

# Google Cloud Fundamentals: Getting Started with GKE

## Overview

In this lab, you create a Google 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

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

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, **1:15:00**), and make sure you can finish within that time.

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

3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.  
If you use other credentials, you'll receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

# Task 2. Confirm that needed APIs are enabled

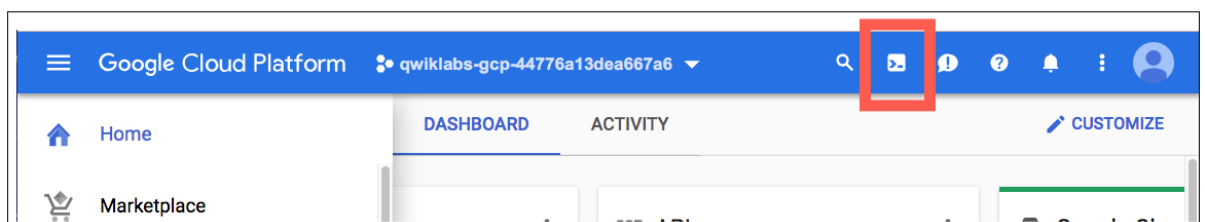
1. Make a note of the name of your Google Cloud project. This value is shown in the top bar of the Google Cloud Console. It will be of the form **qwiklabs-gcp-** followed by hexadecimal numbers.

2. In the Google Cloud 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 Google Cloud console, on the top right toolbar, click the **Activate Cloud Shell** button.



2. Click **Continue**.
3. At the Cloud Shell prompt, type the following command to export the environment variable called **MY\_ZONE**.
4. `export MY_ZONE="Zone"`
5. Copied!
6. content\_copy

7. Start a Kubernetes cluster managed by Kubernetes Engine. Name the cluster **webfrontend** and configure it to run 2 nodes:
8. `gcloud container clusters create webfrontend --zone $MY_ZONE --num-nodes 2`
9. Copied!
10. `content_copy`
11. It takes several minutes to create a cluster as Kubernetes Engine provisions virtual machines for you.
12. After the cluster is created, check your installed version of Kubernetes using the `kubectl version` command:
13. `kubectl version`
14. Copied!
15. `content_copy`
16. The `gcloud container clusters create` command automatically authenticated `kubectl` for you.
17. 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`
3. Copied!

4. `content_copy`
5. 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 versions, you can simply ignore it for now and can proceed further.

2. View the pod running the nginx container:
3. `kubectl get pods`
4. Copied!
5. `content_copy`
6. Expose the nginx container to the Internet:
7. `kubectl expose deployment nginx --port 80 --type LoadBalancer`
8. Copied!
9. `content_copy`
10. 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.
11. View the new service:
12. `kubectl get services`
13. Copied!
14. `content_copy`
15. 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.

16. 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.

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

18. `kubectl scale deployment nginx --replicas 3`

19. Copied!

20. `content_copy`

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

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

23. `kubectl get pods`

24. Copied!

25. `content_copy`

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

27. `kubectl get services`

28. Copied!

29. `content_copy`

30. 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.