GCP Fundamentals: Getting Started with Kubernetes Engine

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 cluster using Kubernetes Engine.
- Deploy and manage Docker containers using `kubectl`.

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

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click Start Lab, shows how long Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access the Google Cloud Platform for the duration of the lab.

What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal GCP account or project, do not use it for this lab.

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. On the **Google Cloud Platform** menu, click **Activate Cloud Shell** (). If a dialog box appears, click **Start Cloud Shell**.
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
```

4. Start a Kubernetes cluster managed by Kubernetes Engine. Name the cluster **webfrontend** and configure it to run 2 nodes:

```
5. 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.

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

```
7. kubectl version
```

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

8. 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 run nginx --image=nginx:1.10.0
```

   In Kubernetes, all containers run in pods. This use of the `kubectl run` 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.
3. View the pod running the nginx container:

```
4. kubectl get pods
```

5. Expose the nginx container to the Internet:

```
6. 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.

7. View the new service:

```
8. 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 **ExternalIP** 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.
9. 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.

10. Scale up the number of pods running on your service:

```
11.  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.

12. Confirm that Kubernetes has updated the number of pods:

```
13. kubectl get pods
```

14. Confirm that your external IP address has not changed:

```
15. kubectl get services
```

16. 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.