Getting started with Google Kubernetes Engine (GKE)

Rafay Systems

530 Lakeside Drive, Suite 210

Sunnyvale, CA 94086

rafay.co

info@ray.co

Getting started with Google Kubernetes Engine(GKE)

Google Kubernetes Engine (GKE) is a managed environment for deploying, managing and scaling containerized applications using the Google Cloud Platform infrastructure. The environment that Google Kubernetes Engine provides consists of multiple machines, specifically Google Compute Engine instances, which are grouped together to form a cluster. Google Kubernetes Engine draws on the same reliable infrastructure and design principles that run popular Google services and provides the same benefits like automatic management, monitoring and liveness probes for application containers, automatic scaling, rolling updates, and more.

Overview

This guide is designed to help you to get started with Google Kubernetes Engine(GKE). We will be using Google cloud shell to setup the GKE cluster and host a multi-container application. This guide will walk through the steps to:

- Prerequisites
- Creating Clusters
- Administering Cluster
- Configuring and expanding Clusters
- Deploying workloads to clusters
- Deploying applications

Prerequisites

In this section, we will enable the kubernetes Engine API. Along with this we will setup a cloud shell and Local shell for deploying the Kubernetes Cluster.

Choosing a shell

To complete this tutorial, you can either use a cloud shell or local Linux terminal.

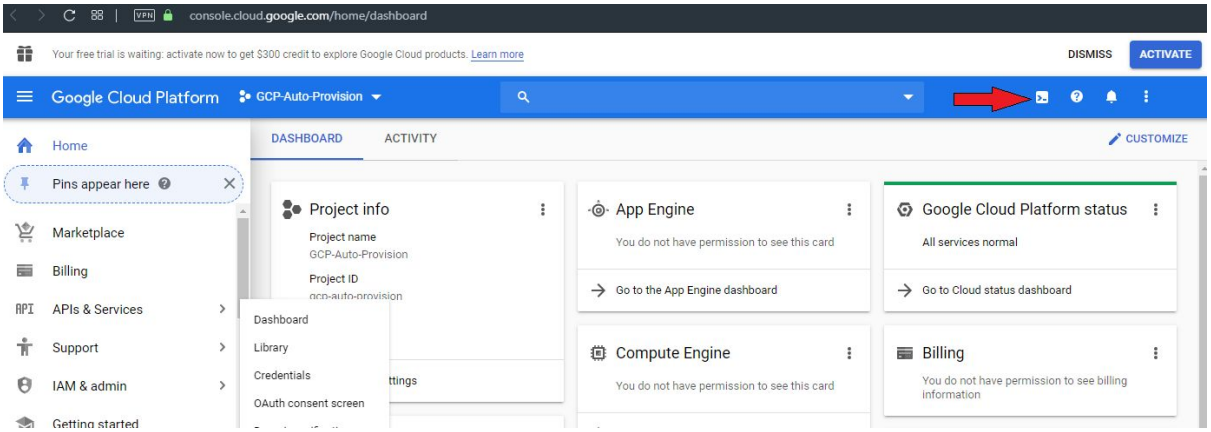
Cloud shell

Cloud Shell is a shell environment for managing resources hosted on Google Cloud. Cloud Shell comes preinstalled with the gcloud command-line tool and kubectl command-line tool. The gcloud tool provides the primary command-line interface for Google Cloud, and kubectl provides the primary command-line interface for running commands against Kubernetes clusters.

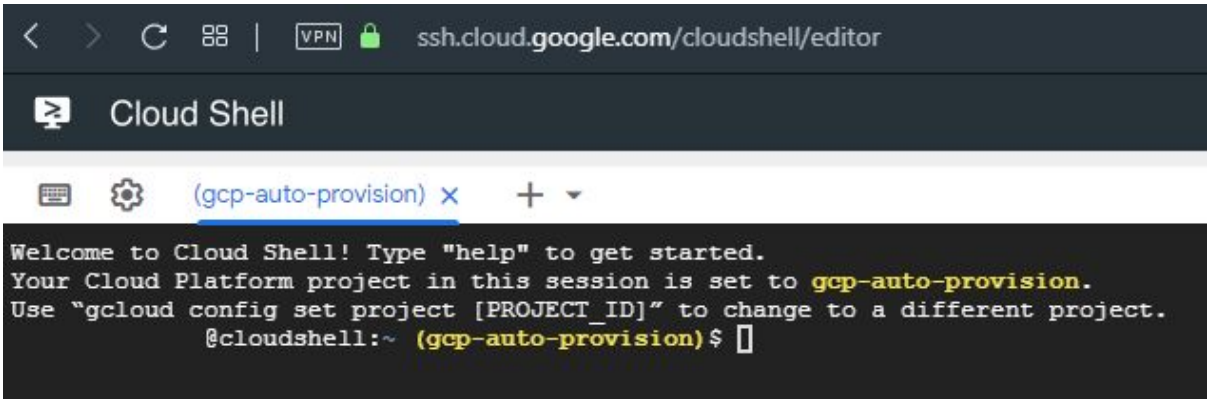
To launch Cloud shell, perform the following steps.

- Go to Google cloud console with your gmail account (https://console.cloud.google.com) Link.

- From the upper-right corner of the console, click the **Activate Cloud Shell**.



- This will launch the google cloudshell.



Local Shell

If you prefer using your local shell, you must install the gcloud tool and kubectl tool in your environment.

To install gcloud and kubectl, perform the following steps:

- Install the Cloud SDK, which includes the gcloud command-line tool.

```
[ec2-user@ip-172-31-83-88 ~]$ sudo yum install google-cloud-sdk
Azure CLI                                                                    29 kB/s | 2.
Docker CE Stable - x86_64                                                    93 kB/s | 3.
Google Cloud SDK                                                           1.8 kB/s | 4
Google Cloud SDK                                                            43 kB/s | 1.
Importing GPG key 0xA7317B0F:
 Userid     : "Google Cloud Packages Automatic Signing Key <gc-team@google.com>"
 Fingerprint: D0BC 747F D8CA F711 7500 D6FA 3746 C208 A731 7B0F
 From       : https://packages.cloud.google.com/yum/doc/yum-key.gpg
Is this ok [y/N]: y
```

- After installing Cloud SDK, install the kubectl command-line tool by running the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ sudo gcloud components install kubectl
```

Setting a default Project

To set a default project, run the following command from Cloud Shell:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud config set project gcp-auto-provision
Updated property [core/project].
```

To set a default compute zone, run the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud config set compute/zone us-west1-a
Updated property [compute/zone].
```

Now we are ready to launch a cluster and deploy an application.

Creating a GKE cluster

A cluster consists of at least one cluster master machine and multiple worker machines called nodes. Nodes are Compute Engine virtual machine (VM) instances that run the Kubernetes processes necessary to make them part of the cluster. You deploy applications to clusters, and the applications run on the nodes.

Creating a single-zone cluster

To create a cluster with the gcloud command-line tool, use one of the following gcloud container clusters commands.

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters create gcloud-demo --zone us-west1-a
WARNING: Currently VPC-native is not the default mode during cluster creation. In the future, this will become the default
 mode and can be disabled using `--no-enable-ip-alias` flag. Use `--[no-]enable-ip-alias` flag to suppress this warning.
WARNING: Newly created clusters and node-pools will have node auto-upgrade enabled by default. This can be disabled using
the `--no-enable-autoupgrade` flag.
WARNING: Starting in 1.12, default node pools in new clusters will have their legacy Compute Engine instance metadata endp
oints disabled by default. To create a cluster with legacy instance metadata endpoints disabled in the default node pool,
run `clusters create` with the flag `--metadata disable-legacy-endpoints=true`.
WARNING: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
This will enable the autorepair feature for nodes. Please see https://cloud.google.com/kubernetes-engine/docs/node-auto-re
pair for more information on node autorepairs.
Creating cluster gcloud-demo in us-west1-a... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-west1-a/gc
loud-demo?project=dulcet-cable-259410
kubeconfig entry generated for gcloud-demo.
NAME         LOCATION    MASTER_VERSION  MASTER_IP     MACHINE_TYPE   NODE_VERSION    NUM_NODES  STATUS
gcloud-demo  us-west1-a  1.13.11-gke.14  35.197.79.76  n1-standard-1  1.13.11-gke.14  3          RUNNING
[ec2-user@ip-172-31-83-88 ~]$
```

Creating a multi-zonal cluster

To create a multi-zonal cluster, set --zone to the zone for the cluster control plane, and set --node-locations to a comma-separated list of compute zones where the control plane and nodes are created. Use one of the following commands.

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters create example-cluster --zone us-central1-a --node-locations us-ce
ntral1-a,us-central1-b,us-central1-c
```

Administering Cluster

Viewing your clusters

To view a specific cluster, run the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters describe gcloud-demo
addonsConfig:
  kubernetesDashboard:
    disabled: true
  networkPolicyConfig:
    disabled: true
clusterIpv4Cidr: 10.48.0.0/14
createTime: '2019-12-14T11:07:47+00:00'
currentMasterVersion: 1.13.11-gke.14
currentNodeCount: 3
```

Setting a default cluster for gcloud

To set a default cluster for gcloud commands

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud config set container/cluster gcloud-demo
Updated property [container/cluster].
[ec2-user@ip-172-31-83-88 ~]$
```

Configuring cluster access for Kubectl

To run kubectl commands against a cluster created in Cloud Console, from another computer, or by another member of the project, you need to generate a kubeconfig entry in your environment.

Generate a kubeconfig entry by running the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters get-credentials gcloud-demo
Fetching cluster endpoint and auth data.
kubeconfig entry generated for gcloud-demo.
```

Upgrading the cluster

You can manually upgrade your cluster using the Cloud Console or the gcloud command-line tool.

To upgrade to the latest version, run the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters upgrade gcloud-demo --master
Master of cluster [gcloud-demo] will be upgraded from version
[1.13.11-gke.14] to version [1.13.11-gke.14]. This operation is
long-running and will block other operations on the cluster (including
 delete) until it has run to completion.

Do you want to continue (Y/n)?  Y

Upgrading gcloud-demo...done.
Updated [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
```

Resizing a Cluster

You can resize a cluster to increase or decrease the number of nodes in that cluster.

To increase the size of your cluster

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters resize gcloud-demo --node-pool default-pool --num-nodes 4
Pool [default-pool] for [gcloud-demo] will be resized to 4.

Do you want to continue (Y/n)?  Y

Resizing gcloud-demo...done.
Updated [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
```

To decrease the size of your cluster

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters resize gcloud-demo --node-pool default-pool --num-nodes 3
Pool [default-pool] for [gcloud-demo] will be resized to 3.

Do you want to continue (Y/n)?  Y

Resizing gcloud-demo...done.
Updated [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
```

AutoScaling a Cluster

To enable autoscaling for an existing node pool, run the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters update gcloud-demo --enable-autoscaling --min-nodes 1 --max-nodes
10 --node-pool default-pool
Updating gcloud-demo...done.
Updated [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-west1-a/gc
loud-demo?project=dulcet-cable-259410
```

Deleting a Cluster

To delete an existing cluster, run the following command:

```
[ec2-user@ip-172-31-83-88 ~]$ gcloud container clusters delete gcloud-demo
The following clusters will be deleted.
 - [gcloud-demo] in [us-west1-a]

Do you want to continue (Y/n)?  Y

Deleting cluster gcloud-demo...done.
Deleted [https://container.googleapis.com/v1/projects/dulcet-cable-259410/zones/us-west1-a/clusters/gcloud-demo].
```

Deploying workloads to clusters

To deploy and manage your containerized applications and other workloads on your Google Kubernetes Engine (GKE) cluster, you use the Kubernetes system to create Kubernetes controller objects. These controller objects represent the applications, daemons, and batch jobs running on your clusters.

You can create these controller objects using the Kubernetes API or by using kubectl, a command-line interface to Kubernetes installed by gcloud. Typically, you build a

representation of your desired Kubernetes controller object as a YAML configuration file, and then use that file with the Kubernetes API or the kubectl command-line interface.

Types of workloads

Kubernetes provides different kinds of controller objects that correspond to different kinds of workloads you can run. Certain controller objects are better suited to representing specific types of workloads. The following sections describe some common types of workloads and the Kubernetes controller objects you can create to run them on your cluster, including:

- Stateless applications: A stateless application does not preserve its state and saves no data to persistent storage — all user and session data stays with the client.

Some examples of stateless applications include web front ends like Nginx, web servers like Apache Tomcat, and other web applications.

- Stateful applications: A stateful application requires that its state be saved or persistent. Stateful applications use persistent storage, such as persistent volumes, to save data for use by the server or by other users.

Examples of stateful applications include databases like MongoDB and message queues like Apache ZooKeeper.

- Batch jobs: Batch jobs represent finite, independent, and often parallel tasks which run to their completion. Some examples of batch jobs include automatic or scheduled tasks like sending emails, rendering video, and performing expensive computations.

- Daemons: Daemons perform ongoing background tasks in their assigned nodes without the need for user intervention. Examples of daemons include log collectors like Fluentd and monitoring services.

Deploy a stateless application

Stateless applications are applications which do not store data or application state to the cluster or to persistent storage. Instead, data and application state stay with the client, which makes stateless applications more scalable. For example, a frontend application is stateless: you deploy multiple replicas to increase its availability and scale down when demand is low, and the replicas have no need for unique identities.

Kubernetes uses the Deployment controller to deploy stateless applications as uniform, non-unique Pods. Deployments manage the desired state of your application: how many Pods should run your application, what version of the container image should run, what the Pods

should be labelled, and so on. The desired state can be changed dynamically through updates to the Deployment's Pod specification.

Stateless applications are in contrast to stateful applications, which use persistent storage to save data and which use StatefulSets to deploy Pods with unique identities.

Creating a Deployment

The following is an example of a simple Deployment manifest file. This Deployment creates three replicated Pods labelled app=my-app that run the hello-app image stored in Container Registry:

You can declaratively create and update Deployments from manifest files using kubectl apply. This method also retains updates made to live resources without merging the changes back into the manifest files.

To create a Deployment from its manifest file, run the following command:

```
[ec2-user@ip-172-31-83-88 gcloud]$ kubectl apply -f config.yaml
deployment.apps/my-app created
```

To get detailed information about the Deployment, run the following command:

```
[ec2-user@ip-172-31-83-88 gcloud]$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
my-app-69b64b7f4c-d9j79   1/1     Running   0          4m16s
my-app-69b64b7f4c-hrnsl   1/1     Running   0          4m16s
my-app-69b64b7f4c-phwww   1/1     Running   0          4m16s
```

Deploying a stateful application

Stateful applications save data to persistent disk storage for use by the server, by clients, and by other applications. An example of a stateful application is a database or key-value store to which data is saved and retrieved by other applications.

Persistent storage can be dynamically provisioned, so that the underlying volumes are created on demand. In Kubernetes, you configure dynamic provisioning by creating a StorageClass. In GKE, a default StorageClass allows you to dynamically provision Compute Engine persistent disks.

Kubernetes uses the StatefulSet controller to deploy stateful applications as StatefulSet objects. Pods in StatefulSets are not interchangeable: each Pod has a unique identifier that is maintained no matter where it is scheduled.

Stateful applications are different from stateless applications, in which client data is not saved to the server between sessions.

Requesting persistent storage in a StatefulSets

Applications can request persistent storage with a [PersistentVolumeClaim]persistent disk storage.

Normally, PersistentVolumeClaim objects have to be created by the user in addition to the Pod. However, StatefulSets include a volumeClaimTemplates array, which automatically generates the PersistentVolumeClaim objects. Each StatefulSet replica gets its own PersistentVolumeClaim object.

kubectl apply uses manifest files to create, update, and delete resources in your cluster. This is a declarative method of object configuration. This method retains writes made to live objects without merging the changes back into the object configuration files.

The following is a simple example of a StatefulSet governed by a Service that has been created separately:

Where:

[STATEFULSET_NAME] is the name you choose for the StatefulSet

[SERVICE_NAME] is the name you choose for the Service

[APP_NAME] is the name you choose for the application run in the Pods

[CONTAINER_NAME] is name you choose for the containers in the Pods

[PORT_NAME] is the name you choose for the port opened by the StatefulSet

[PVC_NAME] is the name you choose for the PersistentVolumeClaim

In this file, the kind field specifies that a StatefulSet object should be created with the specifications defined in the file. This example StatefulSet produces three replicated Pods, and opens port 80 for exposing the StatefulSet to the Internet.

Deploying a containerized web application

This tutorial shows you how to package a web application in a Docker container image, and run that container image on a Google Kubernetes Engine cluster as a load-balanced set of replicas that can scale to the needs of your users.

Build the container image

GKE accepts Docker images as the application deployment format. To build a Docker image, you need to have an application and a Dockerfile.

For this tutorial, you will deploy a sample web application called hello-app, a web server written in Go that responds to all requests with the message “Hello, World!” on port 80.

The application is packaged as a Docker image, using the Dockerfile that contains instructions on how the image is built. You will use this Dockerfile to package your application.

To download the hello-app source code, run the following commands:

```
[ec2-user@ip-172-31-83-88 webapp]$ git clone https://github.com/GoogleCloudPlatform/kubernetes-engine-samples
Cloning into 'kubernetes-engine-samples'...
remote: Enumerating objects: 597, done.
remote: Total 597 (delta 0), reused 0 (delta 0), pack-reused 597
Receiving objects: 100% (597/597), 413.06 KiB | 20.65 MiB/s, done.
Resolving deltas: 100% (258/258), done.
```

Set the PROJECT_ID environment variable to your Google Cloud project ID. This variable will be used to associate the container image with your project's Container Registry.

```
[ec2-user@ip-172-31-83-88 webapp]$ cd kubernetes-engine-samples/hello-app
[ec2-user@ip-172-31-83-88 hello-app]$ export PROJECT_ID=dulcet-cable-259410
```

To build the container image of this application and tag it for uploading, run the following command:

```
[ec2-user@ip-172-31-83-88 hello-app]$ docker build -t gcr.io/${PROJECT_ID}/hello-app:v1 .
Sending build context to Docker daemon  9.728kB
Step 1/7 : FROM golang:1.8-alpine
1.8-alpine: Pulling from library/golang
```

You can run docker images command to verify that the build was successful:

```
[ec2-user@ip-172-31-83-88 hello-app]$ docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
gcr.io/dulcet-cable-259410/hello-app      v1                  f1727cc0eb4f        9 seconds ago       11.4MB
```

Upload the container image

You need to upload the container image to a registry so that GKE can download and run it.

First, configure Docker command-line tool to authenticate to Container Registry (you need to run this only once):

```
[ec2-user@ip-172-31-83-88 hello-app]$ gcloud auth configure-docker
The following settings will be added to your Docker config file
located at [/home/ec2-user/.docker/config.json]:
 {
  "credHelpers": {
    "gcr.io": "gcloud",
    "us.gcr.io": "gcloud",
    "eu.gcr.io": "gcloud",
    "asia.gcr.io": "gcloud",
    "staging-k8s.gcr.io": "gcloud",
    "marketplace.gcr.io": "gcloud"
  }
}

Do you want to continue (Y/n)?  Y

Docker configuration file updated.
```

You can now use the Docker command-line tool to upload the image to your Container Registry:

```
[ec2-user@ip-172-31-83-88 hello-app]$ docker push gcr.io/${PROJECT_ID}/hello-app:v1
The push refers to repository [gcr.io/dulcet-cable-259410/hello-app]
e78d7caf0811: Pushed
77cae8ab23bf: Layer already exists
v1: digest: sha256:5144c34b18f2294fcd0a86693a613f41cf1eb7ef0b4acfd5d303f4a18f943acd size: 739
```

Run your container locally

To test your container image using your local Docker engine, run the following command:

Deploy your application

To deploy and manage applications on a GKE cluster, you must communicate with the Kubernetes cluster management system. You typically do this by using the kubectl command-line tool.

Kubernetes represents applications as Pods, which are units that represent a container (or group of tightly-coupled containers). The Pod is the smallest deployable unit in Kubernetes. In this tutorial, each Pod contains only your hello-app container.

The kubectl create deployment command below causes Kubernetes to create a Deployment named hello-web on your cluster. The Deployment manages multiple copies of your application, called replicas, and schedules them to run on the individual nodes in your cluster. In this case, the Deployment will be running only one Pod of your application.

Run the following command to deploy your application:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl create deployment hello-web --image=gcr.io/${PROJECT_ID}/hello-app:v1
deployment.apps/hello-web created
```

To see the Pod created by the Deployment, run the following command:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
hello-web-fb5c599b4-5glc6    1/1     Running   0          32s
my-app-69b64b7f4c-d9j79      1/1     Running   0          73m
my-app-69b64b7f4c-hrnsl      1/1     Running   0          73m
my-app-69b64b7f4c-phwww      1/1     Running   0          73m
```

Expose your application to the Internet

By default, the containers you run on GKE are not accessible from the Internet, because they do not have external IP addresses. You must explicitly expose your application to traffic from the Internet, run the following command:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-port 8080
service/hello-web exposed
```

To get the external-IP of your application, run the below command:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl get service
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
hello-web    LoadBalancer   10.51.242.66   34.82.238.249   80:31779/TCP   102s
kubernetes   ClusterIP      10.51.240.1    <none>          443/TCP        94m
```

Scale up your application

You add more replicas to your application's Deployment resource by using the kubectl scale command. To add two additional replicas to your Deployment (for a total of three), run the following command:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl scale deployment hello-web --replicas=3
deployment.extensions/hello-web scaled
```

You can see the new replicas running on your cluster by running the following commands:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl get deployment hello-web
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
hello-web   3/3     3            3           4m12s
```

Now, you have multiple instances of your application running independently of each other and you can use the kubectl scale command to adjust capacity of your application.

The load balancer you provisioned in the previous step will start routing traffic to these new replicas automatically.

Deploy a new version of your app

GKE's rolling update mechanism ensures that your application remains up and available even as the system replaces instances of your old container image with your new one across all the running replicas.

You can create an image for the v2 version of your application by building the same source code and tagging it as v2 (or you can change the "Hello, World!" string to "Hello, GKE Version 2" before building the image):

```
// hello responds to the request with a plain-text "Hello, world" message.
func hello(w http.ResponseWriter, r *http.Request) {
        log.Printf("Serving request: %s", r.URL.Path)
        host, _ := os.Hostname()
        fmt.Fprintf(w, "Hello, GKE Version 2\n")
        fmt.Fprintf(w, "Version: 1.0.0\n")
        fmt.Fprintf(w, "Hostname: %s\n", host)
}
// [END all]
```

Now build the image:

```
[ec2-user@ip-172-31-83-88 hello-app]$ docker build -t gcr.io/${PROJECT_ID}/hello-app:v2 .
Sending build context to Docker daemon  9.728kB
Step 1/7 : FROM golang:1.8-alpine
 ---> 4cb86d3661bf
Step 2/7 : ADD . /go/src/hello-app
 ---> 64166fe5cba8
Step 3/7 : RUN go install hello-app
 ---> Running in bd06a765abd4
Removing intermediate container bd06a765abd4
 ---> 29ec84619bc3
Step 4/7 : FROM alpine:latest
 ---> 965ea09ff2eb
Step 5/7 : COPY --from=0 /go/bin/hello-app .
 ---> 69d0d1ae1ea5
Step 6/7 : ENV PORT 8080
 ---> Running in f20412b72425
Removing intermediate container f20412b72425
 ---> 9878934d8dd0
Step 7/7 : CMD ["./hello-app"]
 ---> Running in 919101659bf2
Removing intermediate container 919101659bf2
 ---> 1f1cee4b863b
Successfully built 1f1cee4b863b
Successfully tagged gcr.io/dulcet-cable-259410/hello-app:v2
```

Push the image to the Google Container Registry:

```
[ec2-user@ip-172-31-83-88 hello-app]$ docker push gcr.io/${PROJECT_ID}/hello-app:v2
The push refers to repository [gcr.io/dulcet-cable-259410/hello-app]
5bd74fde8243: Pushed
77cae8ab23bf: Layer already exists
v2: digest: sha256:fd2674d0af4dfddc5c6135e23f413f25741b699ece449a42b21adb01d086e376 size: 739
```

Now, apply a rolling update to the existing deployment with an image update:

```
[ec2-user@ip-172-31-83-88 hello-app]$ kubectl set image deployment/hello-web hello-app=gcr.io/${PROJECT_ID}/hello-app:v2
deployment.extensions/hello-web image updated
```

Visit your application again at http://[EXTERNAL_IP], and observe the changes you made take effect.


```
Hello, GKE Version 2
Version: 1.0.0
Hostname: hello-web-77dfc7779f-4v6fj
```

Conclusion

In this guide you have seen how easily we build a GKE cluster on Google environment. GKE offers a platform for managing your containers across a variety of operating environments, significantly reducing the time necessary to build, deploy, and scale them. As an open source next-generation virtualization tool, Google Kubernetes service provides you with all the functionality you need to optimize containerization usage with your existing IT resources.