Managing Deployments Using Kubernetes Engine

Configure Compute Zone

gcloud config set compute/zone <ZONE>

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-02-a91874aa85c8.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
student_01_b76a55fe2a8b@cloudshell:~ (qwiklabs-gcp-02-a91874aa85c8)$ gcloud config set compute/zone us-west1-c
Updated property [compute/zone].
student_01_b76a55fe2a8b@cloudshell:~ (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ gcloud container cluste
rs 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).
```

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

```
    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.
```

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

```
(qwiklabs-gcp-02-a91874aa85c8)  Open Editor
    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>
```

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

kubectl explain deployment --recursive

```
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


student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

Task 2. Create a deployment

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

vi deployments/auth.yaml

2. Start the editor:

i

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

```
...
containers:
- name: auth
  image: "kelseyhightower/auth:1.0.0"
```

```
      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:
                                                                    18,41          41%
```

4. 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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl create -f deployments/auth.yaml

```
            initialDelaySeconds: 5
            timeoutSeconds: 1
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl create -f deplo
yments/auth.yaml
deployment.apps/auth created
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl get deployments

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
auth   1/1     1            1           25s
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get replicasets
NAME              DESIRED   CURRENT   READY   AGE
auth-69d588f955   1         1         1       39s
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
auth-69d588f955-w7t7z   1/1     Running   0          57s
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl create -f servi
ces/auth.yaml
service/auth created
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl create -f deplo
yments/hello.yaml
kubectl create -f services/hello.yaml
deployment.apps/hello created
service/hello created
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl create secret g
eneric 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
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl get services frontend

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

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ curl -ks https://35.197
.125.51
{"message":"Hello"}
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ curl -ks https://`kubec
tl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`
{"message":"Hello"}
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl explain deploym
ent.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.


student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl scale deployment hello --replicas=5

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl scale deploymen
t hello --replicas=5
deployment.apps/hello scaled
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl get pods | grep hello- | wc -l

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get pods | grep
 hello- | wc -l
5
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

4. Now scale back the application:

kubectl scale deployment hello --replicas=3

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl scale deploymen
t hello --replicas=3
deployment.apps/hello scaled
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

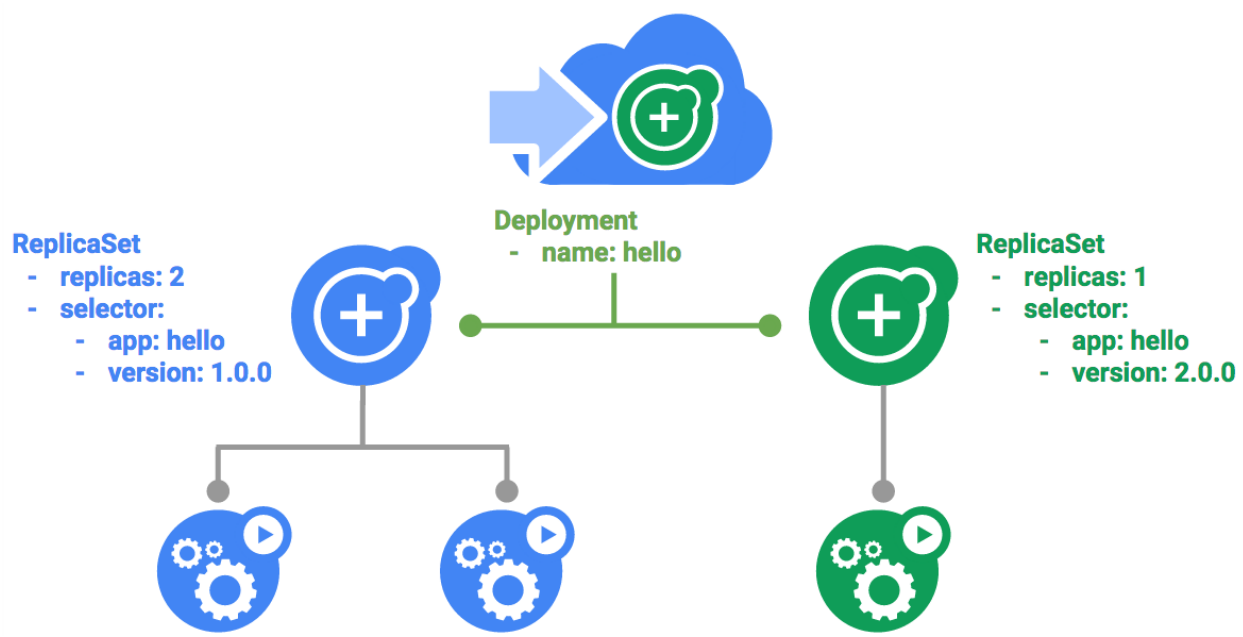
5. Again, verify that you have the correct number of Pods:

kubectl get pods | grep hello- | wc -l

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get pods | grep
 hello- | wc -l
3
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

```
(qwiklabs-gcp-02-a91874aa85c8)                                   Open Editor
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2025-05-30T04:38:17Z"
  generation: 3
  name: hello
  namespace: default
  resourceVersion: "17485801315332630 16"
  uid: cbb1cb8a-d1dc-476e-a420-bef527b5bb4e
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
"/tmp/kubectl-edit-1178417925.yaml" 96L, 2591B                                    1,1           Top
```

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

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
auth-69d588f955       1         1         1       7m11s
frontend-9c7c7c45b    1         1         1       5m22s
hello-57d9c6cd57      1         1         0       13s
hello-65b56477b8      3         3         3       5m39s
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl rollout history deployment/hello

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout history
 deployment/hello
deployment.apps/hello
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout pause d
eployment/hello
deployment.apps/hello paused
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

2. Verify the current state of the rollout:

kubectl rollout status deployment/hello

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout status
deployment/hello
deployment "hello" successfully rolled out
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get pods -o jso
npath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'
auth-69d588f955-w7t7z           kelseyhightower/auth:1.0.0
frontend-9c7c7c45b-d2tgp                nginx:1.9.14
hello-57d9c6cd57-8rtk6          kelseyhightower/hello:2.0.0
hello-57d9c6cd57-g9tsm          kelseyhightower/hello:2.0.0
hello-57d9c6cd57-pvv9j          kelseyhightower/hello:2.0.0
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout resume
deployment/hello
deployment.apps/hello resumed
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl rollout status deployment/hello

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout status
deployment/hello
deployment "hello" successfully rolled out
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout undo de
ployment/hello
deployment.apps/hello rolled back
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

2. Verify the roll back in the history:

kubectl rollout history deployment/hello

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl rollout history
 deployment/hello
deployment.apps/hello
REVISION  CHANGE-CAUSE
2         <none>
3         <none>

student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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}'

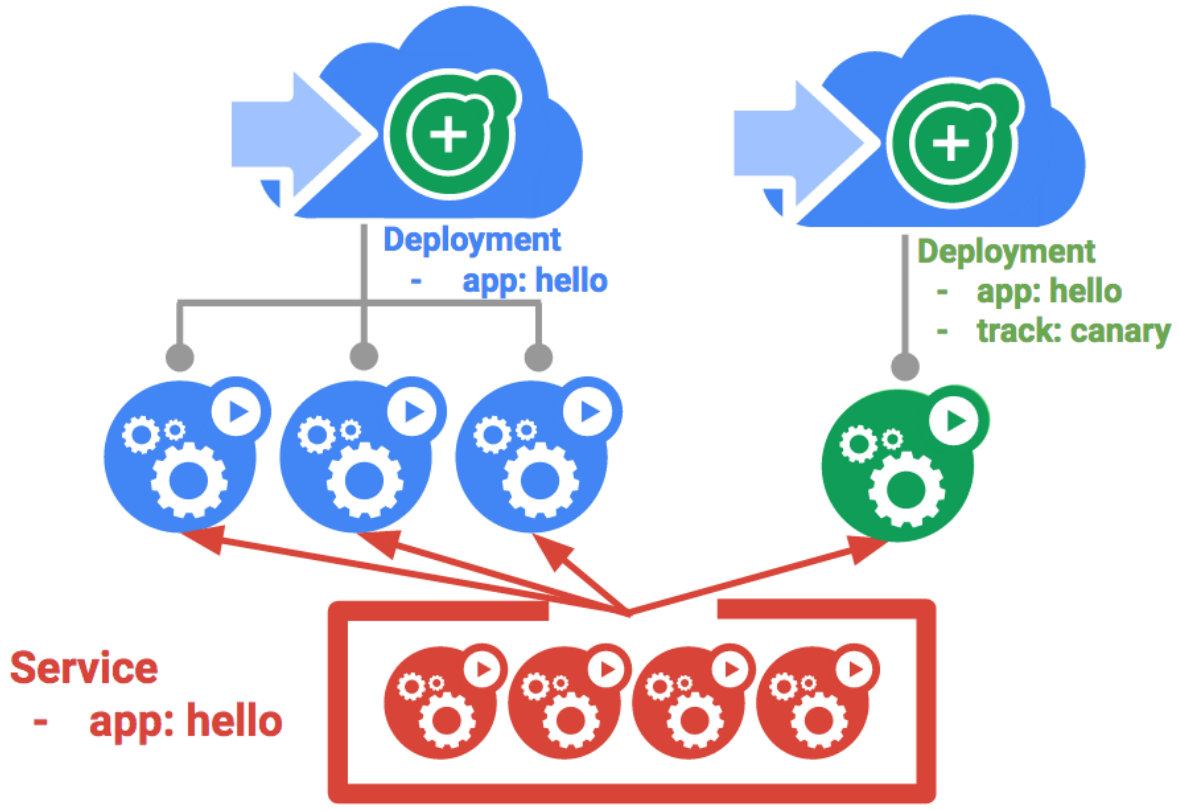
```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get pods -o jso
npath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'
auth-69d588f955-w7t7z           kelseyhightower/auth:1.0.0
frontend-9c7c7c45b-d2tgp                nginx:1.9.14
hello-57d9c6cd57-8rtk6          kelseyhightower/hello:2.0.0
hello-65b56477b8-55wfh          kelseyhightower/hello:1.0.0
hello-65b56477b8-7ljmb          kelseyhightower/hello:1.0.0
hello-65b56477b8-cmxm2          kelseyhightower/hello:1.0.0
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

2. Now create the canary deployment:

kubectl create -f deployments/hello-canary.yaml

```
            timeoutSeconds: 1
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl create -f deplo
yments/hello-canary.yaml
deployment.apps/hello-canary created
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kuber
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
auth          1/1     1            1           10m
frontend      1/1     1            1           9m2s
hello         3/3     3            3           9m19s
```

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

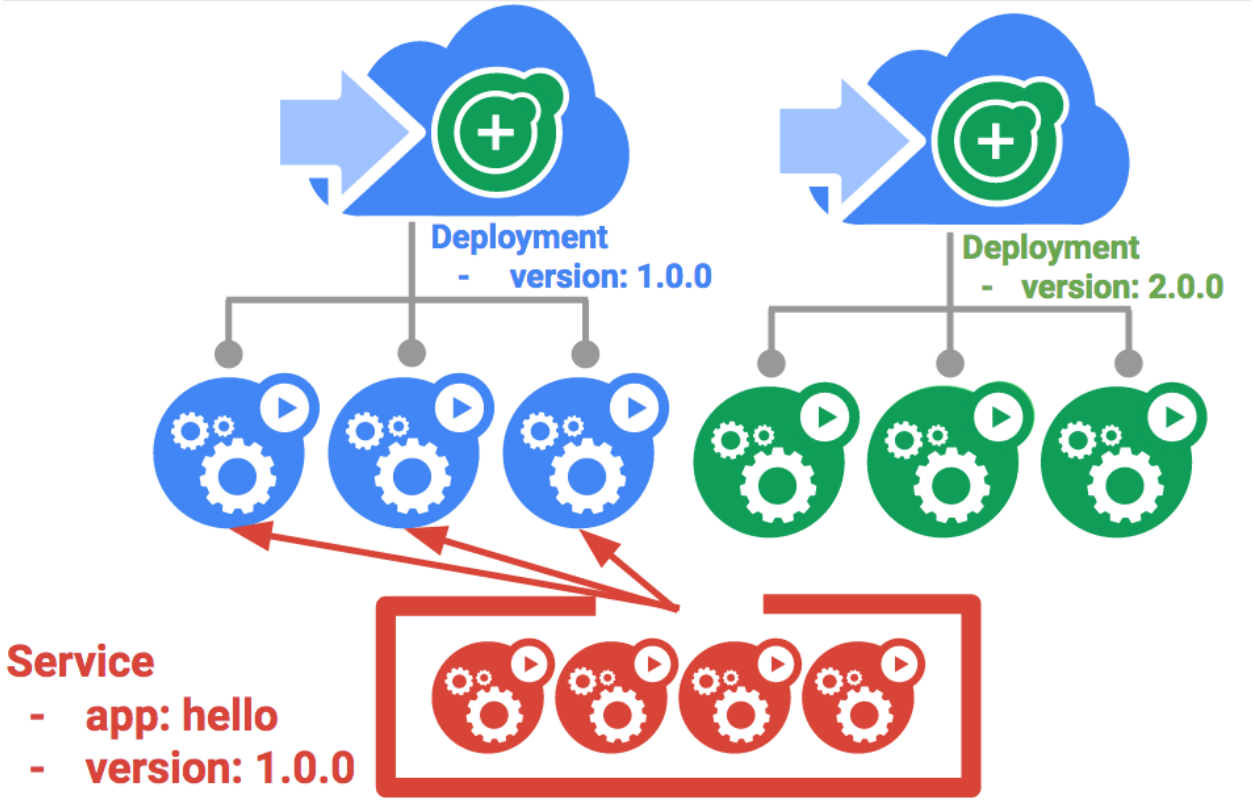
```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
auth           1/1     1            1           10m
frontend       1/1     1            1           9m2s
hello          3/3     3            3           9m19s
hello-canary   1/1     1            1           14s
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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.

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

1. Create the green deployment:

kubectl create -f deployments/hello-green.yaml

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ curl -ks https://`kubec
tl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

kubectl apply -f services/hello-green.yaml

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl apply -f servic
es/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
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

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

```
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ curl -ks https://`kubec
tl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

Blue-Green rollback

If necessary, you can roll back to the old version in the same way.

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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ kubectl apply -f servic
es/hello-blue.yaml
service/hello configured
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```

2. e 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_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$ curl -ks https://`kubec
tl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
{"version":"1.0.0"}
student_01_b76a55fe2a8b@cloudshell:~/orchestrate-with-kubernetes/kubernetes (qwiklabs-gcp-02-a91874aa85c8)$
```