Kubernetes Engine Tutorials (https://cloud.google.com/kubernetes-engine/docs/tutorials/)

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.

Objectives

To package and deploy your application on GKE, you must:

1. Package your app into a Docker image
2. Run the container locally on your machine (optional)
3. Upload the image to a registry
4. Create a container cluster
5. Deploy your app to the cluster
6. Expose your app to the Internet
7. Scale up your deployment
8. Deploy a new version of your app

Before you begin

Take the following steps to enable the Kubernetes Engine API:

1. Visit the Kubernetes Engine page (https://console.cloud.google.com/projectselector/kubernetes) in the Google Cloud Platform Console.
2. Create or select a project.
3. Wait for the API and related services to be enabled. This can take several minutes.
4. Make sure that billing is enabled for your Google Cloud Platform project. Learn how to enable billing (https://cloud.google.com/billing/docs/how-to/modify-project).

Option A: Use Google Cloud Shell

You can follow this tutorial using Google Cloud Shell (https://cloud.google.com/shell), which comes preinstalled with the **`gcloud`**, **`docker`**, and **`kubectl`** command-line tools used in this tutorial. If you use Cloud Shell, you don't need to install these command-line tools on your workstation.

To use Google Cloud Shell:

1. Go to the Google Cloud Platform Console (https://console.cloud.google.com/).
2. Click the **Activate Cloud Shell** button at the top of the console window.

A Cloud Shell session opens inside a new frame at the bottom of the console and displays a command-line prompt.

Option B: Use command-line tools locally

If you prefer to follow this tutorial on your workstation, you need to install the following tools:

1. Install the Google Cloud SDK (https://cloud.google.com/sdk/docs/quickstarts), which includes the **`gcloud`** command-line tool.
2. Using the **`gcloud`** command line tool, install the Kubernetes (https://kubernetes.io) command-line tool. **`kubectl`** is used to communicate with Kubernetes, which is the cluster orchestration system of GKE clusters:

```
gcloud components install kubectl
```

3. Install Docker Community Edition (CE) (https://docs.docker.com/engine/installation/) on your workstation. You will use this to build a container image for the application.

4. Install the Git source control (https://git-scm.com/downloads) tool to fetch the sample application from GitHub.

Step 1: 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 (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/master/hello-app) called **`hello-app`**, a web server written in Go (https://golang.org/) that responds to all requests with the message "Hello, World!" on port 80.

The application is packaged as a Docker image, using the Dockerfile (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/master/hello-app/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:

```
git clone https://github.com/GoogleCloudPlatform/kubernetes-engine-samples
cd kubernetes-engine-samples/hello-app
```

Set the **`PROJECT_ID`** environment variable to your GCP project ID (https://cloud.google.com/resource-manager/docs/creating-managing-projects#identifying_projects). This variable will be used to associate the container image with your project's Container Registry (https://cloud.google.com/container-registry).

```
export PROJECT_ID=[PROJECT_ID]
```

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

```
docker build -t gcr.io/${PROJECT_ID}/hello-app:v1 .
```

This command instructs Docker to build the image using the **`Dockerfile`** in the current directory and tag it with a name, such as **`gcr.io/my-project/hello-app:v1`**. The **`gcr.io`**

prefix refers to Google Container Registry (https://cloud.google.com/container-registry), where the image will be hosted. Running this command does not upload the image yet.

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

```
docker images
```

Output:

```
REPOSITORY                    TAG                 IMAGE ID            CREATED
gcr.io/my-project/hello-app   v1                  25cfadb1bf28        10 second
```

Step 2: 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 (https://cloud.google.com/container-registry) (you need to run this only once):

```
gcloud auth configure-docker
```

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

```
docker push gcr.io/${PROJECT_ID}/hello-app:v1
```

Step 3: Run your container locally (optional)

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

```
docker run --rm -p 8080:8080 gcr.io/${PROJECT_ID}/hello-app:v1
```

If you're on Cloud Shell, you can click "Web preview" button on the top right to see your application running in a browser tab. Otherwise, open a new terminal window (or a Cloud Shell tab) and run to verify if the container works and responds to requests with "Hello, World!":

```
curl http://localhost:8080
```

Once you've seen a successful response, you can shut down the container by pressing **Ctrl+C** in the tab where **`docker run`** command is running.

Step 4: Create a container cluster

Now that the container image is stored in a registry, you need to create a container cluster (https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture) to run the container image. A cluster consists of a pool of Compute Engine VM instances (https://cloud.google.com/compute) running Kubernetes (https://kubernetes.io), the open source cluster orchestration system that powers GKE.

Once you have created a GKE cluster, you use Kubernetes to deploy applications to the cluster and manage the applications' lifecycle.

Set your project ID (https://cloud.google.com/resource-manager/docs/creating-managing-projects#identifying_projects) and Compute Engine zone (https://cloud.google.com/compute/docs/zones#available) options for the **`gcloud`** tool:

```
gcloud config set project $PROJECT_ID
gcloud config set compute/zone [COMPUTE_ENGINE_ZONE]
```

Run the following command to create a two-node cluster named **`hello-cluster`**:

```
gcloud container clusters create hello-cluster --num-nodes=2
```

It may take several minutes for the cluster to be created. Once the command has completed, run the following command and see the cluster's three worker VM instances:

```
gcloud compute instances list
```

Output:

```
NAME                                          ZONE           MACHINE_TYPE   PREE
gke-hello-cluster-default-pool-07a63240-822n  us-central1-b  n1-standard-1
gke-hello-cluster-default-pool-07a63240-kbtq  us-central1-b  n1-standard-1
```

Note: If you are using an existing Google Kubernetes Engine cluster or if you have created a cluster through Google Cloud Platform Console, you need to run the following command to retrieve cluster credentials and configure **`kubectl`** command-line tool with them:

```
gcloud container clusters get-credentials hello-cluster
```

If you have already created a cluster with the **`gcloud container clusters create`** command listed above, this step is not necessary.

Step 5: 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 (https://kubernetes.io/docs/concepts/workloads/pods/pod/), 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 (https://kubernetes.io/docs/concepts/workloads/controllers/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:

```
kubectl create deployment hello-web --image=gcr.io/${PROJECT_ID}/hello-app:v1
```

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

```
kubectl get pods
```

Output:

```
NAME                           READY     STATUS    RESTARTS   AGE
hello-web-4017757401-px7tx     1/1       Running   0          3s
```

Step 6: 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:

```
kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-po
```

The **`kubectl expose`** command above creates a Service (https://kubernetes.io/docs/user-guide/services/) resource, which provides networking and IP support to your application's Pods. GKE creates an external IP and a Load Balancer (subject to billing (https://cloud.google.com/compute/pricing#lb)) for your application.

The **`--port`** flag specifies the port number configured on the Load Balancer, and the **`--target-port`** flag specifies the port number that the **`hello-app`** container is listening on.

Note: GKE assigns the external IP address to the **Service** resource—not the Deployment. If you want to find out the external IP that GKE provisioned for your application, you can inspect the Service with the **`kubectl get service`** command:

```
kubectl get service
```

Output:

```
NAME         CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
hello-web    10.3.251.122    203.0.113.0     80:30877/TCP     3d
```

Once you've determined the external IP address for your application, copy the IP address. Point your browser to this URL (such as **`http://203.0.113.0`**) to check if your application is accessible.

Step 7: 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:

```
kubectl scale deployment hello-web --replicas=3
```

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

```
kubectl get deployment hello-web
```

Output:

```
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-web   3         3         3            2           1m
```

```
kubectl get pods
```

Output:

```
NAME                         READY     STATUS    RESTARTS   AGE
hello-web-4017757401-ntgdb   1/1       Running   0          9s
hello-web-4017757401-pc4j9   1/1       Running   0          9s
hello-web-4017757401-px7tx   1/1       Running   0          1m
```

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.

Step 8: 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!`**" before building the image):

```
docker build -t gcr.io/${PROJECT_ID}/hello-app:v2 .
```

Then push the image to the Google Container Registry:

```
docker push gcr.io/${PROJECT_ID}/hello-app:v2
```

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

```
kubectl set image deployment/hello-web hello-app=gcr.io/${PROJECT_ID}/hello-a
```

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

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

After completing this tutorial, follow these steps to remove the following resources to prevent unwanted charges incurring on your account:

1. **Delete the Service:** This step will deallocate the Cloud Load Balancer created for your Service:

   ```
   kubectl delete service hello-web
   ```

2. **Delete the container cluster:** This step will delete the resources that make up the container cluster, such as the compute instances, disks and network resources.

   ```
   gcloud container clusters delete hello-cluster
   ```

What's next

- Read the Load Balancers (https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer) tutorial, which demonstrates advanced load balancing configurations for web applications.
- Learn how to store persistent data in your application through the MySQL and WordPress (https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk) tutorial.
- Configure static IP and domain name (https://cloud.google.com/kubernetes-engine/docs/tutorials/configuring-domain-name-static-ip) for your application.
- Explore other Kubernetes Engine tutorials (https://cloud.google.com/kubernetes-engine/docs/tutorials).
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).
