# Managing Deployments Using Kubernetes Engine

## Preface:

I set the zone to us-central1-a and cloned the lab repo to **GCP**.   
Now I forgot to end the Kubernetes process on GCP, so it *ate* all of my money. I had to start a free trial.  
Therefore, my project is not under the same id and user.

### The Setup

Text

Description automatically generated

Command created cluster with **5** nodes. These nodes will be doing the work which is why they are called *worker* nodes. I think it was important to specify what region they are in because nodes **can be** virtual or physical machines. Thus, it is important for google to know where to serve from so they can benefit from virtualization.

## Deployment Creation

In this part I changed the version of the container to deploy. I did not put this image because it was a minor change. I then created the deployment object via kubectl create.

A replicaset is the process that runs many instances of a pod to keep the number of pods running to the number a user specified. We note that a replicaset was created as per this deployment. Also I did notice we said we wanted **one** replicas in out deployment.yaml and that is what happened:

We now have a pod running as per the deployment file. Graphical user interface, text

Description automatically generated with medium confidence

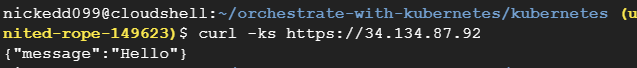
I see online that many people are *confused* about replicaset vs deployment. The point is that the *deployment* is what controls the replicaset. The replicaset is one of the many jobs of a deployment.

I then created services and deployments for the rest our applicaction.

Table

Description automatically generated with medium confidence

This is where the last lab ended.



Proof that the front-end is set up correctly.

## Scaling Pods and Containers

We recall that our deployment.spec.replicas is set to 1. We only have 1 copy of a pod running. If this pod is down our app will be down ☹.

After running the scaling command:

kubectl scale deployment hello --replicas=5

A screenshot of a computer

Description automatically generated with medium confidence

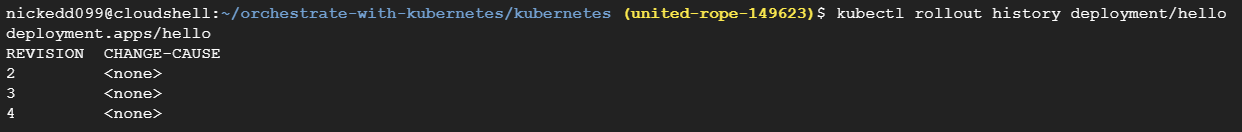
Our “Hello” pod that returns the message now has **5** copies.

This command can also be used to **downscale** the lab.

## Rolling Update

The point of this delivery method is to increase the number of replicas carrying the **new** version while decreasing the ones with the older containers.

For this method I updated the hello/deployment.yaml first, the container version will be set to 2.0.0 again.

****

I was playing around with it, there should only be two entries.

Since there are only a few replicas, the redeployment was quick: 

Then I did a **rollback**:

A picture containing text

Description automatically generated

This time around I watched it log out the updates. It does each pod one by one and **deletes** the old pods. Pods are meant to be *disposable*.

This appended another entry on our history for the *hello* service*.*

## Canary deployments

This will expose the new deployment to only a few users. (relative)

I first created a canary-hello deployment:

Text

Description automatically generated with low confidence

Now we notice we have two deployments: hello and hello-canary.

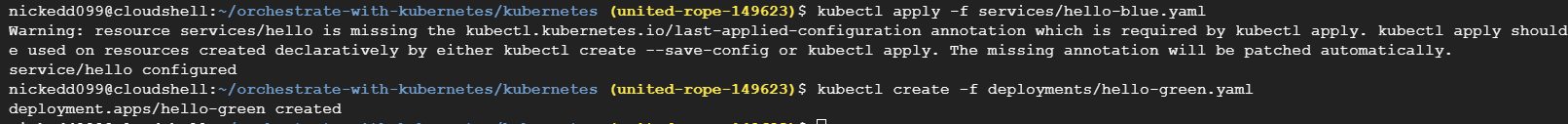
Text

Description automatically generated

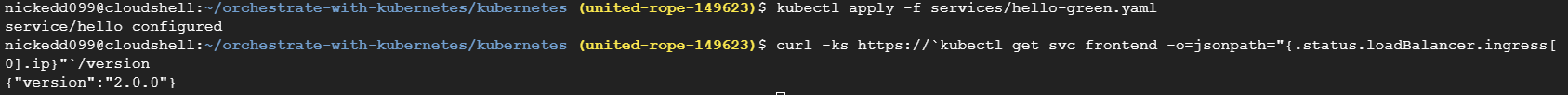
This screenshot shows the difference between hitting a canary pod vs a regular deployment.

## Blue-green deployments

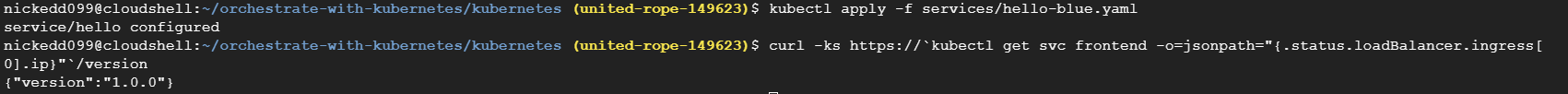
This allows is to load deployments and then make them active after testing and the mass of the new “green”-nodes hits a critical percentage. At first I did not realize the purpose of this. Why use Blue-green when we can just call rollout? The answer to this question is that bigger containers take longer to rollout; this can mean that if we want to rollout we will be slowly “flipping” pods one by one. This can cause major concurrency problems. To solve this, we can load in the new pods and then choose to release them pre the Blue-green ideology.

We apply the blue label to mark the service as current. Then create a green deployment which is the update that needs to be pushed out. 

Then we push the update by simply changing the service to serve v2 instead of v1.



The image shows the hello returning the v2 string.



Rollback back to blue. It is as simple as swapping the active service that is facing a pod.

To finish off the lab, I deleted the cluster.