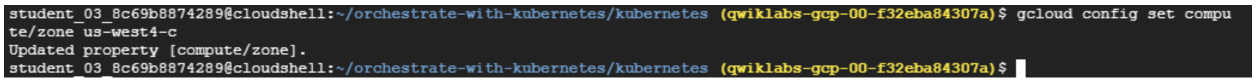
Managing Deployments Using Kubernetes Engine

Set the zone

Set your working Google Cloud zone by running the following command, substituting the local zone as ZONE:

gcloud config set compute/zone ZONE

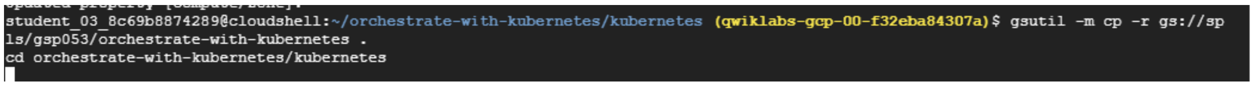


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00-f32eba84307a)$ gcloud config set compu
te/zone us-west4-c
Updated property [compute/zone].
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00-f32eba84307a)$
```

Get sample code for this lab

1. Get the sample code for creating and running containers and deployments:

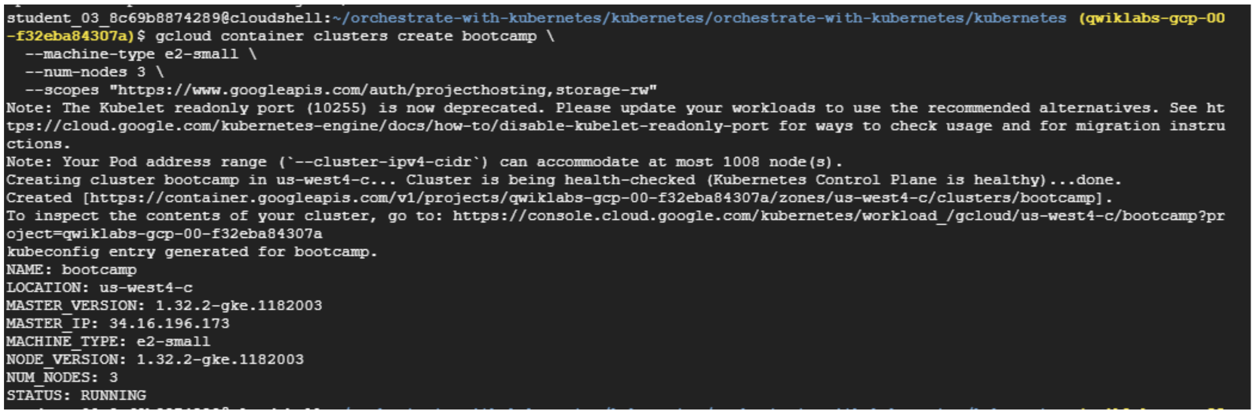
gsutil -m cp -r gs://spls/gsp053/orchestrate-with-kubernetes .
cd orchestrate-with-kubernetes/kubernetes



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00-f32eba84307a)$ gsutil -m cp -r gs://sp
ls/gsp053/orchestrate-with-kubernetes .
cd orchestrate-with-kubernetes/kubernetes
```

2. Create a cluster with 3 nodes (this will take a few minutes to complete):

gcloud container clusters create bootcamp \
--machine-type e2-small \
--num-nodes 3 \
--scopes https://www.googleapis.com/auth/projecthosting,storage-rw



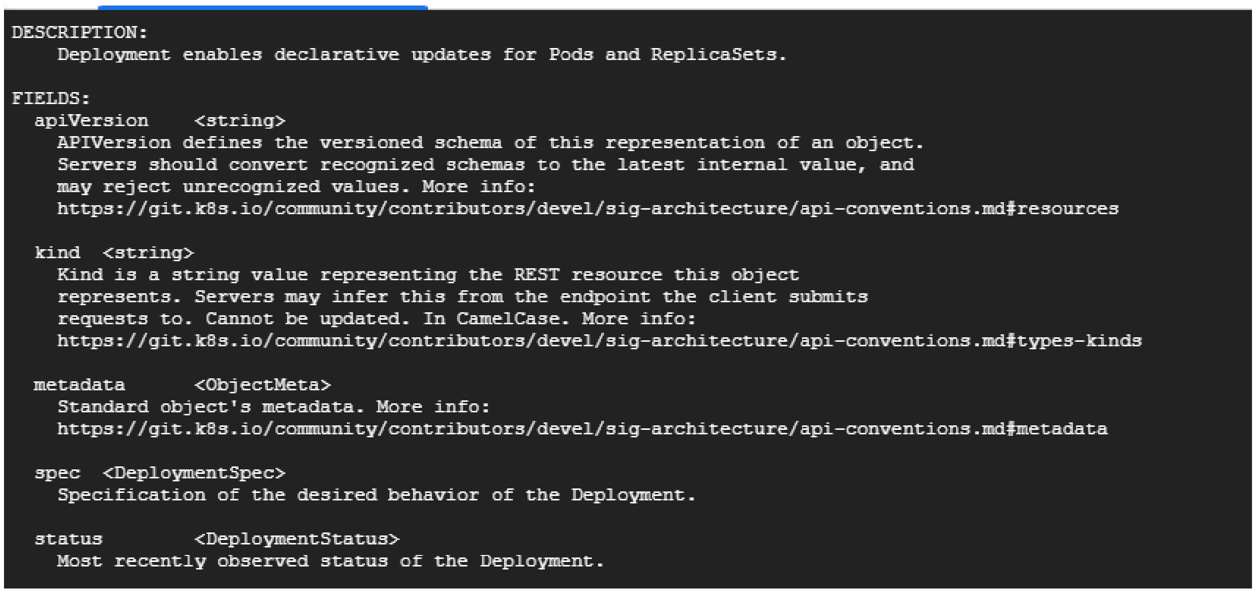
```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ gcloud container clusters create bootcamp \
  --machine-type e2-small \
  --num-nodes 3 \
  --scopes "https://www.googleapis.com/auth/projecthosting,storage-rw"
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See ht
tps://cloud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-port for ways to check usage and for migration instru
ctions.
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster bootcamp in us-west4-c... Cluster is being health-checked (Kubernetes Control Plane is healthy)...done.
Created [https://container.googleapis.com/v1/projects/qwiklabs-gcp-00-f32eba84307a/zones/us-west4-c/clusters/bootcamp].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-west4-c/bootcamp?pr
oject=qwiklabs-gcp-00-f32eba84307a
kubeconfig entry generated for bootcamp.
NAME: bootcamp
LOCATION: us-west4-c
MASTER_VERSION: 1.32.2-gke.1182003
MASTER_IP: 34.16.196.173
MACHINE_TYPE: e2-small
NODE_VERSION: 1.32.2-gke.1182003
NUM_NODES: 3
STATUS: RUNNING
```

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:

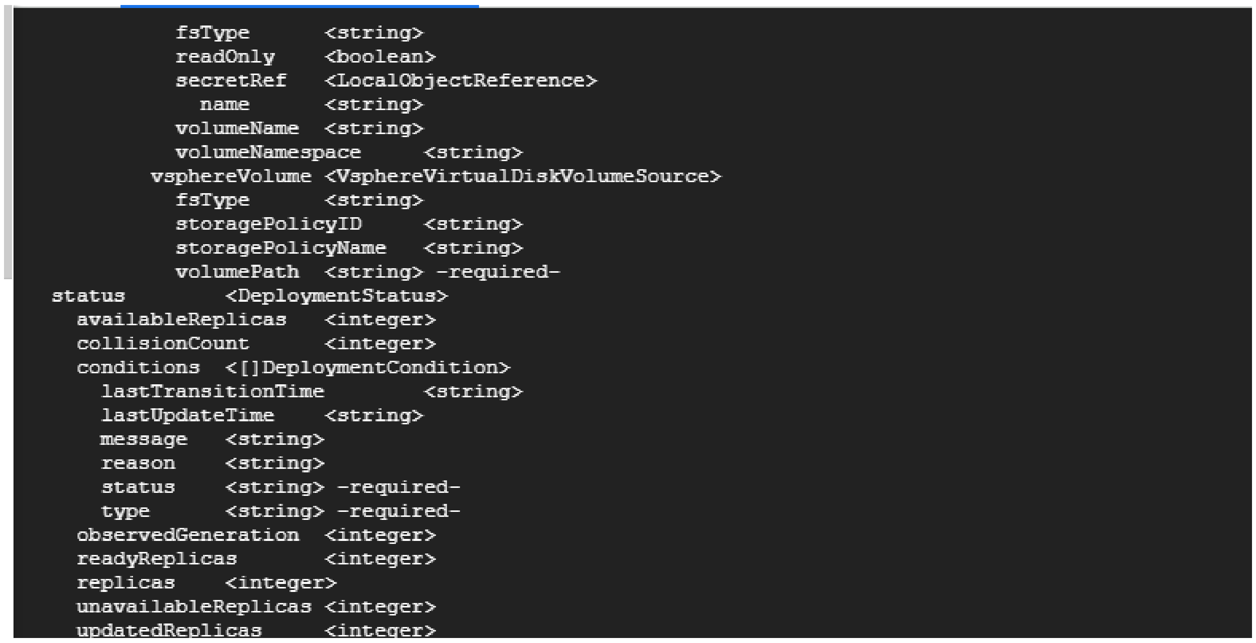
kubectl explain deployment



```
DESCRIPTION:
    Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
  apiVersion    <string>
    APIVersion defines the versioned schema of this representation of an object.
    Servers should convert recognized schemas to the latest internal value, and
    may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

  kind  <string>
    Kind is a string value representing the REST resource this object
    represents. Servers may infer this from the endpoint the client submits
    requests to. Cannot be updated. In CamelCase. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  metadata      <ObjectMeta>
    Standard object's metadata. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

  spec  <DeploymentSpec>
    Specification of the desired behavior of the Deployment.

  status        <DeploymentStatus>
    Most recently observed status of the Deployment.
```

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

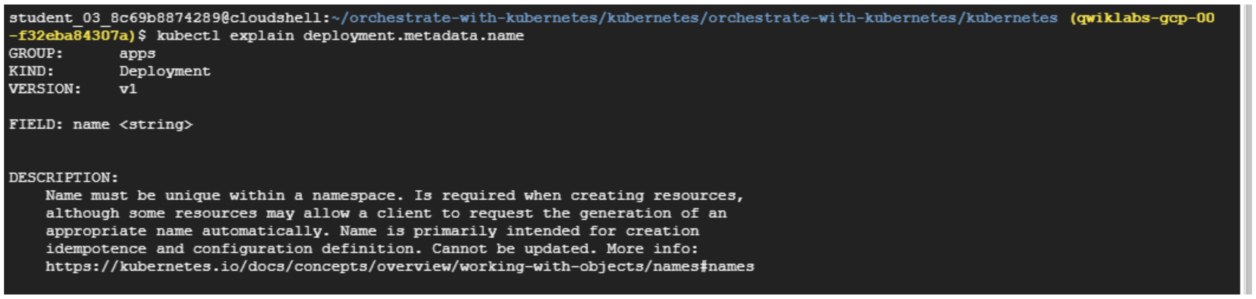
kubectl explain deployment –recursive



```
            fsType      <string>
            readOnly    <boolean>
            secretRef   <LocalObjectReference>
              name      <string>
            volumeName  <string>
            volumeNamespace     <string>
          vsphereVolume <VsphereVirtualDiskVolumeSource>
            fsType      <string>
            storagePolicyID     <string>
            storagePolicyName   <string>
            volumePath  <string> -required-
  status        <DeploymentStatus>
    availableReplicas   <integer>
    collisionCount      <integer>
    conditions  <[]DeploymentCondition>
      lastTransitionTime        <string>
      lastUpdateTime    <string>
      message   <string>
      reason    <string>
      status    <string> -required-
      type      <string> -required-
    observedGeneration  <integer>
    readyReplicas       <integer>
    replicas    <integer>
    unavailableReplicas <integer>
    updatedReplicas     <integer>
```

3. 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

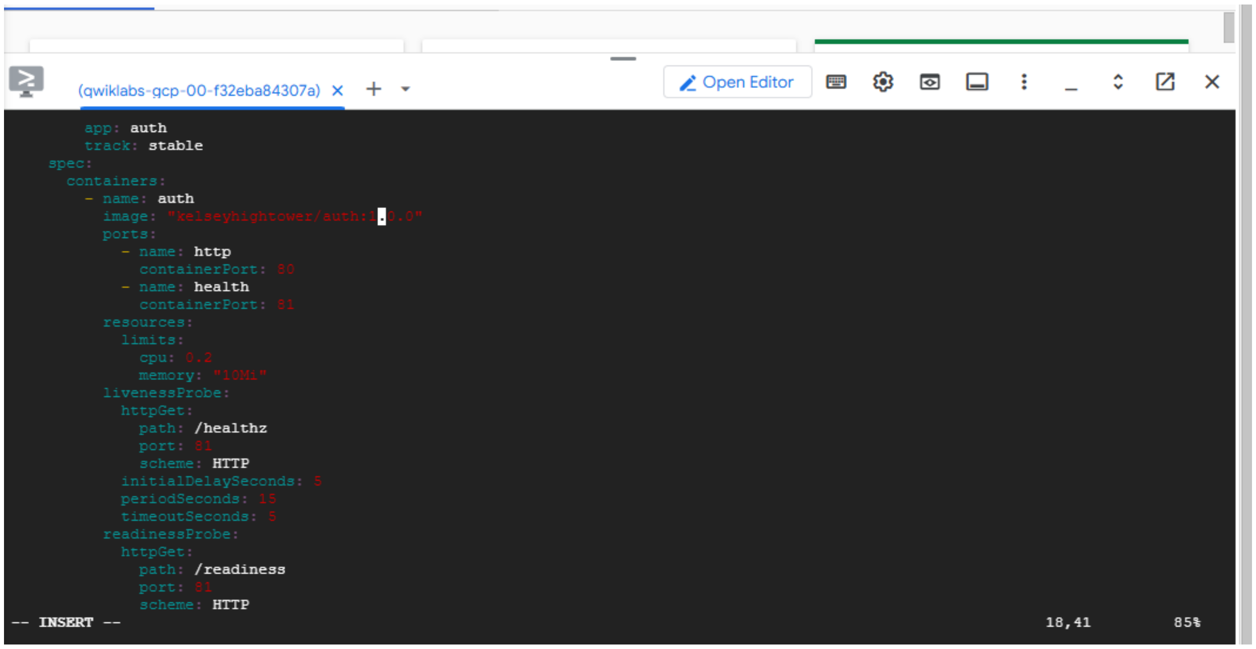


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl explain deployment.metadata.name
GROUP:      apps
KIND:       Deployment
VERSION:    v1

FIELD: name <string>


DESCRIPTION:
    Name must be unique within a namespace. Is required when creating resources,
    although some resources may allow a client to request the generation of an
    appropriate name automatically. Name is primarily intended for creation
    idempotence and configuration definition. Cannot be updated. More info:
    https://kubernetes.io/docs/concepts/overview/working-with-objects/names#names
```

Task 2. Create a deployment

1. Update the deployments/auth.yaml configuration file:

vi deployments/auth.yaml



```
        app: auth
        track: stable
    spec:
      containers:
        - name: auth
          image: "kelseyhightower/auth:1.0.0"
          ports:
            - name: http
              containerPort: 80
            - name: health
              containerPort: 81
          resources:
            limits:
              cpu: 0.2
              memory: "10Mi"
          livenessProbe:
            httpGet:
              path: /healthz
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            periodSeconds: 15
            timeoutSeconds: 5
          readinessProbe:
            httpGet:
              path: /readiness
              port: 81
              scheme: HTTP
-- INSERT --                                                          18,41         85%
```

2. Start the editor:

i

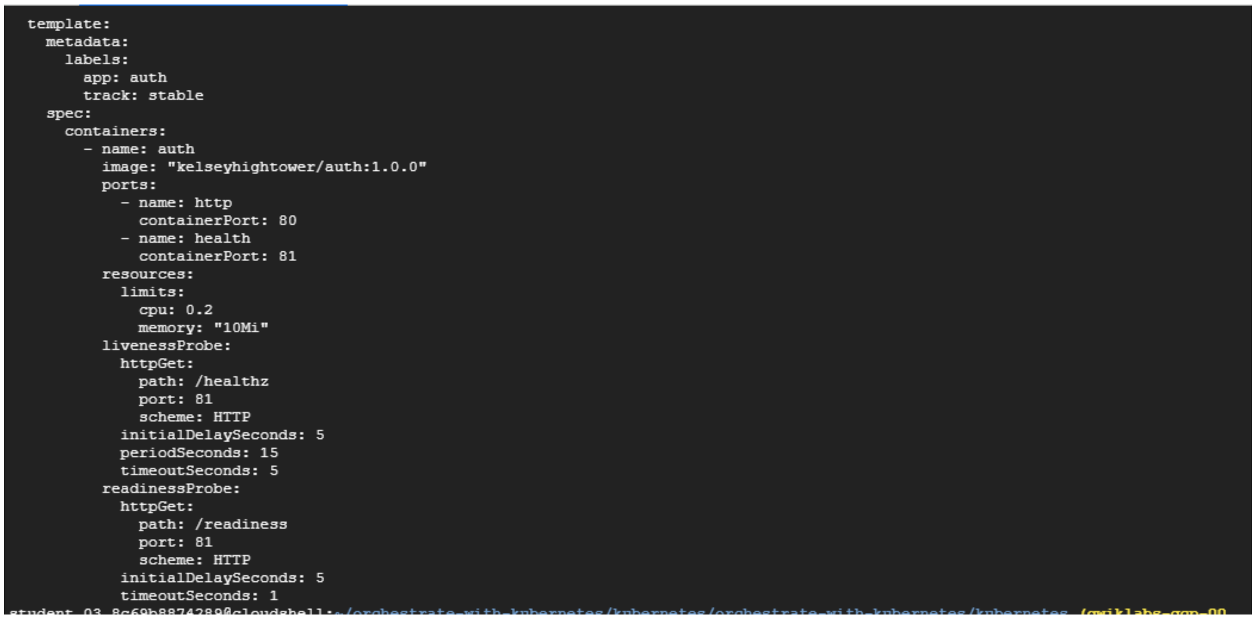
3. Change the image in the containers section of the deployment to the following:

4. **...**

containers:

- name: auth

image: "kelseyhightower/auth:1.0.0"



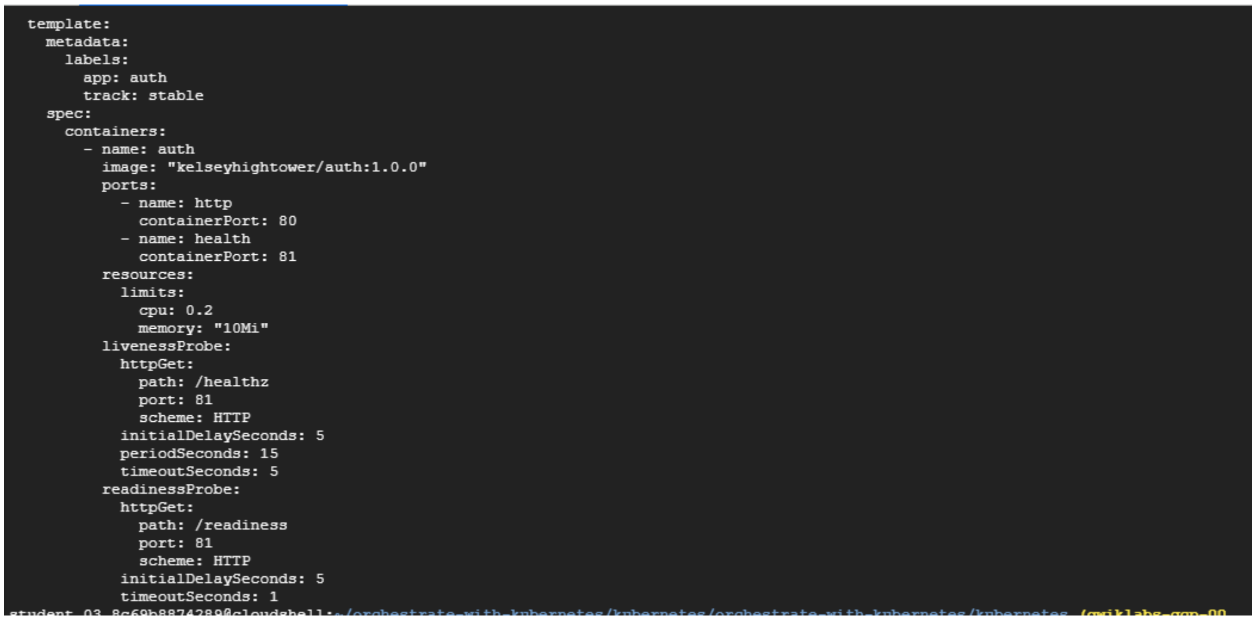
```
  template:
    metadata:
      labels:
        app: auth
        track: stable
    spec:
      containers:
        - name: auth
          image: "kelseyhightower/auth:1.0.0"
          ports:
            - name: http
              containerPort: 80
            - name: health
              containerPort: 81
          resources:
            limits:
              cpu: 0.2
              memory: "10Mi"
          livenessProbe:
            httpGet:
              path: /healthz
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            periodSeconds: 15
            timeoutSeconds: 5
          readinessProbe:
            httpGet:
              path: /readiness
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            timeoutSeconds: 1
student_03_8c69b88742890@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
```

Save the auth.yaml file: press <Esc> then type:

:wq

5. Press <Enter>. Now create a simple deployment. Examine the deployment configuration file:

cat deployments/auth.yaml



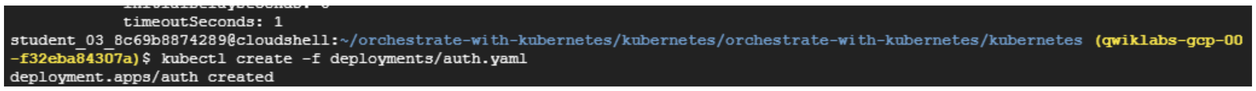
```
  template:
    metadata:
      labels:
        app: auth
        track: stable
    spec:
      containers:
        - name: auth
          image: "kelseyhightower/auth:1.0.0"
          ports:
            - name: http
              containerPort: 80
            - name: health
              containerPort: 81
          resources:
            limits:
              cpu: 0.2
              memory: "10Mi"
          livenessProbe:
            httpGet:
              path: /healthz
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            periodSeconds: 15
            timeoutSeconds: 5
          readinessProbe:
            httpGet:
              path: /readiness
              port: 81
              scheme: HTTP
            initialDelaySeconds: 5
            timeoutSeconds: 1
student_03_8c69b88742890@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
```

Notice how the deployment is creating one replica and it's using version 1.0.0 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.

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

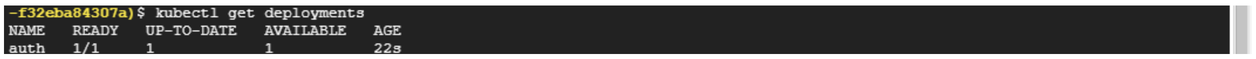
kubectl create -f deployments/auth.yaml



```
          timeoutSeconds: 1
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create -f deployments/auth.yaml
deployment.apps/auth created
```

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

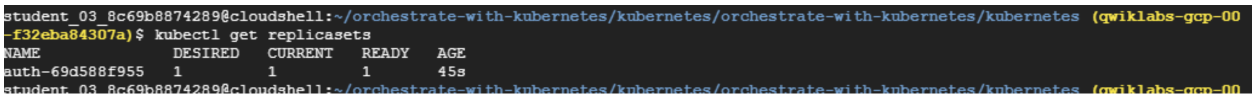
kubectl get deployments



```
-f32eba84307a)$ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
auth   1/1     1            1           22s
```

8. 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

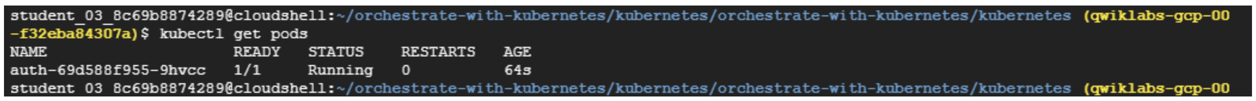


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get replicasets
NAME              DESIRED   CURRENT   READY   AGE
auth-69d588f955   1         1         1       45s
```

You should see a ReplicaSet with a name like auth-xxxxxxx

9. View the Pods that were created as part of the deployment. The single Pod is created by the Kubernetes when the ReplicaSet is created:

kubectl get pods

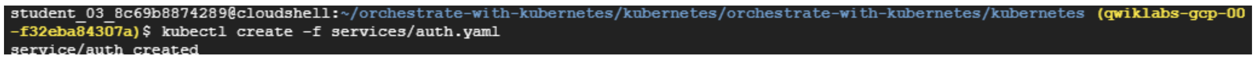


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
auth-69d588f955-9hvcc   1/1     Running   0          64s
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
```

It's time to create a service for the auth deployment. You've already seen service manifest files, so the details won't be shared here.

10. Use the kubectl create command to create the auth service:

kubectl create -f services/auth.yaml

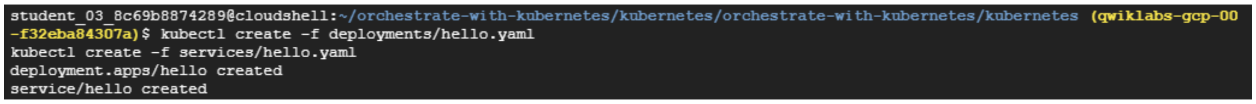


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create -f services/auth.yaml
service/auth created
```

11. Now, do the same thing to create and expose the hello deployment:

kubectl create -f deployments/hello.yaml

kubectl create -f services/hello.yaml



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create -f deployments/hello.yaml
kubectl create -f services/hello.yaml
deployment.apps/hello created
service/hello created
```

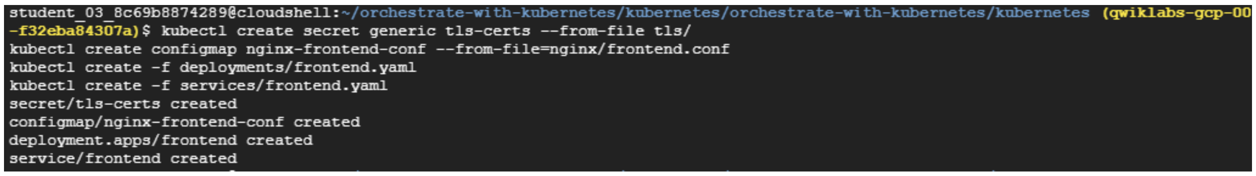
12. And one more time to create and expose the frontend deployment:

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

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

kubectl create -f deployments/frontend.yaml

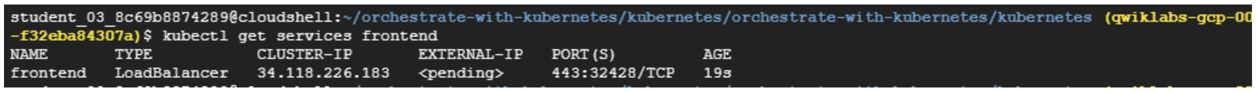
kubectl create -f services/frontend.yaml



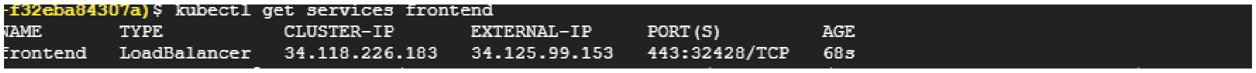
```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create secret generic tls-certs --from-file tls/
kubectl create configmap nginx-frontend-conf --from-file=nginx/frontend.conf
kubectl create -f deployments/frontend.yaml
kubectl create -f services/frontend.yaml
secret/tls-certs created
configmap/nginx-frontend-conf created
deployment.apps/frontend created
service/frontend created
```

13. Interact with the frontend by grabbing its external IP and then curling to it:

kubectl get services frontend

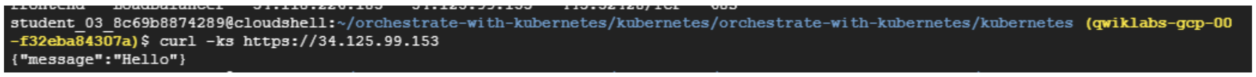


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get services frontend
NAME       TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
frontend   LoadBalancer   34.118.226.183   <pending>     443:32428/TCP   19s
```



```
f32eba84307a)$ kubectl get services frontend
NAME       TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
frontend   LoadBalancer   34.118.226.183   34.125.99.153   443:32428/TCP   68s
```

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

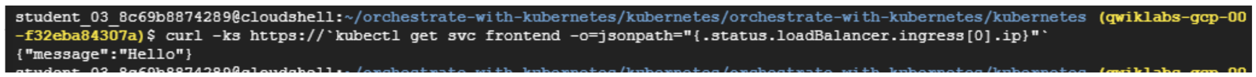


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://34.125.99.153
{"message":"Hello"}
```

And you get the hello response back.

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

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`



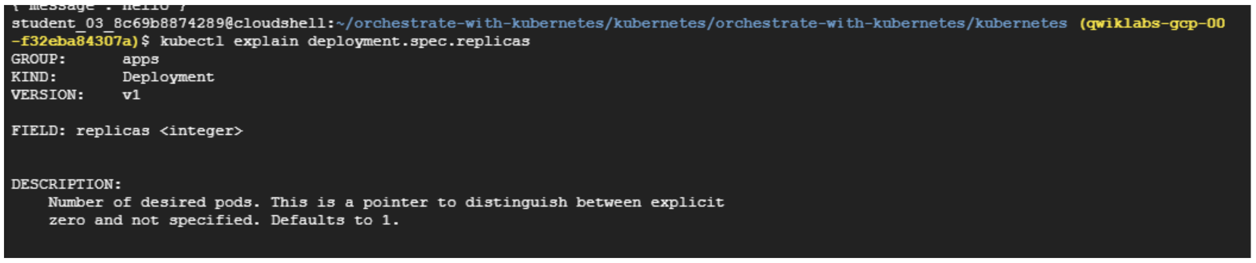
```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`
{"message":"Hello"}
```

Scale a deployment

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

1. Look at an explanation of this field using the kubectl explain command again:

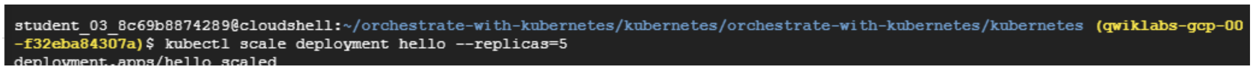
kubectl explain deployment.spec.replicas



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl explain deployment.spec.replicas
GROUP:      apps
KIND:       Deployment
VERSION:    v1

FIELD: replicas <integer>


DESCRIPTION:
    Number of desired pods. This is a pointer to distinguish between explicit
    zero and not specified. Defaults to 1.
```

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

kubectl scale deployment hello --replicas=5

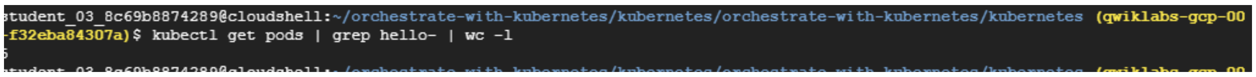


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl scale deployment hello --replicas=5
deployment.apps/hello scaled
```

After the deployment is updated, Kubernetes will automatically update the associated ReplicaSet and start new Pods to make the total number of Pods equal 5.

3. Verify that there are now 5 hello Pods running:

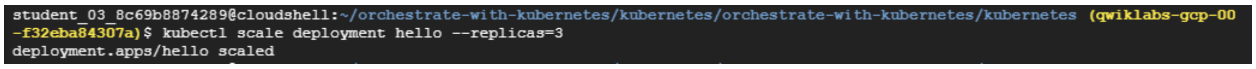
kubectl get pods | grep hello- | wc –l



```
tudent_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
f32eba84307a)$ kubectl get pods | grep hello- | wc -l
5
```

4. Now scale back the application:

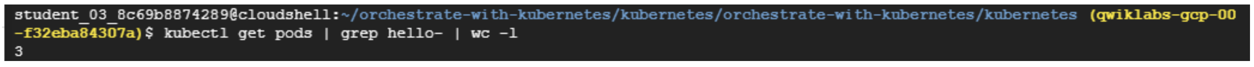
kubectl scale deployment hello --replicas=3



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl scale deployment hello --replicas=3
deployment.apps/hello scaled
```

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

kubectl get pods | grep hello- | wc –l

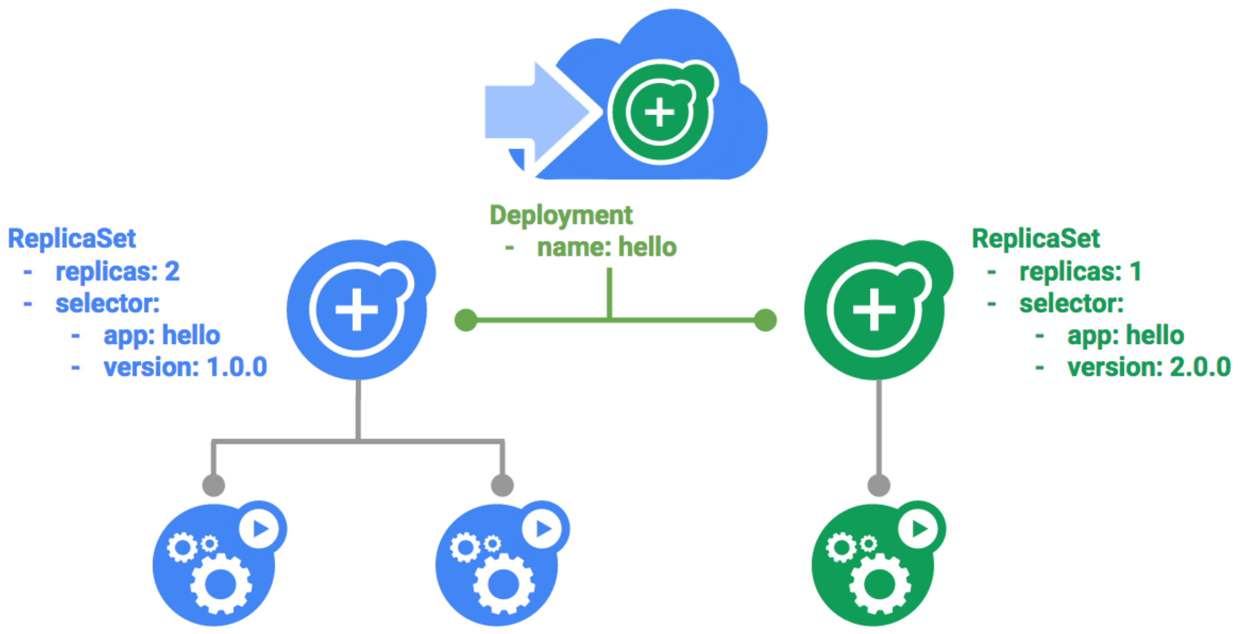


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get pods | grep hello- | wc -l
3
```

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

Task 3. Rolling update

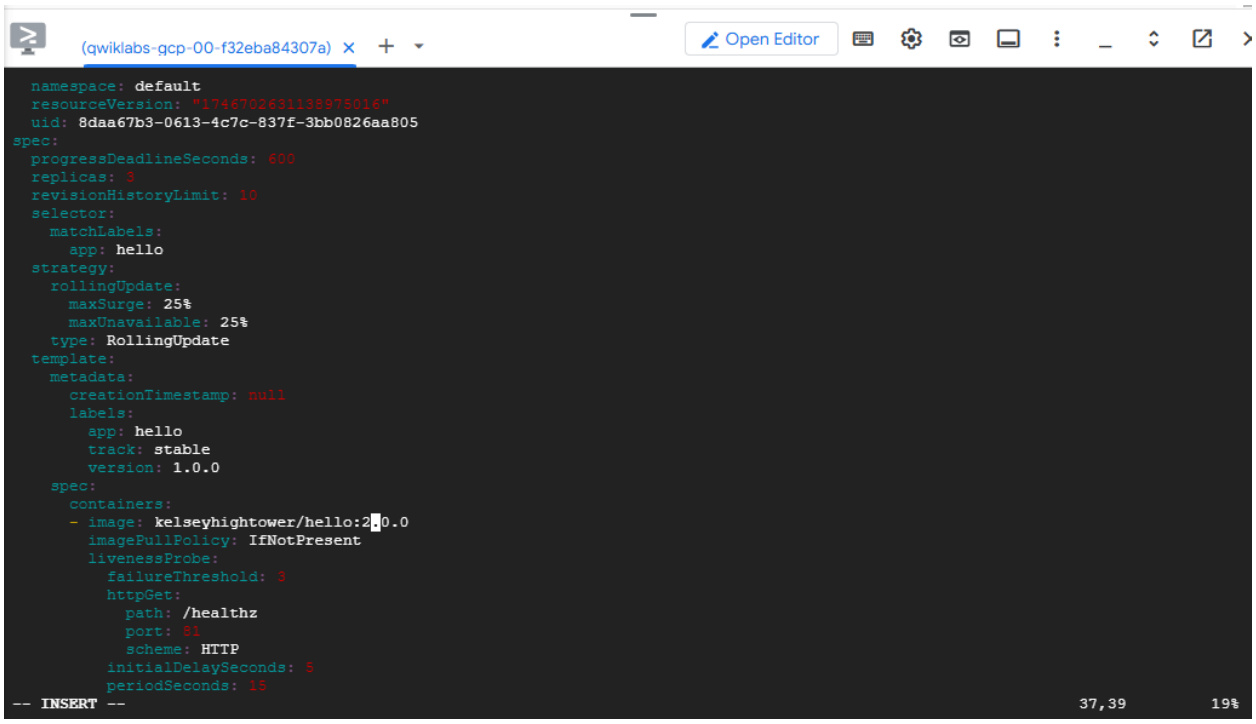
Deployments support updating images to a new version through a rolling update mechanism. When a deployment is updated with a new version, it creates a new ReplicaSet and slowly increases the number of replicas in the new ReplicaSet as it decreases the replicas in the old ReplicaSet.



Trigger a rolling update

1. To update your deployment, run the following command:

kubectl edit deployment hello



```
  namespace: default
  resourceVersion: "1746702631138975016"
  uid: 8daa67b3-0613-4c7c-837f-3bb0826aa805
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: hello
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: hello
        track: stable
        version: 1.0.0
    spec:
      containers:
      - image: kelseyhightower/hello:2.0.0
        imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 81
            scheme: HTTP
          initialDelaySeconds: 5
          periodSeconds: 15
-- INSERT --                                                                  37,39         19%
```

2. Change the image in the containers section of the deployment to the following:

...

containers:

image: kelseyhightower/hello:2.0.0

...

3. **Save** and **exit**.

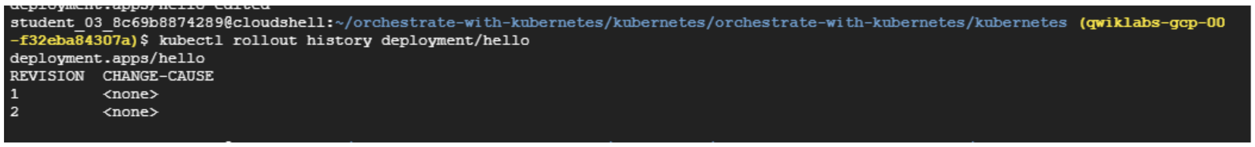
The updated deployment will be saved to your cluster and Kubernetes will begin a rolling update.

4. See the new ReplicaSet that Kubernetes creates.:

kubectl get replicaset

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

kubectl rollout history deployment/hello



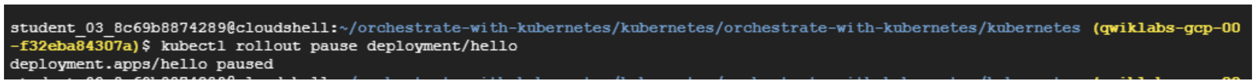
```
student_03_8c69b88742890cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout history deployment/hello
deployment.apps/hello
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

Pause a rolling update

If you detect problems with a running rollout, pause it to stop the update.

1. Run the following to pause the rollout:

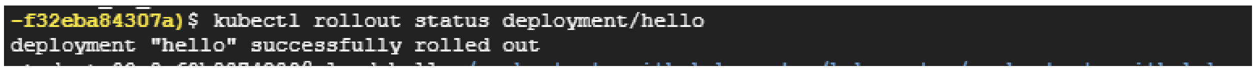
kubectl rollout pause deployment/hello



```
student_03_8c69b88742890cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout pause deployment/hello
deployment.apps/hello paused
```

2. Verify the current state of the rollout:

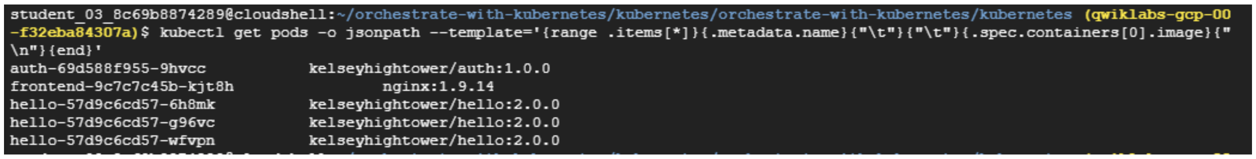
kubectl rollout status deployment/hello



```
-f32eba84307a)$ kubectl rollout status deployment/hello
deployment "hello" successfully rolled out
```

3. You can also verify this on the Pods directly:

kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'

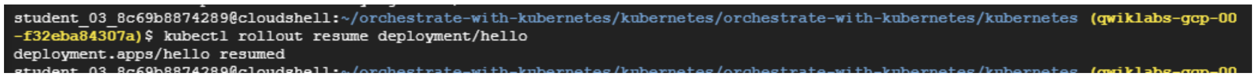

```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"
\n"}{end}'
auth-69d588f955-9hvcc          kelseyhightower/auth:1.0.0
frontend-9c7c7c45b-kjt8h               nginx:1.9.14
hello-57d9c6cd57-6h8mk         kelseyhightower/hello:2.0.0
hello-57d9c6cd57-g96vc         kelseyhightower/hello:2.0.0
hello-57d9c6cd57-wfvpn         kelseyhightower/hello:2.0.0
```

Resume a rolling update

The rollout is paused which means that some pods are at the new version and some pods are at the older version.

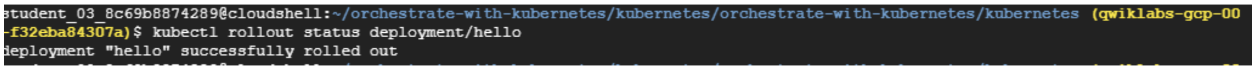
1. Continue the rollout using the resume command:

kubectl rollout resume deployment/hello


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout resume deployment/hello
deployment.apps/hello resumed
```

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

kubectl rollout status deployment/hello


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout status deployment/hello
deployment "hello" successfully rolled out
```

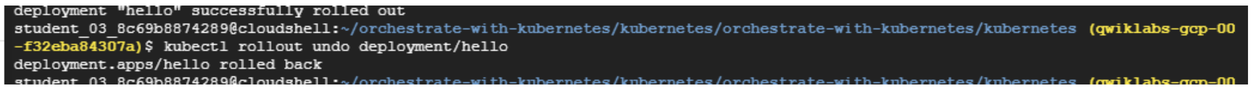
Roll back an update

Assume that a bug was detected in your new version. Since the new version is presumed to have problems, any users connected to the new Pods will experience those issues.

You will want to roll back to the previous version so you can investigate and then release a version that is fixed properly.

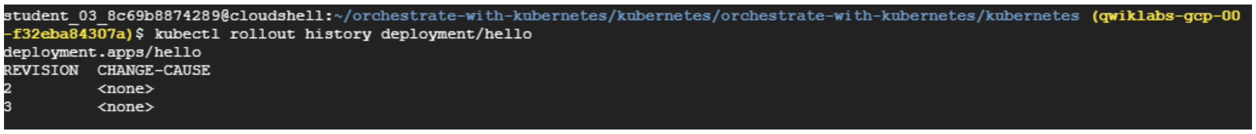
1. Use the rollout command to roll back to the previous version:

kubectl rollout undo deployment/hello


```
deployment "hello" successfully rolled out
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout undo deployment/hello
deployment.apps/hello rolled back
```

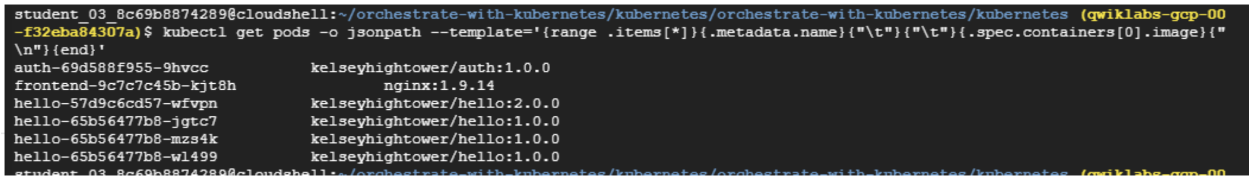
2. Verify the roll back in the history:

kubectl rollout history deployment/hello


```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl rollout history deployment/hello
deployment.apps/hello
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

3. Finally, verify that all the Pods have rolled back to their previous versions:

kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'



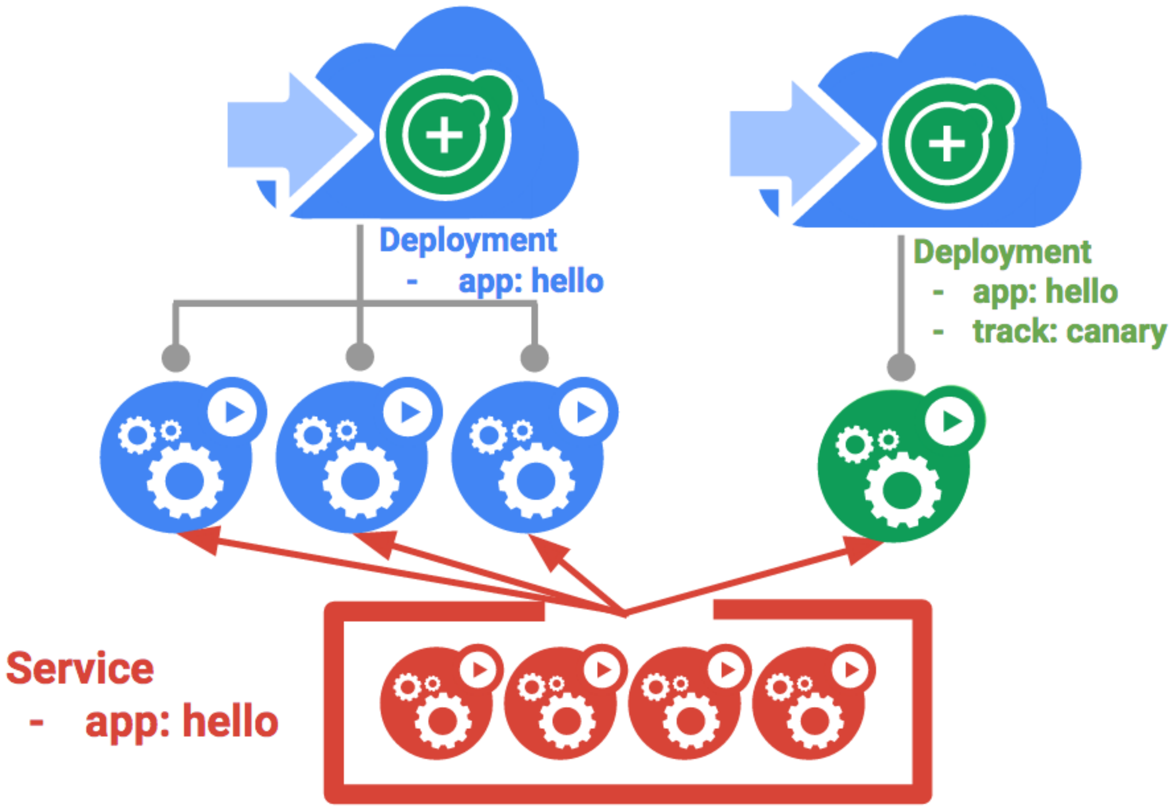
```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"
\n"}{end}'
auth-69d588f955-9hvcc           kelseyhightower/auth:1.0.0
frontend-9c7c7c45b-kjt8h                nginx:1.9.14
hello-57d9c6cd57-wfvpn          kelseyhightower/hello:2.0.0
hello-65b56477b8-jgtc7          kelseyhightower/hello:1.0.0
hello-65b56477b8-mzs4k          kelseyhightower/hello:1.0.0
hello-65b56477b8-w1499          kelseyhightower/hello:1.0.0
```

Task 4. Canary deployments

When you want to test a new deployment in production with a subset of your users, use a canary deployment. Canary deployments allow you to release a change to a small subset of your users to mitigate risk associated with new releases.

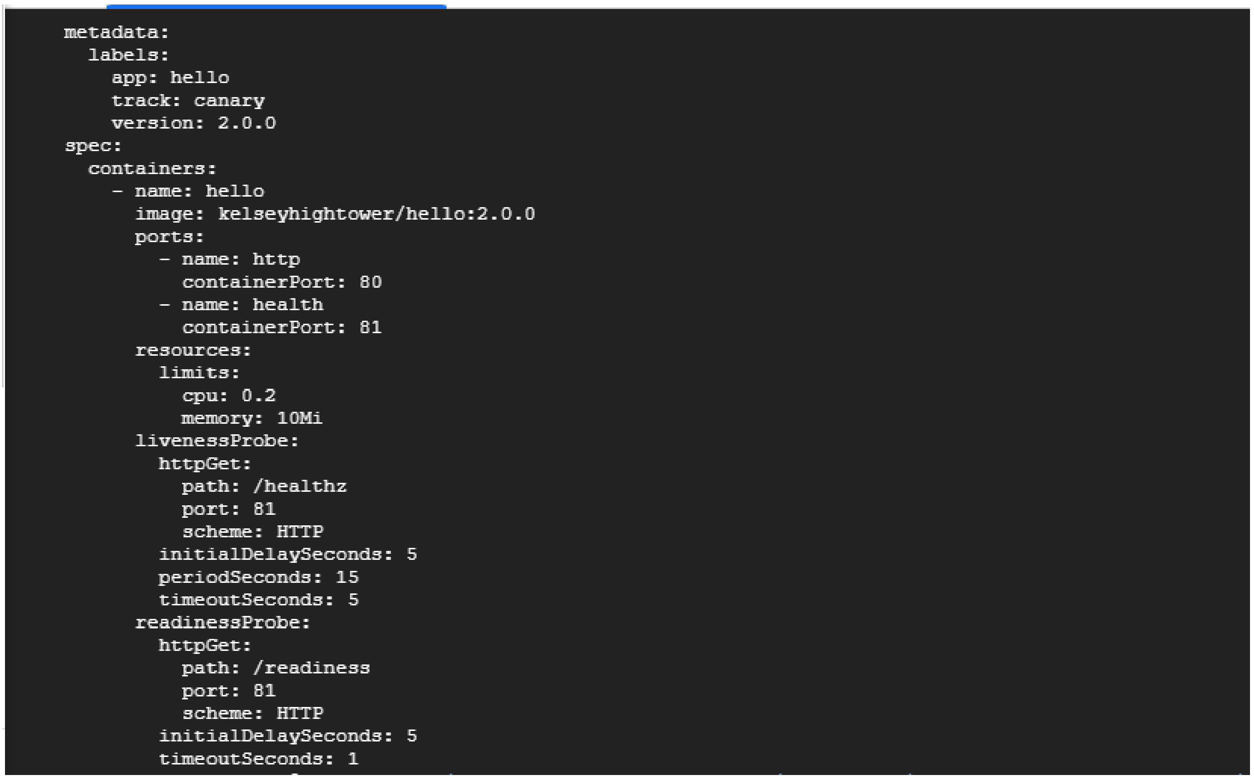
Create a canary deployment

A canary deployment consists of a separate deployment with your new version and a service that targets both your normal, stable deployment as well as your canary deployment.



1. First, create a new canary deployment for the new version:

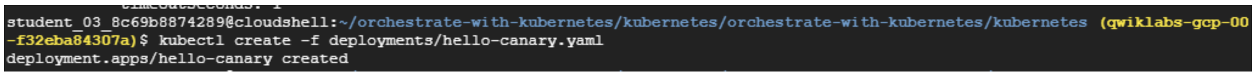
cat deployments/hello-canary.yaml



```
metadata:
  labels:
    app: hello
    track: canary
    version: 2.0.0
spec:
  containers:
    - name: hello
      image: kelseyhightower/hello:2.0.0
      ports:
        - name: http
          containerPort: 80
        - name: health
          containerPort: 81
      resources:
        limits:
          cpu: 0.2
          memory: 10Mi
      livenessProbe:
        httpGet:
          path: /healthz
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        periodSeconds: 15
        timeoutSeconds: 5
      readinessProbe:
        httpGet:
          path: /readiness
          port: 81
          scheme: HTTP
        initialDelaySeconds: 5
        timeoutSeconds: 1
```

2. Now create the canary deployment:

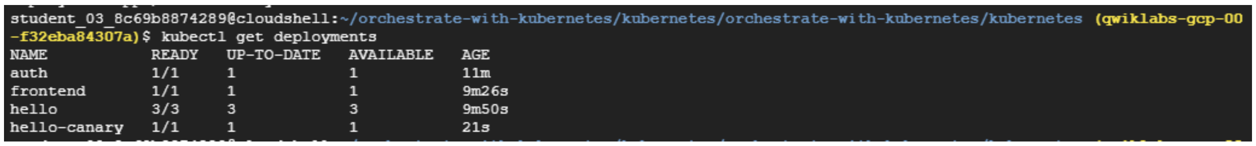
kubectl create -f deployments/hello-canary.yaml



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create -f deployments/hello-canary.yaml
deployment.apps/hello-canary created
```

3. After the canary deployment is created, you should have two deployments, hello and hello-canary. Verify it with this kubectl command:

kubectl get deployments



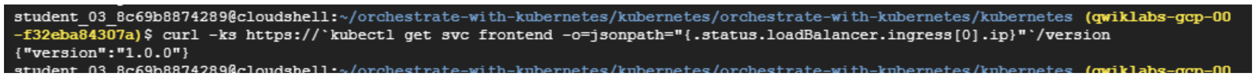
```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
auth           1/1     1            1           11m
frontend       1/1     1            1           9m26s
hello          3/3     3            3           9m50s
hello-canary   1/1     1            1           21s
```

On the hello service, the app:hello selector will match pods in **both** the prod deployment and canary deployment. However, because the canary deployment has a fewer number of pods, it will be visible to fewer users.

Verify the canary deployment

1. You can verify the hello version being served by the request:

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"` /version



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
```

2. Run this several times and you should see that some of the requests are served by hello 1.0.0 and a small subset (1/4 = 25%) are served by 2.0.0.

Canary deployments in production - session affinity

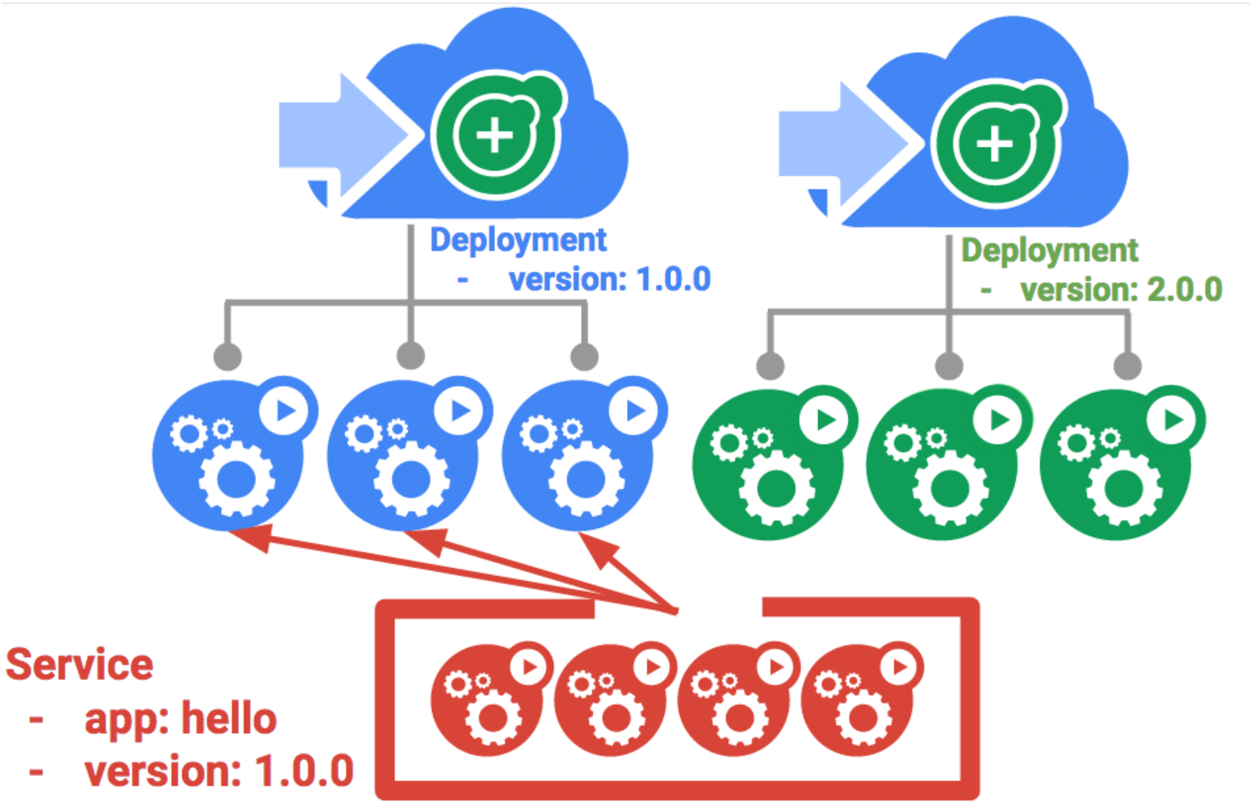
In this lab, each request sent to the Nginx service had a chance to be served by the canary deployment. But what if you wanted to ensure that a user didn't get served by the canary deployment? A use case could be that the UI for an application changed, and you don't want to confuse the user. In a case like this, you want the user to "stick" to one deployment or the other.

You can do this by creating a service with session affinity. This way the same user will always be served from the same version. In the example below, the service is the same as before, but a new sessionAffinity field has been added, and set to ClientIP. All clients with the same IP address will have their requests sent to the same version of the hello application.

Task 5. Blue-green deployments

Rolling updates are ideal because they allow you to deploy an application slowly with minimal overhead, minimal performance impact, and minimal downtime. There are instances where it is beneficial to modify the load balancers to point to that new version only after it has been fully deployed. In this case, blue-green deployments are the way to go.

Kubernetes achieves this by creating two separate deployments; one for the old "blue" version and one for the new "green" version. Use your existing hello deployment for the "blue" version. The deployments will be accessed via a service which will act as the router. Once the new "green" version is up and running, you'll switch over to using that version by updating the service.

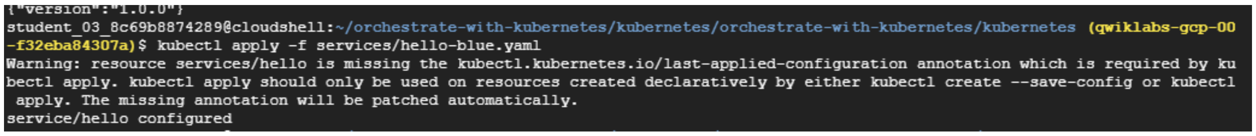


The service

Use the existing hello service, but update it so that it has a selector app:hello, version: 1.0.0. The selector will match the existing "blue" deployment. But it will not match the "green" deployment because it will use a different version.

- First update the service:

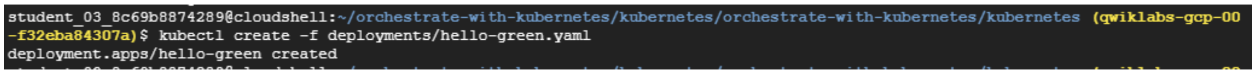
kubectl apply -f services/hello-blue.yaml



```
{"version":"1.0.0"}
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl apply -f services/hello-blue.yaml
Warning: resource services/hello is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by ku
bectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl
 apply. The missing annotation will be patched automatically.
service/hello configured
```

1. Create the green deployment:

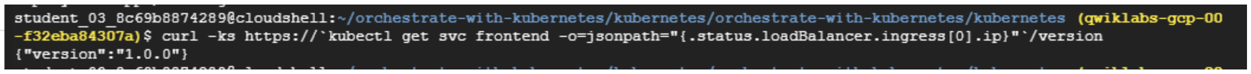
kubectl create -f deployments/hello-green.yaml



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl create -f deployments/hello-green.yaml
deployment.apps/hello-green created
```

2. Once you have a green deployment and it has started up properly, verify that the current version of 1.0.0 is still being used:

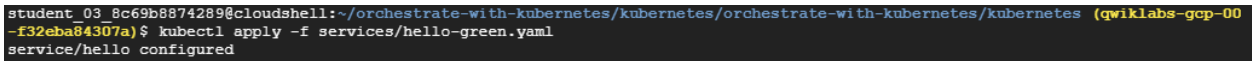
curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
```

3. Now, update the service to point to the new version:

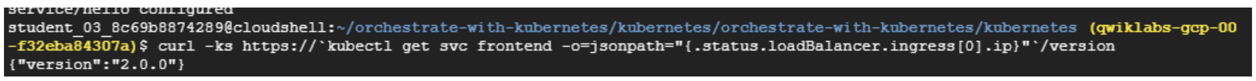
kubectl apply -f services/hello-green.yaml



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl apply -f services/hello-green.yaml
service/hello configured
```

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

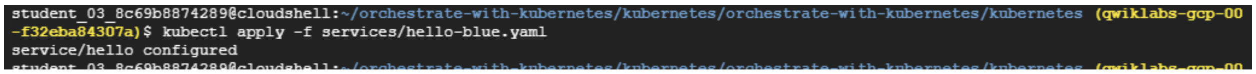
curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version



```
service/hello configured
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"2.0.0"}
```

1. While the "blue" deployment is still running, just update the service back to the old version:

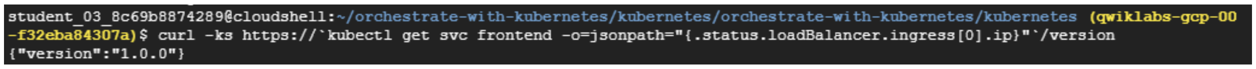
kubectl apply -f services/hello-blue.yaml



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ kubectl apply -f services/hello-blue.yaml
service/hello configured
```

2. Once you have updated the service, your rollback will have been successful. Again, verify that the right version is now being used:

curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version



```
student_03_8c69b8874289@cloudshell:~/orchestrate-with-kubernetes/kubernetes/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-00
-f32eba84307a)$ curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
```