Name : Nishita Manahrsinh Solanki

Roll No.: 22BCE211

Subject : DevOps

Practical – 8

**Task 1. Get the sample code**

To get the code, copy the sample code from a Google Cloud Storage bucket.

1. In Cloud Shell, copy the source code from the Cloud Shell command line:

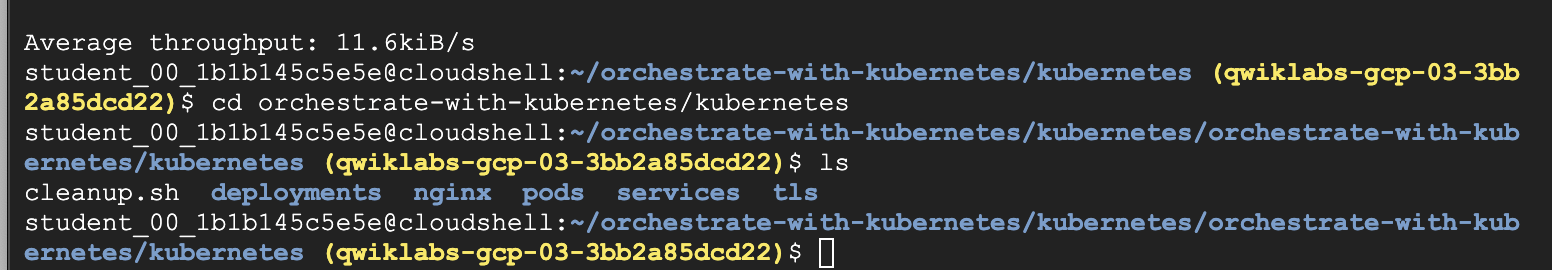
gcloud storage cp -r gs://spls/gsp021/\* .

1. Change into the directory needed for this lab:

cd orchestrate-with-kubernetes/kubernetes

1. List the files to see what you're working with:

ls

**Task 2. A quick Kubernetes demo**

The easiest way to get started with Kubernetes is to use the kubectl create command.

1. Use it to launch a single instance of the nginx container:

kubectl create deployment nginx --image=nginx:1.27.0

Kubernetes has created a Deployment—more about Deployments later, but for now all you need to know is that Deployments keep the Pods up and running even when the nodes they run on fail.

In Kubernetes, all containers run in a Pod.

1. Use the kubectl get pods command to view the running nginx container:

kubectl get pods

1. Once the nginx container has a Running status you can expose it outside of Kubernetes using the kubectl expose command:

kubectl expose deployment nginx --port 80 --type LoadBalancer

So what just happened? Behind the scenes Kubernetes created an external load balancer with a public IP address attached to it. Any client who hits that public IP address will be routed to the Pods behind the service. In this case that would be the nginx Pod.

1. List the services now using the kubectl get services command:

kubectl get services

**Note:** It may take a few seconds before the ExternalIP field is populated for your service. This is normal—just re-run the kubectl get services command every few seconds until the field populates.

1. Add the External IP to this command to hit the Nginx container remotely:

curl http://<External IP>:80

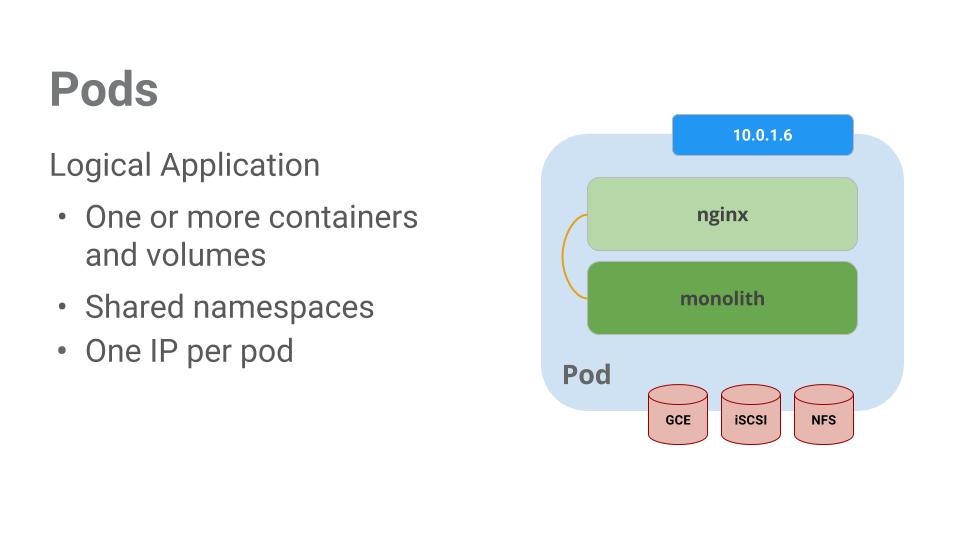
And there you go! Kubernetes supports an easy to use workflow out of the box using the kubectl run and expose commands.

## Task 3. About Pods

At the core of Kubernetes is the [Pod](https://kubernetes.io/docs/concepts/workloads/Pods/).

Pods represent and hold a collection of one or more containers. Generally, if you have multiple containers with a hard dependency on each other, you package the containers inside a single Pod.

The following example shows a Pod that contains the monolith and nginx containers.



Pods also have [Volumes](https://kubernetes.io/docs/concepts/storage/volumes/). Volumes are data disks that live as long as the Pods live, and can be used by the containers in that Pod. Pods provide a shared namespace for their contents which means that the two containers inside of our example Pod can communicate with each other, and they also share the attached volumes.

Pods also share a network namespace. This means that there is one IP Address per Pod.

Next, a deeper dive into Pods.

**Task 4. Create Pods**

Pods can be created using Pod configuration files. Take a moment to explore the fortune-app Pod configuration file.

1. Go to the directory:

cd ~/orchestrate-with-kubernetes/kubernetes

1. Run the following to view a configuration file:

cat pods/fortune-app.yaml

The output shows the open configuration file:

apiVersion: v1

kind: Pod

metadata:

name: fortune-app

labels:

app: fortune-app

spec:

containers:

- name: fortune-app

image: "us-central1-docker.pkg.dev/qwiklabs-resources/spl-lab-apps/fortune-app:1.0.0"

ports:

- name: http

containerPort: 8080

resources:

limits:

cpu: 0.2

memory: "20Mi"

There's a few things to notice here:

* Your Pod is made up of one container (the fortune-app).
* You're opening port 8080 for http traffic.

1. Create the fortune-app Pod using kubectl:

kubectl create -f pods/fortune-app.yaml

1. Examine your Pods. Use the kubectl get pods command to list all Pods running in the default namespace:

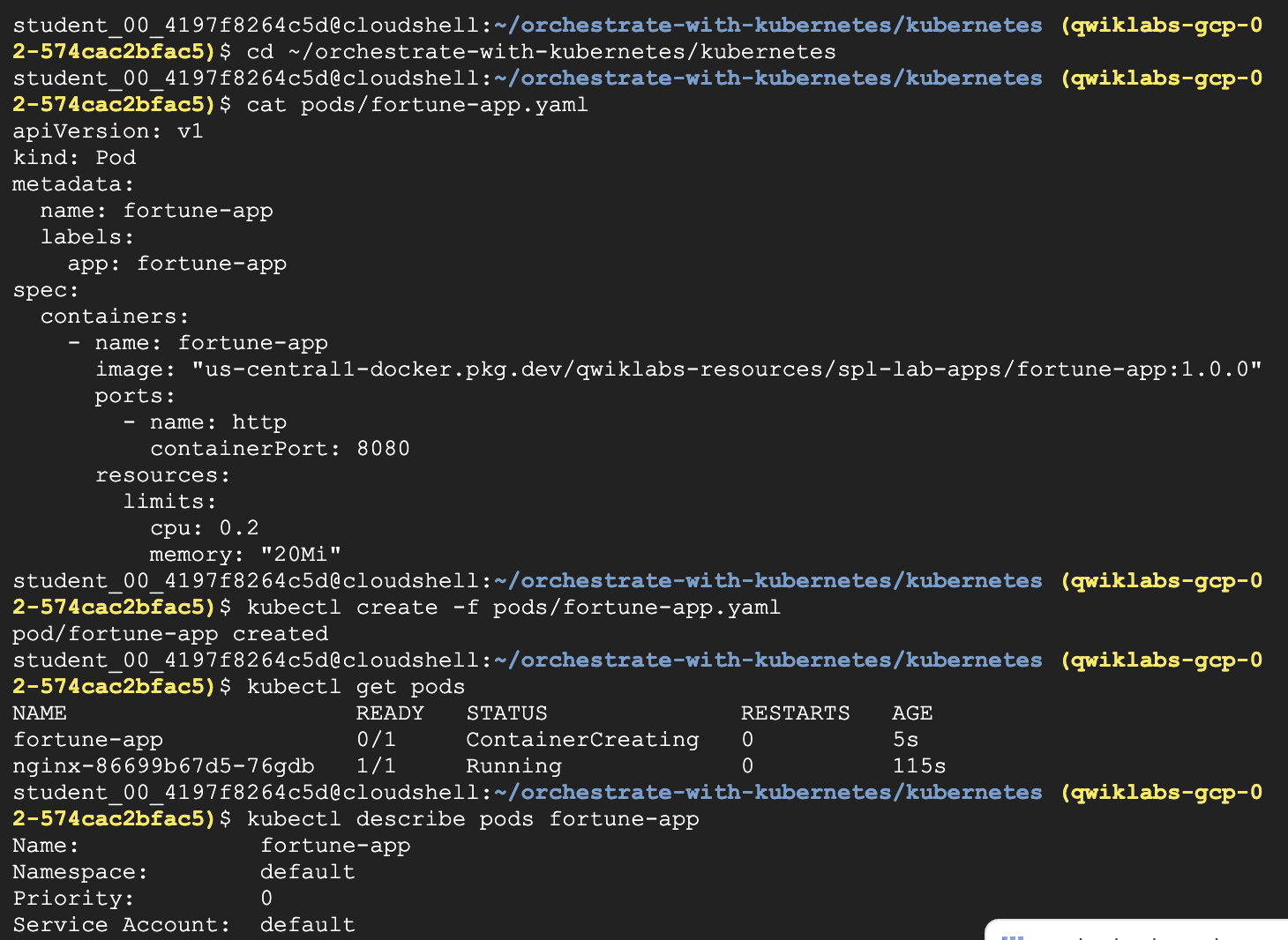
kubectl get pods

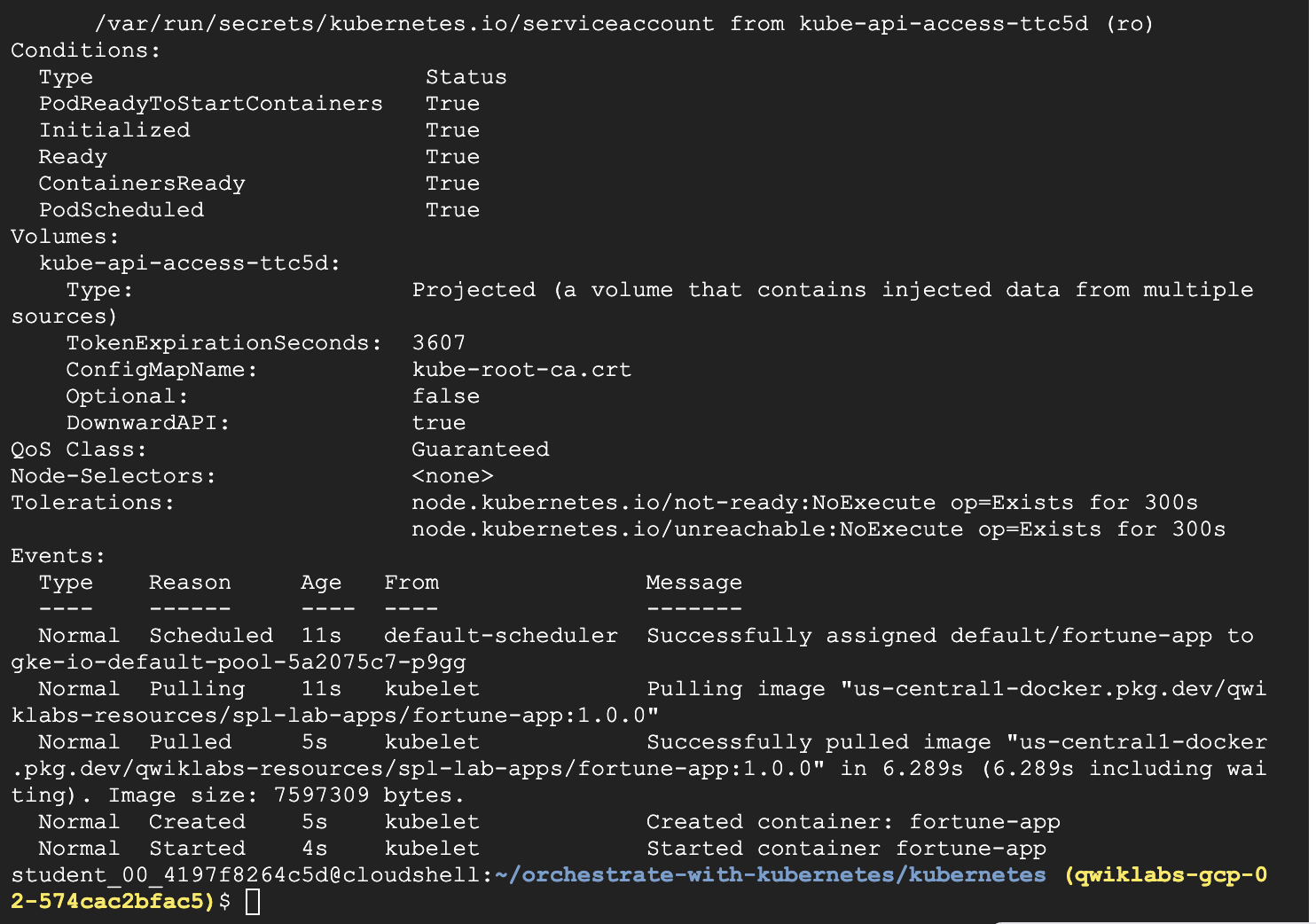
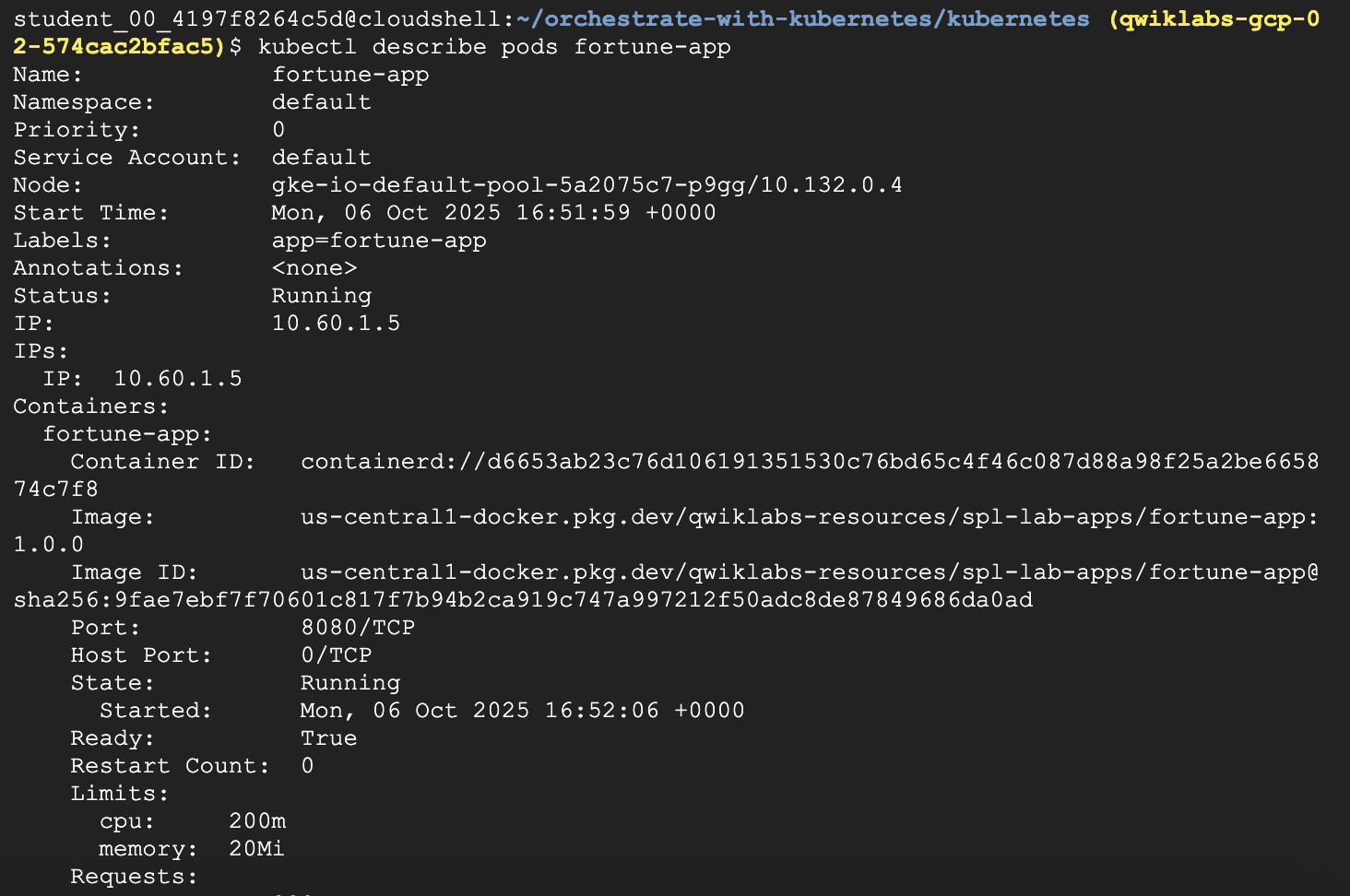
**Note:** It may take a few seconds before the fortune-app Pod is up and running. The container image needs to be pulled from the Artifact Registry before you can run it.

1. Once the Pod is running, use the kubectl describe command to get more information about the fortune-app Pod:

kubectl describe pods fortune-app

You see a lot of the information about the fortune-app Pod, including the Pod IP address and the event log. This information comes in handy when troubleshooting.





**Task 5. Interact with Pods**

By default, Pods are allocated a private IP address and cannot be reached outside the cluster. Use the kubectl port-forward command to map a local port to a port inside the fortune-app Pod.

**Note:**From this point on, this lab will ask you to work in multiple cloud shell tabs to set up communication between the Pods. Any commands that are executed in a second or third command shell will be denoted in the command's instructions.

1. Open a second Cloud Shell terminal. Now you have two terminals, one to run the kubectl port-forward command, and the other to issue curl commands.
2. In the **2nd terminal**, run this command to set up port-forwarding:

kubectl port-forward fortune-app 10080:8080

1. Now in the **1st terminal** start talking to your Pod using curl:

curl http://127.0.0.1:10080

You got a fortune cookie back from your container!

1. Now use the curl command to see what happens when you hit a secure endpoint. This endpoint requires authentication.

curl http://127.0.0.1:10080/secure

You'll receive an "Unauthorized" error, which is expected.

1. To get an auth token back from the app, try logging in:

curl -u user http://127.0.0.1:10080/login

Copied!

1. At the login prompt, use the super-secret password password to login.

Logging in will return a JWT token.

1. Since Cloud Shell does not handle copying long strings well, create an environment variable for the token.

TOKEN=$(curl -u user http://127.0.0.1:10080/login | jq -r '.token')

Copied!

1. Enter the super-secret password password again when prompted for the host password.
2. Use this command to copy and then use the token to hit the secure endpoint with curl:

curl -H "Authorization: Bearer $TOKEN" http://127.0.0.1:10080/secure

Copied!

At this point you should get a fortune back from the application, letting you know everything is right in the world again.

You have accessed the secure fortune!

1. Use the kubectl logs command to view the logs for the fortune-app Pod.

kubectl logs fortune-app

Copied!

1. Open a **3rd terminal** and use the -f flag to get a stream of the logs happening in real-time:

kubectl logs -f fortune-app

1. Now if you use curl in the **1st terminal** to interact with the fortune-app, you can see the logs updating (in the **3rd terminal**):

curl http://127.0.0.1:10080

1. Use the kubectl exec command to run an interactive shell inside the fortune-app Pod. This comes in handy when you want to troubleshoot from within a container:

kubectl exec fortune-app --stdin --tty -c fortune-app -- /bin/sh

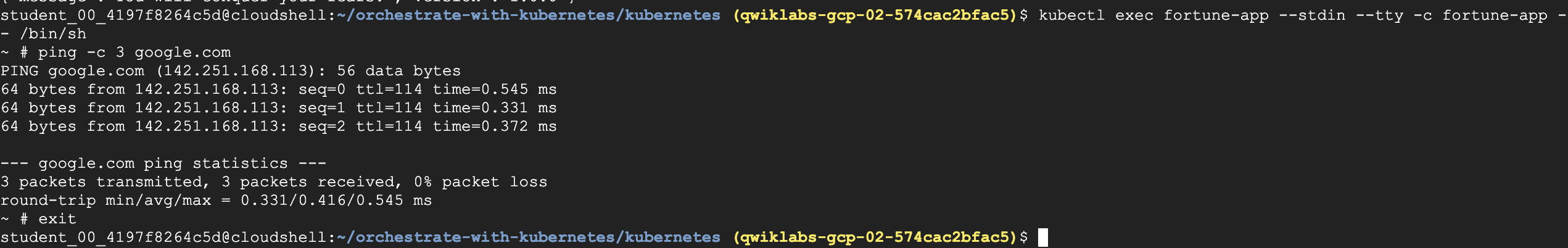
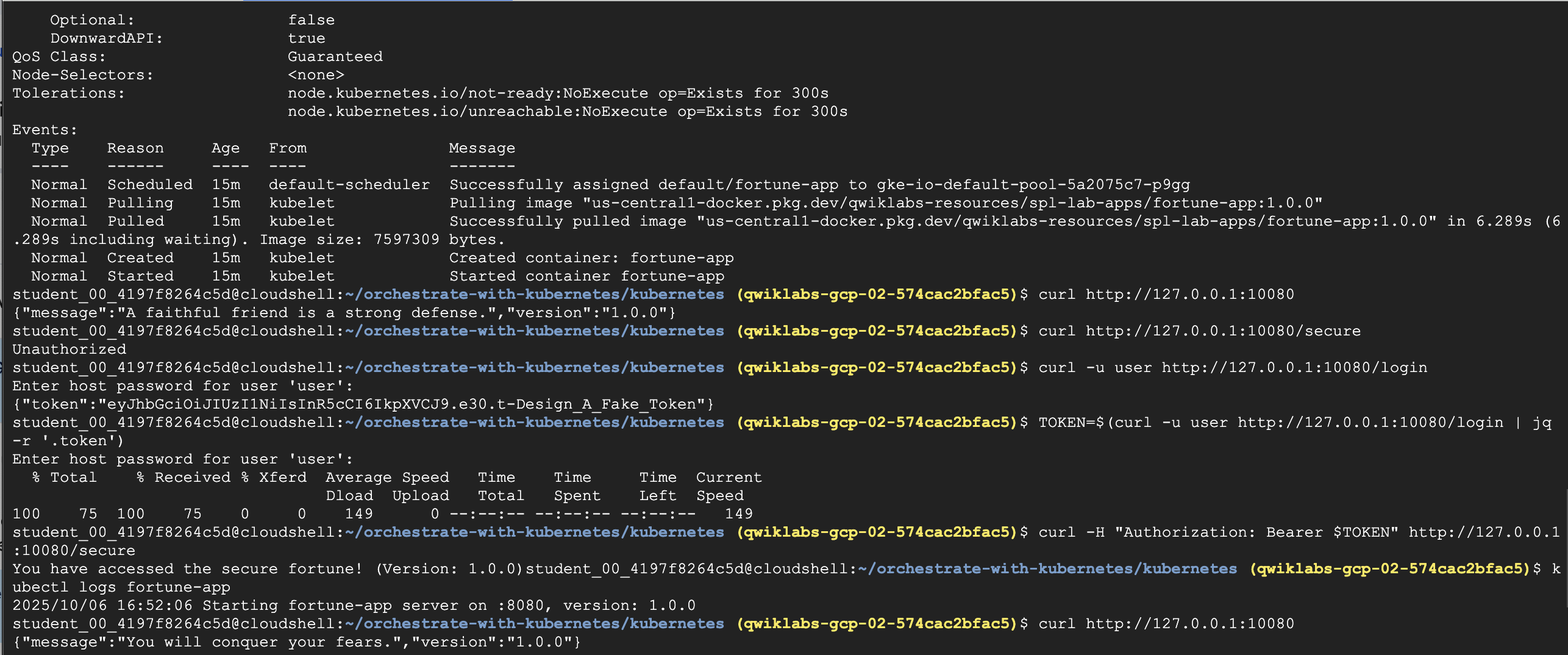
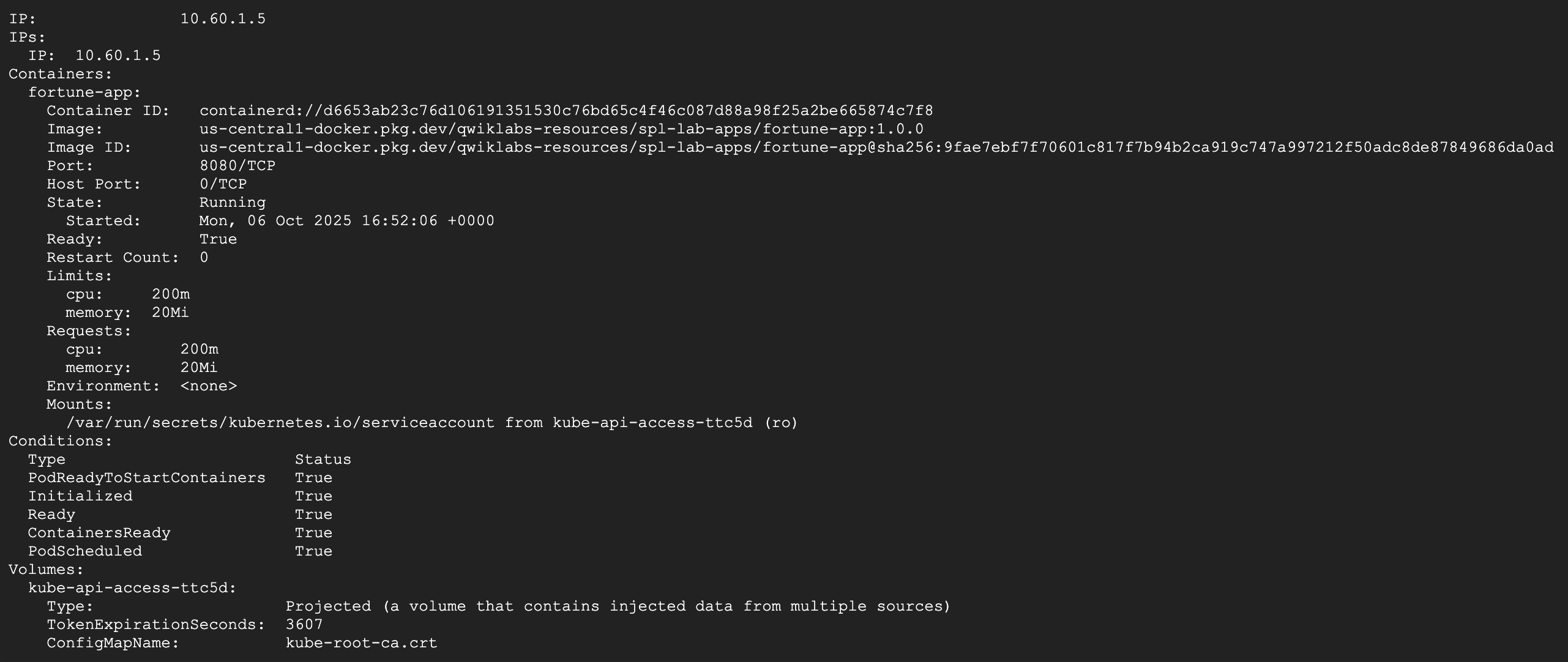
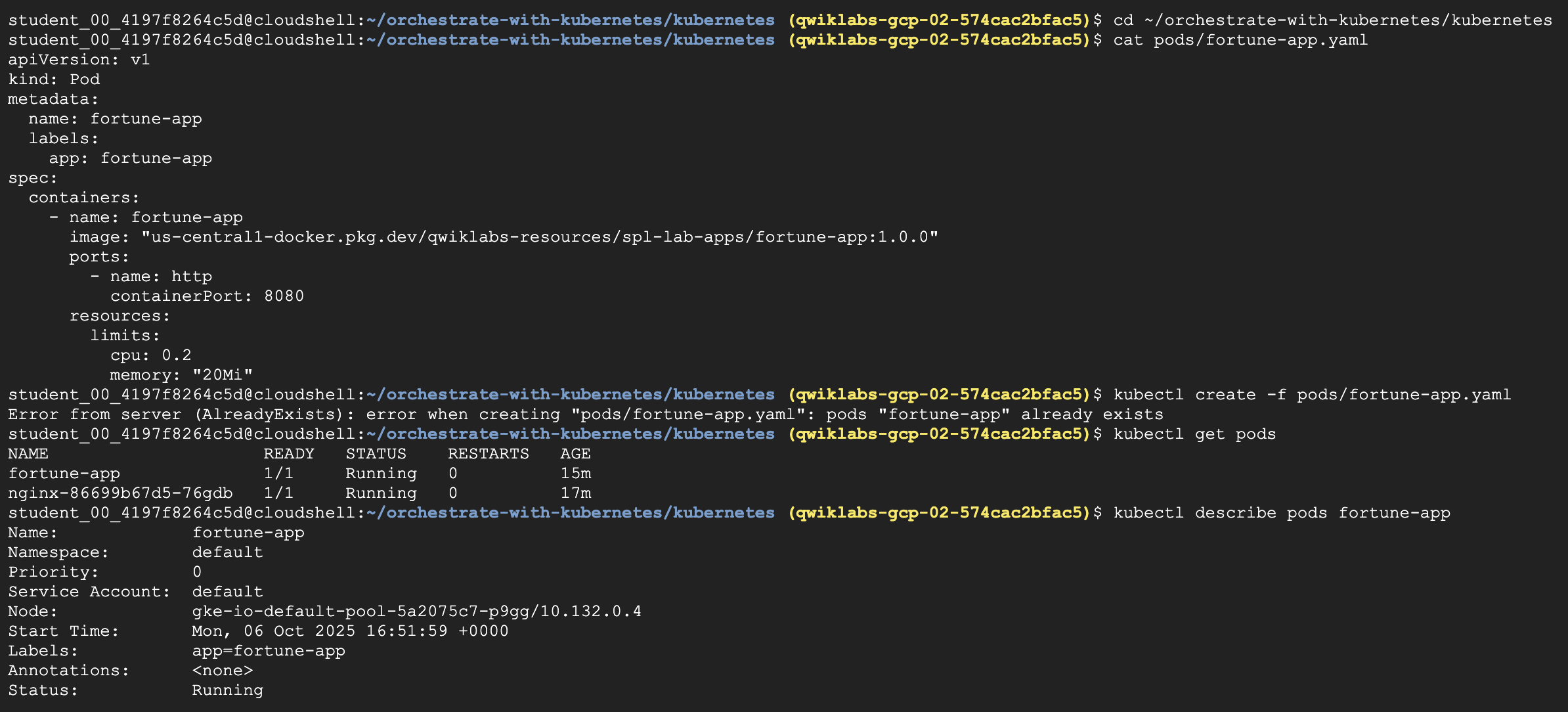
1. For example, once you have a shell into the fortune-app container you can test external connectivity using the ping command:

ping -c 3 google.com

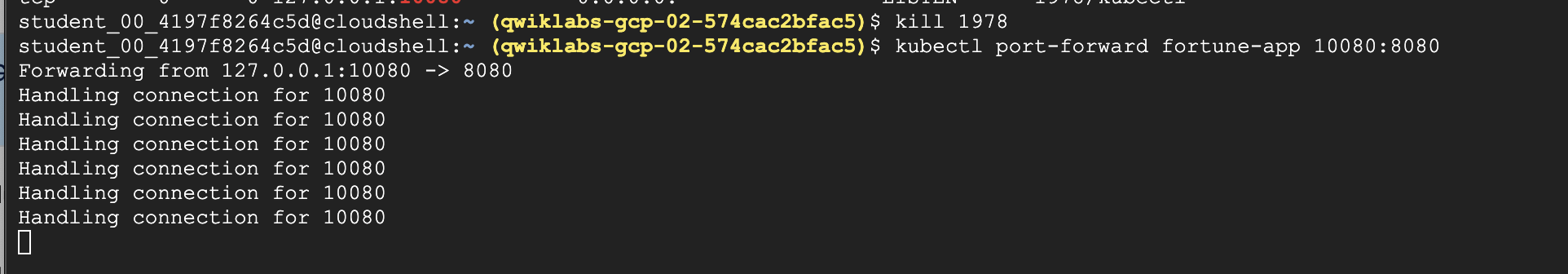
1. Be sure to log out when you're done with this interactive shell.

exit

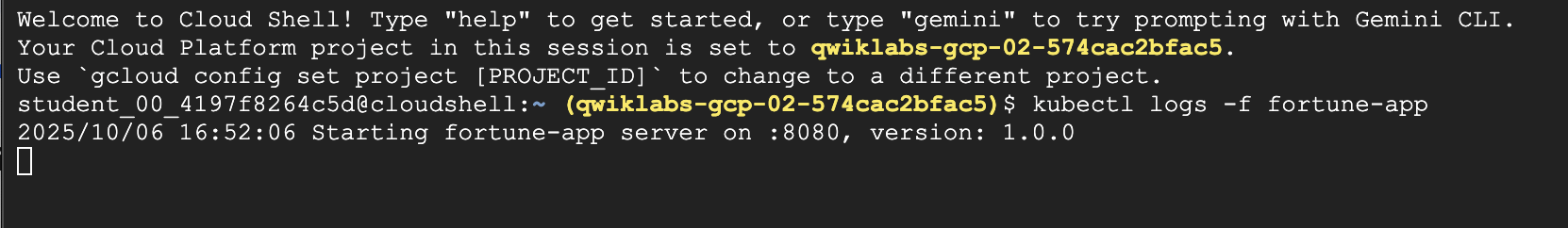
As you can see, interacting with Pods is as easy as using the kubectl command. If you need to hit a container remotely, or get a login shell, Kubernetes provides everything you need to get up and going.



Terminal – 2



Terminal – 3



**Task 6. About Services**

Pods aren't meant to be persistent. They can be stopped or started for many reasons, like failed liveness or readiness checks, which leads to a problem:

What happens if you want to communicate with a set of Pods? When they get restarted they might have a different IP address.

That's where [Services](https://kubernetes.io/docs/concepts/services-networking/service/) come in. Services provide stable endpoints for Pods.

Services use labels to determine what Pods they operate on. If Pods have the correct labels, they are automatically picked up and exposed by our services.

The level of access a service provides to a set of Pods depends on the Service's type. Currently there are three types:

* ClusterIP (internal) is the default type. This Service is only visible inside the cluster.
* NodePort gives each node in the cluster an externally accessible IP.
* LoadBalancer adds a load balancer from the cloud provider which forwards traffic from the Service to Nodes within it.

Now you'll learn how to:

* Create a Service
* Use label selectors to expose a limited set of Pods externally

## Task 7. Create a Service

Before you can create Services, first create a secure Pod that can handle https traffic.

1. If you've changed directories, make sure you return to the ~/orchestrate-with-kubernetes/kubernetes directory:

cd ~/orchestrate-with-kubernetes/kubernetes

1. Explore the secure fortune-app service configuration file:

cat pods/secure-fortune.yaml

1. Create the secure-fortune Pods and their configuration data:

kubectl create secret generic tls-certs --from-file tls/

kubectl create configmap nginx-proxy-conf --from-file nginx/proxy.conf

kubectl create -f pods/secure-fortune.yaml

Now that you have a secure Pod, it's time to expose it externally. To do that, create a Kubernetes Service.

1. Explore the fortune-app service configuration file:

cat services/fortune-app.yaml

(Output):

kind: Service

apiVersion: v1

metadata:

name: "fortune-app"

spec:

selector:

app: "fortune-app"

secure: "enabled"

ports:

- protocol: "TCP"

port: 443

targetPort: 443

nodePort: 31000

type: NodePort

**Note:**

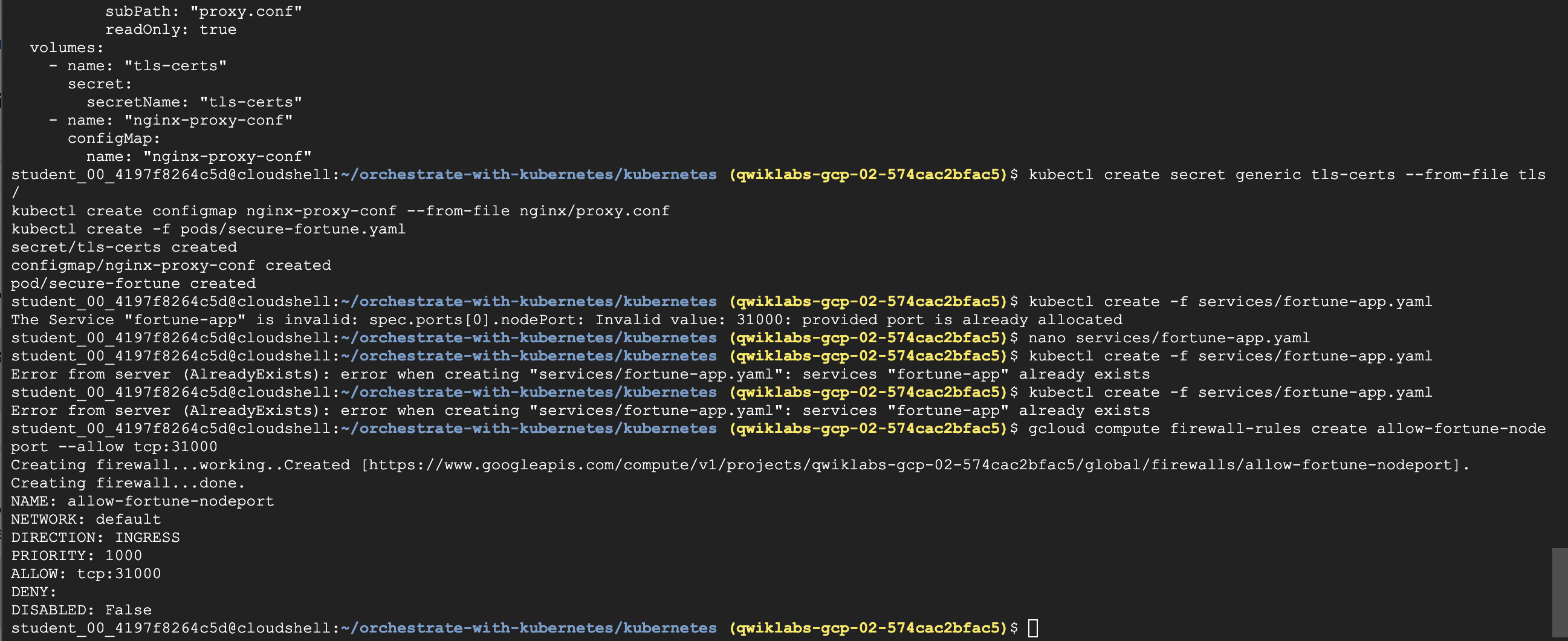
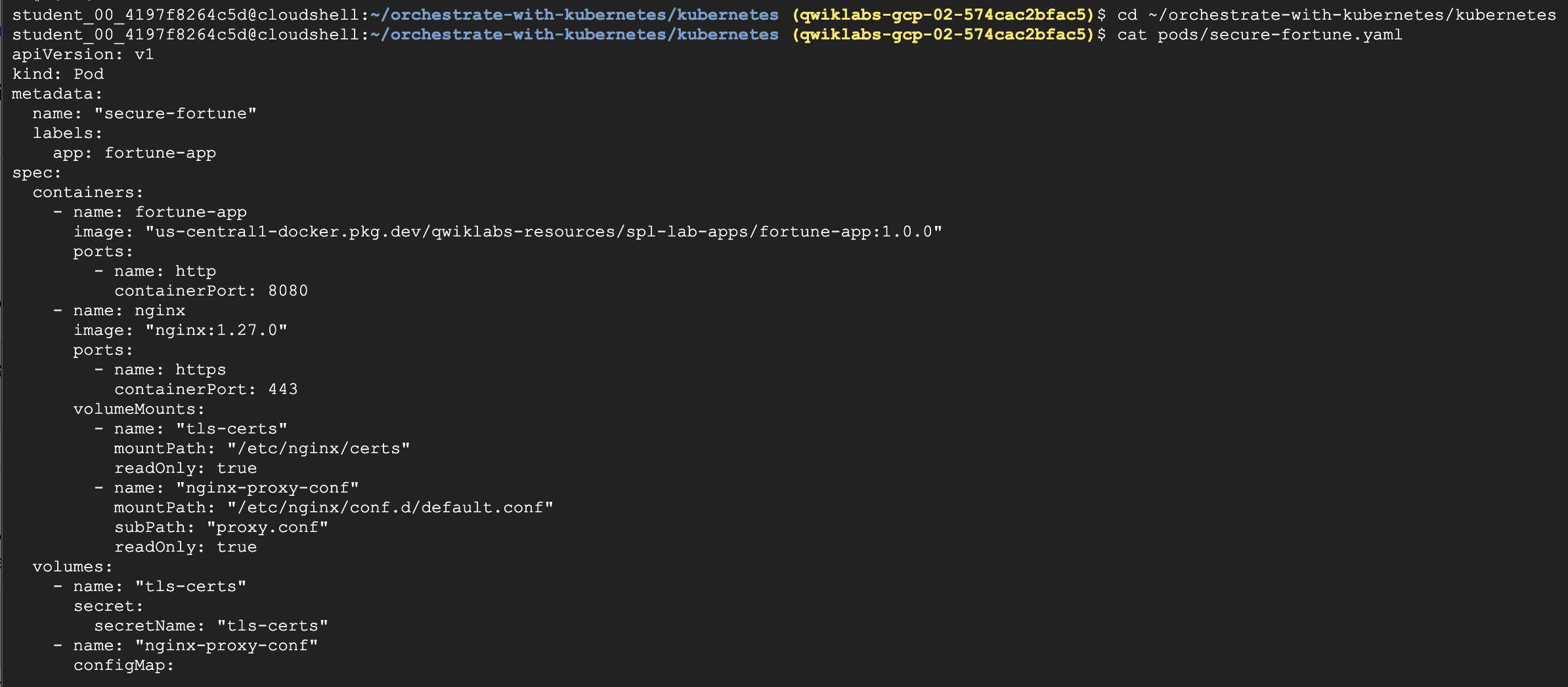
There's a selector which is used to automatically find and expose any Pods with the labels app: fortune-app and secure: enabled.

You have to expose the nodeport here because this is how you forward external traffic from port 31000 to nginx (on port 443).5. Use the kubectl create command to create the fortune-app service from the configuration file:

kubectl create -f services/fortune-app.yaml

(Output):

service/fortune-app created



## ****Task 8. Add labels to Pods****

Currently the fortune-app service does not have endpoints. One way to troubleshoot an issue like this is to use the kubectl get pods command with a label query.

1. You can see that you have a Pod running with the fortune-app label:

kubectl get pods -l "app=fortune-app"

1. But what about "app=fortune-app" and "secure=enabled"?

kubectl get pods -l "app=fortune-app,secure=enabled"

Notice this label query does not print any results. It seems you need to add the "secure=enabled" label to them.

1. Use the kubectl label command to add the missing secure=enabled label to the secure-fortune Pod. Afterwards, you can check and see that your labels have been updated.

kubectl label pods secure-fortune 'secure=enabled'

kubectl get pods secure-fortune --show-labels

1. Now that your Pods are correctly labeled, view the list of endpoints on the fortune-app service:

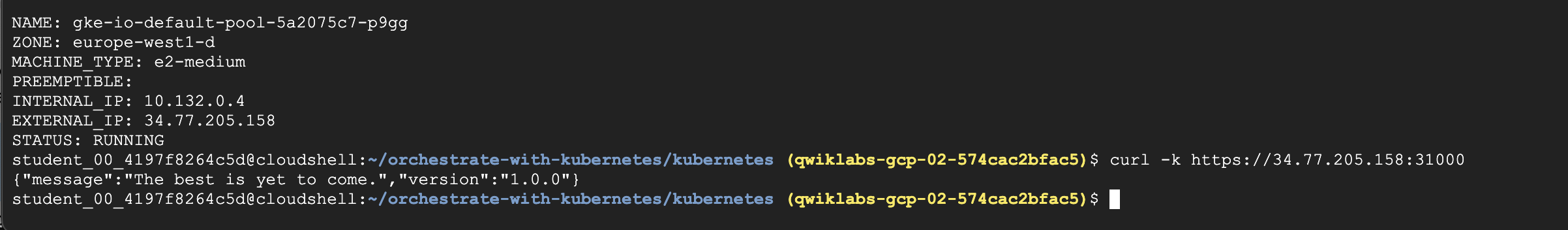
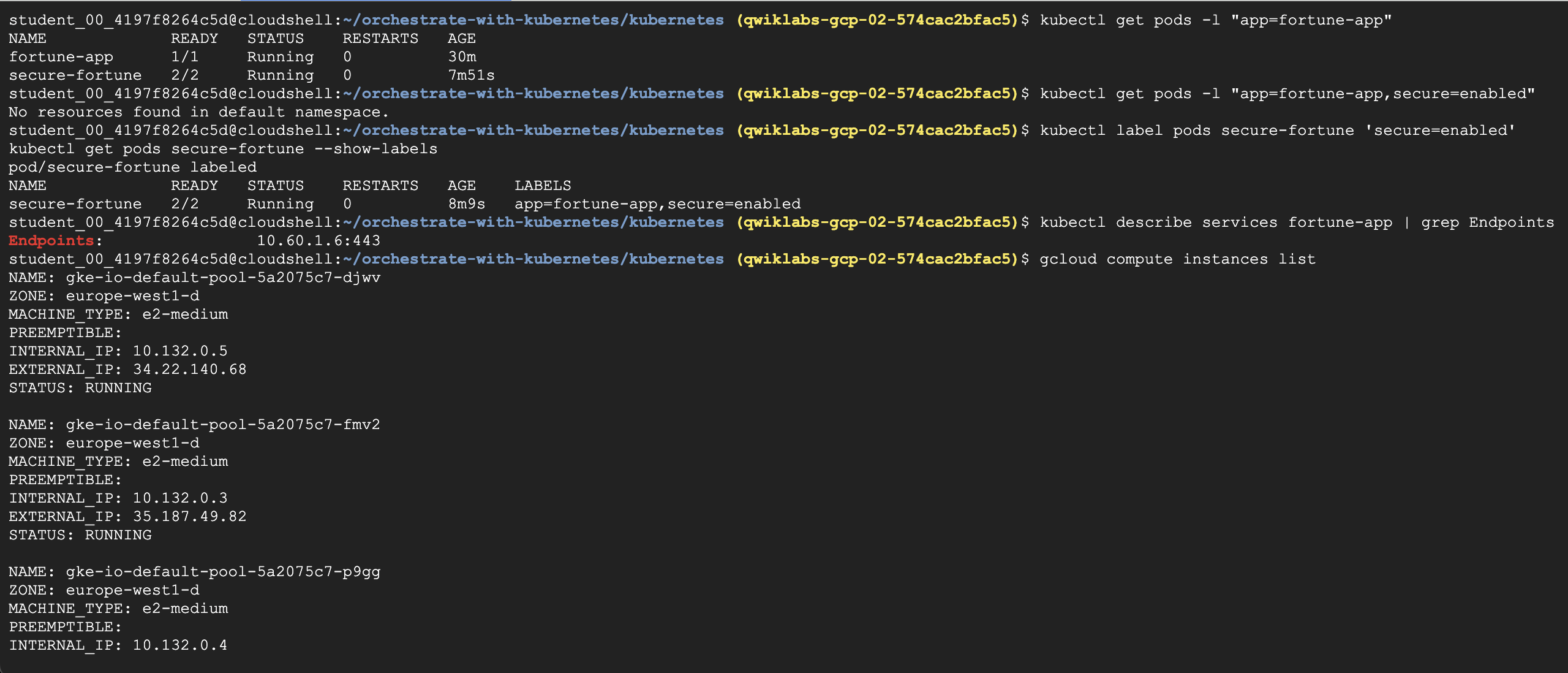
kubectl describe services fortune-app | grep Endpoints

1. Get an external IP address for one of the nodes.

gcloud compute instances list

1. Try hitting the secure-fortune service using curl:

curl -k https://<EXTERNAL\_IP>:31000



## Task 9. About Deployments

The goal of this lab is to get you ready for scaling and managing containers in production. That's where [Deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#what-is-a-deployment) come in. Deployments are a declarative way to ensure that the number of Pods running is equal to the desired number of Pods, specified by the user.

The main benefit of Deployments is in abstracting away the low level details of managing Pods. Behind the scenes Deployments use [Replica Sets](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/) to manage starting and stopping the Pods. If Pods need to be updated or scaled, the Deployment will handle that. Deployment also handles restarting Pods if they happen to go down for some reason.

Look at a quick example:

Pods are tied to the lifetime of the Node they are created on. In the example above, Node3 went down (taking a Pod with it). Instead of manually creating a new Pod and finding a Node for it, your Deployment created a new Pod and started it on Node2.

That's pretty cool!

It's time to combine everything you learned about Pods and Services to break up the fortune-app application into smaller Services using Deployments.

## ****Task 10. Create Deployments****

You're going to break the fortune-app app into three separate pieces:

* **auth** - Generates JWT tokens for authenticated users.
* **fortune** - Serves fortunes to authenticated users.
* **frontend** - Routes traffic to the auth and fortune services.

You are ready to create Deployments, one for each service. Afterwards, you'll define internal services for the auth and fortune Deployments and an external service for the frontend Deployment. Once finished, you'll be able to interact with the microservices just like with the monolith, only now each piece is able to be scaled and deployed, independently!

1. Get started by examining the auth Deployment configuration file:

cat deployments/auth.yaml

(Output)

apiVersion: apps/v1

kind: Deployment

metadata:

name: auth

spec:

selector:

matchLabels:

app: auth

replicas: 1

template:

metadata:

labels:

app: auth

spec:

containers:

- name: auth

image: "us-central1-docker.pkg.dev/qwiklabs-resources/spl-lab-apps/auth-service:1.0.0"

ports:

- name: http

containerPort: 8080

The Deployment creates 1 replica of the auth container.

When you run the kubectl create command to create the auth Deployment it will make one Pod that conforms to the data in the Deployment manifest. This means you can scale the number of Pods by changing the number specified in the Replicas field.

1. Go ahead and create your Deployment object:

kubectl create -f deployments/auth.yaml

1. It's time to create a service for your auth Deployment. Use the kubectl create command to create the auth service:

kubectl create -f services/auth.yaml

1. Do the same thing to create and expose the fortune Deployment:

kubectl create -f deployments/fortune-service.yaml

kubectl create -f services/fortune-service.yaml

1. And one more time to create and expose the frontend Deployment.

kubectl create configmap nginx-frontend-conf --from-file=nginx/frontend.conf

kubectl create -f deployments/frontend.yaml

kubectl create -f services/frontend.yaml

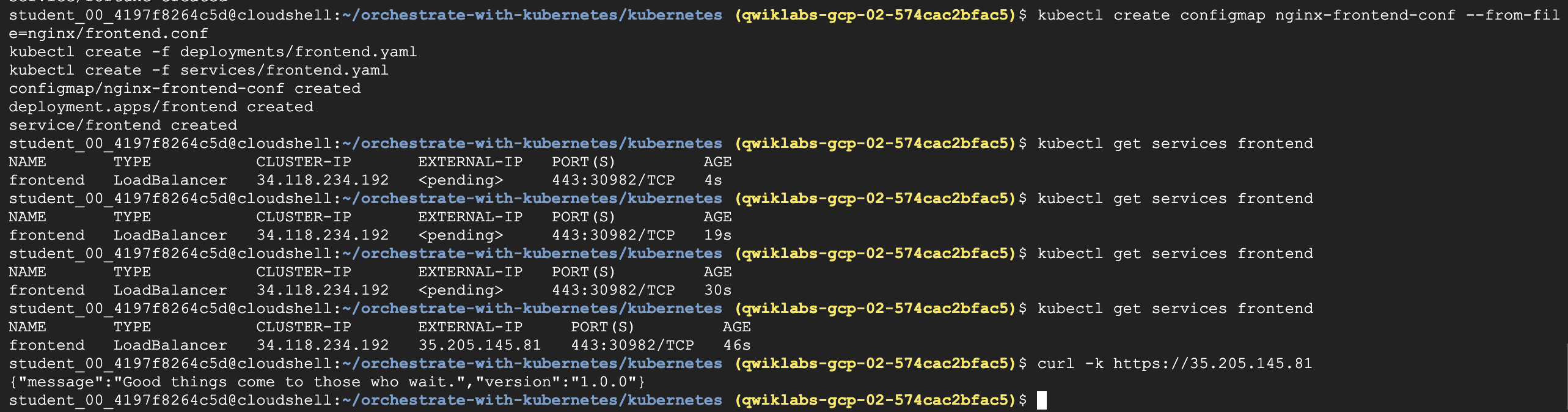
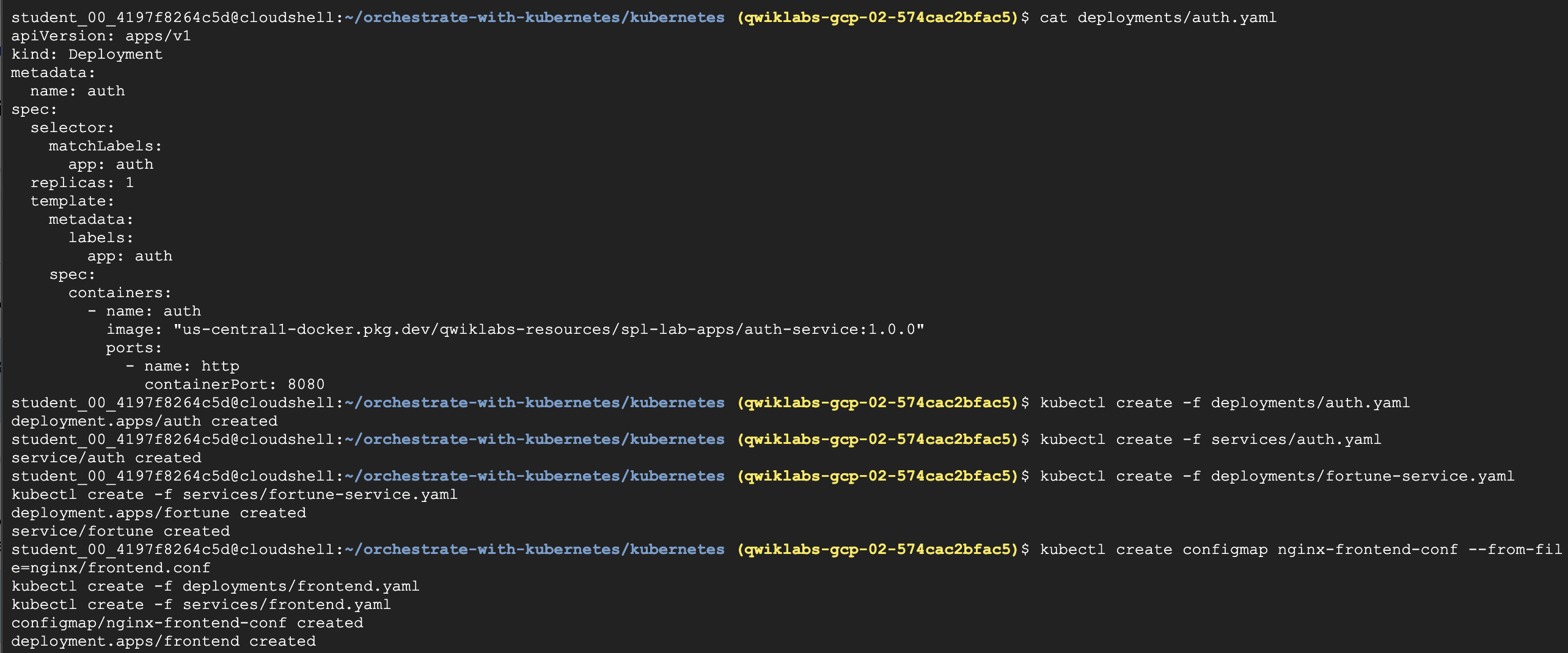
**Note:**There is one more step to creating the frontend because you need to store some configuration data with the container.

1. Interact with the frontend by grabbing its External IP and then curling to it:

kubectl get services frontend

**Note:**It might take a minute for the external IP address to be generated. Run the above command again if the EXTERNAL-IP column status is pending.

curl -k https://<EXTERNAL-IP>



## Task 1. Learn about the deployment object

To get started, take a look at the deployment object.

1. The explain command in kubectl can tell us about the deployment object:

kubectl explain deployment

Copied!

1. You can also see all of the fields using the --recursive option:

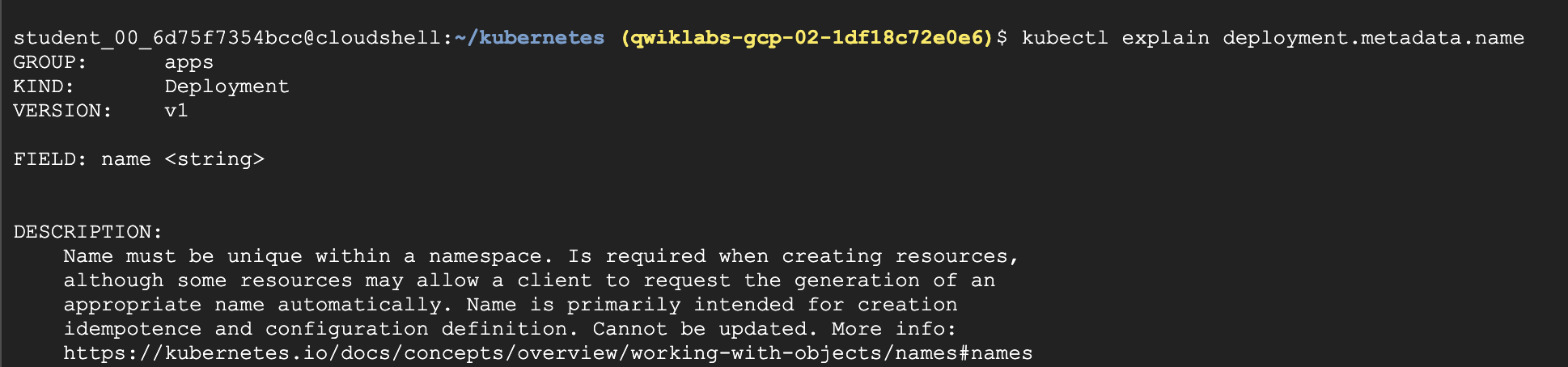
kubectl explain deployment --recursive

Copied!

1. You can use the explain command as you go through the lab to help you understand the structure of a deployment object and understand what the individual fields do:

kubectl explain deployment.metadata.name

Copied!



## Task 2. Create a deployment

1. Create the fortune-app deployment. Examine the deployment configuration file:

cat deployments/fortune-app-blue.yaml

Copied!

Output:

# orchestrate-with-kubernetes/kubernetes/deployments/fortune-app-blue.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: fortune-app-blue

spec:

replicas: 3

selector:

matchLabels:

app: fortune-app

template:

metadata:

labels:

app: fortune-app

track: stable

version: "1.0.0"

spec:

containers:

- name: fortune-app

# The new, centralized image path

image: "us-central1-docker.pkg.dev/qwiklabs-resources/spl-lab-apps/fortune-service:1.0.0"

ports:

- name: http

containerPort: 8080

...

Notice how the deployment is creating three replicas and it's using version 1.0.0 of the fortune-service container.

1. Go ahead and create your deployment object using kubectl create:

kubectl create -f deployments/fortune-app-blue.yaml

Copied!

1. Once you have created the deployment, you can verify that it was created:

kubectl get deployments

Copied!

1. Once the deployment is created, Kubernetes will create a ReplicaSet for the deployment. You can verify that a ReplicaSet was created for the deployment:

kubectl get replicasets

Copied!

You should see a ReplicaSet with a name like fortune-app-blue-xxxxxxx

1. View the Pods that were created as part of the deployment:

kubectl get pods

Copied!

1. Now, create a service to expose the fortune-app deployment externally.

kubectl create -f services/fortune-app.yaml

Copied!

1. Interact with the fortune-app by grabbing its external IP and then curling the /version endpoint:

kubectl get services fortune-app

Copied!

**Note:** It may take a few seconds before the External-IP field is populated for your service. This is normal. Just re-run the above command every few seconds until the field is populated.

curl http://<EXTERNAL-IP>/version

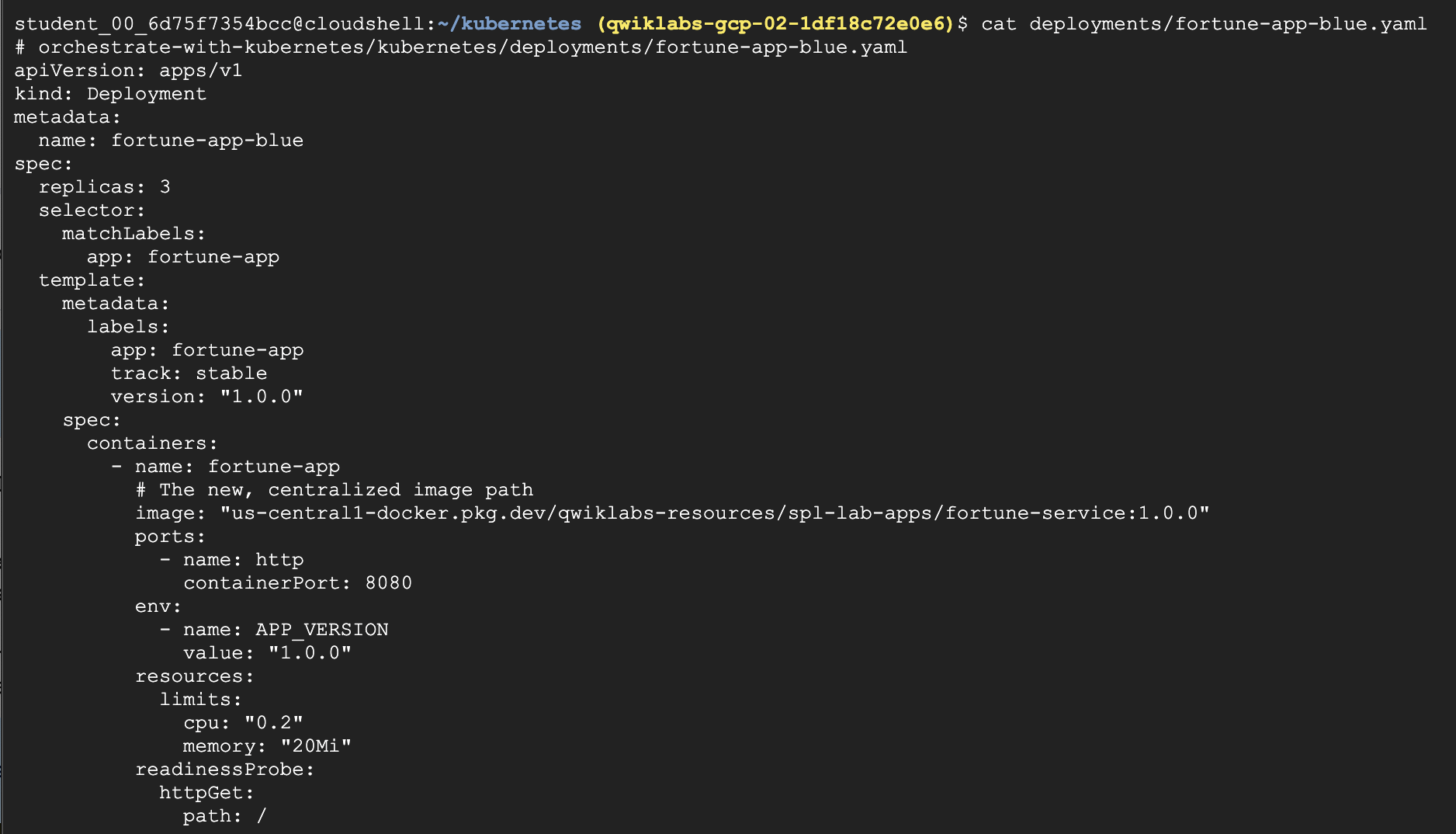
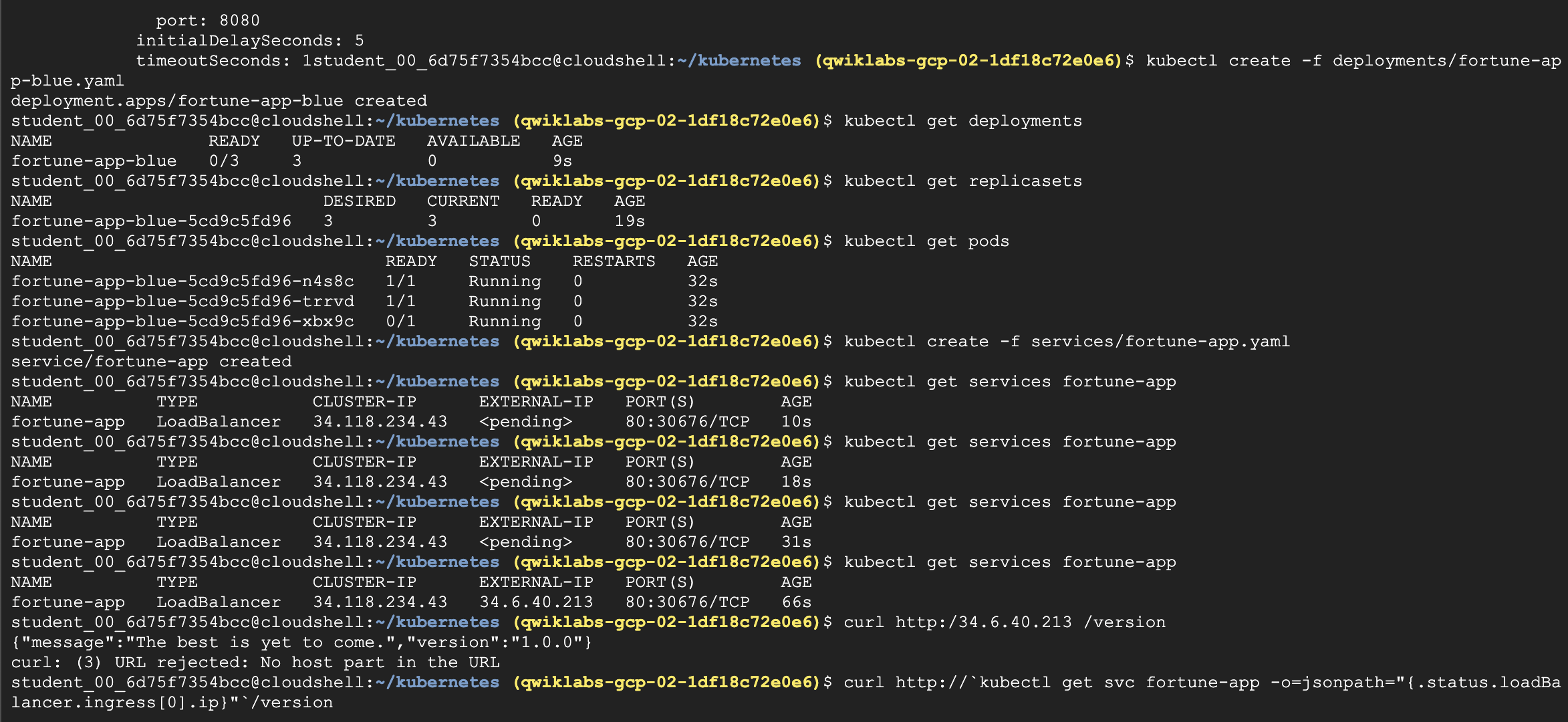
Copied!

You should get back a JSON response indicating {"version":"1.0.0"}.

1. You can also use the output templating feature of kubectl to use curl as a one-liner:

curl http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Copied!

### **Scale a deployment**

Now that you have a deployment created, you can scale it. Do this by updating the spec.replicas field.

1. The replicas field can be most easily updated using the kubectl scale command:

kubectl scale deployment fortune-app-blue --replicas=5

Copied!

**Note:** It may take a minute or so for all the new pods to start up.

1. Verify that there are now 5 fortune-app-blue Pods running:

kubectl get pods | grep fortune-app-blue | wc -l

Copied!

1. Now scale back the application:

kubectl scale deployment fortune-app-blue --replicas=3

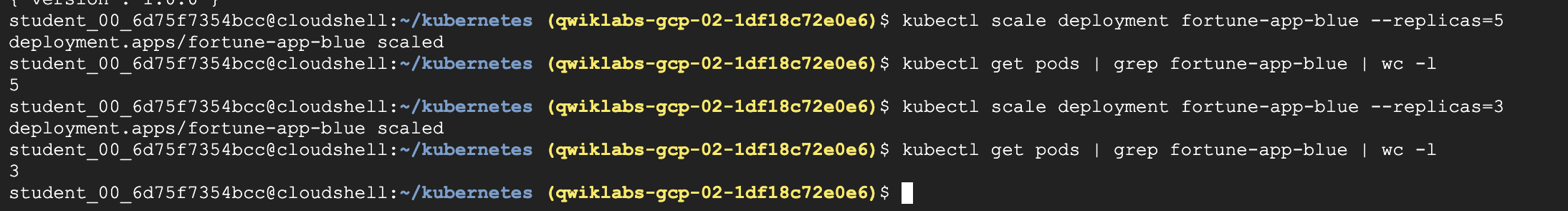
Copied!

1. Again, verify that you have the correct number of Pods:

kubectl get pods | grep fortune-app-blue | wc -l

Copied!

Now you know about Kubernetes deployments and how to manage & scale a group of Pods.



## Task 3. Rolling update

Deployments support updating images to a new version through a rolling update mechanism.

### **Trigger a rolling update**

1. To trigger a rolling update, you can simply **apply** the configuration of the "green" deployment. Kubernetes is smart enough to see the existing deployment (fortune-app-blue) and will "roll" the changes from the new file onto it.

kubectl edit deployment fortune-app-blue

Copied!

1. In the editor, find the image line and change the version tag from 1.0.0 to 2.0.0. You can edit the file by pressing i on your keyboard to enter "insert mode".

* **First, change the image tag**:
  + Find this line: image: "us-central1-docker.pkg.dev/qwiklabs-resources/spl-lab-apps/fortune-service:1.0.0"
  + And change it to: image: "us-central1-docker.pkg.dev/qwiklabs-resources/spl-lab-apps/fortune-service:2.0.0"
* **Next, update the environment variable**:
  + Find the env section and the APP\_VERSION variable.
  + Change value: "1.0.0" to value: "2.0.0"

1. Save and close the editor. You can do this by pressing Esc, then typing :wq, and pressing Enter. This will trigger the rolling update on the correct deployment and properly record its history. This will trigger the rolling update on the correct deployment and properly record its history.
2. See the new ReplicaSet that Kubernetes creates:

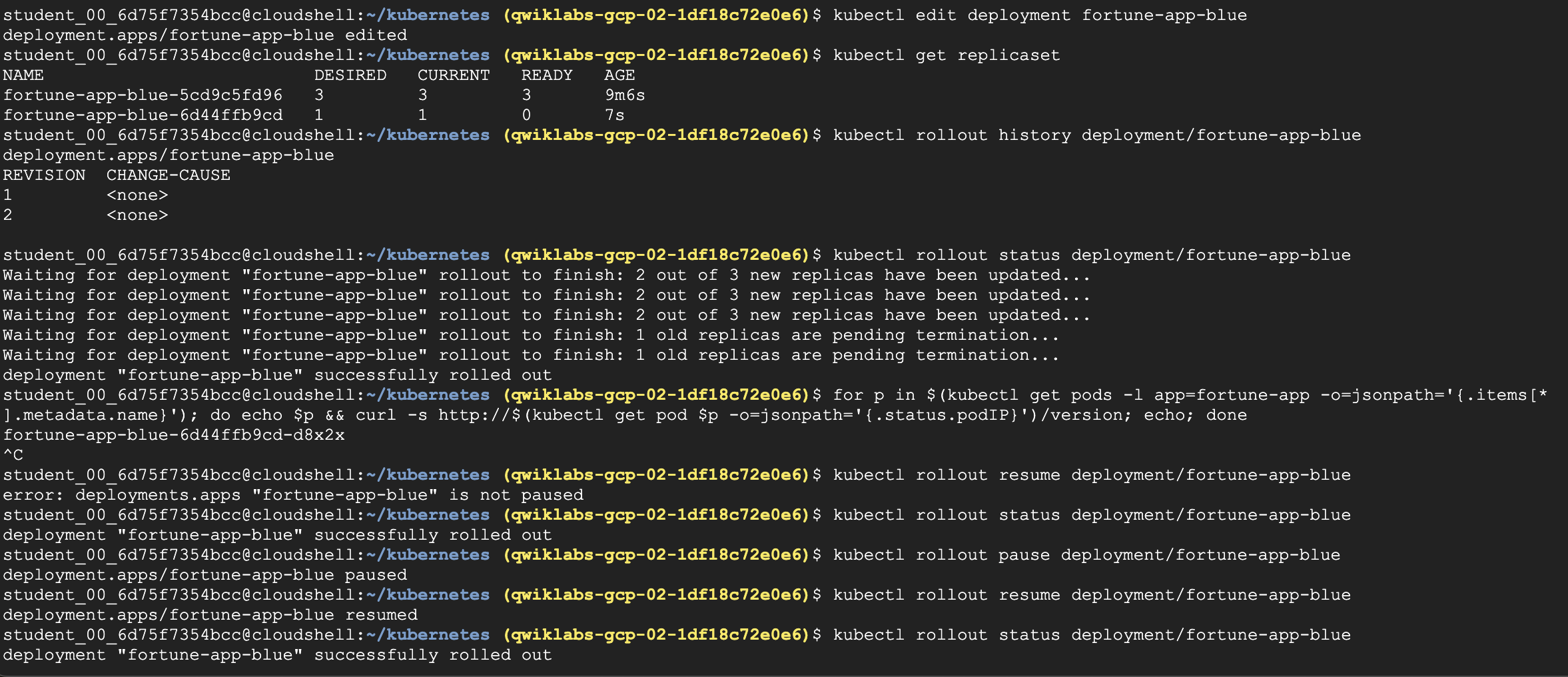
kubectl get replicaset

Copied!

1. You can also see a new entry in the rollout history:

kubectl rollout history deployment/fortune-app-blue

Copied!



### **Pause a rolling update**

1. Run the following to pause the rollout:

kubectl rollout pause deployment/fortune-app-blue

Copied!

1. Verify the current state of the rollout:

kubectl rollout status deployment/fortune-app-blue

Copied!

**Note:** The status command might immediately report "deployment "fortune-app-blue" successfully rolled out". This is expected and indicates that the pause command itself was successful. It does not mean the version update is complete.

1. Check the version of each pod. You'll see a mix of 1.0.0 and 2.0.0 pods, confirming the rollout is paused mid-way.

for p in $(kubectl get pods -l app=fortune-app -o=jsonpath='{.items[\*].metadata.name}'); do echo $p && curl -s http://$(kubectl get pod $p -o=jsonpath='{.status.podIP}')/version; echo; done

Copied!

1. Press Ctrl+C to exit the loop.

### **Resume a rolling update**

1. Continue the rollout using the resume command:

kubectl rollout resume deployment/fortune-app-blue

Copied!

1. When the rollout is complete, you should see the following when running the status command:

kubectl rollout status deployment/fortune-app-blue

Copied!

### **Roll back an update**

Assume that a bug was detected in your new version.

1. Use the rollout command to roll back to the previous version:

kubectl rollout undo deployment/fortune-app-blue

Copied!

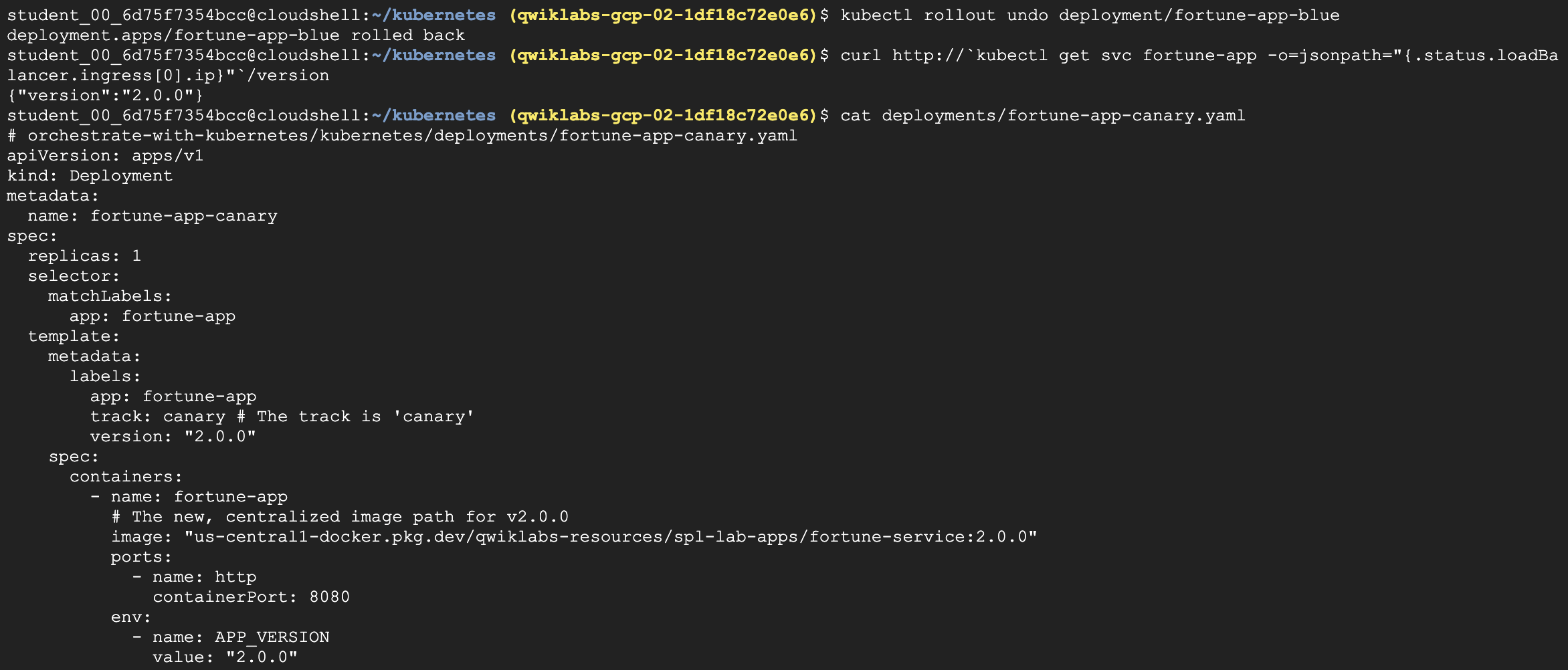
**Note:** the rollback may take a few moments to complete.

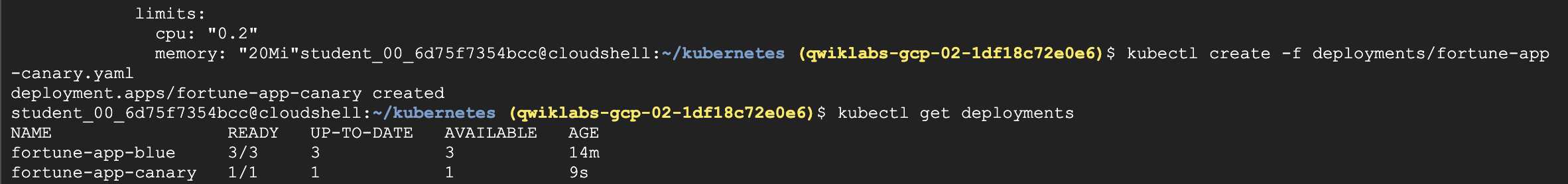
1. Verify that all the Pods have rolled back to version 1.0.0:

curl http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Copied!

Great! You learned how to do a rolling update for Kubernetes deployments.





## Task 4. Canary deployments

When you want to test a new deployment in production with a subset of your users, use a canary deployment.

### **Create a canary deployment**

1. First, create a new canary deployment for the new version using the fortune-app-canary.yaml file:

cat deployments/fortune-app-canary.yaml

Copied!

1. Now create the canary deployment:

kubectl create -f deployments/fortune-app-canary.yaml

Copied!

1. After the canary deployment is created, you should have two deployments. Verify with this command:

kubectl get deployments

Copied!

The fortune-app service has a selector for app: fortune-app, which will match pods in **both** the fortune-app-blue (prod) and fortune-app-canary deployments.

### **Verify the canary deployment**

1. You can verify the version being served by making requests to the service.

for i in {1..10}; do curl -s http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version; echo;

done

Copied!

1. Run this several times and you should see that most of the requests are served by version 1.0.0 and a small subset are served by 2.0.0.

#### Test completed task

Click **Check my progress** below to check your lab progress. If you successfully created Canary deployment, you'll see an assessment score.



## Task 5. Blue-green deployments

For blue-green deployments, you'll create two separate deployments and switch traffic between them by updating the service selector.

### **The service**

1. First, update the service to point only to the "blue" version (1.0.0).

kubectl apply -f services/fortune-app-blue-service.yaml

Copied!

### **Updating using Blue-Green deployment**

1. Now, create the new "green" deployment for version 2.0.0.

kubectl create -f deployments/fortune-app-green.yaml

Copied!

1. Once the green deployment has started, verify that the current version being served is still 1.0.0.

curl http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Copied!

1. Now, update the service to point to the new "green" version:

kubectl apply -f services/fortune-app-green-service.yaml

Copied!

1. When the service is updated, the "green" deployment will be used immediately. You can now verify that version 2.0.0 is always being served:

curl http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Copied!

### **Blue-Green rollback**

1. To roll back, simply re-apply the service manifest for the "blue" deployment:

kubectl apply -f services/fortune-app-blue-service.yaml

Copied!

1. Once you have updated the service, your rollback will have been successful. Verify that version 1.0.0 is now being used:

curl http://`kubectl get svc fortune-app -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version

Copied!

You did it! You learned about blue-green deployments and how to deploy updates to applications that need to switch versions all at once.

