**Deploying to Google Kubernetes Engine with Helm**

Toggle Lab Panel



**Deploying to Google Kubernetes Engine with Helm**

1 hour Free

Rate Lab

**Overview**

In this lab, you use a Helm chart to deploy a full Kubernetes solution.

**Objectives**

In this lab, you learn how to perform the following tasks:

* Use Helm charts to deploy a Kubernetes solution
* Use kubectl commands to validate that solution resources have been deployed
* Test the functionality of a Redis solution running on Kubernetes Engine

**Task 0. Lab Setup**

**Access Qwiklabs**

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

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, img/time.pngand make sure you can finish in that time block.

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

1. When ready, click img/start_lab.png.
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

**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. Click **Continue**. 

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) .

**Task 1. Use Helm charts to deploy a solution on Kubernetes Engine**

Typical Kubernetes systems consist of many different resources. Though you could maintain configuration files for each of these resources, managing versions of different resources and compatibility among those versions can be difficult. Helm is a popular tool used with Kubernetes for packaging resources.

In this task, you use Helm charts to deploy Redis, an open source in memory key-value database service, on a Kubernetes Engine cluster.

**Connect to the lab GKE cluster**

1. In Cloud Shell, type the following command to set the environment variable for the zone and cluster name.

export my\_zone=us-central1-a

export my\_cluster=standard-cluster-1

1. Configure kubectl tab completion in Cloud Shell.

source <(kubectl completion bash)

1. In Cloud Shell, configure access to your cluster for the kubectl command-line tool, using the following command:

gcloud container clusters get-credentials $my\_cluster --zone $my\_zone

**Download the Helm binary, deploy a Helm Chart**

You download helm, configure admin and service accounts and then initialize Helm for your Kubernetes cluster.

1. Execute the following command to download the Helm installation shell script.

curl -LO https://git.io/get\_helm.sh

**Note:** You can find information on this, and other methods, for installing Helm on the Helm installation page, <https://helm.sh/docs/using_helm/#installing-helm>.

1. Make the installation script executable.

chmod 700 get\_helm.sh

1. Execute the Helm installation script.

./get\_helm.sh

1. Ensure your user account has the cluster-admin role in your cluster.

kubectl create clusterrolebinding user-admin-binding \

--clusterrole=cluster-admin \

--user=$(gcloud config get-value account)

1. Create a Kubernetes service account called Tiller. this will be used by Tiller, the server side of Helm, for deploying Helm charts.

kubectl create serviceaccount tiller --namespace kube-system

1. Grant the Tiller service account the cluster-admin role in your cluster:

kubectl create clusterrolebinding tiller-admin-binding \

--clusterrole=cluster-admin \

--serviceaccount=kube-system:tiller

1. Execute the following commands to initialize Helm using the Tiller service account.

helm init --service-account=tiller

Click *Check my progress* to verify the objective. Create a tiller service account and initialize the Helm

1. Execute the following commands to update the Helm repositories.

helm repo update

1. In Cloud Shell, execute the following command to verify the Helm installation and configuration:

helm version

The output should look like the example, although you will see a newer version.

**Output (do not copy)**

Client: &version...{SemVer:"v2.16.6", GitCommit:"be3...State:"clean"}

Server: &version...{SemVer:"v2.16.6", GitCommit:"be3...State:"clean"}

1. Execute the following command to deploy a set of resources to create a Redis service on the active context cluster:

helm install stable/redis

A Helm chart is a package of resource configuration files, along with configurable parameters. This single command deployed a collection of resources.

**Task 2. Validate and test a solution deployed using Helm**

In this task you use kubectl to validate that Helm has deployed the resource components for the Redis application and you then use Redis to store and retrieve sample data to confirm that it has been successfully deployed.

**Use kubectl to inspect the Kubernetes resources deployed via Helm**

You use kubectl to list the Kubernetes Services, StatefulSets, ConfigMaps and Secrets that were deployed using Helm.

1. A Kubernetes Service defines a set of Pods and a stable endpoint by which network traffic can access them. In Cloud Shell, execute the following command to view Services that were deployed through the Helm chart:

kubectl get services

The output should look like the example.

**Output (do not copy)**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

zinc-...redis-master ClusterIP 10.11.244.2 <none> 6379/TCP 26s

zinc-...redis-slave ClusterIP 10.11.240.101 <none> 6379/TCP 26s

kubernetes ClusterIP 10.11.240.1 <none> 443/TCP 10m

1. A Kubernetes StatefulSet manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods. In Cloud Shell, execute the following commands to view a StatefulSet that was deployed through the Helm chart:

kubectl get statefulsets

The output should look like the example.

**Output (do not copy)**

NAME Ready AGE

zinc-leopard-redis-master 1/1 1m

zinc-leopard-redis-slave 2/2 1m

1. A Kubernetes ConfigMap lets you store and manage configuration artifacts, so that they are decoupled from container-image content. In Cloud Shell, execute the following commands to view ConfigMaps that were deployed through the Helm chart:

kubectl get configmaps

The output should look like the example.

**Output (do not copy)**

NAME DATA AGE

lumpy-mandrill-redis 3 2m

lumpy-mandrill-redis-health 3 2m

1. A Kubernetes Secret, like a ConfigMap, lets you store and manage configuration artifacts, but it's specially intended for sensitive information such as passwords and authorization keys. In Cloud Shell, execute the following commands to view some of the Secret that was deployed through the Helm chart:

kubectl get secrets

The output should look like the example.

**Output (do not copy)**

NAME TYPE DATA AGE

default-token-vl8bh kubernetes.io/service-account-token 3 12m

lumpy-mandrill-redis Opaque 1 3m

Click *Check my progress* to verify the objective. Install redis and check resources- services, secrets,deployments and configmaps

1. You can inspect the Helm chart directly using the following command:

helm inspect stable/redis

1. If you want to see the templates that the Helm chart deploys you can use the following command:

helm install stable/redis --dry-run --debug

**Test Redis functionality**

You store and retrieve values in the new Redis deployment running in your Kubernetes Engine cluster.

1. Execute the following command in the Cloud Shell to store the service ip-address for the Redis cluster in an environment variable.

export REDIS\_IP=$(kubectl get services -l app=redis -o json | jq -r '.items[].spec | select(.selector.role=="master")' | jq -r '.clusterIP')

1. Retrieve the Redis password and store it in an environment variable.

export REDIS\_PW=$(kubectl get secret -l app=redis -o jsonpath="{.items[0].data.redis-password}" | base64 --decode)

1. Display the Redis cluster address and password.

echo Redis Cluster Address : $REDIS\_IP

echo Redis auth password : $REDIS\_PW

1. Open an interactive shell to a temporary Pod running inside your GKE cluster, passing in the Redis cluster address and password as environment variables.

kubectl run redis-test --rm --tty -i --restart='Never' \

--env REDIS\_PW=$REDIS\_PW \

--env REDIS\_IP=$REDIS\_IP \

--image docker.io/bitnami/redis:4.0.12 -- bash

1. Enter the following command in the interactive shell to connect to the Redis cluster using the redis-cli.

redis-cli -h $REDIS\_IP -a $REDIS\_PW

1. In the redis cli set a key value.

set mykey this\_amazing\_value

This will display OK if successful.

1. In the redis cli retrieve the key value.

get mykey

This will return the value you stored indicating that the Redis cluster can successfully store and retrieve data.

**End your lab**