUSTER\_ZONE --project $GCLOUD\_PROJECT

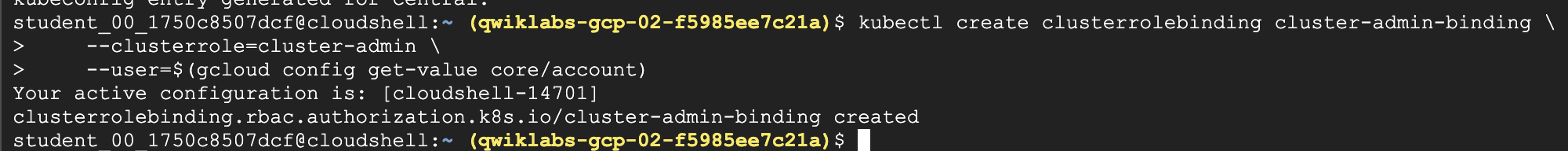


1. Grant admin permissions in the cluster to the current gcloud user:

kubectl create clusterrolebinding cluster-admin-binding \

--clusterrole=cluster-admin \

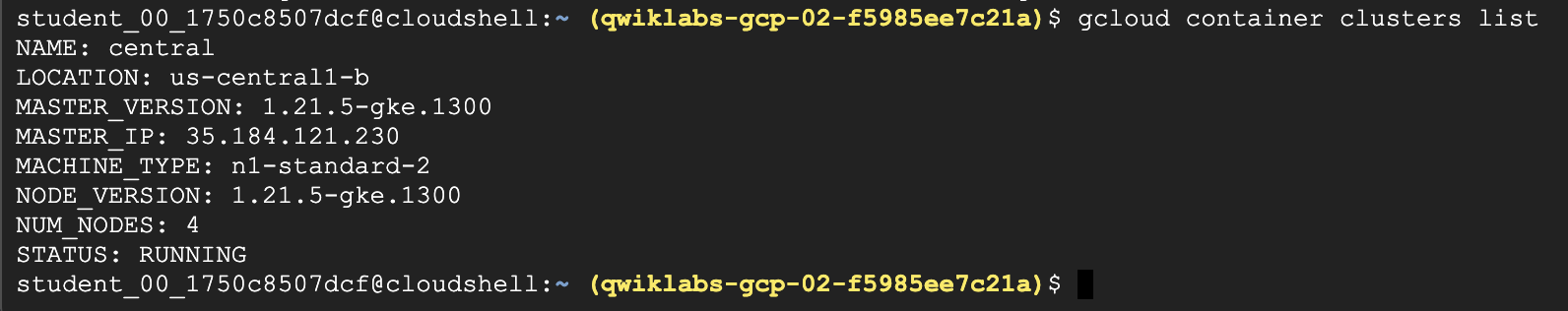
--user=$(gcloud config get-value core/account)

****

### **Verify installation**

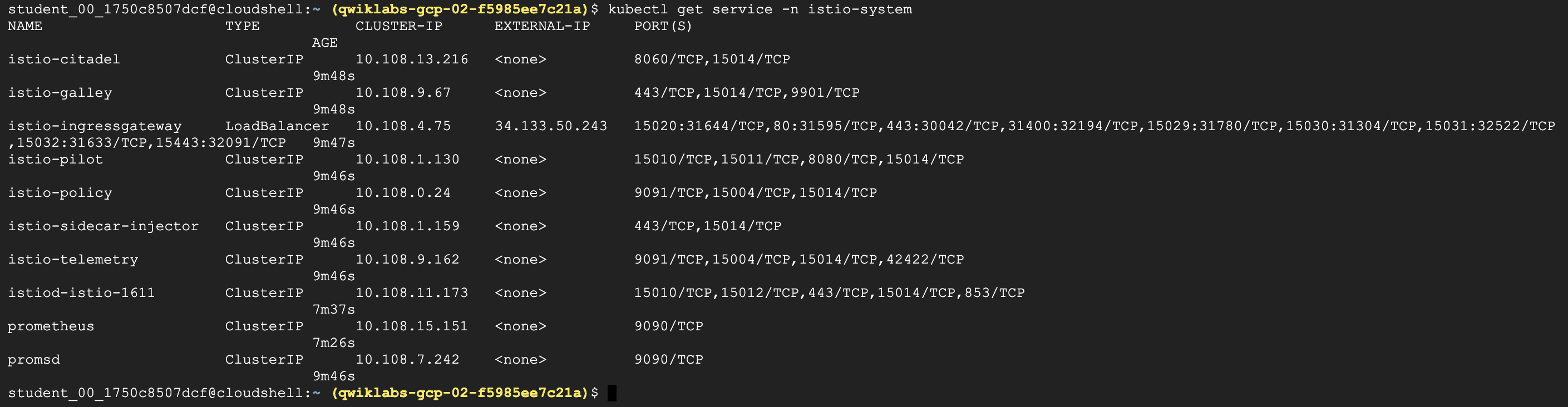
1. Check that your cluster is up and running.

gcloud container clusters list



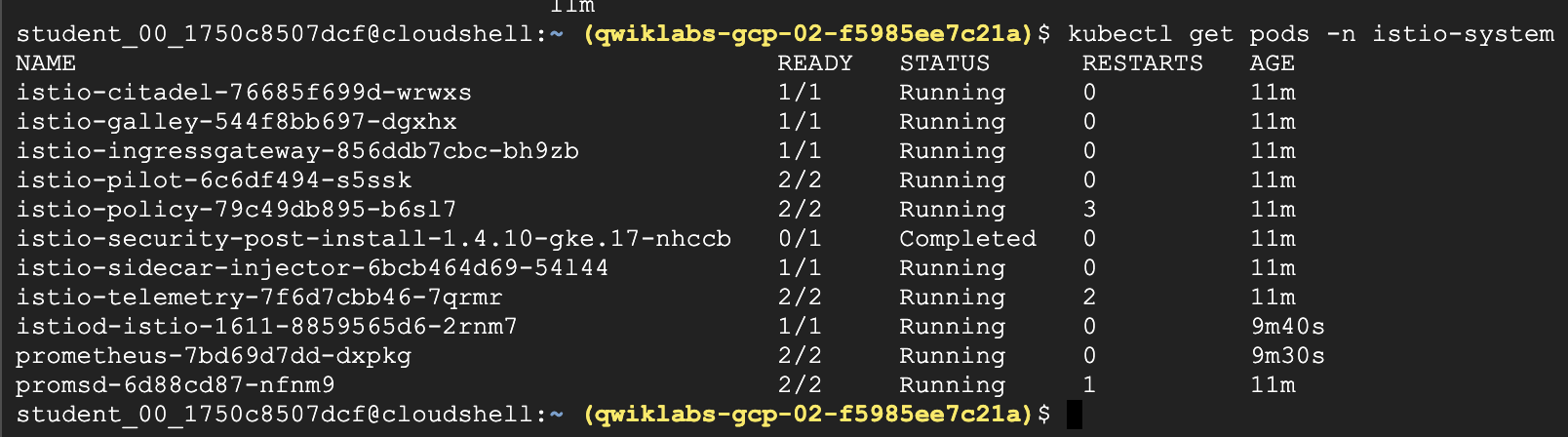
1. Ensure the following Kubernetes services are deployed: istio-citadel, istio-galley, istio-pilot, istio-ingressgateway, istio-policy, istio-sidecar-injector, and istio-telemetry. You'll also see other deployed services:

kubectl get service -n istio-system



1. Ensure the corresponding Kubernetes pods are deployed and all containers are up and running: istio-pilot, istio-galley, istio-policy, istio-telemetry, istio-ingressgateway, istio-sidecar-injector, and istio-citadel.

kubectl get pods -n istio-system



## Deploy Bookinfo, an Istio-enabled multi-service application

In this task, you enable the istioctl tool, set up the **Bookinfo** sample microservices application, and explore the app.

### **Download and configure istioctl**

1. Use Cloud Shell to download and extract the Istio release, with the istioctl tool, and sample code:

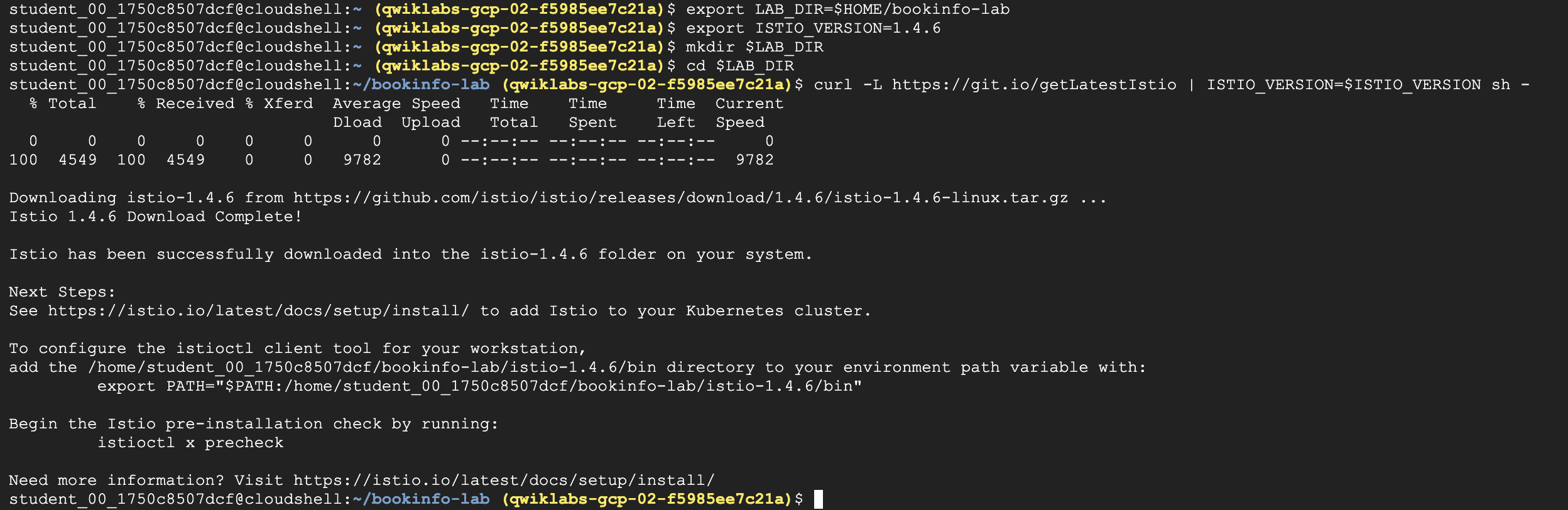
export LAB\_DIR=$HOME/bookinfo-lab

export ISTIO\_VERSION=1.4.6

mkdir $LAB\_DIR

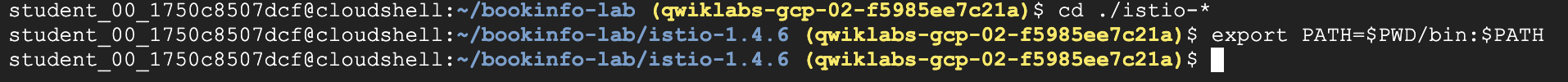
cd $LAB\_DIR

curl -L https://git.io/getLatestIstio | ISTIO\_VERSION=$ISTIO\_VERSION sh -



1. Make the Istio tools visible to your environment, by adding bin to your **PATH**:
2. cd ./istio-\*

export PATH=$PWD/bin:$PATH



The installation directory contains the following files which we'll use:

* + Sample applications in samples/.
  + The istioctl client binary in the bin/ directory. Similar to kubectl for Kubernetes, this is the tool used to manage Istio, including network routing and security policies.

1. Verify istioctl works:

istioctl version

Copied!

content\_copy

**Output (do not copy)**

client version: 1.4.6

control plane version: 1.4.6-gke.0

data plane version: 1.4.6-gke.0 (1 proxies)

### **Bookinfo Overview**

Now that Istio is configured and verified, you can deploy one of the sample applications provided with the installation — [BookInfo](https://istio.io/docs/guides/bookinfo.html" \t "_blank). This is a simple mock bookstore application made up of four microservices - all managed using Istio. Each microservice is written in a different language, to demonstrate how you can use Istio in a multi-language environment, without any changes to code.

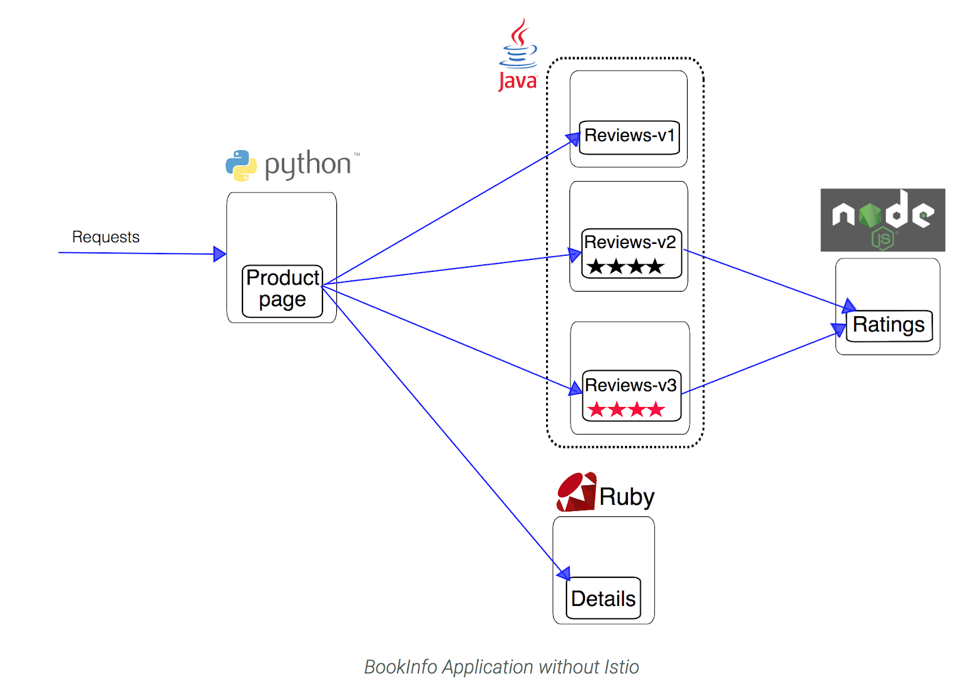
The microservices are:

* **productpage:** calls the details and reviews microservices to populate the page.
* **details**: contains book information.
* **reviews:** contains book reviews. It also calls the ratings microservice.
* **ratings**: contains book ranking information that accompanies a book review.

There are 3 versions of the **reviews** microservice:

* Reviews **v1** doesn't call the ratings service.
* Reviews **v2** calls the ratings service, and displays each rating as 1 - 5 black stars.
* Reviews **v3** calls the ratings service, and displays each rating as 1 - 5 red stars.

The end-to-end architecture of the application looks like this:



You can find the source code and all the other files used in this example in your Istio [samples/bookinfo](https://github.com/istio/istio/tree/master/samples/bookinfo) directory.

### **Deploy Bookinfo**

1. Look at the .yaml which describes the bookInfo application:

cat samples/bookinfo/platform/kube/bookinfo.yaml

Copied!

content\_copy

Look for containers to see that each Deployment, has **one** container, for each version of each service in the Bookinfo application.

1. Look at the same .yaml with Istio proxy sidecars injected using istioctl:

istioctl kube-inject -f samples/bookinfo/platform/kube/bookinfo.yaml

Copied!

content\_copy

Now, when you scroll up, and look for containers, you can see extra configuration describing the proxy sidecar containers that will be deployed.

istioctl kube-inject takes a Kubernetes YAML file as input, and outputs a version of that YAML which includes the Istio proxy. You can look at one of the Deployments in the output from istio kube-inject. It includes a second container, the **Istio sidecar**, along with all the configuration necessary.

1. In **Cloud Shell**, use the following command to inject the proxy sidecar along with each application Pod that is deployed.

kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/bookinfo.yaml)

Copied!

content\_copy

Istio uses an extended version of the open-source [Envoy proxy](https://www.envoyproxy.io/envoy/), a high-performance proxy developed in C++, to mediate all inbound and outbound traffic for all services in the service mesh. Istio leverages Envoy's many built-in features including dynamic service discovery, load balancing, TLS termination, HTTP/2 & gRPC proxying, circuit breakers, health checks, staged rollouts with %-based traffic split, fault injection, and rich metrics.

**Output (do not copy)**

service/details created

serviceaccount/bookinfo-details created

deployment.apps/details-v1 created

service/ratings created

serviceaccount/bookinfo-ratings created

deployment.apps/ratings-v1 created

service/reviews created

serviceaccount/bookinfo-reviews created

deployment.apps/reviews-v1 created

deployment.apps/reviews-v2 created

deployment.apps/reviews-v3 created

service/productpage created

serviceaccount/bookinfo-productpage created

deployment.apps/productpage-v1 created

**Note**: Automatic sidecar injection can be enabled by labeling the namespace hosting the application with istio-injection=enabled, using a command like: kubectl label namespace default istio-injection=enabled

Click Check my progress to verify the objective.

Deploy Bookinfo, an Istio-enabled multi-service application.

Check my progress

### **Enable external access using an Istio Ingress Gateway**

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

1. Look at the .yaml which describes the configuration for the application ingress gateway:

cat samples/bookinfo/networking/bookinfo-gateway.yaml

Copied!

content\_copy

Look for the Gateway and VirtualService mesh resources which get deployed. The Gateway exposes services to users outside the service mesh, and allows Istio features such as monitoring and route rules to be applied to traffic entering the cluster.

1. Configure the **ingress gateway** for the application, which exposes an external IP you will use later:

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

Copied!

content\_copy

**Output (do not copy)**

gateway.networking.istio.io/bookinfo-gateway created

virtualservice.networking.istio.io/bookinfo created

Copied!

content\_copy

### **Verify the Bookinfo deployments**

1. Confirm that the application has been deployed correctly, review services, pods, and the ingress gateway:

kubectl get services

Copied!

content\_copy

**Output (do not copy)**

NAME TYPE CLUSTER-IP ...

details ClusterIP 10.107.0.77 ...

kubernetes ClusterIP 10.107.0.1 ...

productpage ClusterIP 10.107.8.22 ...

ratings ClusterIP 10.107.13.29 ...

reviews ClusterIP 10.107.12.100 ...

1. Review running application pods:

kubectl get pods

Copied!

content\_copy

**Output (do not copy)**

NAME READY STATUS

details-v1-1520924117-48z17 2/2 Running

productpage-v1-560495357-jk1lz 2/2 Running

ratings-v1-734492171-rnr5l 2/2 Running

reviews-v1-874083890-f0qf0 2/2 Running

reviews-v2-1343845940-b34q5 2/2 Running

reviews-v3-1813607990-8ch52 2/2 Running

You may need to re-run this command until you see that all six pods are in **Running** status.

**Note:** See how each pod has two containers? That's the application container and the Istio proxy sidecar.

1. Confirm that the Bookinfo application is running by sending a curl request to it from some pod, within the cluster, for example from ratings:
2. kubectl exec -it $(kubectl get pod -l app=ratings -o jsonpath='{.items[0].metadata.name}') \

-c ratings -- curl productpage:9080/productpage | grep -o "<title>.\*</title>"

Copied!

content\_copy

**Output (do not copy)**

<title>Simple Bookstore App</title>

1. Confirm the ingress gateway has been created:

kubectl get gateway

Copied!

content\_copy

**Output (do not copy)**

NAME AGE

bookinfo-gateway 20m

1. Get the **external IP** address of the **ingress gateway**:

kubectl get svc istio-ingressgateway -n istio-system

NAME TYPE CLUSTER-IP EXTERNAL-IP

istio-ingressgateway LoadBalancer 10.107.15.123 104.154.143.236

In this example, the external IP of the ingress gateway is 104.154.143.236.

1. Check that the Bookinfo app is running by sending a curl request to it from outside the cluster (be sure to update [EXTERNAL-IP] with the output from the previous command):
2. export GATEWAY\_URL=[EXTERNAL-IP]

curl -I http://${GATEWAY\_URL}/productpage

Copied!

content\_copy

**Output (do not copy)**

HTTP/1.1 200 OK

content-type: text/html; charset=utf-8

content-length: 4415

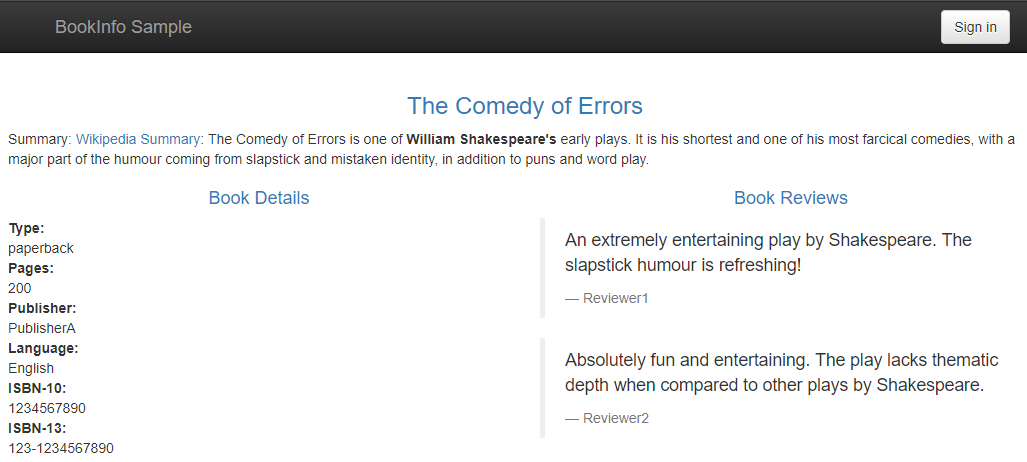
server: istio-envoy

...

## Use the Bookinfo application

Congratulations! You have deployed an Istio-enabled application. Next, let's see the BookInfo application in action.

1. Point your browser to http://[$GATEWAY\_URL]/productpage to see the BookInfo web page. Don't forget to replace [$GATEWAY\_URL] with your working external IP address.



1. Refresh the page several times.

Notice how you see three different versions of reviews, since we have not yet used Istio to control the version routing.

There are three different book review services being called in a round-robin style:

* + No stars
  + Black stars
  + Red stars

Switching among the three is normal Kubernetes routing/balancing behavior.