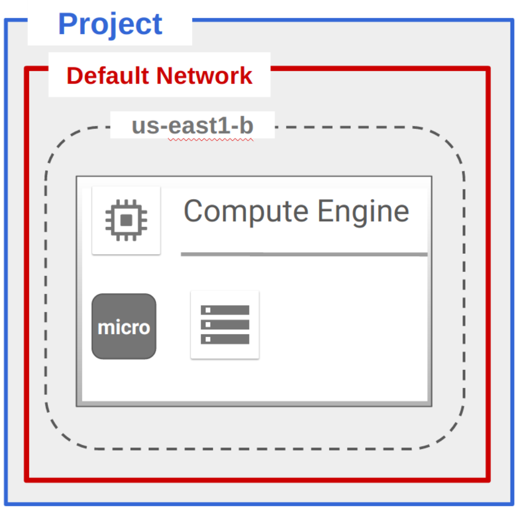
Lab 11

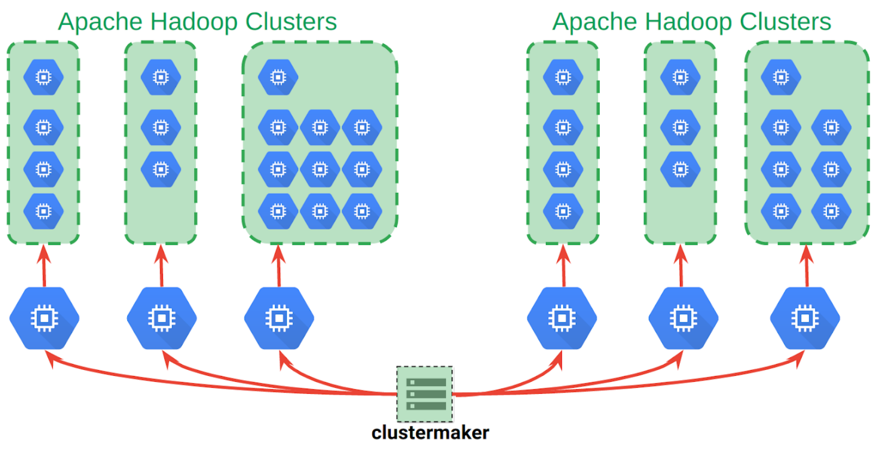
Google Cloud Platform API Infrastructure Automation

Overview

In this lab, you build your own global IaaS Hadoop cluster deployment service based on open-source software. The service gives you a reliable and secure way to deploy any size Hadoop Cluster in any GCP region in minutes.
To accomplish this, you create an IAM service account with the role of Project Editor. You authorize and initialize the Google Cloud SDK on a VM. Then you "bake" that authority into a reusable snapshot that can reconstitute the "clustermaker" VM in any GCP region.
The clustermaker VM is a micro type machine with just enough capacity to do the work needed to create and deploy Hadoop clusters using the Google Cloud SDK.



The clustermaker VM must be able to use the Google Cloud API. To do this, you create an IAM service account with a Project Editor role, download the private key, and copy it to the VM. Then you go through an authorization and initialization process on the VM. You modify the VM by installing Git, and then use Git to install an open-source tool called bdutil (Big Data Utility). You modify the configuration to select a machine type for the workers in the cluster. bdutil uses a GCS bucket to stage and install the Hadoop software, and Hadoop is configured to use GCS rather than HDFS for its file system.



After you verify that the Hadoop cluster created by clustermaker is working, you create a snapshot of the boot persistent disk using the snapshot service.
The snapshot can be used to recreate the persistent disk in any region, and the persistent disk can be used to launch a VM in that region. You test this by creating a new clustermaker VM in a different region and using it to launch another Hadoop cluster.

Objectives

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

- Create an IAM service account
- Create a VM
- Authorize it to use Google Cloud API using the Service Account
- Install open-source software on the VM, configure the software, and test it by deploying a Hadoop cluster
- Create a snapshot of the boot disk with the Service Account authority "baked in"
- Recreate the clustermaker VM in a different region and test it by deploying another Hadoop cluster in the new region

Task 1: Create and authorize a VM to use the Cloud SDK

Create a service account

- In the GCP Console, on the **Navigation menu** (☰), click **IAM & admin** > **Service accounts**.
- Click **Create service account**.
- Specify the following:

Property	**Value** (type value or select option as specified)
Service account name	**clustermaker**
Role	**Project** > **Editor**

- Click **Furnish a new private key**.
- For **Key type**, click **JSON**.
- Click **Create**. A JSON key file will download. You will need to find this key file and upload it in into the VM in a later step.
- Rename the JSON key file on your local machine **credentials.json**
- Click **Close**.

Create a VM instance

You create the VM with a persistent disk that remains when the VM is deleted. This disk is used to create the pattern you use later to create a cluster provisioning VM.

- On the **Navigation menu**, click **Compute Engine** > **VM instances**.
- Click **Create**.
- Specify the following, and leave the remaining settings as their defaults:

Property	**Value** (type value or select option as specified)
Name	**clustermaker**
Zone	**us-east1-b**
Machine type	**micro (1 shared vCPU)**

- Click **Management, disks, networking, SSH keys**.
- Click **Disks**.
- Disable **Delete boot disk when instance is deleted**.
- Click **Create**.

Authorize the VM to use the Cloud SDK

This VM will use API calls to create the Hadoop cluster, so it must have the Google Cloud SDK installed and it must be authorized to use the API.

The SDK is required for the gcloud command line tool, so you can verify that the Google Cloud SDK is installed if the gcloud tool is installed. The -v (version) option lists the Google Cloud SDK version.

- For **clustermaker**, click **SSH** to launch a terminal and connect.
- To check whether the SDK is installed on this VM, run the following:

```
gcloud -v
```

Are the gcloud tool and the SDK installed on this standard GCP base image?

- To see whether the permissions are set to make calls to Google Cloud, run the following:

```
gcloud compute zones list
```

What was the result?

- To upload **credentials.json** through the SSH VM terminal, click on the gear icon in the upper-right corner, and then click **Upload file**.
- Select **credentials.json** and upload it.
- Click **Close** in the File Transfer window.
- To authorize the VM with the credentials you just uploaded, run the following:

```
gcloud auth activate-service-account --key-file credentials.json
```

The image you are using has the Google Cloud SDK pre-installed; therefore, you don't need to initialize the Google Cloud SDK. If you are attempting this lab in a different environment, make sure you have followed these procedures regarding installing the Google Cloud SDK: https://cloud.google.com/sdk/downloads

- To verify that the VM is now authorized to use Google Cloud Platform API, run the following:

```
gcloud compute zones list
```

This time the command should succeed.

At this point you will delete the credentials.json file from the VM. If you need the credentials again, you can upload the file that was downloaded to your computer.

- To delete the credentials file, run the following:

```
rm credentials.json
```

Task 2: Customize the VM

You customize the VM and install necessary software. The final step is to create the Hadoop cluster using the customized VM to verify that it is working.

You use an open-source cluster automation tool called bdutil. The code is open source and is hosted in a Git repository. To copy that code to the VM, you first need to install Git.

Install Git and clone the source code.

- In the **clustermaker** SSH terminal, make sure the package index is up to date:

```
sudo apt-get -qq update
```

- To install Git, run the following:

```
sudo apt-get install -y -qq git
```

- To download the code to the VM, run the following:

```
git clone \
https://github.com/GoogleCloudPlatform/bdutil
```

- Navigate to the bdutil directory:

```
cd bdutil
```

Modify the configuration

- Open bdutil_env.sh with nano:

```
nano bdutil_env.sh
```

- Locate the line GCE_MACHINE_TYPE=. The machine type should be set to: n1-standard-4. That will create worker nodes with 4 vCPUs each. That's a good size for a Hadoop cluster. But for demonstration purposes you will use a smaller machine.
- Edit the file by changing the value of GCE_MACHINE_TYPE to n1-standard-1.
- To save the file and exit nano, press **Ctrl+O**, **ENTER**, **Ctrl+X**.
- Exit the SSH terminal.

Task 3: Test the clustermaker

You need a Google Cloud Storage bucket. The application stages intermediate files during cluster VM creation and configuration. It also serves as the base for the file system used by Hadoop. Hadoop can use the Cloud Storage file system instead of the Hadoop Distributed File System (HDFS). (HDFS was derived from an earlier version of Cloud Storage).

Create a Cloud Storage bucket

5 In the GCP Console, on the **Navigation menu** (☰), click **Storage** > **Browser**.
6 Click **Create Bucket**.
7 Specify the following:

Property	**Value** (type value or select option as specified)
Name	Enter a globally unique name (common practice is to use part of the Project ID - {part of project id}-hadoop-storage) Do not exceed 64 characters.
Default storage class	**Multi-regional**
Multi-Regional location	**United States**

- Click **Create.**
- Make a note of the bucket name; it will be referred to as **[YOUR_BUCKET_NAME]**.

Create a Hadoop Cluster

- On the **Navigation menu**, click **Compute Engine** > **VM instances**.
- For **clustermaker**, click **SSH** to launch a terminal and connect.
- Create an environment variable for the bucket you just created:

```
export BUCKET_NAME=<Enter YOUR_BUCKET_NAME here>
```

- Choose a unique ID for your hadoop cluster. For example, use the local-part of your email address (the part before the @symbol). Note: The unique ID cannot be greater than 64 characters and cannot contain any special characters. This is a JVM limitation, and the script will fail if the name is too long. This unique ID will be referred to as [YOUR_UNIQUE_ID].
- Store the unique ID for your Hadoop cluster in an environment variable:

```
export UNIQUE_ID=<Enter YOUR_UNIQUE_ID here>
```

- Navigate to the bdutil folder:

```
cd bdutil
```

- Create the first Hadoop cluster with your unique ID:

```
./bdutil -b $BUCKET_NAME \
-z us-east1-b \
-n 2 \
-P $UNIQUE_ID \
deploy
```

- When prompted, type **y**, then press **ENTER**.

Wait for completion and persist environment variables

- Return to the **VM instances** page in the GCP Console. Notice the VM instances being created by the clustermaker VM. Each instance is named after your unique ID. The master VM has the letter -m appended, and the workers have the letter -w appended.

Wait until bdutil completes in the terminal before proceeding to the next step.

- Open the ~/.bashrc file using nano:

```
nano ~/.bashrc
```

- To persist your environment variables, copy and then paste the following at the end of the file (these are the same commands you ran earlier to store the environment variables locally):

```
export BUCKET_NAME=<Enter YOUR_BUCKET_NAME here>
export UNIQUE_ID=<Enter YOUR_UNIQUE_ID here>
```

- To save the file and exit nano, press **Ctrl+O**, **ENTER**, **Ctrl+X**.

Task 4: Verify the Hadoop cluster

You now verify that the cluster is working. Although you could run a MapReduce job, in the interest of time, just verify that the Hadoop file system is functioning correctly.

- In the **clustermaker** SSH terminal, SSH into the Hadoop Master VM by running the following:

```
gcloud compute ssh --zone=us-east1-b $UNIQUE_ID-m
```

  - To create a Hadoop file system directory, run the following:

```
hadoop fs -mkdir testsetup
```

  - To download a file to use for the test, run the following:

```
curl \
http://hadoop.apache.org/docs/current/\
hadoop-project-dist/hadoop-common/\
ClusterSetup.html > setup.html
```

This is a Hadoop cluster setup document in HTML format from the Apache.org website.

  - To copy the file into the directory you created, run the following:

```
hadoop fs -copyFromLocal setup.html testsetup
```

  - To verify that the file is in Hadoop, run the following:

```
hadoop fs -ls testsetup
```

  - To dump the contents of the file, run the following:

```
hadoop fs -cat testsetup/setup.html
```

Task 5: Take the custom solution global

Exit the terminals

  - In the SSH terminal, type exit to exit the Hadoop Master SSH terminal session, and then type exit again to exit the clustermaker SSH terminal session. This closes the terminal window.

Delete all the VMs

Recall that when you created the clustermaker VM, you specified that the disk should not be deleted when the VM is deleted.

- Return to the **VM instances** page in the GCP Console.
- Select all instances, and then click **Delete**.
- In the confirmation dialog, click **Delete**.

Create a persistent disk snapshot

- On the **Navigation menu**, click **Compute Engine** > **Disks**. The **clustermaker** disk should still exist.
- To make a snapshot from the disk, click on the three vertical dots at the end of the **clustermaker** row, and click **Create snapshot**.
- For **Name**, type **clustermaker**.
- Click **Create**. Wait until the snapshot has been created before proceeding to the next task.

Task 6: Test your global deployment tool

Reconstitute the disk from the snapshot

- In the left pane, click **Disks**.
- Click **Create disk**.
- Specify the following:

Property	**Value** (type value or select option as specified)
Name	**newdisk1**
Zone	**us-central1-c**
Source type	**Snapshot**
Source snapshot	**clustermaker**

Notice that the new disk is in a completely different region and zone than the original.

- Click **Create**. Wait until the disk has been created before proceeding to the next step.

Start a VM from the disk

Create a VM from the new disk.

- At the end of the row for **newdisk1**, click the three vertical dots, and then click **Create instance**.
- For **Name**, type **new-clustermaker**.
- Click **Create**.

Launch a new Hadoop Cluster in a new region

When the VM is operational, connect via SSH to it and launch a 2-worker cluster in the new region.

- For **new-clustermaker**, click **SSH** to launch a terminal and connect.
- To create the new Hadoop cluster, run the following:

```
cd bdutil

./bdutil -b $BUCKET_NAME \
-z us-central1-c \
-n 2 \
-P $UNIQUE_ID \
deploy
```

- When prompted, type **y**, then press **ENTER**.

In the GCP Console, view the VM instances to see them being created by the new-clustermaker VM.

In practice, you could now delete the new-clustermaker VM and the newdisk1 disk, until you needed to deploy another cluster.

Congratulations! You created a custom snapshot from which to launch Hadoop clusters of any size in any region as needed.

Task 7: Review

What you learned:

- How to create an IAM service account.
- How to create a VM.
- How to authorize a VM to use Google Cloud API using the service account, which is a useful skill for creating automation tools.
- How to Install open-source software on the VM.
- How to configure and test the VM by deploying a Hadoop cluster.
- How to create a global solution by generating a snapshot of the boot disk with the service account authority "baked in."
- How to recreate the clustermaker VM in a different region and test it by deploying another Hadoop cluster in the new region.

There are many ways to provide big data processing in GCP, including BigQuery, Cloud Dataproc, and Cloud Dataflow. You can also use third-party installation tools to deploy Hadoop clusters. In this lab, you learned how to create your own IaaS solution where you have access to and control over all the source code.

During this lab, you learned many IaaS skills that can be leveraged to automate activities through the Google Cloud SDK. This is important for Site Reliability Engineering (SRE). You can build on what you learned here by studying the bdutil source code to see how it works with the Cloud API.

Cleanup

2 In the **Cloud Platform Console**, sign out of the Google account.

3 Close the browser tab.

Last Updated: 2018-06-25

End your lab

Lab 12
Deployment Manager
Overview

Deployment Manager is an infrastructure deployment service that automates the creation and management of Google Cloud Platform (GCP) resources for you. Write flexible template and configuration files and use them to create deployments that have a variety of GCP services, such as Cloud Storage, Compute Engine, and Cloud SQL, configured to work together.

Objectives

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

- Download and review a set of Deployment Manager templates
- Use the templates to deploy two networks and a VM instance

Task 1: Preparation

Create a directory for the Deployment Manager lab

- In the GCP Console, click **Activate Google Cloud Shell** (). If prompted, click **Start Cloud Shell**.
- Run the following commands:

```
mkdir deplab
cd deplab
```

Download and unzip the Deployment Manager configuration files

- To download the files, run the following commands:

```
gsutil cp gs://cloud-training/archinfra/dm* .
```

- To unzip the archive, run the following commands:

```
unzip dm-net-v01.zip
unzip dm-class.zip
```

Verify that the Deployment Manager API is enabled

- In the GCP Console, on the **Navigation menu** (), click **APIs & services** > **Library**.
- In the search bar, type **Deployment Manager** and click the result for **Google Cloud Deployment Manager V2 API**.
- Verify that the API is enabled (a green checkmark is displayed).

Task 2: Examine the configuration files

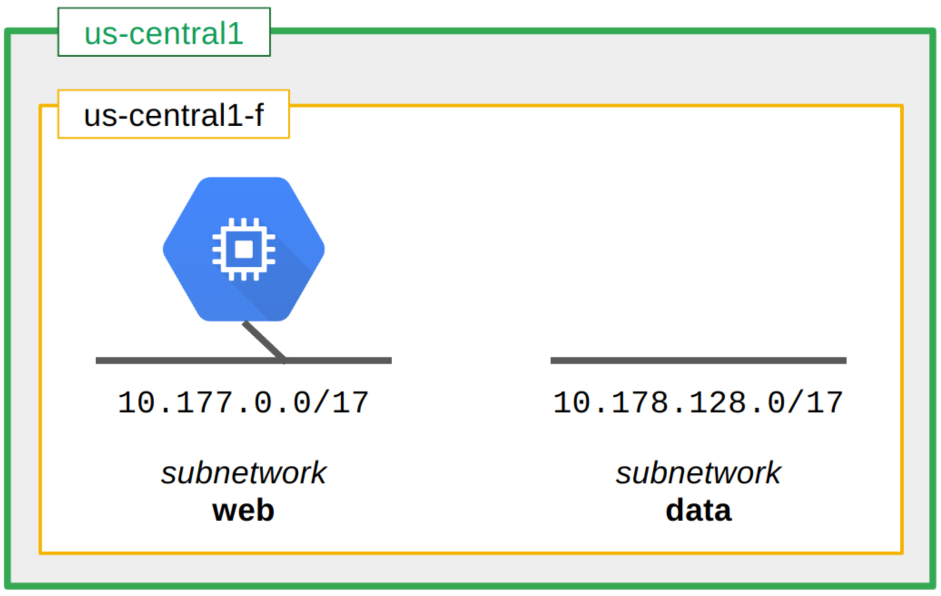
Deployment Manager uses a system of templates to consistently deploy GCP resources. You create files to instruct Deployment Manager.
The files are of three types:

- A configuration file, which drives the deployment; this is a YAML file.
- Template files which implement specific resources; these are written in either Jinja 2.7.3 or Python 2.7.
- Schema files, which are a set of rules that a configuration file must meet to be allowed to use a particular template; you won't use schema files in this lab.

View the *.yaml file

- In Cloud Shell, hover over the pencil icon and click **Launch code editor**.

- Expand the **deplab** folder and click on the **net-config.yaml** file.
- Examine this file:
- The imports: section identifies the *.jinja templates that will be used.
- The resources: section defines GCP resources to be created.
- The first resource to be created is an object called 'network' that is based on the network.jinja template. It will pass in the properties which will create a couple of subnets named 'web' and 'data' in region us-central1.
- The second resource to be created is an object called 'web-instance' that is based on the instance.jinja template. This resource will pass properties to create an n1-standard-1 VM in the us-central1-f zone on the 'web' subnetwork.
- The resources that Deployment Manager will create look like this:



To learn more about how to create a basic configuration file, see the documentation: https://cloud.google.com/deployment-manager/docs/configuration/create-basic-configuration

Customize the net-config.yaml file

Using the code editor window, make the following changes:

- Change the region to a **region of your choice**.
- Change the zone to a **zone of your choice** within your selected region.
- Change the web subnetwork range to **10.5.3.0/29**
- Change the data subnetwork range to **10.29.60.0/24**

For current region and zone names, see: https://cloud.google.com/compute/docs/regions-zones/

Note: All changes are saved automatically.

View the *.jinja files

- In the code editor window, open the network.jinja file.
- Examine the file to explore the configuration of the network and subnetworks.
- In the code editor window, open the instance.jinja file.
- Examine the file to explore the configuration of the VM instance.

To learn more about how to create a template file, see the documentation: https://cloud.google.com/deployment-manager/docs/step-by-step-guide/create-a-template

Task 3: Deploy the configuration

- In Cloud Shell, run the following command:

```
gcloud deployment-manager deployments create gcpinfra --config=net-config.yaml
```

You should see something like this **(do not copy; this is example output)**:

```
NAME               TYPE                  STATE     ERRORS  INTENT
data               compute.v1.subnetwork  COMPLETED  []
gcpinfra-instance  compute.v1.instance    COMPLETED  []
gcpinfra-network   compute.v1.network     COMPLETED  []
web                compute.v1.subnetwork  COMPLETED  []
```

Note: If something goes wrong with the deployment, you must delete any artifacts that were created and delete the Deployment Manager template before you can try again.
Because this creates a custom network, you must delete the subnetworks before you delete the network, and only then can you delete the Deployment Manager template.

Task 4: Explore resources in the Console

- In the GCP Console, on the **Navigation menu**, click **VPC network** > **VPC networks**. Verify that you see the custom **gcpinfra-network** network with two subnets data and web in your chosen region.
- On the **Navigation menu**, click **Compute Engine** > **VM instances**.Verify that the **gcpinfra-instance** VM is running in your chosen zone.
- On the **Navigation menu**, click **Deployment Manager**.
- Click **gcpinfra**.
- Examine the deployment of the network and instance using the *.jinja templates.

Task 5: About Deployment Manager

The concept behind Deployment Manager is "infrastructure as code." You can treat the templates like code, placing them under version control in a repository, and so forth.
In this lab you used commands to run Deployment Manager. However, the same functionality is also available through the API for even greater leverage.

- In Cloud Shell, to list the types of GCP resources that you can control using Deployment Manager, run the following:

```
gcloud deployment-manager types list
```

A huge configuration file can be difficult to manage. Templates are a way to break down configurations into composable units that can be separately updated and can be reused.
Templates are included in the *.yaml configuration using import:.
Here are some of the benefits of Deployment Manager templates:

- Composability: Easily manage and maintain definitions.
- Reusability: Easily reuse definitions across deployments.
- Maintainability: Maintain consistent definitions in one place.

The Deployment Manager configuration and template files use two kinds of variables: template variables and environment variables.
Template variables are abstract properties that allow you to declare the value to be passed to the template in the *.yaml configuration file. You can change the value for each deployment in the *.yaml file without having to make changes to the underlying templates.
Example: Passing zone as a template variable allows you to start a test deployment in a different zone from the production deployment.
Environment variables allow you to reuse templates in different projects and deployments. Instead of representing properties of resources, they represent more global properties such as a Project ID or the name of the deployment. Example: You want to start the same deployment in two projects: proj-east and proj-west. You could declare environment variables for the two Project IDs and then use these in the Deployment Manager configuration files.
For more information, see: https://cloud.google.com/deployment-manager/docs/

Task 6: Review

In this lab you customized Deployment Manager templates to deploy a network with two subnetworks and a VM.

Cleanup

- In the **Cloud Platform Console**, sign out of the Google account.
- Close the browser tab.

Lab 16
Kubernetes Load Balancing
Overview
Google Kubernetes Engine includes integrated support for (L4) network load balancing. To engage network load balancing, all you need to do is include the field type: LoadBalancer in your service configuration file. Kubernetes Engine will set up and connect the network load balancer to your service.
However, if you want more advanced (L7) load balancing features, including cross-region load balancing or content-based load balancing, you need to integrate your service with the HTTP/HTTPS load balancer provided by Compute Engine.
In the first part of this lab, you configure a cluster with network load balancing. In the second part of this lab, you configure a replicated nginx service. Then you use a Kubernetes extension, called ingress, to expose the service behind an HTTPS load balancer.
Objectives
In this lab, you learn how to perform the following tasks:
- Create a Kubernetes cluster using Google Kubernetes Engine with the built-in network load balancer
- Deploy nginx into the cluster and verify that the application is working
- Undeploy the application
- Re-deploy the cluster using ingress to connect it to a Google Compute Engine HTTPS load balancer.
- Redeploy and test.

Important: What is happening during this time?
Your lab is spinning up GCP resources for you behind the scenes, including an account, a project, resources within the project, and permission for you to control the resources you will need to run the lab. This means that instead of spending time manually setting up a project and building resources from scratch as part of your lab, you can begin learning more quickly.
Find Your Lab's GCP Username and Password
To access the resources and console for this lab, locate the Connection Details panel in Qwiklabs. Here you will find the account ID and password for the account you will use to log in to the Google Cloud Platform:

Task 1: Preparation
Set some environment variables and gcloud default values. This practice improves consistency and reduced the chances of typing errors. You set your PROJECT_ID and choose a region and zone for this lab. You set environment variables for these values, and they are referred to as [MY_REGION] and [MY_ZONE]. Select a region and zone that are geographically close to you.
For more information about regions and zones, see:
https://cloud.google.com/compute/docs/regions-zones/regions-zones
Set gcloud default values
- In the GCP Console, click **Activate Google Cloud Shell** (>_). If prompted, click **Start Cloud Shell**.
- Set some environment variables, replacing YOUR_REGION and YOUR_ZONE with your selections:

```
export MY_REGION=[YOUR_REGION]
export MY_ZONE=[YOUR_ZONE]

export CLUSTER_NAME=httploadbalancer
```

- Run the following:

```
gcloud config set project $DEVSHELL_PROJECT_ID

gcloud config set compute/region $MY_REGION

gcloud config set compute/zone $MY_ZONE
```

- To confirm that the project, zone, and region have been set, run the following:

```
gcloud config list
```

Task 2: Create a Kubernetes cluster for network load balancing

- To create a Kubernetes cluster using Kubernetes Engine, run the following:

```
gcloud container clusters create networklb --num-nodes 3
```

Note: the parameter num-nodes 3 tells Kubernetes to start three VMs and configure them as nodes in the cluster.
This takes several minutes to run. It creates Compute Engine instances and configures each instance as a Kubernetes node.
These instances don't include the Kubernetes Master node. In Kubernetes Engine, the Kubernetes Master node is a managed service.

- To see the newly created instances, in the GCP Console, on the **Navigation menu** (☰), click **Compute Engine** > **VM instances**.

Task 3: Deploy nginx in Kubernetes

Deploy nginx into the Kubernetes cluster and expose the service.

- In Cloud Shell, deploy nginx by running the following:

```
kubectl run nginx --image=nginx --replicas=3
```

Result **(do not copy; this is example output)**:

```
deployment "nginx" created
```

This creates a replication controller to spin up 3 pods; each pod runs the nginx container.
Note: The parameter replicas=3 is the action that starts up 3 pods and runs the nginx container on each one.

- To see the status of the deployment, run the following:

```
kubectl get pods -owide
```

Result **(do not copy; this is example output)**:

```
NAME                    READY    STATUS    RESTARTS   AGE      IP         NODE
nginx-4217019353-7rrck   1/1       Running   0          1m       10.X.X.X   gke-networklb-default-
pool-cf87a938-9vqp
nginx-4217019353-bxnlw   1/1       Running   0          1m       10.X.X.X   gke-networklb-default-
pool-cf87a938-z8tr
nginx-4217019353-cnbxg   1/1       Running   0          1m       10.X.X.X   gke-networklb-default-
pool-cf87a938-206q
```

You can see that each nginx pod is now running in a different node (virtual machine).
After all pods have the Running status, you can then expose the nginx cluster as an external service.

- To expose the nginx cluster to allow access to the deployment, run the following:

```
kubectl expose deployment nginx --port=80 --target-port=80 \
--type=LoadBalancer
```

Result **(do not copy; this is example output)**:

```
service "nginx" exposed
```

This command creates a network load balancer to load-balance traffic to the three nginx instances.

Find the network load balancer address

- Run the following:

```
kubectl get service nginx
```

Result **(do not copy; this is example output)**:

```
NAME     CLUSTER-IP     EXTERNAL-IP   PORT(S)       AGE
nginx    10.X.X.X   <pending>    80:30443/TCP   15s
```

It may take several minutes to see the value of EXTERNAL_IP. If you see <pending> as the example above shows, retry every minute or so until the value of EXTERNAL_IP is displayed as in the example below.

Result **(do not copy; this is example output)**:

```
NAME     CLUSTER-IP     EXTERNAL-IP     PORT(S)       AGE
nginx    10.27.251.137   35.192.159.78   80:30443/TCP   1m
```

You can then visit http://EXTERNAL_IP/ to see the server being served through network load balancing.

Task 4: Undeploy nginx

Undeploy nginx before moving on to deploy a full stack application. You will delete the service and then delete the deployment.

For more information, see:

Kubernetes services: https://kubernetes.io/docs/concepts/services-networking/service/

Kubernetes delete: https://kubernetes-v1-4.github.io/docs/user-guide/kubectl/kubectl_delete/

- To delete the service, run the following:

```
kubectl delete service nginx
```

Result **(do not copy; this is example output)**:

```
service "nginx" deleted
```

- To delete the replication controller and delete the pods (all of the nginx instances), run the following:

```
kubectl delete deployment nginx
```

Result **(do not copy; this is example output)**:

```
deployment "nginx" deleted
```

- To delete the cluster, run the following:

```
gcloud container clusters delete networklb
```

- When prompted, type **Y** to continue. Deleting the cluster will take a few minutes.

Result **(do not copy; this is example output)**:

```
Deleting cluster networklb...done.
Deleted [https://container.googleapis.com/v1/projects/qwiklabs-gcp-
3e0eadb8f5747997/zones/us-east1-c/clusters/networklb].
```

Task 5: Create the Kubernetes cluster for HTTP load balancing

- To create the Kubernetes cluster, run the following:

```
gcloud container clusters create $CLUSTER_NAME --zone $MY_ZONE
```

Creating the clusters will take a few minutes.

Result **(do not copy; this is example output)**:

```
Creating cluster httploadbalancer...done.
Created [https://container.googleapis.com/v1/projects/qwiklabs-gcp-
3e0eadb8f5747997/zones/us-east1-c/clusters/httploadbalancer].
kubeconfig entry generated for httploadbalancer.
NAME             ZONE          MASTER_VERSION  MASTER_IP
```

```
MACHINE_TYPE   NODE_VERSION  NUM_NODES  STATUS
httploadbalancer  us-central1-a  1.7.6-gke.1     35.X.X.X  n1-standard-1  1.7.6       3
RUNNING
```

Task 6: Create and expose the services

- To create an instance of the nginx image serving on port 80, run the following:

```
kubectl run nginx --image=nginx --port=80
```

This command runs an instance of the nginx Docker image, serving on port 80. For more information on the nginx Docker image, see: https://hub.docker.com/_/nginx/

8 To create a Kubernetes Engine service that exposes nginx on each Node's IP at a static port (the NodePort), run the following:

```
kubectl expose deployment nginx --target-port=80 --type=NodePort
```

When you create a service of type NodePort with this command, Kubernetes Engine makes your service available on a randomly selected high port number (e.g. 32640) on all the nodes in your cluster.
A ClusterIP service, to which the NodePort service will route, is automatically created. You can contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>. For more information, see:
https://kubernetes.io/docs/concepts/services-networking/service/

Task 7: Create an ingress object

Ingress is an extension to the Kubernetes API that encapsulates a collection of rules for routing external traffic to Kubernetes endpoints. On Kubernetes Engine, ingress is implemented with a Cloud Load Balancing instance.
You need a configuration file that defines an ingress object and configures it to direct traffic to your nginx server.

Create a YAML configuration file

- Use nano to create the configuration file for ingress. Call it: basic-ingress.yaml:

```
nano basic-ingress.yaml
```

- Copy the following and paste it in the .yaml file:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  backend:
    serviceName: nginx
    servicePort: 80
```

- Press **Ctrl+O**, **ENTER** to save the file, and press **Ctrl+X** to exit nano.

The basic ingress configuration file in this lab defines a default backend that directs all traffic to the nginx service on port 80. You can also define rules that direct traffic by host/path to multiple Kubernetes services.
Example **(do not copy; this is an example)**: In the following example, Tomcat is running on 8080 and nginx on 80.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
spec:
  backend:
    serviceName: default-handler
    servicePort: 80
  rules:
```

```
 - host: my.app.com
   http:
     paths:
     - path: /tomcat
       backend:
         serviceName: tomcat
         servicePort: 8080
     - path: /nginx
       backend:
         serviceName: nginx
         servicePort: 80
```

Create the Ingress file, monitor, and verify

- To create the ingress file, run the following:

```
kubectl create -f basic-ingress.yaml
```

Ingress creates the HTTP load balancing resources in Compute Engine and connect them to the deployment. It takes a few minutes for the backend systems to pass health checks and begin serving traffic.

- To monitor the progress, run the following:

```
kubectl get ingress basic-ingress --watch
```

Wait until all three servers are identified, then press **Ctrl+C** to exit.

- To check the service status, run the following:

```
kubectl describe ingress basic-ingress
```

This gives you a list of the HTTP load balancing resources, the backend systems, and their health status.

- To identify the external IP address of the load balancer, run the following:

```
kubectl get ingress basic-ingress
```

Use curl <IP Address> or browse to the address to verify that nginx is being served through the load balancer. If you get a server error, wait a minute and refresh the page.

Task 8: Shut down the cluster

Note: You don't have to shut down the cluster. Qwiklabs will handle that for you. However, if you'd like the experience, here are the steps.

- To delete the ingress object, run the following:

```
kubectl delete -f basic-ingress.yaml
```

- To shut down and delete nginx, run the following:

```
kubectl delete deployment nginx
```

- To delete the cluster, run the following:

```
gcloud container clusters delete $CLUSTER_NAME
```

When prompted, type **Y** to continue. Deleting the cluster will take a few minutes.

Task 9: Review

You deployed a container application using Kubernetes and Google Kubernetes Engine. First, you tested the configuration of your application. After you verified that it was working, you undeployed it and then re-deployed it behind a Compute Engine HTTPS load balancer, using the Ingress extension to Kubernetes.

Cleanup

- In the **Cloud Platform Console**, sign out of the Google account.
- Close the browser tab.

Last Updated: 2018-06-25

End your lab