Using Kubernetes Engine to Deploy Apps with Regional Persistent Disks

1 hour7 Credits

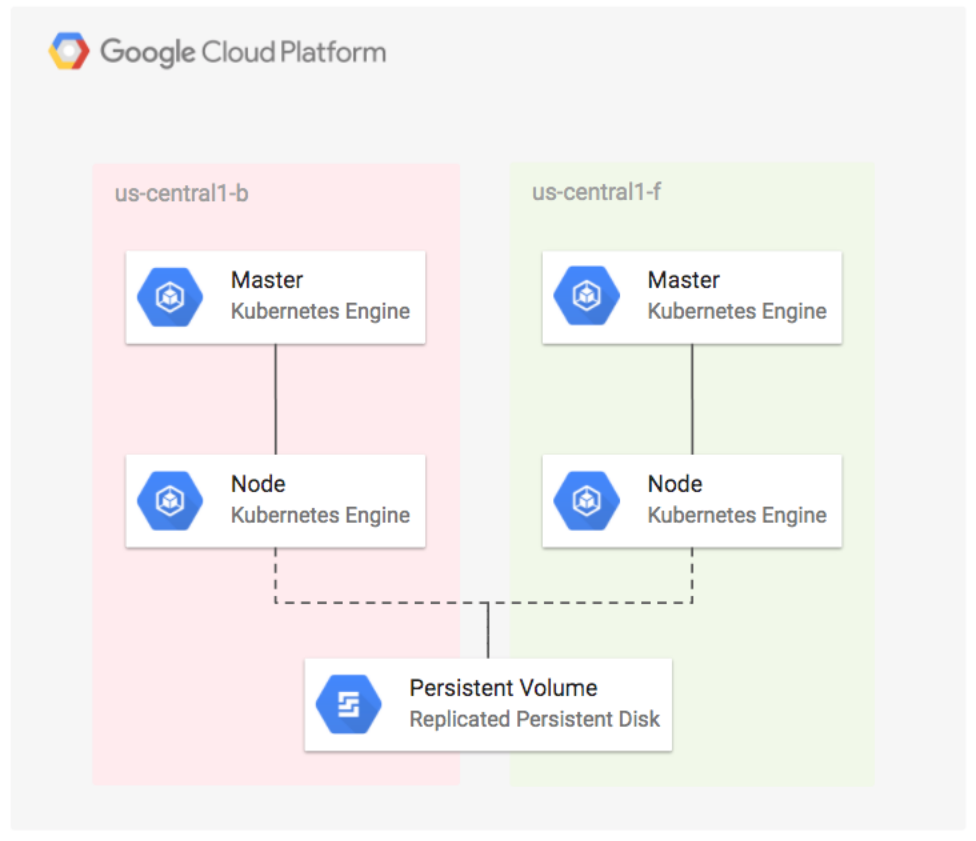
Rate Lab

**GSP200**



**Overview**

In this lab you will learn how to configure a highly available application by deploying WordPress using regional persistent disks on Kubernetes Engine. Regional persistent disks provide synchronous replication between two zones, which keeps your application up and running in case there is an outage or failure in a single zone. Deploying a Kubernetes Engine Cluster with regional persistent disks will make your application more stable, secure, and reliable.



What you'll do

* Create a regional Kubernetes Engine cluster.
* Create a Kubernetes StorageClass resource that is configured for replicated zones.
* Deploy WordPress with a regional disk that uses the StorageClass.
* Simulate a zone failure by deleting a node.
* Verify that the WordPress app and data migrate successfully to another replicated zone.

Prerequisites

This is an **advanced** lab. Before taking it, you should be familiar with at least the basics of Kubernetes and WordPress. Here are some Qwiklabs that can get you up to speed:

* [Kubernetes Engine: Qwik Start](https://google.qwiklabs.com/catalog_lab/911)
* [Running WordPress on App Engine Flexible Environment](https://google.qwiklabs.com/catalog_lab/985)
* [Hello Node Kubernetes](https://google.qwiklabs.com/catalog_lab/468)

Once you're prepared, scroll down to get your lab environment set up.

**Setup**

**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 hand-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.

**How to start your lab and sign in to the Console**

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left you will see a panel populated with the temporary credentials that you must use for this lab.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

***Tip:*** Open the tabs in separate windows, side-by-side.

1. On the Choose an account page, click **Use Another Account**.



1. The Sign in page opens. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

***Important:*** You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

1. Click through the subsequent pages:
   * Accept the terms and conditions.
   * Do not add recovery options or two-factor authentication (because this is a temporary account).
   * Do not sign up for free trials.

After a few moments, the GCP console opens in this tab.

**Note:** You can view the menu with a list of GCP Products and Services by clicking the **Navigation menu** at the top-left, next to “Google Cloud Platform”. 

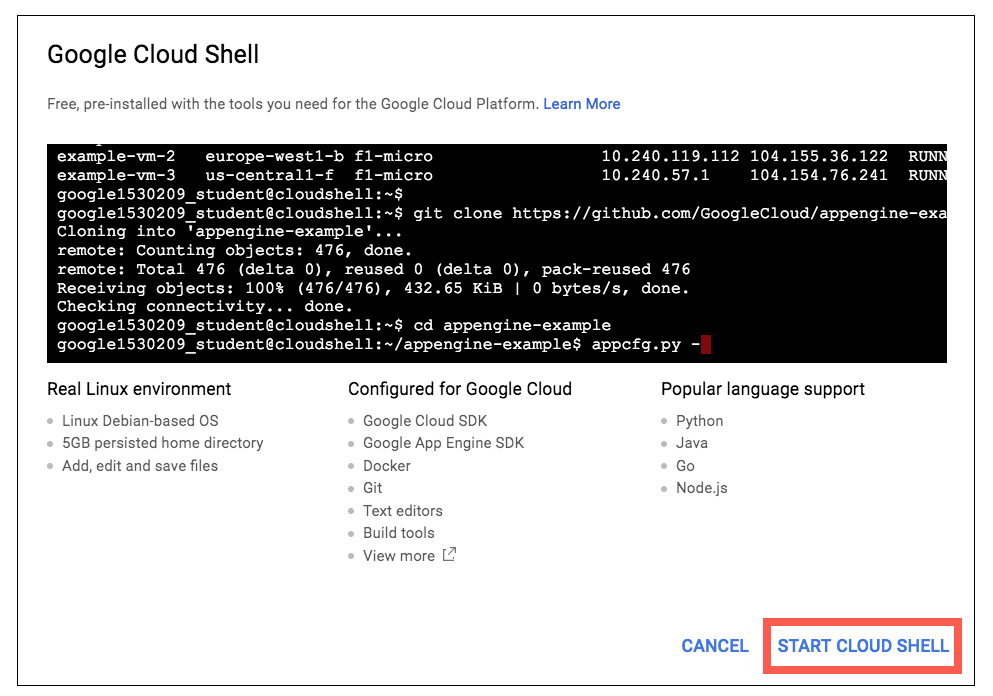
Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



1. In the dialog box that opens, click **START CLOUD SHELL**:



You can click "START CLOUD SHELL" immediately when the dialog box opens.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



**gcloud** is the command-line tool for Google Cloud Platform. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

Example output:

Credentialed accounts:

- google1623327\_student@qwiklabs.net

You can list the project ID with this command:

gcloud config list project

Output:

[core]

project = <project\_ID>

Example output:

[core]

project = qwiklabs-gcp-44776a13dea667a6

Full documentation of **gcloud** is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud).

**Creating the Regional Kubernetes Engine Cluster**

Open a new Cloud Shell session. You will first create a [regional Kubernetes Engine cluster](https://cloud.google.com/kubernetes-engine/docs/concepts/multi-zone-and-regional-clusters) that spans three zones in the us-west1 region. First, fetch the server configuration for the us-west1 region and export environment variables by running:

CLUSTER\_VERSION=$(gcloud container get-server-config --region us-west1 --format='value(validMasterVersions[0])')

export CLOUDSDK\_CONTAINER\_USE\_V1\_API\_CLIENT=false

Now create a standard Kubernetes Engine cluster (this will take a little while, ignore any warnings about node auto repairs):

gcloud container clusters create repd \

--cluster-version=${CLUSTER\_VERSION} \

--machine-type=n1-standard-4 \

--region=us-west1 \

--num-nodes=1 \

--node-locations=us-west1-a,us-west1-b,us-west1-c

**Example Output:**

Creating cluster repd...done.

Created [https:*//container.googleapis.com/v1beta1/projects/qwiklabs-gcp-e8f5f22705c770ab/zones/us-west1/clusters/repd].*

To inspect the contents of your cluster, go to: https:*//console.cloud.google.com/kubernetes/workload\_/gcloud/us-west1/repd?project=qwiklabs-gcp-e8f5f22705c770ab*

kubeconfig entry generated for repd.

NAME LOCATION MASTER\_VERSION MASTER\_IP MACHINE\_TYPE NODE\_VERSION NUM\_NODES STATUS

repd us-west1 1.12.6-gke.7 35.247.50.133 n1-standard-4 1.12.6-gke.7 3 RUNNING

You just created a regional cluster (located in us-west1) with one node in each zone (us-west1-a,us-west1-b,us-west1-c). Navigate to **Compute Engine** from the left-hand menu to view your instances:



The gcloud command has also automatically configured the kubectl command to connect to the cluster.

**Deploying the App with a Regional Disk**

Now that you have your Kubernetes cluster running, you'll do the following three things:

* Install [Helm](https://helm.sh/) (a toolset for managing Kubernetes packages)
* Create the [Kubernetes StorageClass](https://kubernetes.io/docs/concepts/storage/storage-classes/) that is used by the regional persistent disk
* Deploy WordPress

Install and initialize Helm to install the chart package

The chart package, which is installed with Helm, contains everything you need to run WordPress.

1. Install Helm locally in your Cloud Shell instance by running:

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get\_helm.sh

chmod 700 get\_helm.sh

./get\_helm.sh

1. Initialize Helm:

kubectl create serviceaccount tiller --namespace kube-system

kubectl create clusterrolebinding tiller-cluster-rule \

--clusterrole=cluster-admin \

--serviceaccount=kube-system:tiller

helm init --service-account=tiller

until (helm version --tiller-connection-timeout=1 >/dev/null 2>&1); do echo "Waiting for tiller install..."; sleep 2; done && echo "Helm install complete"

Helm is now installed in your cluster.

Create the StorageClass

Next you'll create the StorageClass used by the chart to define the zones of the regional disk. The zones listed in the StorageClass will match the zones of the Kubernetes Engine cluster.

Create a StorageClass for the regional disk by running:

kubectl apply -f - <<EOF

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: repd-west1-a-b-c

provisioner: kubernetes.io/gce-pd

parameters:

type: pd-standard

replication-type: regional-pd

zones: us-west1-a, us-west1-b, us-west1-c

EOF

**Example Output:**

storageclass "repd-west1-a-b-c" created

You now have a StorageClass that is capable of provisioning [PersistentVolumes](https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes)that are replicated across the us-west1-a, us-west1-b and us-west1-c zones.

List the available storageclass with:

kubectl get storageclass

**Example Output:**

NAME PROVISIONER AGE

repd-west1-a-b-c kubernetes.io/gce-pd 26s

standard (default) kubernetes.io/gce-pd 1h

**Create Persistent Volume Claims**

In this section, you will create persistentvolumeclaims for your application.

Create data-wp-repd-mariadb-0 PVC with standard StorageClass.

kubectl apply -f - <<EOF

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: data-wp-repd-mariadb-0

namespace: default

labels:

app: mariadb

component: master

release: wp-repd

spec:

accessModes:

- ReadOnlyMany

resources:

requests:

storage: 8Gi

storageClassName: standard

EOF

Create wp-repd-wordpress PVC with repd-west1-a-b-c StorageClass.

kubectl apply -f - <<EOF

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: wp-repd-wordpress

namespace: default

labels:

app: wp-repd-wordpress

chart: wordpress-5.7.1

heritage: Tiller

release: wp-repd

spec:

accessModes:

- ReadOnlyMany

resources:

requests:

storage: 200Gi

storageClassName: repd-west1-a-b-c

EOF

List the available persistentvolumeclaims with:

kubectl get persistentvolumeclaims

**Example Output:**

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

data-wp-repd-mariadb-0 Bound pvc-8a10ed04-56ca-11e9-a020-42010a8a003d 8Gi ROX standard 21m

wp-repd-wordpress Bound pvc-ad5ddb0b-56ca-11e9-9af5-42010a8a0047 200Gi ROX repd-west1-a-b-c 20m

**Deploy WordPress**

Now that we have our StorageClass configured, Kubernetes automatically attaches the persistent disk to an appropriate node in one of the available zones.

1. Deploy the WordPress chart that is configured to use the StorageClass that you created earlier:

helm install --name wp-repd \

--set smtpHost= --set smtpPort= --set smtpUser= \

--set smtpPassword= --set smtpUsername= --set smtpProtocol= \

--set persistence.storageClass=repd-west1-a-b-c \

--set persistence.existingClaim=wp-repd-wordpress \

--set persistence.accessMode=ReadOnlyMany \

stable/wordpress

1. List out available wordpress pods:

kubectl get pods

**Example Output:**

NAME READY STATUS RESTARTS AGE

wp-repd-mariadb-79444cd49b-lx8jq 1/1 Running 0 35m

wp-repd-wordpress-7654c85b66-gz6nd 1/1 Running 0 35m

1. Run the following command which waits for the service load balancer's external IP address to be created:

while [[ -z $SERVICE\_IP ]]; do SERVICE\_IP=$(kubectl get svc wp-repd-wordpress -o jsonpath='{.status.loadBalancer.ingress[].ip}'); echo "Waiting for service external IP..."; sleep 2; done; echo http://$SERVICE\_IP/admin

1. Verify that the persistent disk was created:

while [[ -z $PV ]]; do PV=$(kubectl get pvc wp-repd-wordpress -o jsonpath='{.spec.volumeName}'); echo "Waiting for PV..."; sleep 2; done

kubectl describe pv $PV

1. Get the URL for the WordPress admin page :

echo http://$SERVICE\_IP/admin

1. Click on the link to open WordPress in a new tab in your browser.
2. Back in Cloud Shell, get a username and password so you can log in to the app:

cat - <<EOF

Username: user

Password: $(kubectl get secret --namespace default wp-repd-wordpress -o jsonpath="{.data.wordpress-password}" | base64 --decode)

EOF

1. Go to the WordPress tab and log in with the username and password that was returned.

You now have a working deployment of WordPress that is backed by regional persistent disks in three zones.

**Simulating a zone failure**

Next you will simulate a zone failure and watch Kubernetes move your workload to the other zone and attach the regional disk to the new node.

1. Obtain the current node of the WordPress pod:

NODE=$(kubectl get pods -l app=wp-repd-wordpress -o jsonpath='{.items..spec.nodeName}')

ZONE=$(kubectl get node $NODE -o jsonpath="{.metadata.labels['failure-domain\.beta\.kubernetes\.io/zone']}")

IG=$(gcloud compute instance-groups list --filter="name~gke-repd-default-pool zone:(${ZONE})" --format='value(name)')

echo "Pod is currently on node ${NODE}"

echo "Instance group to delete: ${IG} for zone: ${ZONE}"

**Example Output:**

Pod is currently on node gke-repd-default-pool-b8cf37cd-bc5q

Instance group to delete: gke-repd-default-pool-b8cf37cd-grp for zone: us-west1-c

You can also verify it with:

kubectl get pods -l app=wp-repd-wordpress -o wide

**Example Output:**

NAME READY STATUS RESTARTS AGE IP NODE

wp-repd-wordpress-7654c85b66-gz6nd 1/1 Running 0 1h 10.20.0.11 gke-repd-default-pool-b8cf37cd-bc5q

Take note of Node column. You are going to delete this node to simulate the zone failure.

1. Now run the following to delete the instance group for the node where the WordPress pod is running, click **Y** to continue deleting:

gcloud compute instance-groups managed delete ${IG} --zone ${ZONE}

Kubernetes is now detecting the failure and migrates the pod to a node in another zone.

1. Verify that both the WordPress pod and the persistent volume migrated to the node that is in the other zone:

kubectl get pods -l app=wp-repd-wordpress -o wide

**Example Output:**

NAME READY STATUS RESTARTS AGE IP NODE

wp-repd-wordpress-7654c85b66-xqb78 1/1 Running 0 1m 10.20.1.14 gke-repd-default-pool-9da1b683-h70h

Make sure the node that is displayed is different from the node in the previous step.

1. Once the new service has a Running status, open the WordPress admin page in your browser from the link displayed in the command output:

echo http://$SERVICE\_IP/admin

You have attached a regional persistent disk to a node that is in a different zone.

**Congratulations!**

This concludes this hands-on lab with Regional Kubernetes cluster.