**Creating Google Kubernetes Engine Deployments**

Toggle Lab Panel



**Creating Google Kubernetes Engine Deployments**

1 hour Free

Rate Lab

**Overview**

In this lab, you explore the basics of using deployment manifests. Manifests are files that contain configurations required for a deployment that can be used across different Pods. Manifests are easy to change.

**Objectives**

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

* Create deployment manifests, deploy to cluster, and verify Pod rescheduling as nodes are disabled
* Trigger manual scaling up and down of Pods in deployments
* Trigger deployment rollout (rolling update to new version) and rollbacks
* Perform a Canary deployment

**Task 0. Lab Setup**

**Access Qwiklabs**

For each lab, you get a new GCP project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, img/time.pngand make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

1. When ready, click img/start_lab.png.
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

After you complete the initial sign-in steps, the project dashboard appears.

**Activate Google Cloud Shell**

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



1. Click **Continue**. 

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



**gcloud** is the command-line tool for Google Cloud Platform. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

Example output:

Credentialed accounts:

- google1623327\_student@qwiklabs.net

You can list the project ID with this command:

gcloud config list project

Output:

[core]

project = <project\_ID>

Example output:

[core]

project = qwiklabs-gcp-44776a13dea667a6

Full documentation of **gcloud** is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud) .

Cluster is already created

**Task 1. Create deployment manifests and deploy to the cluster**

In this task, you create a deployment manifest for a Pod inside the cluster.

**Connect to the lab GKE cluster**

1. In Cloud Shell, type the following command to set the environment variable for the zone and cluster name.

export my\_zone=us-central1-a

export my\_cluster=standard-cluster-1

1. Configure kubectl tab completion in Cloud Shell.

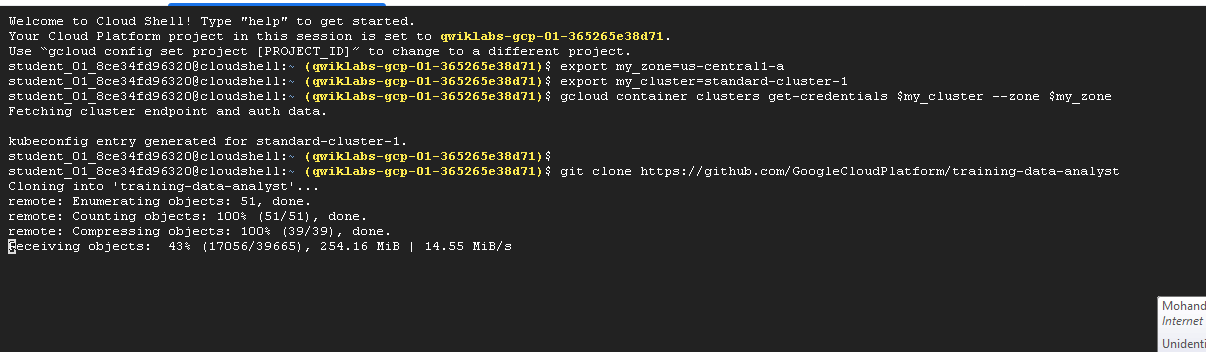
source <(kubectl completion bash)

1. In Cloud Shell, configure access to your cluster for the kubectl command-line tool, using the following command:

gcloud container clusters get-credentials $my\_cluster --zone $my\_zone

1. In Cloud Shell enter the following command to clone the repository to the lab Cloud Shell.

git clone <https://github.com/GoogleCloudPlatform/training-data-analyst>



1. Create a soft link as a shortcut to the working directory.

ln -s ~/training-data-analyst/courses/ak8s/v1.1 ~/ak8s

1. Change to the directory that contains the sample files for this lab.

cd ~/ak8s/Deployments/

**Create a deployment manifest**

You will create a deployment using a sample deployment manifest called nginx-deployment.yaml that has been provided for you. This deployment is configured to run three Pod replicas with a single nginx container in each Pod listening on TCP port 80.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

1. To deploy your manifest, execute the following command:

kubectl apply -f ./nginx-deployment.yaml

Click *Check my progress* to verify the objective. Create and deploy manifest nginx deployment

1. To view a list of deployments, execute the following command:

kubectl get deployments

The output should look like this example.

**Output (do not copy)**

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 0/3 3 0 3s

1. Wait a few seconds, and repeat the command until the number listed for CURRENT deployments reported by the command matches the number of DESIRED deployments.

The final output should look like the example.

**Output (do not copy)**

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 3/3 3 3 42s

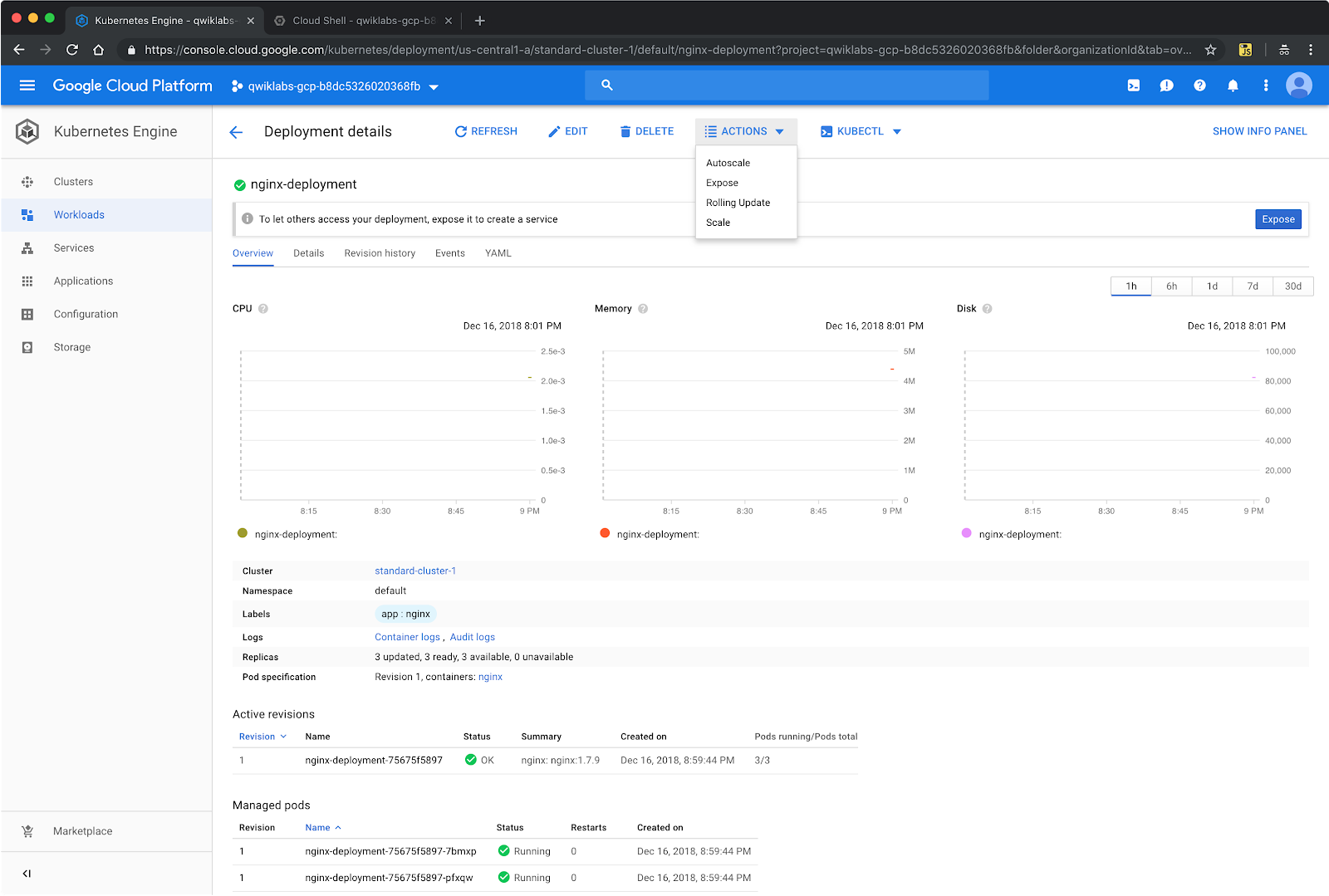
**Task 2. Manually scale up and down the number of Pods in deployments**

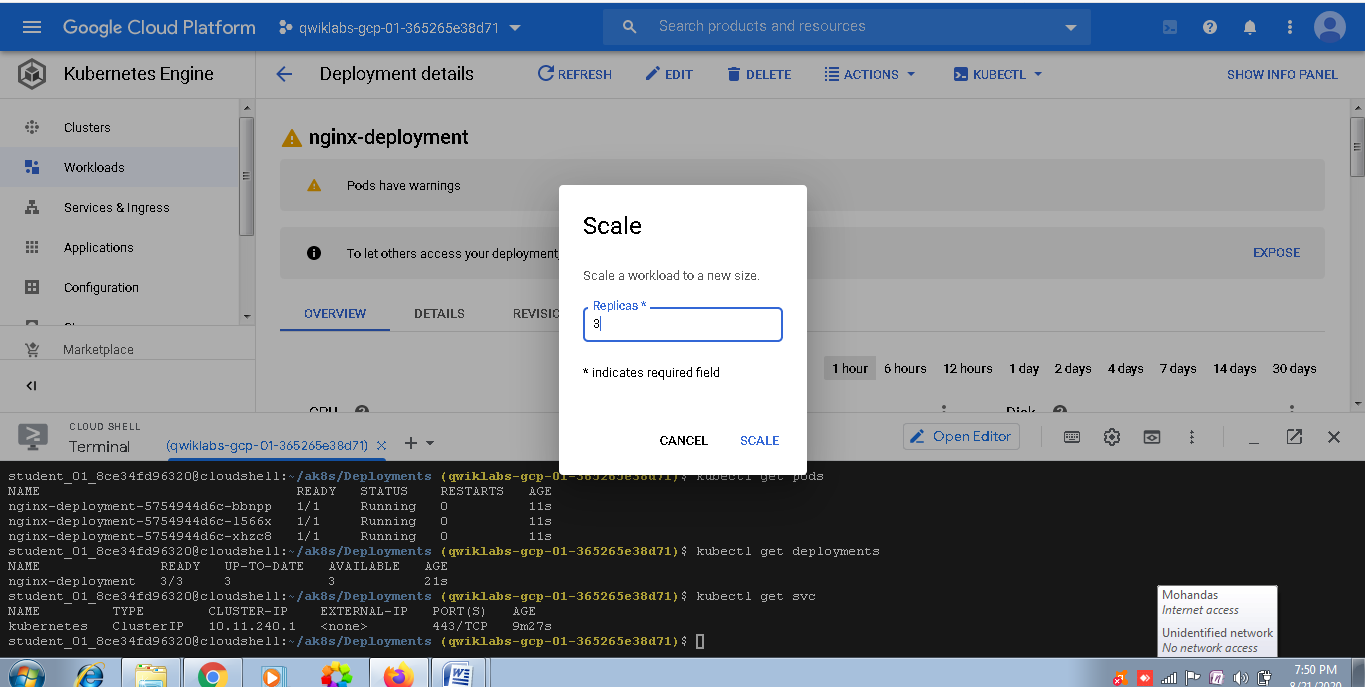
Sometimes, you want to shut down a Pod instance. Other times, you want ten Pods running. In Kubernetes, you can scale a specific Pod to the desired number of instances. To shut them down, you scale to zero.

In this task, you scale Pods up and down in the GCP Console and Cloud Shell.

**Scale Pods up and down in the console**

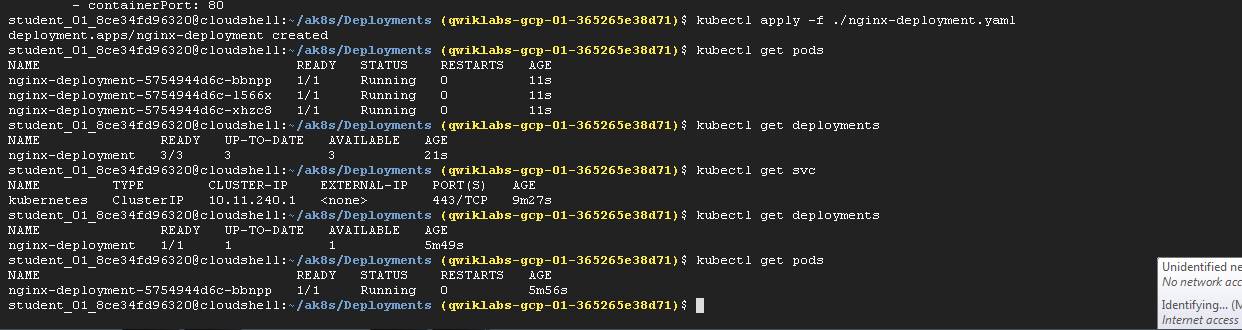
1. Switch to the GCP Console tab.
2. On the **Navigation menu** ( 9a951fa6d60a98a5.png), click **Kubernetes Engine** > **Workloads**.
3. Click **nginx-deployment** (your deployment) to open the Deployment details page.
4. At the top, click **ACTIONS > Scale**.



1. Type **1** and click **SCALE**.
2. 

This action scales down your cluster. You should see the Pod status being updated under **Managed Pods**. You might have to click **Refresh**.

**Scale Pods up and down in the shell**

****

1. Switch back to the Cloud Shell browser tab.
2. In the Cloud Shell, to view a list of Pods in the deployments, execute the following command:

kubectl get deployments

**Output (do not copy)**

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 1/1 1 1 3m

1. To scale the Pod back up to three replicas, execute the following command:

kubectl scale --replicas=3 deployment nginx-deployment

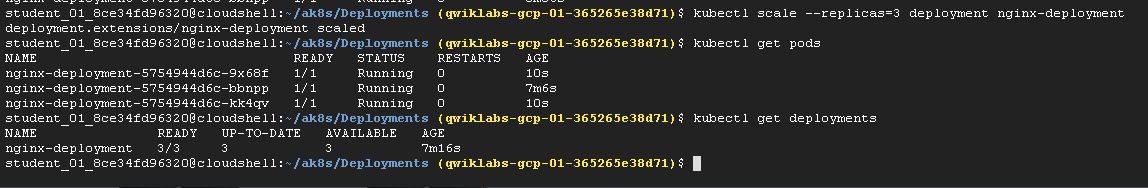
1. To view a list of Pods in the deployments, execute the following command:

kubectl get deployments

**Output (do not copy)**

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 3/3 3 3 4m



**Task 3. Trigger a deployment rollout and a deployment rollback**

A deployment's rollout is triggered if and only if the deployment's Pod template (that is, .spec.template) is changed, for example, if the labels or container images of the template are updated. Other updates, such as scaling the deployment, do not trigger a rollout.

In this task, you trigger deployment rollout, and then you trigger deployment rollback.

**Trigger a deployment rollout**

1. To update the version of nginx in the deployment, execute the following command:

kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record

This updates the container image in your Deployment to nginx v1.9.1.

Click *Check my progress* to verify the objective. Update version of nginx in the deployment

1. To view the rollout status, execute the following command:

kubectl rollout status deployment.v1.apps/nginx-deployment

The output should look like the example.

**Output (do not copy)**

Waiting for rollout to finish: 1 out of 3 new replicas updated...

Waiting for rollout to finish: 1 out of 3 new replicas updated...

Waiting for rollout to finish: 1 out of 3 new replicas updated...

Waiting for rollout to finish: 2 out of 3 new replicas updated...

Waiting for rollout to finish: 2 out of 3 new replicas updated...

Waiting for rollout to finish: 2 out of 3 new replicas updated...

Waiting for rollout to finish: 1 old replicas pending termination...

Waiting for rollout to finish: 1 old replicas pending termination...

deployment "nginx-deployment" successfully rolled out

1. To verify the change, get the list of deployments.

kubectl get deployments

The output should look like the example.

**Output (do not copy)**

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 3/3 3 3 6m

1. View the rollout history of the deployment.

kubectl rollout history deployment nginx-deployment

The output should look like the example. Your output might not be an exact match.

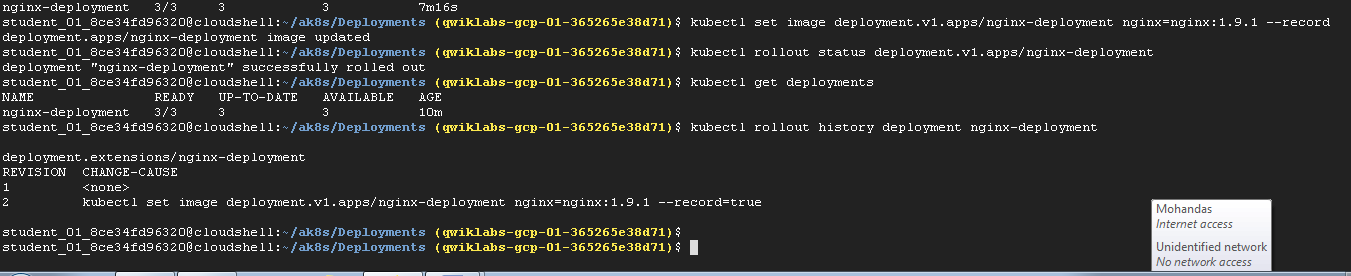
**Output (do not copy)**

deployments "nginx-deployment"

REVISION CHANGE-CAUSE

1 <none>

2 kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record=true



**Trigger a deployment rollback**

To roll back an object's rollout, you can use the kubectl rollout undo command.

1. To roll back to the previous version of the nginx deployment, execute the following command:

kubectl rollout undo deployments nginx-deployment

1. View the updated rollout history of the deployment.

kubectl rollout history deployment nginx-deployment

The output should look like the example. Your output might not be an exact match.

**Output (do not copy)**

deployments "nginx-deployment"

REVISION CHANGE-CAUSE

2 kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record=true

3 <none>

1. View the details of the latest deployment revision

kubectl rollout history deployment/nginx-deployment --revision=3

The output should look like the example. Your output might not be an exact match but it will show that the current revision has rolled back to nginx:1.7.9.

**Output (do not copy)**

deployments "nginx-deployment" with revision #3

Pod Template:

Labels: app=nginx

pod-template-hash=3123191453

Containers:

nginx:

Image: nginx:1.7.9

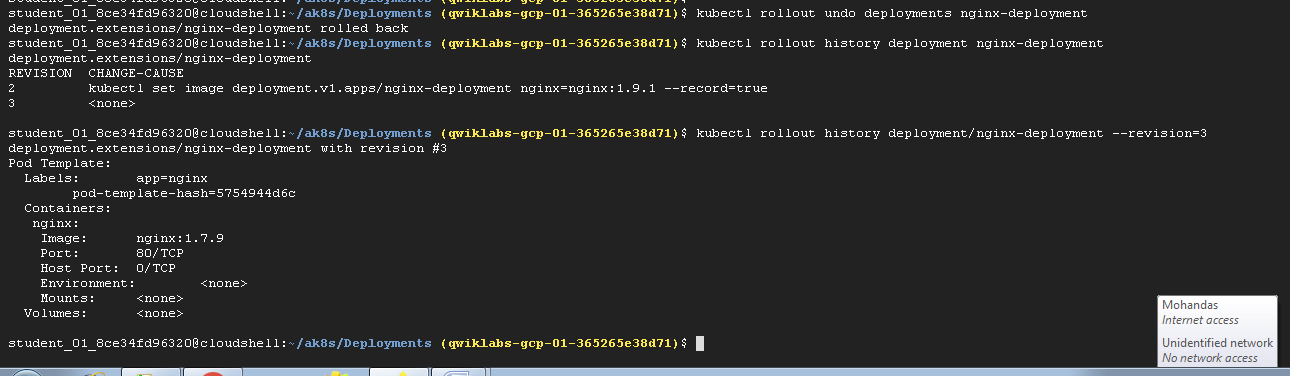
Port: 80/TCP

Host Port: 0/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>



**Task 4. Define the service type in the manifest**

In this task, you create and verify a service that controls inbound traffic to an application. Services can be configured as ClusterIP, NodePort or LoadBalancer types. In this lab you configure a LoadBalancer.

**Define service types in the manifest**

A manifest file called service-nginx.yaml that deploys a LoadBalancer service type has been provided for you. This service is configured to distribute inbound traffic on TCP port 60000 to port 80 on any containers that have the label app: nginx.

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

type: LoadBalancer

selector:

app: nginx

ports:

- protocol: TCP

port: 60000

targetPort: 80

1. In the Cloud Shell, to deploy your manifest, execute the following command:

kubectl apply -f ./service-nginx.yaml

This manifest defines a service and applies it to Pods that correspond to the selector. In this case, the manifest is applied to the nginx container that you deployed in task 1. This service also applies to any other Pods with the app: nginx label, including any that are created after the service.

Click *Check my progress* to verify the objective. Deploy manifest file that deploys LoadBalancer service type

**Verify the LoadBalancer creation**

1. To view the details of the nginx service, execute the following command:

kubectl get service nginx

The output should look like the example.

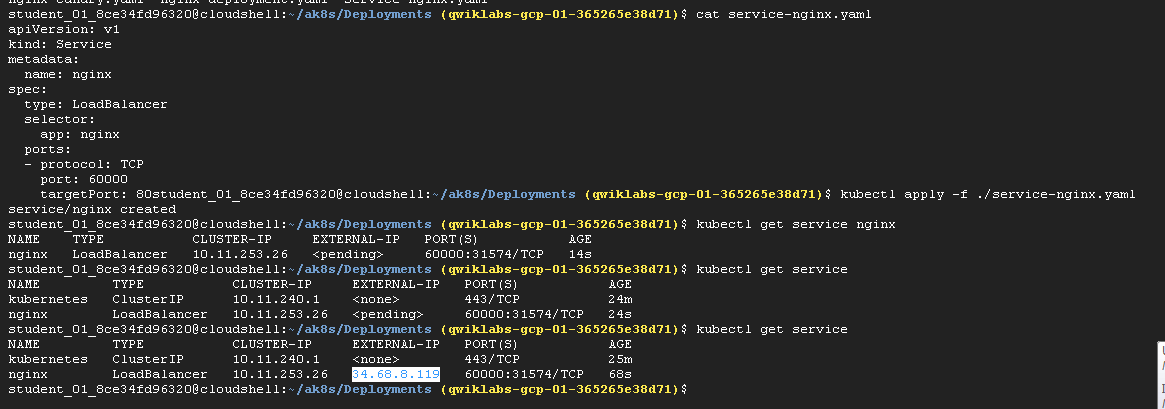
**Output (do not copy)**

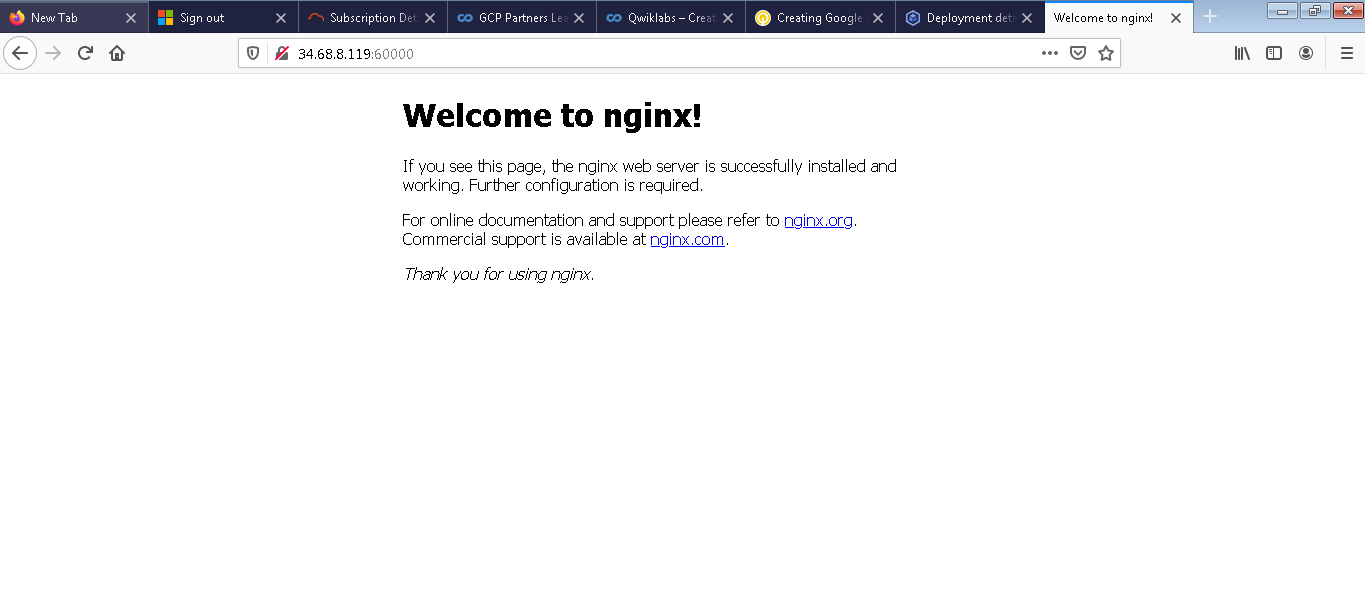
NAME CLUSTER\_IP EXTERNAL\_IP PORT(S) SELECTOR AGE

nginx 10.X.X.X X.X.X.X 60000/TCP run=nginx 1m

1. When the external IP appears, open http://[EXTERNAL\_IP]:60000/ in a new browser tab to see the server being served through network load balancing.

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 nginx command every few seconds until the field is populated.





**Task 5. Perform a canary deployment**

A canary deployment is a separate deployment used to test a new version of your application. A single service targets both the canary and the normal deployments. And it can direct a subset of users to the canary version to mitigate the risk of new releases. The manifest file nginx-canary.yaml that is provided for you deploys a single pod running a newer version of nginx than your main deployment. In this task, you create a canary deployment using this new deployment file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-canary

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

track: canary

Version: 1.9.1

spec:

containers:

- name: nginx

image: nginx:1.9.1

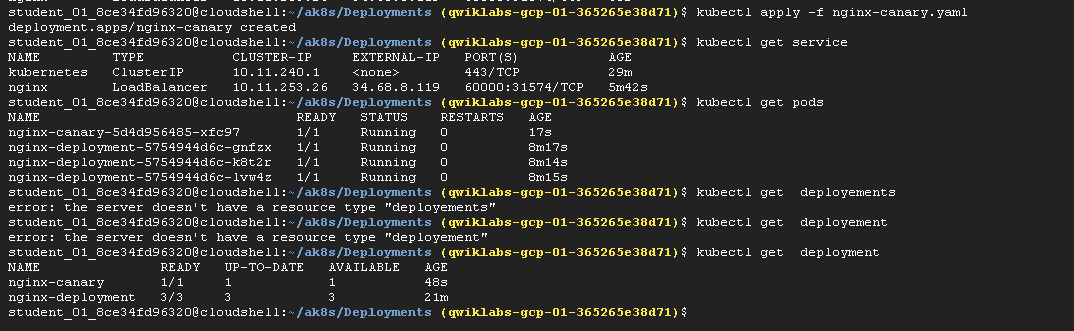
ports:

- containerPort: 80

The manifest for the nginx Service you deployed in the previous task uses a label selector to target the Pods with the app: nginx label. Both the normal deployment and this new canary deployment have the app: nginx label. Inbound connections will be distributed by the service to both the normal and canary deployment Pods. The canary deployment has fewer replicas (Pods) than the normal deployment, and thus it is available to fewer users than the normal deployment.

1. Create the canary deployment based on the configuration file.

kubectl apply -f nginx-canary.yaml



Click *Check my progress* to verify the objective. Create a Canary Deployment

1. When the deployment is complete, verify that both the nginx and the nginx-canary deployments are present.

kubectl get deployments

1. Switch back to the browser tab that is connected to the external LoadBalancer service ip and refresh the page. You should continue to see the standard "Welcome to nginx" page.
2. Switch back to the Cloud Shell and scale down the primary deployment to 0 replicas.

kubectl scale --replicas=0 deployment nginx-deployment

1. Verify that the only running replica is now the Canary deployment:

kubectl get deployments

1. Switch back to the browser tab that is connected to the external LoadBalancer service ip and refresh the page. You should continue to see the standard "Welcome to nginx" page showing that the Service is automatically balancing traffic to the canary deployment.

**Note: Session affinity**

The Service configuration used in the lab does not ensure that all requests from a single client will always connect to the same Pod. Each request is treated separately and can connect to either the normal nginx deployment or to the nginx-canary deployment. This potential to switch between different versions may cause problems if there are significant changes in functionality in the canary release. To prevent this you can set the sessionAffinity field to ClientIP in the specification of the service if you need a client's first request to determine which Pod will be used for all subsequent connections.

For example:

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

type: LoadBalancer

sessionAffinity: ClientIP

selector:

app: nginx

ports:

- protocol: TCP

port: 60000

targetPort: 80

**End your lab**