Appendix B. Setting Up a Kubernetes Cluster on Google Cloud

This appendix provides a brief overview of how to create a Kubernetes cluster on Google Cloud that can run our example project. If Kubernetes is new to you, take a look at Appendix A and our suggested reading at the end of Chapter 9. While the exact commands we will cover only apply to Google Cloud, the overall setup process is the same with other managed Kubernetes services like AWS EKS or Microsoft Azure's AKS.

Before You Get Started

For the following installation steps, we assume you have an account with Google Cloud. If you don't have an account, you can create one. Furthermore, we assume that you have installed Kubernetes `kubectl` (client version 1.18.2 or higher) on your local computer and that you can also execute Google Cloud's SDK `gcloud` (version 289.0.0 or higher).

WATCH YOUR CLOUD INFRASTRUCTURE COSTS

Operating Kubernetes clusters can accumulate significant infrastructure costs. Therefore, we highly recommend to watch your infrastructure costs by setting billing alerts and budgets. Details can be found in the Google Cloud documentation. We also recommend turning off idling compute instances because they accrue costs even if they are idling and no pipeline task is being computed.

Steps on how to install a `kubectl` client for your operating system can be found as part of the Kubernetes documentation. The Google Cloud documentation provides step-by-step details on how to install their client for your operating system.

Kubernetes on Google Cloud

In the following five sections, we take you through the step-by-step process of creating a Kubernetes cluster from scratch with Google Cloud.

Selecting a Google Cloud Project

For the Kubernetes cluster, we need to create a new Google Cloud project or select an existing project in the Google Cloud Project dashboard.

Please note the project ID for the following steps. We will deploy our cluster in the project with the ID `oreilly-book`, as shown in Figure B-1.

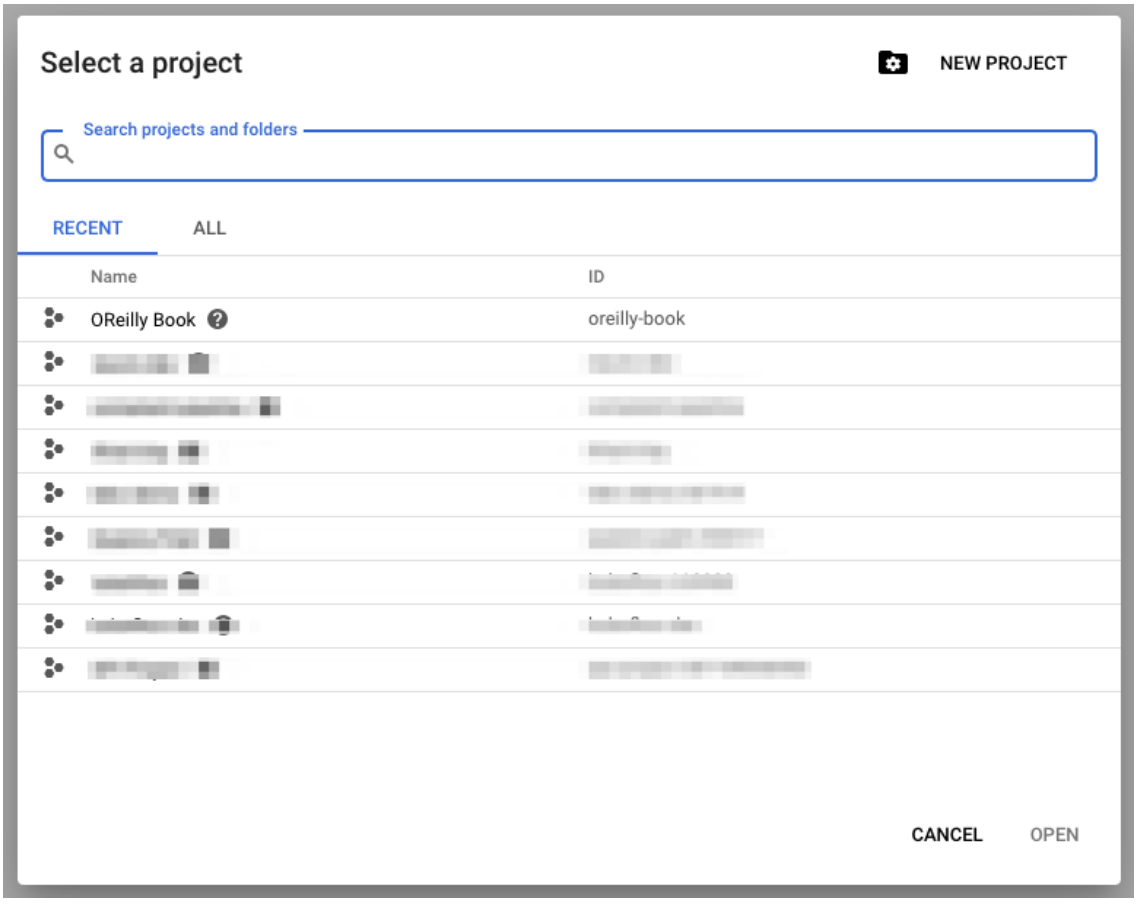

Figure B-1. Google Cloud Project dashboard

Setting Up Your Google Cloud Project

Before creating a Kubernetes cluster, let's set up your Google Cloud project. In the terminal of your operating system, you can authenticate your Google Cloud SDK client with:

```
$ gcloud auth login
```

Then update the SDK client with:

```
$ gcloud components update
```

After you have successfully authenticated and updated the SDK client, let's configure a few basics. First, we'll set the GCP project as the default project and pick a compute zone as a default zone. In our example, we have chosen `us-central-1`. You can find a list of all available zones in the Google Cloud documentation. Pick a zone either closest to your physical location or where the required Google Cloud services are available (not all services are available in all zones).

By setting these default values, we don't have to specify them later on in following commands. We also will request to enable Google Cloud's container APIs. The last step is only needed once per project:

```
$ export PROJECT_ID=<your gcp project id> ❶
$ export GCP_REGION=us-central1-c ❷
$ gcloud config set project $PROJECT_ID
$ gcloud config set compute/zone $GCP_REGION
$ gcloud services enable container.googleapis.com ❸
```

❶ Replace with the project ID from the previous step.

❷ Select your preferred zone or region.

❸ Enable APIs.

Creating a Kubernetes Cluster

With our Google Cloud project ready to go, we can now create a Kubernetes cluster with a number of compute nodes as part of the cluster. In our example cluster called `kfp-oreilly-book`, we allow the cluster to run between zero and five nodes at any point in time in our pool called `kfp-pool`, and the desired number of available nodes is three. We also assign a service account to the cluster. Through the service account,

we can control access permissions for requests from the cluster nodes. To learn more about service accounts at Google Cloud, we recommend the online documentation:

```
$ export CLUSTER_NAME=kfp-oreilly-book
$ export POOL_NAME=kfp-pool
$ export MAX_NODES=5
$ export NUM_NODES=3
$ export MIN_NODES=0
$ export SERVICE_ACCOUNT=service-account@oreilly-book.iam.gserviceaccount.com
```

With the cluster parameters now defined in an environment variable, we can execute the following command:

```
$ gcloud container clusters create $CLUSTER_NAME \
      --zone $GCP_REGION \
      --machine-type n1-standard-4 \
      --enable-autoscaling \
      --min-nodes=$MIN_NODES \
      --num-nodes=$NUM_NODES \
      --max-nodes=$MAX_NODES \
      --service-account=$SERVICE_ACCOUNT
```

For our demo pipeline, we selected the instance type `n1-standard-4`, which provides 4 CPUs and 15 GB of memory per node. These instances provide enough compute resources to train and evaluate our machine learning model and its datasets. You can find a complete list of available instance types by running the following SDK command:

```
$ gcloud compute machine-types list
```

If you would like to add a GPU to the cluster, you can specify the GPU type and the number of GPUs by adding the `accelerator` argument, as shown in the following example:

```
$  gcloud container clusters create $CLUSTER_NAME \
      ...
```

```
        --accelerator=type=nvidia-tesla-v100,count=1
```

The creation of the Kubernetes cluster can take a few minutes until all the resources are fully assigned to your project and available. The time depends on your requested resources and the number of nodes. For our demo cluster, you can expect to wait approximately 5 minutes until all the resources are available.

Accessing Your Kubernetes Cluster with kubectl

When your newly created cluster is available, you can set up your `kubectl` to access the cluster. The Google Cloud SDK provides a command to register the cluster with your local `kubectl` configuration:

```
$ gcloud container clusters get-credentials $CLUSTER_NAME --zone $GCP_REGION
```

After updating the `kubectl` configuration, you can check if the correct cluster is selected by running the following command:

```
$ kubectl config current-context
gke_oreilly-book_us-central1-c_kfp-oreilly-book
```

Using Your Kubernetes Cluster with kubectl

Because your local `kubectl` can connect with your remote Kubernetes cluster, all `kubectl` commands, such as our Kubeflow Pipelines steps mentioned in the following and in Chapter 12, will be executed on the remote cluster:

```
$ export PIPELINE_VERSION=0.5.0
$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/"\
                   "cluster-scoped-resources?ref=$PIPELINE_VERSION"
$ kubectl wait --for condition=established \
                --timeout=60s crd/applications.app.k8s.io
$ kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/"\
                   "env/dev?ref=$PIPELINE_VERSION"
```

Persistent Volume Setups for Kubeflow Pipelines

In "Exchange Data Through Persistent Volumes", we'll discuss the setup of persistent volumes in our Kubeflow Pipelines setup. The complete configuration of the persistent volume and its claim can be seen in the following code blocks. The presented setup is specific to the Google Cloud environment.

Example B-1 shows the configuration of the persistent volume for our Kubernetes cluster:

Example B-1. Persistent volume configuration

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: tfx-pv
  namespace: kubeflow
  annotations:
    kubernetes.io/createdby: gce-pd-dynamic-provisioner
    pv.kubernetes.io/bound-by-controller: "yes"
    pv.kubernetes.io/provisioned-by: kubernetes.io/gce-pd
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 20Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: tfx-pvc
    namespace: kubeflow
  gcePersistentDisk:
    fsType: ext4
    pdName: tfx-pv-disk
  nodeAffinity:
    required:
      nodeSelectorTerms:
```

```
        - matchExpressions:
          - key: failure-domain.beta.kubernetes.io/zone
            operator: In
            values:
            - us-central1-c
          - key: failure-domain.beta.kubernetes.io/region
            operator: In
            values:
            - us-central1
    persistentVolumeReclaimPolicy: Delete
    storageClassName: standard
    volumeMode: Filesystem
  status:
    phase: Bound
```

Once the persistent volume is created, we can claim a portion or all of the available storage through a persistent volume claim. The configuration file can be seen in Example B-2:

Example B-2. Persistent volume claim configuration

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tfx-pvc
  namespace: kubeflow
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

With the presented configuration, we have now created a persistent volume and its claim in the Kubernetes cluster. The volume can now be mounted as discussed in "Pipeline Setup" or used as discussed in the section "Exchange Data Through Persistent Volumes" of the following appendix.