Creating Google Kubernetes Engine Deployments

# 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. This allows a container to be deployed multiple times simultaneously giving scalability and resilience. It also allows for the controlled updating of the image from which the container is created. Manifests are easy to create and 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 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

Start your Linux virtual machine and connect to it using SSH

Install and configure kubectl, needed to communicate directly to the Kubernetes cluster by running the following 3 commands..

sudo apt-get install kubectl

sudo apt-get install google-cloud-sdk-gke-gcloud-auth-plugin

export USE\_GKE\_GCLOUD\_AUTH\_PLUGIN=True

Configure kubectl tab completion (pressing tab key to auto-complete entries):

source <(kubectl completion bash)

Configure access to your cluster for the kubectl command-line tool, using the following command, inserting your cluster name and region where appropriate:

gcloud container clusters get-credentials {your cluster name} --region {the region you created your cluster in}

## Create a deployment manifest

You will create a deployment using a sample deployment manifest called nginx-deployment.yaml. This deployment is configured to run three Pod replicas, each with a single container running the nginx web service and listening on TCP port 80.

Create and open a file called {yourname}-nginx.yaml with nano using the following command:

nano {yourname}-nginx.yaml

Once nano has opened, paste the following into the yaml file:

apiVersion: apps/v1

kind: Deployment

metadata:

name: {yourname}-nginx

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

Press Ctrl+O, and then press Enter to save your edited file.

Press Ctrl+X to exit the nano text editor.

To deploy your manifest, execute the following command:

kubectl apply -f ./{yourname}-nginx.yaml

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

kubectl get deployments

The output should look like this example.

Output:

Wait a few minutes 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.

A screen shot of a computer

Description automatically generated

# 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 may want ten more 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 will scale Pods up and down in the Google Cloud Console and Cloud SDK.

## Scale Pods up and down in the console

Switch to the Google Cloud Console tab on your local laptop.

Navigate to Kubernetes Engine > Workloads.

Click {your-name}-nginx to open the Deployment details page.

At the top, click ACTIONS > Scale > Edit Replicas….

A screenshot of a computer

Description automatically generated

Type 1 and click SCALE.

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

Switch back to your Linux VM

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

kubectl get deployments

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

kubectl scale --replicas=3 deployment {yourname}-nginx

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

kubectl get deployments

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

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

## Trigger a deployment rollout

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

kubectl set image deployment.v1.apps/{yourname}-nginx nginx=nginx:1.9.1

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

To annotate the rollout with details on the change, execute the following command:

kubectl annotate deployment {yourname}-nginx kubernetes.io/change-cause="version change to 1.9.1" --overwrite=true

To view the rollout status, execute the following command:

kubectl rollout status deployment.v1.apps/{yourname}-nginx

Trigger a deployment rollback

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

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

kubectl rollout undo deployments {yourname}-nginx

View the updated rollout history of the deployment:

kubectl rollout history deployment {yourname}-nginx

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

Output:

A black screen with white text

Description automatically generated

Note: The most recent update is blank for the CHANGE-CAUSE as we did not use the kubectl annotate command .

View the details of the latest deployment revision:

kubectl rollout history deployment/{yourname}-nginx --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:

: A screen shot of a computer

Description automatically generated

Volumes:

# Task 4. Define the service type in the manifest

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

## Define service types in the manifest

We must create a manifest file called {yourname}-nginx-service.yaml that deploys a LoadBalancer service. This service is configured to distribute inbound traffic received on TCP port 60000 to port 80 on any containers in your cluster that have the label app: nginx.

Create and open a file called {yourname}-nginx-service.yaml with nano using the following command:

nano {yourname}-ngnix-service.yaml

Once nano has opened, paste the following into the {yourname}-ngnix-service.yaml:

apiVersion: v1

kind: Service

metadata:

name: {yourname}-nginx-service

spec:

type: LoadBalancer

selector:

app: nginx

ports:

- protocol: TCP

port: 60000

targetPort: 80

Press Ctrl+O, and then press Enter to save your edited file.

Press Ctrl+X to exit the nano text editor.

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

kubectl apply -f ./{yourname}-ngnix-service.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.

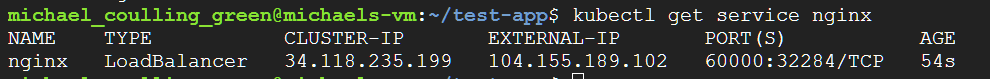
## Verify the LoadBalancer creation

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

kubectl get service nginx

The output should look similar to the example.

Output:



Note: It may take a few seconds before the External IP 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.

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.

# 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 normal deployments. And it can direct a subset of users to the canary version to mitigate the risk of new releases.

In this task, you create a canary deployment to deploys a single pod running a newer version of nginx than your main deployment.

Create and open a file called {yourname}-nginx-canary.yaml with nano using the following command:

nano {yourname}-nginx-canary.yaml

Once nano has opened, paste the following into the {yourname}-nginx-canary.yaml file:

apiVersion: apps/v1

kind: Deployment

metadata:

name: {yourname}-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

Press Ctrl+O, and then press Enter to save your edited file.

Press Ctrl+X to exit the nano text editor.

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 same app: nginx label. Inbound connections will therefore 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.

Create the canary deployment based on the configuration file:

kubectl apply -f ./ {yourname}-nginx-canary.yaml

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

kubectl get deployments

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.

Switch back to the SDK and scale down the primary deployment to 0 replicas:

kubectl scale --replicas=0 deployment {yourname}-nginx

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

kubectl get deployments

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 page showing that the Service is automatically balancing traffic to the canary deployment.

End of lab