CHAPTER 4

Google Compute Engine

Google Compute Engine is an infrastructure service provided as part of the Google Cloud Platform. Compute Engine is made up of three major components: virtual machines, persistent disks, and networks. It is available at several Google datacenters worldwide and is provided exclusively on an on-demand basis. This means Compute Engine neither charges any upfront fees for deployment nor locks down the customer. At the same time, Compute Engine provides steep discounts for sustained use. These sustained-use discounts, without any upfront subscription fees, are an industry first; this pricing innovation shows that Google innovates outside of technology as well.

In this chapter, you continue to use the Cloud Platform project that you created earlier using the web-based Google Developers Console. Most of the examples in this chapter use the `gcloud` command-line interface tool.

The objective of this chapter is to introduce you to all the major components of Compute Engine. The chapter includes examples of the various components to make it easier for you to understand them. This chapter is written in a prescriptive manner, meaning it provides step-by-step instructions for how to create most common deployment architectures and solutions using Compute Engine components. Let us begin with an overview.

Virtual Machines

The core component of Compute Engine is the virtual machine (VM). Compute Engine allows you to create clusters of high-performance VMs comprising of thousands of virtual CPU cores. The VMs are available in multiple hardware configurations, making them suitable for different computational workloads. Changing the ratio of CPU cores and main memory per VM creates the hardware variations. This results in three types of VM instances:

- *Standard instances* have 3.75GB of RAM per CPU core.
- *High-memory instances* have 6.5GB of RAM per CPU core.
- *High-CPU instances* have 0.9GB of RAM per CPU core.

The graph in Figure 4-1 shows the difference between these instance types and lists all the types of VMs that can be created as of this writing. The specific instance types may change over time; see `https://cloud.google.com/compute/pricing` for current instance types and prices.

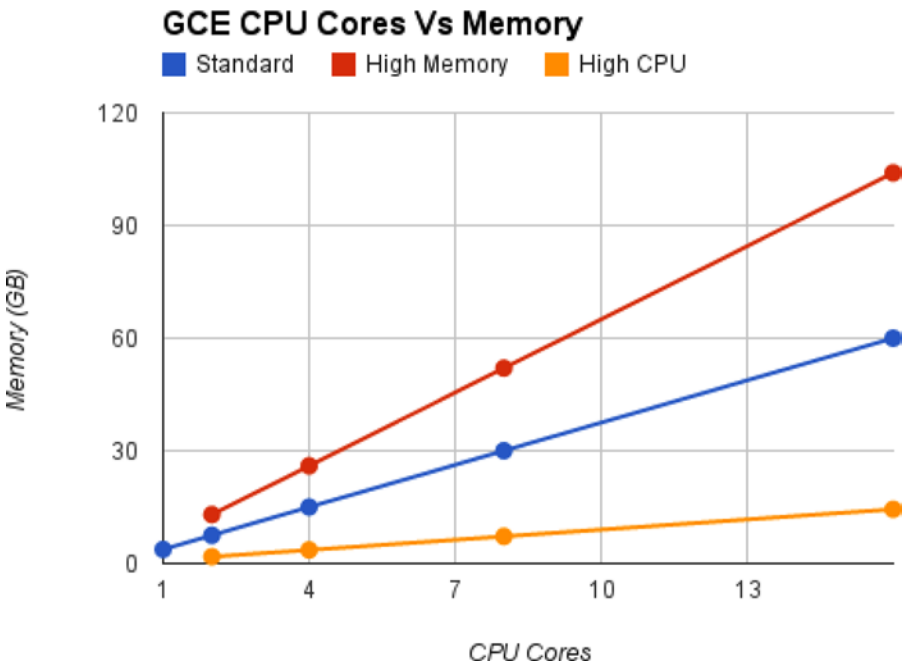

Figure 4-1. *VM classes and instance types*

WHAT IS A VIRTUAL MACHINE?

In computing, a *virtual machine* is a representation of hardware. The representation can be either emulation or simulation based. Emulation is the default method and is more efficient, whereas simulation is typically used for smaller targets. The hardware that is being represented can be either real or hypothetical. For example, semiconductor companies typically emulate hypothetical hardware to test its performance for common workloads before making a physical prototype. The emulation target can be either the local machine (that is, the underlying host hardware) or a remote target. Some enterprises are consolidating servers as VMs, and these VMs run on emulated hardware. Virtualization is one of the core technologies in cloud computing, as well.

Compute Engine, while offering consistent performance, supports several standard and customized operating systems. These include several distributions of Linux, UNIX and Windows. On the hardware front, Compute Engine supports 64-bit versions of x86-based processors only. An *instance* is a per-zone resource, meaning you have to specifically state the Cloud Platform zone in which you want to start the instance.

The action of adding an instance to a project also automatically starts the instance. Although it is not possible to add an instance without starting it, you can shut down an instance without terminating it and removing it from the project.

Persistent Disks

A persistent disk (PD) is network-based block storage that is attached to one or more VMs and used as a storage device. Hosting a PD external to an instance is critical because it helps to retain data in case of hardware failures in the physical host. Google Cloud Platform offers both magnetic and solid-state disk-based PDs. Magnetic disk-based persistent storage is the default, and its performance is sufficient for most applications. You can format the device using any file system you choose. Compute Engine also offers solid-state disks; you can use these for applications that require performance beyond what a magnetic medium can provide.

Each PD can be attached to a single VM instance in read-write mode or can be attached to multiple VM instances in read-only mode. When multiple disks are attached to a single VM instance, some of them can be in read-only mode while others are in read-write modes. A total of 16 persistent disks can be added to a VM, with cumulative storage space of 10TB.

From a system design perspective, persistent disks are hosted in multiple vantage points in a data center. This arrangement eliminates bottlenecks that would be present if the storage was hosted in single network location. Persistent disk data is also replicated for additional redundancy.

In addition to network-hosted persistent block storage, Compute Engine also offers local SSD-based storage for extreme low latency, high IOPS, and high throughput requirements. Local SSD is optimal for temporary data or for hosting high-performance databases. However, because the data is local to the instance, there is no redundancy in case of failure. For database systems, you should configure replication or regular backups. As with PDs, the contents are encrypted-at-rest. If a VM is migrated between physical hosts, the local SSD-based data is automatically migrated as well.

Networks and Firewalls

Networks are the means through which a VM communicates with the external world. This includes communicating with its PD as well, because the PDs are hosted external to the VM instance. The Cloud Platform's philosophy of network bandwidth is to provide dedicated bandwidth on a per-CPU-core basis. This means a VM that has more CPU cores has more network bandwidth compared to another VM with fewer CPU cores. The current bandwidth/core is set at 2 Gbits/sec.

There can be multiple networks in a Cloud Platform project. Each network can host several VM instances, but one VM instance can be attached to only one network. This means the Cloud Platform does not support multi-homed instance configurations. However, it is possible to achieve the same result by using a combination of IP forwarding and Compute Engine network components.

An associated feature of a Compute Engine network is its integrated firewall capability. By default, Compute Engine allows only a small set of ingress (incoming) traffic to reach an instance. The following standard firewalls are attached to Compute Engine's default network (this is the list at the time of this writing; see https://cloud.google.com/compute/docs/networking#firewalls_1 for an updated list):

- `default-allow-internal`: Allows network connections of any protocol and port between any two instances
- `default-allow-ssh`: Allows TCP connections from any source to any instance on the network, over port 22
- `default-allow-icmp`: Allows ICMP traffic from any source to any instance on the network
- `default-allow-rdp`: Allows remote desktop protocol traffic to port 3389

All other ingress traffic must be explicitly allowed through firewall rules. Specifically, no ingress HTTP traffic is allowed to reach an instance. This "deny all by default" approach is very important from a security perspective.

You can customize the scope of Compute Engine network firewall rules. For example, a firewall rule can be applied to all VM instances in a network or to a single VM instance in a network. An example would be to allow incoming SSH from source IP 1.2.3.4 to access an instance called `vm1-nextbigthing-com`. All communication between instances, even if they are within the same network, needs to be explicitly allowed by corresponding firewall rules, whether user-defined or predefined by default.

The Compute Engine network firewall is capable of filtering ingress traffic only. The recommended way to filter egress (outgoing) traffic is to use routes (https://cloud.google.com/compute/docs/networking#routing) to force all outbound traffic to a specific instance or load balancer, and do the filtering there. This lets you configure software VPN connections to other sites. It is also possible to filter egress traffic using other methods in two special cases. First, when the destination is another VM in Compute Engine, the ingress rules at the target can be used to filter ingress traffic. When the OS platform in the source VM is a Linux distribution, you can use host-based firewall filters like `iptables to filter egress traffic`.

Deploying High-Performance Virtual Machines Using Compute Engine

This section describes the steps required to deploy VM instances in Compute Engine. Along the way, you learn about associated Compute Engine components. You use the `gcloud` command-line tool exclusively on this journey.

The following steps are required to realize this goal:

1. Associate the `gcloud` command-line tool with a Google account.
2. Select a Google Cloud Platform project.
3. Create and start your instance.
4. Allow ingress network access.

Let us delve into the technicalities involved in these steps.

Associating the gcloud Command-Line Tool with a Google Account

Before you can use the `gcloud` command-line tool to manage a Google cloud project, you need to associate the `gcloud` tool with a Google account. The following command makes this association:

```
$ gcloud auth login [--no-launch-browser]
```

When you type this command in a terminal window and press the Enter key, your default web browser launches. Depending on whether you have already logged in to a Google account, such as Gmail, you may encounter either one or both of the windows shown in Figure 4-2 and Figure 4-3.

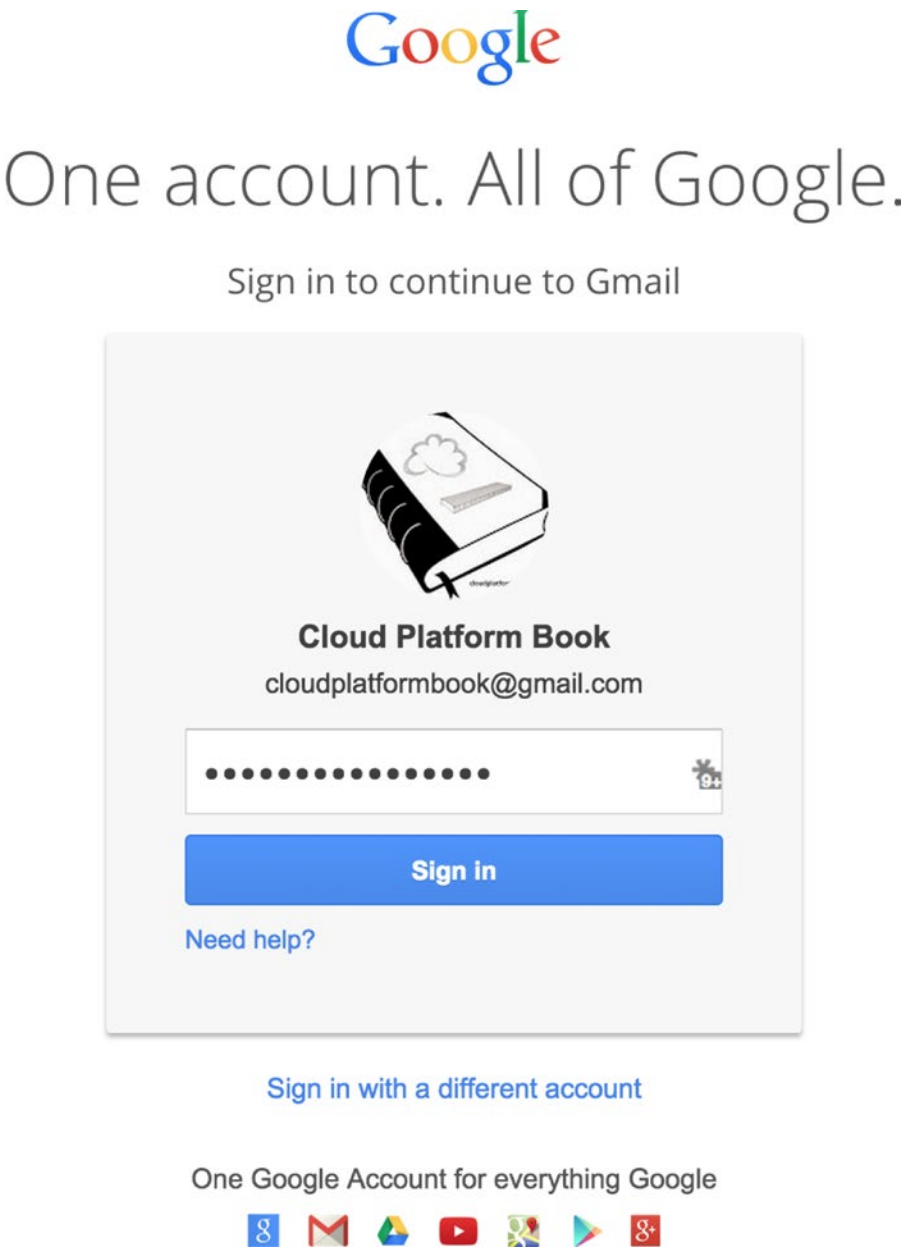

Figure 4-2. *Google login screen if you are not currently logged in*

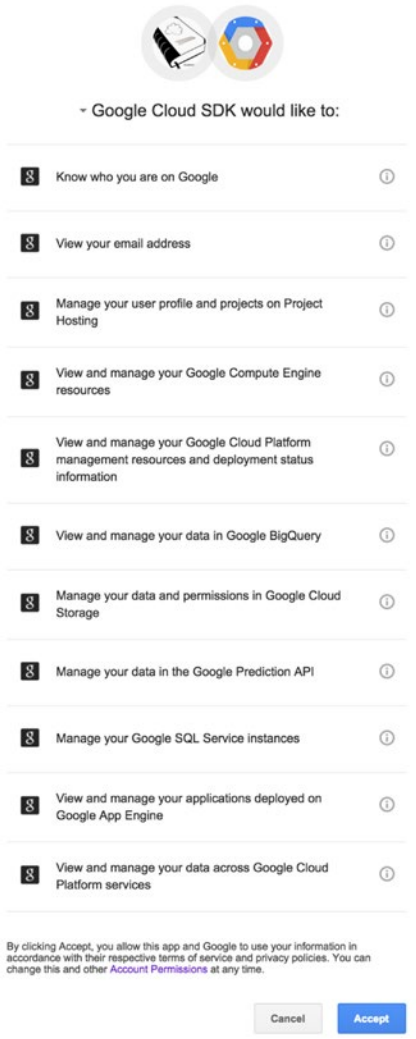

Figure 4-3. *Google authentication screen after you have logged in*

You may be running `gcloud` in an environment where there is no active GUI environment, such as a server or a remote terminal connection. In this case, you can invoke the previous command with the option `--no-launch-browser`, and `gcloud` will print a long URL and request an access code. Copy and paste the code into a web browser (on any system), which will return an access code that you then copy and paste into the command-line prompt. Figure 4-4 illustrates this workflow.

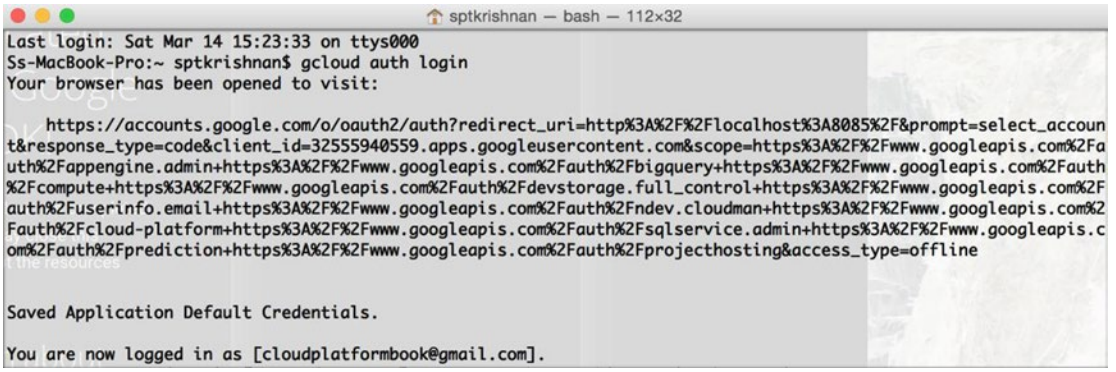

Figure 4-4. *Screenshot showing the long URL and code*

You may use different Google identities for different roles in your life—work, personal, and community project. If so, you have to repeat this process for each account the first time. After you have done this once, you can easily switch between accounts anytime. The following command sequence lists all linked account and lets you choose one of them:

```
$ gcloud auth list
$ gcloud config set account <email>
```

Selecting a Google Cloud Platform Project

You need to declare a `project-id` for use by the `gcloud` tool. You can get the `project-id` from the Developers Console at `https://console.developers.google.com`. Once you have the `project-id`, issue the following command from a terminal window to set the project:

```
$ gcloud config set project <project-id>
```

At this stage, it is important to know about the following limitations when selecting a project using `gcloud`:

- It is currently not possible to create a new project.
- It is currently not possible to list existing projects.
- The declared `project-id` is not validated on entry.
- Access rights to a project are not checked on entry.

Creating and Starting an Instance

Now it is time to create your first VM instance in the Compute Engine platform. You need to do decide which geographical region and zone you want to host your VM and the operating system to use. A geographical region may contain one or more zones. Compute Engine has packaged several widely used OSs as disk images. *Disk images*, or simply *images*, are templates for creating new disks. Root disks usually have an operating system installed, but an image may just be a data disk.

The following command lists all Google Cloud Platform regions and the count of certain resources you have created in each of them:

```
$ gcloud compute regions list
```

Google has multiple zones in each of these regions. You can use the following command to list the zones in each region:

```
$ gcloud compute zones list
```

For this first instance, you use a standard operating system. To determine which operating system images are currently supported in Compute Engine, use this command:

```
$ gcloud compute images list
```

With this information, using the defaults for the rest of the options, you can now create a VM instance using the following command:

```
$ gcloud compute instances create <instance-name> --image <image-name> --zone <zone>
```

This command lists all running instances in the Compute Engine platform:

```
$ gcloud compute instances describe <instance-name>
```

Allowing Ingress Network Access

With the instance up and running, all that remains is to allow access to it from the Internet. You do so by creating firewall rules. The Compute Engine network firewall processes all inbound traffic; hence the source addresses are always external, whereas the target addresses are hosted on the Compute Engine platform. The following command shows how to add a new firewall rule to allow ingress network traffic to access an instance:

```
$ gcloud compute firewall-rules create <name> --description <description> --allow
<protocol:port(s)>
```

With the completion of these commands, you have a VM instance that is accessible from the Internet. In the next section, using these command templates, you launch a new VM instance to host a web-based blog.

Creating a Web Presence with Compute Engine in 8 Minutes Flat

In this section, you use the `gcloud` commands you learned in the previous section to launch a VM instance with a web application that is accessible from anywhere on the Internet. The following instructions create a VM with default specifications running the Linux Debian 7 operating system. The instructions are in **bold** font, and the output is shown:

```
$ gcloud config set account cloudplatformbook@gmail.com

$ gcloud config set project cloud-platform-book
```

```
$ gcloud config list
[core]
account = cloudplatformbook@gmail.com
disable_usage_reporting = False
project = cloud-platform-book
user_output_enabled = True

$ gcloud compute instances list
NAME ZONE MACHINE_TYPE INTERNAL_IP EXTERNAL_IP STATUS
```

■ **Note** Some of "gcloud" command-line tool argument list and output string is longer than single line. We have manually split such long lines into two lines for ease of reading throughout this chapter. For input argument list we have added a " \" at the end of list just like with traditional bash shell inputs. You can ignore these two characters and input the two lines as a single continuous line in your console. For return values we have split them at ".../cloud-platform-book/" part of the output.

```
$ gcloud compute instances create frontend-master --image debian-7 --zone asia-east1-a
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/asia-east1-a/instances/frontend-master].
NAME            ZONE         MACHINE_TYPE  INTERNAL_IP    EXTERNAL_IP    STATUS
frontend-master asia-east1-a n1-standard-1 10.240.135.214 104.155.206.70 RUNNING
```

This command explicitly states the Compute Engine zone using the `--zone` flag. If you are going to work with instances that are in a single zone most of the time, you are advised to set that zone as the default in the project configuration for Compute Engine. This saves you the hassle of specifying the zone in all the commands. When you need to work with instances that are in other zones, you can explicitly use the `--zone` flag to override the default temporarily.

The following `gcloud` command sets the default zone for Compute Engine. Consequently, you can drop the `--zone` flag from subsequent commands:

```
$ gcloud config set compute/zone asia-east1-a

$ gcloud compute instances describe frontend-master
canIpForward: false
creationTimestamp: '2015-01-10T22:21:33.244-08:00'
disks:
- autoDelete: true
  boot: true
  deviceName: persistent-disk-0
  index: 0
  interface: SCSI
  kind: compute#attachedDisk
  mode: READ_WRITE
  source: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/asia-east1-a/disks/frontend-master
  type: PERSISTENT
id: '13106361493425027 62'
kind: compute#instance
```

```
machineType: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/asia-east1-a/machineTypes/n1-standard-1
metadata:
  fingerprint: P8RS5MYe1aA=
  kind: compute#metadata
name: frontend-master
networkInterfaces:
- accessConfigs:
  - kind: compute#accessConfig
    name: external-nat
    natIP: 104.155.206.70
    type: ONE_TO_ONE_NAT
  name: nic0
  network: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
networks/default
  networkIP: 10.240.135.214
scheduling:
  automaticRestart: true
  onHostMaintenance: MIGRATE
selfLink: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/asia-east1-a/instances/frontend-master
serviceAccounts:
- email: 261451503272-compute@developer.gserviceaccount.com
  scopes:
  - https://www.googleapis.com/auth/devstorage.read_only
status: RUNNING
tags:
  fingerprint: 42WmSpB8rSM=
zone: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/zones/asia-east1-a
```

By default, Compute Engine configures the network firewall to allow only a few default ports (SSH, ICMP, and RDP) from any source IP address to the VM instances. You need to add firewall rules to allow inbound connections to other software network ports. The most common such rule allows HTTP and HTTPS traffic to ports 80 and 443. You are setting a web server in the newly created instance, so you can now configure the network firewall to allow incoming HTTP and HTTPS traffic to this instance:

```
$ gcloud compute firewall-rules create allow-http --description "Incoming http allowed."
--allow tcp:80
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/firewalls/allow-http].
NAME       NETWORK SRC_RANGES RULES  SRC_TAGS TARGET_TAGS
allow-http default 0.0.0.0/0  tcp:80
```

You can tweak the command to allow HTTPS connections as well:

```
$ gcloud compute firewall-rules create allow-https --description "Incoming https allowed."
--allow tcp:443
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/firewalls/allow-https].
NAME        NETWORK SRC_RANGES RULES   SRC_TAGS TARGET_TAGS
allow-https default 0.0.0.0/0  tcp:443
```

Let us dissect one of these commands in detail, to understand some of the implicit assumptions made by Compute Engine when no explicit options are provided. Compute Engine uses these default options:

- When a firewall rule does not specify an instance, the firewall rule is applied to all live instances.
- When a firewall rule does not specify a network, the firewall rule is applied to the default network.
- When a firewall rule does not specify a source, the firewall rule allows *all* source IPs (both internal and external) to make requests to the target. In this case, all instances are added to the default network.

Now that you have added your own custom firewall rules, you may want to list all the active firewall rules. This can be accomplished using the following command:

```
$ gcloud compute firewall-rules list
This NAME                NETWORK SRC_RANGES    RULES         SRC_TAGS TARGET_TAGS
allow-http               default 0.0.0.0/0     tcp:80
allow-https              default 0.0.0.0/0     tcp:443
default-allow-icmp       default 0.0.0.0/0     icmp
default-allow-internal default 10.240.0.0/16 tcp:1-65535,udp:1-65535,icmp
default-allow-rdp        default 0.0.0.0/0     tcp:3389
default-allow-ssh        default 0.0.0.0/0     tcp:22
```

If you want to describe any one rule in more detail, you can do so using the following command:

```
$ gcloud compute firewall-rules describe allow-http
allowed:
- IPProtocol: tcp
  ports:
  - '80'
creationTimestamp: '2015-01-18T21:35:09.045-08:00'
description: Incoming http allowed.
id: '15660039426868416436'
kind: compute#firewall
name: allow-http
network: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/networks/default
selfLink: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/firewalls/allow-http
sourceRanges:
- 0.0.0.0/0
```

The default options may not be suitable for every VM instance. For example, you may want to restrict SSH access from certain IP addresses, such as a bastion host, or allow HTTP from internal IP addresses only. In such cases, you can use the applicable options to these commands. To list the available options for any command, add the suffix `--help` to it. As an example, the following command shows the help page for firewall rule creation:

```
$ gcloud compute firewall-rules create -help
```

You have set up your instance and allowed both inbound HTTP(S) and SSH connections. Let us log in to the system and set up the required software to serve incoming requests. You use the following command to log in to the system:

```
$ gcloud compute ssh frontend-master
Warning: Permanently added '104.155.206.70' (RSA) to the list of known hosts.
Linux frontend-master 3.2.0-4-amd64 #1 SMP Debian 3.2.65-1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jan 11 07:17:48 2015 from 103.224.116.114
user@frontend-master:~$
```

Let us examine some of this command's features. First, the `gcloud` command-line tool has the built-in ability to SSH into the remote system. Second, the default authentication method is to use public keys instead of passwords. This is safer and easier to use, as well. At the same time, you can use any other SSH tool to access VM instances. See the online documentation at https://cloud.google.com/compute/docs/console for more information. You can also log in over SSH using your browser from the web-based Developers Console; look for the SSH button when viewing information about a VM.

Once you are logged in, you can do anything on this Compute Engine VM instance that you can do on a standard Linux system. You have full root access in the VM. Next, you need to run the following three commands to update the Linux VM with any critical updates that may have been issued by the distribution owner since the image was created. Remote Compute Engine VM commands have the prefix `VM` before the `$` symbol:

```
VM$ sudo apt-get update
<Snip>

VM$ sudo apt-get dist-upgrade
<Snip>

VM$ sudo reboot
<Snip>
```

After the VM has rebooted, you need to log in again via SSH. Let us set up a web server to serve the default web site from your instance. This example uses the most popular web server software: the Apache web server. Use the following command to set up the Apache web server software:

```
VM$ sudo apt-get install apache2
<Snip>
```

You should now be able to access the instance over HTTP using the URL http://EXTERNAL_IP. You can obtain the external IP address using the following command:

```
$ gcloud compute instances list
NAME            ZONE         MACHINE_TYPE  INTERNAL_IP    EXTERNAL_IP    STATUS
frontend-master asia-east1-a n1-standard-1 10.240.135.214 104.155.206.70 RUNNING
```

It is important to know that billing begins on VM deployment, and you are charged every month for the resources consumed: compute, storage, or network bandwidth. Hence, you should delete deployed resources when you do not need them anymore. The following command stops and deletes the VM instance and, depending on the setting, the associated PD as well. The default setting is to delete PDs on VM instance deletion:

```
$ gcloud compute instances delete frontend-master
```

Handling Unpredictable Traffic with the Compute Engine Load Balancer

Compute Engine offers VM instances of varying sizes. But each of these instances has a maximum computing capacity, as measured by CPU processing and/or network throughput. Hence, it is important for system architects to know how to design Internet-facing systems that can scale up as the incoming requests increase and scale down to shed extra capacity when not required, thereby leading to cost savings. You can achieve this automatic reconfiguration of systems using two entities called the *load balancer* and the *autoscaler*.

This section discusses the Compute Engine load balancer and shows an example of setting up a frontend web tier comprising a load balancer and two web servers in a single geographic region. The following section expands this architecture to build a truly global infrastructure consisting of nodes in multiple geographic regions. Let us start by defining a load balancer.

A *load balancer* is a device that acts as a proxy; it can be considered the first network hop for incoming network or application traffic. In a traditional three-tier architecture, consisting of web, application, and database tiers, the load balancer is in front of the web tier and distributes incoming traffic to healthy web server nodes. A load balancer can be either physical or virtual.

The Google Cloud Platform currently offers two types of load balancing: network and HTTP. *Network load balancing* is a regional capability that can load-balance incoming network traffic to a set of healthy compute nodes in a single region. Network load balancing operates on a low-level network protocol, such as TCP, and software ports on a VM instance, such as port 80. On the other hand, *HTTP load balancing* is a global capability; it can distribute incoming application-level traffic, such as web traffic, and distribute that traffic to VM instances in different Google Cloud Platform regions and zones. Let us look in detail at network load balancing in the following paragraphs; the following section covers HTTP load balancing.

The Compute Engine load balancer has two main building blocks—forwarding rules and target pools—and a supporting service called the health checker. Let us examine these along with the load-distribution algorithm.

■ **Note** Some components of the Google Cloud Platform are in beta as of this writing. This includes HTTP load balancing. To access these components, you need to install the `gcloud` preview component, using the following command: `gcloud components update preview`.

Forwarding Rules

A *forwarding rule* is like a filter that extracts selected traffic, which is interesting to you, from the total traffic entering a Cloud Platform zone. Technically, it can be seen as a regular expression that matches a subset of network traffic from the overall traffic. Forwarding rules reside in a forwarding rule collection managed by Compute Engine. A forwarding rule matches an external IP address and a protocol (TCP/UDP). Optionally, the forwarding rule can be restricted to a certain range of ports. The filtered traffic matching a forwarding rule is forwarded to a target pool.

Target Pool

A *target pool* is a resource that consists of one or more VM instances. Incoming network traffic is automatically assigned to a healthy VM instance based on a load-distribution algorithm (explained in the next section). The advantage of using network load balancing is that the endpoint VM receives the actual network traffic as if it was the instance exposed to the Internet in the first place.

Load-Distribution Algorithm

It is common for a load balancer to use a round-robin method to assign new incoming connections to healthy VM instances. Compute Engine uses a different and unique method: a hash of incoming IP and port, and target IP and port, to decide the target instance to which to deliver the connection. All subsequent connections that match the same hash will be delivered to the same VM instance. This characteristic ensures that a session is handled by the same VM instance. You can change this default behavior using a different hashing algorithm if better session affinity is required.

Health Checks

Compute Engine has a built-in ability to detect whether a VM instance is healthy. It determines this by sending an HTTP request to a preset URL hosted in each of the VM instances; it expects a valid response with HTTP code 200 OK, and it expected the HTTP to be closed normally within the timeout duration. If an instance does not respond within the timeout duration or does not return the expected response for a configurable X number of times, then it is marked as unhealthy and removed from the target pool. The target pool does not send any new requests to this VM instance; however, existing connections in the session are served by the same instance to allow for graceful shutdown. The Compute Engine health checker keeps pinging the unhealthy VM instance, and once it receives a configurable Y number of good responses, it marks the VM instance as healthy and returns the instance to the target pool.

It is important to note that the VM instance should be running a web server in order to respond to the health checker, even if the expected network traffic to the VM instance is not HTTP traffic. This usually is not a major issue, because you can limit incoming network traffic to the IP range of the health checker, thereby not exposing the web server to the Internet. Further, you can use a lightweight web server if the system resources are not sufficient to run a full-fledged web server like Apache. For HTTP load balancing, the health check requests come from addresses in the range 130.211.0.0/22. For network load balancing, the health check requests come from 169.254.169.254. See https://cloud.google.com/compute/docs/load-balancing/health-checks#listing_health_checks for the current health-checker IP range. This is useful if you configure firewall rules with narrower scopes.

Going Live

This section explores the details of setting up a network load balancer with two VM instances. Begin by following the instructions from the previous section and setting up a second VM instance. Then, log in to each of these instances and create different default `index.html` files to uniquely identify them over the Web. The following instructions achieve this (the commands are documented individually):

```
# Create a new instance in the same region but in a different zone
$ gcloud compute instances create frontend-slave --image debian-7 --zone asia-east1-b

# Login to the new instance
$ gcloud compute ssh frontend-slave --zone asia-east1-b
```

```
# Update the installed packages and install apache web server
VM$ sudo apt-get update
<Snip>

VM$ sudo apt-get dist-upgrade
<Snip>

VM$ sudo reboot
<Snip>

VM$ sudo apt-get install apache2
<Snip>
```

Now SSH into both VM instances and update the default `index.html` file to show different messages. You will use this for a future test case where you load-test the load balancer; by having different default pages, you can see which instance is responding to incoming queries. The following code shows the modified `index.html` files for the two instances:

```
user@frontend-master:/var/www$ pwd
/var/www

user@frontend-master:/var/www$ ls -l
total 4
-rw-r--r-- 1 root root 193 Jan 24 01:09 index.html

user@frontend-master:/var/www$ cat index.html
<html><body><h1>Frontend Master</h1>
<p>Welcome to FRONTEND MASTER server.</p>
</body></html>

user@frontend-slave:/var/www$ pwd
/var/www

user@frontend-slave:/var/www$ ls -l
total 4
-rw-r--r-- 1 root root 192 Jan 24 01:10 index.html

user@frontend-slave:/var/www$ cat index.html
<html><body><h1>Frontend Slave</h1>
<p>Welcome to FRONTEND SLAVE server.</p>
</body></html>
```

At this stage, you should have two Linux-powered VM instances that have Apache web server software installed and are accessible from the Internet. Verify that you can access them using a web browser:

```
# Verify the VM instances are accessible from a web browser
http://130.211.244.80/
http://104.155.206.70/
```

Note that this code uses the IP addresses of my instances. Your IP addresses will be different. You can find the external IP addresses by running `gcloud compute instances list`.

Next, you need to set up a health check, create a target pool, and associate the health check with the target pool. After this, you add these two instances to the target pool and create the required forwarding rules to direct traffic to the target pool. The following are the commands to run on your workstation, with explanations in the comments:

```
# Create an http health check using default settings
$ gcloud compute http-health-checks create http-check
Created
[https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/httpHealthChecks/http-check].
NAME       HOST PORT REQUEST_PATH
http-check      80   /
```

Now let us add a target pool in the same region where you have created the instances. The instances can be in different zones in the same region, as is the case with the two example instances. The following commands list the regions, set an environment variable to remember the region, and then create the target pool linked with the HTTP check in the same region:

```
$ gcloud compute regions list
NAME            CPUS          DISKS_GB     ADDRESSES RESERVED_ADDRESSES STATUS TURNDOWN_DATE
asia-east1      2.00/24.00    20/10240     2/23      0/7                UP
europe-west1    0.00/24.00     0/10240     0/23      0/7                UP
us-central1     0.00/24.00     0/10240     0/23      0/7                UP

$ REGION="asia-east1"

$ gcloud compute target-pools create www-pool --region $REGION --health-check http-check
Created
[https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
regions/asia-east1/targetPools/www-pool].
NAME     REGION     SESSION_AFFINITY BACKUP HEALTH_CHECKS
www-pool asia-east1 NONE                    http-check

$ gcloud compute target-pools add-instances www-pool --instances frontend-master --zone
asia-east1-a
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
regions/asia-east1/targetPools/www-pool].

$ gcloud compute target-pools add-instances www-pool --instances frontend-slave --zone
asia-east1-b
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
regions/asia-east1/targetPools/www-pool].
```

In this example, if the instances are in same zone, you can combine the two statements and list the instances. Also, it is possible to add a health check to an existing target pool. This is useful when new type of health checks are added to Compute Engine.

As a final step, let us add a forwarding rule to send traffic to the target pool. The forwarding rule needs a public IP address; you can use either static IP or an ephemeral IP address. In this case, use the default option (an ephemeral IP address):

```
$ gcloud compute forwarding-rules create www-rule --region $REGION --port-range 80 --target-pool www-pool
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/regions/asia-east1/forwardingRules/www-rule].
NAME     REGION     IP_ADDRESS      IP_PROTOCOL TARGET
www-rule asia-east1 104.155.234.199 TCP         asia-east1/targetPools/www-pool
```

Let us add the ephemeral IP address assigned to the forwarding rule and load-test the setup you have built to see if the target pool is working as expected:

```
$ IP="104.155.234.199"

$ while true; do curl -m1 $IP; done
<html><body><h1>Frontend Master</h1>
<p>Welcome to FRONTEND MASTER server.</p>
</body></html>
<html><body><h1>Frontend Slave</h1>
<p>Welcome to FRONTEND SLAVE server.</p>
</body></html>
<html><body><h1>Frontend Master</h1>
<p>Welcome to FRONTEND MASTER server.</p>
</body></html>
<html><body><h1>Frontend Slave</h1>
<p>Welcome to FRONTEND SLAVE server.</p>
</body></html>
<html><body><h1>Frontend Master</h1>
<p>Welcome to FRONTEND MASTER server.</p>
</body></html>
<html><body><h1>Frontend Slave</h1>
<p>Welcome to FRONTEND SLAVE server.</p>
</body></html>
```

As you can see, incoming queries to the forwarding-rule IP address are distributed to the VM instances behind the target pool. This completes the setup for the network load balancer.

Before you move on, a final note. It is my observation and experience that users usually set up one VM to begin with; once the software stack is set up and tuned properly, they scale out. You can do so by creating an image of your existing VM, using it as a mold to create a second VM, and attaching both of them to the same target pool. See https://cloud.google.com/compute/docs/images for extensive documentation of Compute Engine images.

Building a Global Multi-Datacenter Web Tier in an Hour

In the previous section, you created a network load balancer that is restricted to one region in the Google Cloud Platform. This type of setup works well for companies whose customers are in one geographical region. For global enterprises that have customers around the globe, having a setup in one geographical region means low-latency network access to customers in one region but medium- to high-latency network

access to other groups of users. Hence, many organizations with customers spread globally need multiple points of presence for their IT footprint. This requirement is well served by the HTTP load balancer and is covered in this section.

The key components of the HTTP load balancer are as follows:

- Global forwarding rule
- Target proxy
- URL map
- Backend services
- Instance groups

Global Forwarding Rules

Global forwarding rules route incoming network traffic by IP address, port, and protocol to a load-balancing configuration, which consists of a target proxy, a URL map, and one or more backend services. Similar to the forwarding rules in network loading balancing, global forwarding rules assign a single global IP address, which can be either static or an ephemeral IP address. This single global IP address can be used in a DNS record, negating the need for traditional DNS-based round-robin load balancing.

Target HTTP Proxy

The *target HTTP proxy*, as the name suggests, is the endpoint on the Compute Engine side that terminates the incoming HTTP/TCP connection from the user. A new connection is then made by the proxy to the next internal network hop, known as the *URL map*. The proxy also adds additional request/response HTTP headers to the HTTP connection, as follows:

```
Via: 1.1 Google (requests and responses)
X-Forwarded-For: <client IP>, <global forwarding rule external IP> (requests only)
```

By inspecting the HTTP headers, the clients and the server can extract key information. For example, an analytics module on the server side should use the `<client IP>` in the `X-Forwarded-For` header instead of the `source-ip` of the TCP connection, because the latter is always the proxy IP. The clients can also parse the HTTP connection and know that a Google proxy is serving them, and not the actual server, although the DNS resolution of the domain provides the server's IP address.

URL Maps

URL maps define regular-expression matching patterns to filter the incoming requests and forward them to different sets of backend services. They are typically used in content-based routing architectures where different backend services serve different type of content, such as images, videos, and text. A default service is defined to handle any requests that are not matched by any other rule. This is sufficient for a multi-region load-balancing setup.

Backend Services

Backend services define groups of backend instances and their serving capacity. The serving capacity is defined either as CPU usage or requests per second (RPS). The backend service also specifies the health checks that will be performed against the available instances. The health checks are the same as described earlier for network load balancing.

Instance Groups

Instance groups are a grouping mechanism that defines VM instances that are available as a backend-services group. A backend service may list a set of instance groups instead of individual instances. You can add instances to and remove them from an instance group, and the instance group can be used for other cloud services in addition to load balancing.

Load-Distribution Algorithm

HTTP load balancing provides two methods of determining an instance load: RPS and CPU utilization modes. Both modes let you specify a maximum value. Incoming requests are sent to the region that is closest to the user and that has remaining capacity.

Going Live

This example uses the two instances you created in the previous two sections. In addition to these two instances in the Asia region, you add four more instances: two in Europe and two in the United States. All the instances are in different zones. In addition, you automate the installation of a web server using a startup script. Compute Engine executes the startup script as part of VM startup. Following are the required Linux commands and the output of these steps:

```
$ echo "apt-get update && apt-get install -y apache2 && hostname > /var/www/index.html" > \
$HOME/gce:startup.sh

$ gcloud compute instances create frontend-us1 --image debian-7 \
--zone us-central1-a --metadata-from-file startup-script=$HOME/gce:startup.sh
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/us-central1-a/instances/frontend-us1].
NAME         ZONE          MACHINE_TYPE  INTERNAL_IP    EXTERNAL_IP     STATUS
frontend-us1 us-central1-a n1-standard-1 10.240.154.170 130.211.186.207 RUNNING

$ gcloud compute instances create frontend-us2 --image debian-7 \
--zone us-central1-b --metadata-from-file startup-script=$HOME/gce:startup.sh
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/us-central1-b/instances/frontend-us2].
NAME         ZONE          MACHINE_TYPE  INTERNAL_IP   EXTERNAL_IP    STATUS
frontend-us2 us-central1-b n1-standard-1 10.240.124.81 104.154.39.159 RUNNING

$ gcloud compute instances create frontend-eu1 --image debian-7 --zone europe-west1-b
--metadata-from-file startup-script=$HOME/gce:startup.sh
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/europe-west1-b/instances/frontend-eu1].
NAME         ZONE           MACHINE_TYPE  INTERNAL_IP   EXTERNAL_IP     STATUS
frontend-eu1 europe-west1-b n1-standard-1 10.240.232.37 146.148.126.134 RUNNING

$ gcloud compute instances create frontend-eu2 --image debian-7 \
--zone europe-west1-c --metadata-from-file startup-script=$HOME/gce:startup.sh
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/europe-west1-c/instances/frontend-eu2].
NAME         ZONE           MACHINE_TYPE  INTERNAL_IP   EXTERNAL_IP   STATUS
frontend-eu2 europe-west1-c n1-standard-1 10.240.95.225 104.155.11.26 RUNNING
```

Using a web browser, verify that you can access each instance through its public IP address. If they are not accessible, check whether the firewall rules have been added to enable access at port 80.

The following are the logical steps to set up the HTTP load-balancing service:

1. Create instance groups, one per zone.
2. In these zones, add instances to the instance groups.
3. Add the required HTTP service to each instance group.
4. Create or reuse an HTTP health-check object.
5. Create a backend service linking the health check.
6. Add instance groups as backends to the backend service.
7. Create a URL map to direct ingress traffic to the backend service.
8. Create a target HTTP proxy to route requests to the URL map.
9. Create a global forwarding rule to send traffic to the HTTP proxy.

Here are the `gcloud` commands to carry out these steps:

■ **Note** Some `gcloud` commands use `preview` instead of the `compute` option. This is because they are in a beta state, and it is possible the API and `gcloud` options may change as the product goes into GA release. If the following commands do not work, check the current versions at http://cloud.google.com/compute/docs.

```
# Step 1 - gcloud commands to create instances groups

$ gcloud preview instance-groups --zone us-central1-a create ig-usc1a
Instance group ig-usc1a created.
$ gcloud preview instance-groups --zone us-central1-b create ig-usc1b
Instance group ig-usc1b created.
$ gcloud preview instance-groups --zone europe-west1-b create ig-euw1b
Instance group ig-euw1b created.
$ gcloud preview instance-groups --zone europe-west1-c create ig-euw1c
Instance group ig-euw1c created.
$ gcloud preview instance-groups --zone asia-east1-a create ig-ape1a
Instance group ig-ape1a created.
$ gcloud preview instance-groups --zone asia-east1-b create ig-ape1b
Instance group ig-ape1b created.

# Step 2 - gcloud commands to add instances to instance groups

$ gcloud preview instance-groups --zone us-central1-a instances \
--group ig-usc1a add frontend-us1
Instances added to instance group ig-usc1a.
$ gcloud preview instance-groups --zone us-central1-b instances \
--group ig-usc1b add frontend-us2
Instances added to instance group ig-usc1b.
$ gcloud preview instance-groups --zone europe-west1-b instances \
--group ig-euw1b add frontend-eu1
Instances added to instance group ig-euw1b.
```

```
$ gcloud preview instance-groups --zone europe-west1-c instances \
--group ig-euw1c add frontend-eu2
Instances added to instance group ig-euw1c.
$ gcloud preview instance-groups --zone asia-east1-a instances \
--group ig-ape1a add frontend-master
Instances added to instance group ig-ape1a.
$ gcloud preview instance-groups --zone asia-east1-b instances \
--group ig-ape1b add frontend-slave
Instances added to instance group ig-ape1b.

# Step 3 - gcloud commands to add required http service to each instance group

$ gcloud preview instance-groups --zone us-central1-a add-service ig-usc1a \
--port 80 --service http
Service http:80 added.
$ gcloud preview instance-groups --zone us-central1-b add-service ig-usc1b \
--port 80 --service http
Service http:80 added.
$ gcloud preview instance-groups --zone europe-west1-b add-service ig-euw1b \
--port 80 --service http
Service http:80 added.
$ gcloud preview instance-groups --zone europe-west1-c add-service ig-euw1c \
--port 80 --service http
Service http:80 added.
$ gcloud preview instance-groups --zone asia-east1-a add-service ig-ape1a \
--port 80 --service http
Service http:80 added.
$ gcloud preview instance-groups --zone asia-east1-b add-service ig-ape1b \
--port 80 --service http
Service http:80 added.
```

You created an HTTP health check earlier, in the network load-balancing section, and you reuse it here. If you have deleted it, you can create a new one using the following command:

```
# Step 4a - List any existing http health check objects
$ gcloud compute http-health-checks list
NAME       HOST PORT REQUEST_PATH
http-check      80   /

# Step 4b - Create a new http health check object (if required)
$ gcloud compute http-health-checks create http-check

# Step 5 - Create a backend service

$ gcloud compute backend-services create web-service --http-health-check http-check
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/backendServices/web-service].
NAME        BACKENDS PROTOCOL
web-service          HTTP
```

```
# Step 6 - Add instance groups as backends to backend service

$ gcloud compute backend-services add-backend web-service \
--group ig-usc1a --zone us-central1-a
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].
$ gcloud compute backend-services add-backend web-service \
--group ig-usc1b --zone us-central1-b
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].
$ gcloud compute backend-services add-backend web-service \
--group ig-euw1b --zone europe-west1-b
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].
$ gcloud compute backend-services add-backend web-service \
--group ig-euw1c --zone europe-west1-c
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].
$ gcloud compute backend-services add-backend web-service \
--group ig-ape1a --zone asia-east1-a
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].
$ gcloud compute backend-services add-backend web-service \
--group ig-ape1b --zone asia-east1-b
Updated [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
backendServices/web-service].

# Step 7 - Create URL Map to direct ingress traffic to backend service

$ gcloud compute url-maps create web-map --default-service web-service
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/urlMaps/web-map].
NAME    DEFAULT_SERVICE
web-map web-service

# Step 8 - Create target HTTP proxy to route requests to URL map

$ gcloud compute target-http-proxies create web-proxy --url-map web-map
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/global/
targetHttpProxies/web-proxy].
NAME      URL_MAP
web-proxy web-map

# Step 9 - Create Global forwarding rule to send traffic to HTTP proxy

$ gcloud compute forwarding-rules create http-rule --global \
--target-http-proxy web-proxy --port-range 80
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/forwardingRules/http-rule].
NAME      REGION IP_ADDRESS     IP_PROTOCOL TARGET
http-rule        107.178.245.89 TCP         web-proxy
```

At this stage, if you open a web browser with `http://<global-forwarding-rule-ip>`, your request is routed to the instance that is nearest to you. Let us send some traffic to the HTTP load balancer you have set up, and see its responses. Note that my workstation was in Singapore for this test:

```
$ while true; do curl -m1 107.178.245.89; done
frontend-master
frontend-master
frontend-master
```

Let us simulate system maintenance in the setup by shutting down the Apache web servers in both of Asian nodes and see if the traffic is automatically routed to the next-best location:

```
$ gcloud compute ssh frontend-master --zone asia-east1-a
...
Last login: Sun Jan 25 04:06:14 2015 from 1.2.1.1
VM$ sudo service apache2 stop
[ ok ] Stopping web server: apache2 ... waiting .
VM$ exit
logout
Connection to 104.155.206.70 closed.

$ gcloud compute ssh frontend-slave --zone asia-east1-b
...
Last login: Sat Jan 24 14:31:53 2015 from 1.2.1.1
VM$ sudo service apache2 stop
[ ok ] Stopping web server: apache2 ... waiting .
VM$ exit
logout
Connection to 130.211.244.80 closed.

$ while true; do curl -m1 107.178.245.89; done
frontend-us2
frontend-us2
frontend-us1
```

This experiment shows that the HTTP load balancer is transparently detecting backend failure and automatically routing traffic to the next-nearest node. When the nodes return to service, the routing will automatically reset. Using the HTTP load balancer, IT operations can be assured that customers always reach a good node.

A variation of the HTTP load-balancing setup is *content-based load balancing*. In this architecture, incoming traffic is distributed to various backend services based on predefined patterns in the request URLs. This is a very useful feature that can be used to select appropriate backend services to serve different type of data such as images, video, and text.

You can obtain this desirable behavior by making a minor modification to the HTTP load-balancing setup discussed in this section. You make the change in the step where you define a URL map. In addition to defining the default map, you also need to define more specific maps that route traffic to those URLs to a different backend. Let us look at an example:

```
# Example URL map
hostRules:
- hosts: ["*"]
  pathMatcher: pathmap
pathMatchers:
- name: pathmap
  defaultService: https://www.googleapis.com/compute/v1/projects/PROJECT_ID/
global/backendServices/www-service
  pathRules:
    - paths: ["/video", "/video/*"]
      service: https://www.googleapis.com/compute/v1/projects/PROJECT_ID/
global/backendServices/video-service
    - paths: ["/static", "/static/*"]
      service: https://www.googleapis.com/compute/v1/projects/PROJECT_ID/
global/backendServices/static-service
```

In this example, requests to URLs `/video/*` are sent to a backend service called `video-service`, and requests to `/static/*` are sent to a backend service called `static-service`. All other requests are sent to a backend service called `www-service`.

Automatically Resizing a Web Tier with the Compute Engine Autoscaler

The previous sections looked at the network load balancer and HTTP load balancer. A network load balancer has regional scale, whereas an HTTP load balancer has global scale, but neither load balancer scales automatically. You need to either scale them up manually (by adding more VM instances) to meet additional workloads or scale them down (by removing VM instances) to save costs.

This section looks at another technology called the *autoscaler* in the Compute Engine portfolio that automatically adds resources on demand and sheds overcapacity when not required, resulting in cost savings. The autoscaler can add resources based on several metrics: CPU utilization, serving metric, and other cloud-monitoring metrics. Let us first learn about the key building blocks and terminology of the Compute Engine autoscaler service.

Managed Instance Group

An *instance group* is a group of VM instances managed as a single entity. A *managed instance group* is a group of homogenous instances that is automatically managed by an instance group manager. The instance group manager can add homogenous instances (instances with the same specs) to the group or remove them.

Utilization Metric

The autoscaler requires a dynamic system metric that it can measure, in order to add and remove the VM resources that it is managing. This is known as the *utilization metric*; and depending on the type of computational resource, it is measured differently.

The following are some standard metrics:

- *CPU utilization metric*: Usage is measured as a percentage of total CPU capacity in the VM cluster.
- *Incoming requests*: Usage is counted as the number of new connections every second.

The utilization metrics are averaged over the total number of VM instances, and the average value is compared against the target utilization level. Selecting a good utilization metric is paramount to the proper functioning of the autoscaler. *Good* mean a metric that reflects the actual load the cluster is experiencing at that point in time. Examples of good metrics include CPU usage across the VMs and actual network ingress/egress traffic from the VMs. An example of an indirect metric is the amount of traffic to/from the persistent storage. Although this is a sign of how busy a VM/app is, it need not produce good results all the time, because apps may be CPU-bound instead of IO-bound. At the same time, choosing a bad metric negates the purpose of the autoscaler. An example of a bad metric is the number of cores in a VM. Because the number of cores is a static measure, the autoscaler is ineffective in this case.

Target Utilization Level

You can think of the *target utilization level* as a tipping point: when exceeded, it triggers the autoscaler to add more resources so the average utilization level is restored to being under this set level. By default, the autoscaler watches the resource every 5 seconds. When resource consumption exceeds this set level for two consecutive checks, additional VM resources are added. Similarly, when resource consumption falls below the set threshold level for two consecutive checks, VMs are discarded, resulting in cost savings.

The frequency of checks and the number of consecutive checks that trigger the autoscaler to either scale up or scale down VM resources are configurable. Also, the specification of the target utilization level is different for various resources. CPU utilization is defined as the average of CPU utilization across all VM instances, whereas RPS for ingress connections is defined per VM. Both the autoscaler and the managed VMs reside in the same zone.

Autoscaling Integrations

This section discusses how autoscaling can be integrated with the load-balancing configurations you saw in the previous sections. To recall, you learned about the regional-scale network load balancer and global-scale HTTP load balancer. In both cases, the number of instances is manually managed. Using an autoscaler, the number of instances can be auto-managed so the VM cluster size is optimal for the load factor.

In the case of a network load balancer, forwarding rule filters traffic based on network characteristics (IP, port, and protocol) and sends the traffic to a network pool, which is the entity that contains an instance group. In an earlier example, you used an unmanaged (static) instance group. When you are using the autoscaler, you can simply replace this unmanaged instance group with a managed (dynamic) instance group. The instance group manager of the managed instance group is passed to the autoscaler, which measures the load factor—say, CPU utilization—while the forwarding rules and network pool continue to deliver ingress traffic to the VM instances.

In the case of an HTTP load balancer, a global forwarding rule filters traffic to a target HTTP proxy that in turn consults an URL map and the source of the request and delivers that traffic to the corresponding backend services. The backend services form the unmanaged instance group in the classic configuration. With the autoscaler, you replace the unmanaged instance group with a managed instance group and pass the corresponding instance-group manager to the autoscaler. This way, each regional cluster autoscales regardless of the load factor at other regional clusters. This behavior is useful for content-based load balancing where certain parts of a web app, such as media (images and video), may require more backend resources compared to text-based resources like HTML, CSS, and JavaScript data. Let us look at the steps required to set up an autoscaler in the network load balancer.

Going Live

The following are the high-level steps to realize an autoscaling backend for a network load balancer:

1. Create an instance template.
2. Create a managed instance group.
3. Create an autoscaler.
4. Add autoscaled managed instance groups as endpoints.

Step 1: Create an Instance Template

Instance templates are deployment configurations for VMs that specify various system settings such as hardware configuration and operating system, among others. Creating an instance template does not automatically create an instance. An instance group manager creates instances in a managed instance group using the template. Let us use the `gcloud` cli tool to create an instance template using the default values:

```
$ gcloud compute instance-templates create debian-template
Created [https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/instanceTemplates/debian-template].
NAME            MACHINE_TYPE  CREATION_TIMESTAMP
debian-template n1-standard-1 2015-01-26T21:38:14.412-08:00
```

The following are the default values for the various parameters. Using the default values has its pros and cons, as discussed in a moment:

- *Machine type*: n1-standard-1
- *Image*: The latest Debian backports image
- *Boot disk*: A new boot disk named after the VM
- *Network*: The default network
- *IP address*: An ephemeral external IP address

Recall that the purpose of creating instance templates is for the autoscaler to use them to create replica VMs in a managed VM. Suppose that a managed instance group contains only one VM under normal workloads. After a few weeks, the autoscaler detects an increased workload and creates a second VM based on the instance template. However, during the time between the creation of the first and second VMs, the Debian distribution is updated, and the "latest" refers to a newer software release version compared to the first VM. This creates variations between the VMs and may lead to unexpected and undesirable outcomes. Hence, it is recommended that you create a deterministic instance template using the custom instance template creation options. Here is an example command:

```
$ gcloud compute instance-templates create example-template-custom \
--machine-type n1-standard-4 --image debian-7 --boot-disk-size 250GB
```

You should be aware of two move caveats while using the instance template. These revolve around the usage of PDs and network IPs. In Compute Engine, a PD can be mounted in read-write mode in a single VM instance or read-only mode in more than one VM instance. Hence, if you specify a custom disk image by name, the VM template can be used to make only one VM instance—negating the purpose of the autoscaler. In order to use customized software in the new VMs, you can either use startup scripts that install required

software or create snapshots from a master VM and use it to create new PDs. The same logic applies to IP address. In Compute Engine, a static IP address can be attached to only one VM. Hence, if you mention static IP addresses as part of the instance template, you cannot use it to create more than one instance. Therefore, it is best to use ephemeral addresses in the instance template specification.

You can list, describe, and delete instance templates using the following commands:

```
$ gcloud compute instance-templates describe debian-template
<snip>

$ gcloud compute instance-templates list
NAME            MACHINE_TYPE  CREATION_TIMESTAMP
debian-template n1-standard-1 2015-01-26T21:38:14.412-08:00

$ gcloud compute instance-templates delete example-template
<snip>
```

Step 2: Create a Managed Instance Group

After you have created an instance template, you can use it to create an instance group manager to manage the instance group. Here is an example command:

```
$ gcloud preview managed-instance-groups --zone ZONE create GROUP \
--base-instance-name BASE_INSTANCE_NAME --size SIZE \
--template TEMPLATE [--target-pool [TARGET_POOL ...]]
```

Let us dissect this example's various options:

- `GROUP`: The name of the managed instance group
- `BASE_INSTANCE_NAME`: The name prefix of all instances that are created in the managed instance group
- `SIZE`: The minimum number of instances in the managed instance group
- `TEMPLATE`: The instance template created in the previous step

You can use this example template to create a new managed instance group, as follows.

```
$ gcloud preview managed-instance-groups --zone asia-east1-a create debian-group \
--base-instance-name debian7c --size 1 --template debian-template
Managed instance group debian-group is being created. Operation: \operation-1422345108442-
8e29ad58-62a0-4a7e-b379-460d2af08e46
```

Let us list all the managed instance groups in this zone and describe the instance group `debian-group`:

```
$ gcloud preview managed-instance-groups --zone asia-east1-a list
https://www.googleapis.com/replicapool/v1beta2/projects/cloud-platform-book/
zones/asia-east1-a/instanceGroupManagers/debian-group

$ gcloud preview managed-instance-groups --zone asia-east1-a describe debian-group
---
baseInstanceName: debian7c
creationTimestamp: '2015-01-27T07:51:48.272Z'
```

```
currentSize: 1
description: ''
fingerprint: 5qidxR57jJo=
group: https://www.googleapis.com/resourceviews/v1beta2/projects/cloud-platform-book/
zones/asia-east1-a/resourceViews/debian-group
id: '3800371457892609647'
instanceTemplate: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/instanceTemplates/debian-template
kind: replicapool#instanceGroupManager
name: debian-group
selfLink: https://www.googleapis.com/replicapool/v1beta2/projects/cloud-platform-book/
zones/asia-east1-a/instanceGroupManagers/debian-group
targetSize: 1
```

To see the number and names of the instances in this instance group, you can use the `get` option of the `instance-groups` command to list the instances:

```
$ gcloud preview instance-groups --zone asia-east1-a get debian-group
---
creationTimestamp: '2015-01-27T07:51:48.272Z'
description: 'This instance group is controlled by Instance Group Manager ''debian-group''.
  To modify instances in this group, use the Instance Group Manager API: https://developers.
google.com/compute/docs/instance-groups/manager/'
fingerprint: dfaa54cf583111db
kind: resourceviews#resourceView
name: debian-group
network: https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
global/networks/default
resources:
- https://www.googleapis.com/compute/v1/projects/cloud-platform-book/
zones/asia-east1-a/instances/debian7c-vpwn
selfLink: https://www.googleapis.com/resourceviews/v1beta2/projects/cloud-platform-book/
zones/asia-east1-a/resourceViews/debian-group
size: 1
```

Step 3: Create an Autoscaler

After you have created your managed instance group, you can create the autoscaler using the `gcloud` CLI tool. The following command template shows the important parts of the command:

```
$ gcloud preview autoscaler --zone ZONE create AUTOSCALER \
    --max-num-replicas MAX_NUM \
    --min-num-replicas MIN_NUM \
    --target INSTANCE_GROUP_MANAGER
```

Here are the details of each part of the command:

- `ZONE`: The same zone where the instance group manager was created.
- `AUTOSCALER`: The name of the autoscaler.

- `MAX_NUM`: The maximum number of VM replicas the autoscaler creates. This is a very important metric and has many indirect effects. You see an example scenario in a moment.
- `MIN_NUM`: The minimum number of VM replicas the autoscaler maintains. This is an optional argument, and the default value is 2. If your web app is not expected to receive a lot of traffic at normal times, you can set this to 1.
- `INSTANCE_GROUP_MANAGER`: The name of the instance group manager created in the previous step.

Suppose an autoscaled managed instance group is attached as an HTTP load balancer backend. Heavy network traffic has caused the load balancer to add new instance replicas, and `MAX_NUM` has been reached. Even if the load further increases, the load balancer won't add any new instance replicas. Let us say the utilization metric of the instance group has crossed the target utilization metric. At this stage, the load balancer shifts traffic away from this instance group to other instance groups. Depending on the setup, this may result in more network latency for customers based in some geographical regions, or in part of the web app experiencing more latency if content-based HTTP load balancing is used. Hence, it is important to understand the consequence of using a high value compared to a low value.

Step 4: Add Autoscaled Managed Instance Groups as Endpoints

This step is similar to what you did for both the classic network load balancer and the HTTP load balancer. Instead of using fixed-size instance groups, you use a managed instance group as the endpoint. In the case of a network load balancer, you add this managed instance group to the target pool; in the case of an HTTP load balancer, you attach the managed instance group as a backend.

Using the combination of the autoscaler and an HTTP load balancer, it is easy to create a scalable architecture for a web site that uses the HTTP protocol only. However, if you need to support HTTPS and other TCP protocols, you can use the combination of the autoscaler and the network load balancer. The latter has regional scale, and the former is global-scale architecture. This way, depending on the needs of your web application, your organization, and the geographical distribution of your customers, it is easy to architect a web site to handle large and spiky traffic automatically using Compute Engine on the Google Cloud Platform.

Summary

In this chapter, you learned about Compute Engine, Google Cloud Platform's Infrastructure-as-a-Service product,. The Compute Engine platform consists primarily of high-performance VMs, PDs, and networks. You learned about each of these components in detail and how they are interdependent with one another.

Following this, you looked at several practical examples of deploying high-performance VMs on Compute Engine. You started small, by deploying a single VM; and then you gradually upgraded the deployment by introducing the regional-scale network load balancer and the global-scale HTTP load balancer, which can load-balance a fixed backend of VMs. Next, you learned about making the backend dynamic; it autoscales, depending on the transient load factor, by adding resources when needed and shedding excess to achieve cost savings. This chapter has given you a foundation in Compute Engine and will enable you to build more sophisticated and customized native cloud architectures by using Compute Engine, integrating with other Cloud Platform products, and consuming public Google services.