GCP Fundamentals: Getting Started with Kubernetes Engine

**Overview**

In this lab, you create a Kubernetes Engine cluster containing several containers, each containing a web server. You place a load balancer in front of the cluster and view its contents.

**Objectives**

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

* Provision a [Kubernetes](http://kubernetes.io/) cluster using [Kubernetes Engine.](https://cloud.google.com/container-engine)
* Deploy and manage Docker containers using kubectl.

**Task 1: Sign in to the Google Cloud Platform (GCP) Console**

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.png and 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.

**Task 2: Confirm that needed APIs are enabled**

1. Make a note of the name of your GCP project. This value is shown in the top bar of the Google Cloud Platform Console. It will be of the form qwiklabs-gcp- followed by hexadecimal numbers.
2. In the GCP Console, on the **Navigation menu** (), click **APIs & Services**.
3. Scroll down in the list of enabled APIs, and confirm that both of these APIs are enabled:

* Kubernetes Engine API
* Container Registry API

If either API is missing, click **Enable APIs and Services** at the top. Search for the above APIs by name and enable each for your current project. (You noted the name of your GCP project above.)

**Task 3: Start a Kubernetes Engine cluster**

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



1. Click **Continue**. 
2. For convenience, place the zone that Qwiklabs assigned you to into an environment variable called MY\_ZONE. At the Cloud Shell prompt, type this partial command:
3. export MY\_ZONE=

followed by the zone that Qwiklabs assigned to you. Your complete command will look like this:

export MY\_ZONE=us-central1-a

1. Start a Kubernetes cluster managed by Kubernetes Engine. Name the cluster **webfrontend** and configure it to run 2 nodes:
2. gcloud container clusters create webfrontend --zone $MY\_ZONE --num-nodes 2

It takes several minutes to create a cluster as Kubernetes Engine provisions virtual machines for you.

1. After the cluster is created, check your installed version of Kubernetes using the kubectl version command:
2. kubectl version

The gcloud container clusters create command automatically authenticated kubectl for you.

1. View your running nodes in the GCP Console. On the **Navigation menu** (), click **Compute Engine > VM Instances**.

Your Kubernetes cluster is now ready for use.

**Task 4: Run and deploy a container**

1. From your Cloud Shell prompt, launch a single instance of the nginx container. (Nginx is a popular web server.)
2. kubectl create deploy nginx --image=nginx:1.17.10

In Kubernetes, all containers run in pods. This use of the kubectl create command caused Kubernetes to create a deployment consisting of a single pod containing the nginx container. A Kubernetes deployment keeps a given number of pods up and running even in the event of failures among the nodes on which they run. In this command, you launched the default number of pods, which is 1.

**Note**: If you see any deprecation warning about future version you can simply ignore it for now and can proceed further.

1. View the pod running the nginx container:
2. kubectl get pods
3. Expose the nginx container to the Internet:
4. kubectl expose deployment nginx --port 80 --type LoadBalancer

Kubernetes created a service and an external load balancer with a public IP address attached to it. The IP address remains the same for the life of the service. Any network traffic to that public IP address is routed to pods behind the service: in this case, the nginx pod.

1. View the new service:
2. kubectl get services

You can use the displayed external IP address to test and contact the nginx container remotely.

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 command every few seconds until the field is populated.

1. Open a new web browser tab and paste your cluster's external IP address into the address bar. The default home page of the Nginx browser is displayed.
2. Scale up the number of pods running on your service:
3. kubectl scale deployment nginx --replicas 3

Scaling up a deployment is useful when you want to increase available resources for an application that is becoming more popular.

1. Confirm that Kubernetes has updated the number of pods:
2. kubectl get pods
3. Confirm that your external IP address has not changed:
4. kubectl get services
5. Return to the web browser tab in which you viewed your cluster's external IP address. Refresh the page to confirm that the nginx web server is still responding.

**Congratulations!**

In this lab, you configured a Kubernetes cluster in Kubernetes Engine. You populated the cluster with several pods containing an application, exposed the application, and scaled the application.