TÉCNICO LISBOA

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

1 Introduction

The objective of the experiments in this Lab assignment series is to continue learning about the deployment of Cloud-based Infrastructure, but now on a Public Cloud, the **Google Cloud Platform (GCP)**, using as Automation Engines **Terraform**, an *orchestration* tool for building, changing, and versioning infrastructures, together with the **Ansible** Configuration Management tool for configuring the compute instances of the infrastructure. Both are outstanding **Infrastructure as Code** products used in the "**DevOps/GitOps**" world that can be used to deploy repeatable environments that meet a vast range of complexity requirements.

2 Google Cloud Platform

The Google Cloud Platform consists of a set of resources, such as Compute products (scalable, high-performance virtual machines, Kubernetes clusters for Container Orchestration, event-driven serverless functions, etc.), Storage and database products (secure object storage, fully-managed MySQL and PostgreSQL database service, etc.), and other Services (cloud load balancing, cloud VPN, Virtual Private Clouds (VPC), managed Spark and Hadoop data processing, etc.), as shown in Figures 1 and 2, that are provided from Google's Data Centers around the globe.

Each Data Center location is in a global region. Regions include Central US, Western Europe, and East Asia. Each region is a collection of zones, which are isolated from each other within the region.

Figure 1: Google Cloud Platform services - 1

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

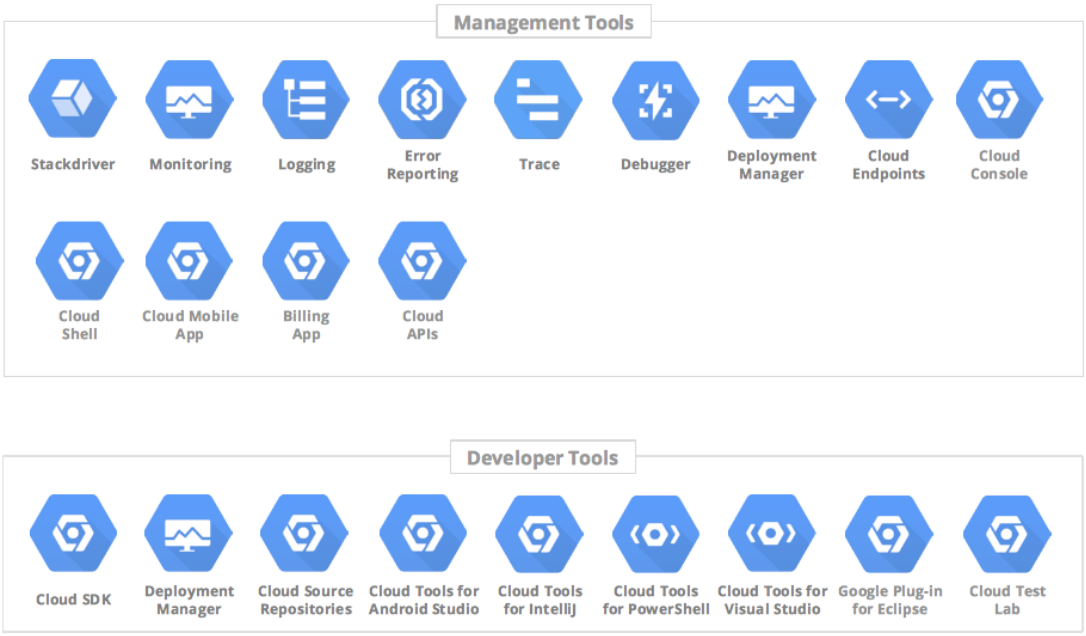

Figure 2: Google Cloud Platform services - 2

2.1 Access to the GCP and Budget

Public cloud providers, such as Google Cloud Platform, charge users per resource used and per time. Therefore, the number of instances created, the uptime, the network components, packet transmissions, disk used and IO, are some of the items charged by the provider. The cost is proportional to the resource utilization.

The Google Cloud Platform Education Grant for the 2020/2021 Academic year, allows students of Instituto Superior Técnico, Universidade de Lisboa to access and use the Platform "freely" during the academic year.

For that purpose, students need to Register in the Platform and observe the following conditions:

1. Each student of the AGISIT course, is assigned (individually) a **Coupon** with a **Face Value of $50.00 USD**, without requiring to register any payment method;

2. In order to Register, Students MUST be using their `@tecnico.ulisboa.pt` email domain, i.e., MUST be signed-in in a browser with their Institutional email account of `@tecnico.ulisboa.pt`, and no other!;

3. By the time you are reading this Guide, you should have already received by email from the AGISIT course, a special **URL** that is necessary to request your Google Cloud Platform Coupon. During the process, as illustrated in Figure 3, you will be asked for your name and email address, which **MUST match the domain** `@tecnico.ulisboa.pt`, after which a confirmation email is sent to you with a Coupon Code. You can only request **ONE code per unique email address**;

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

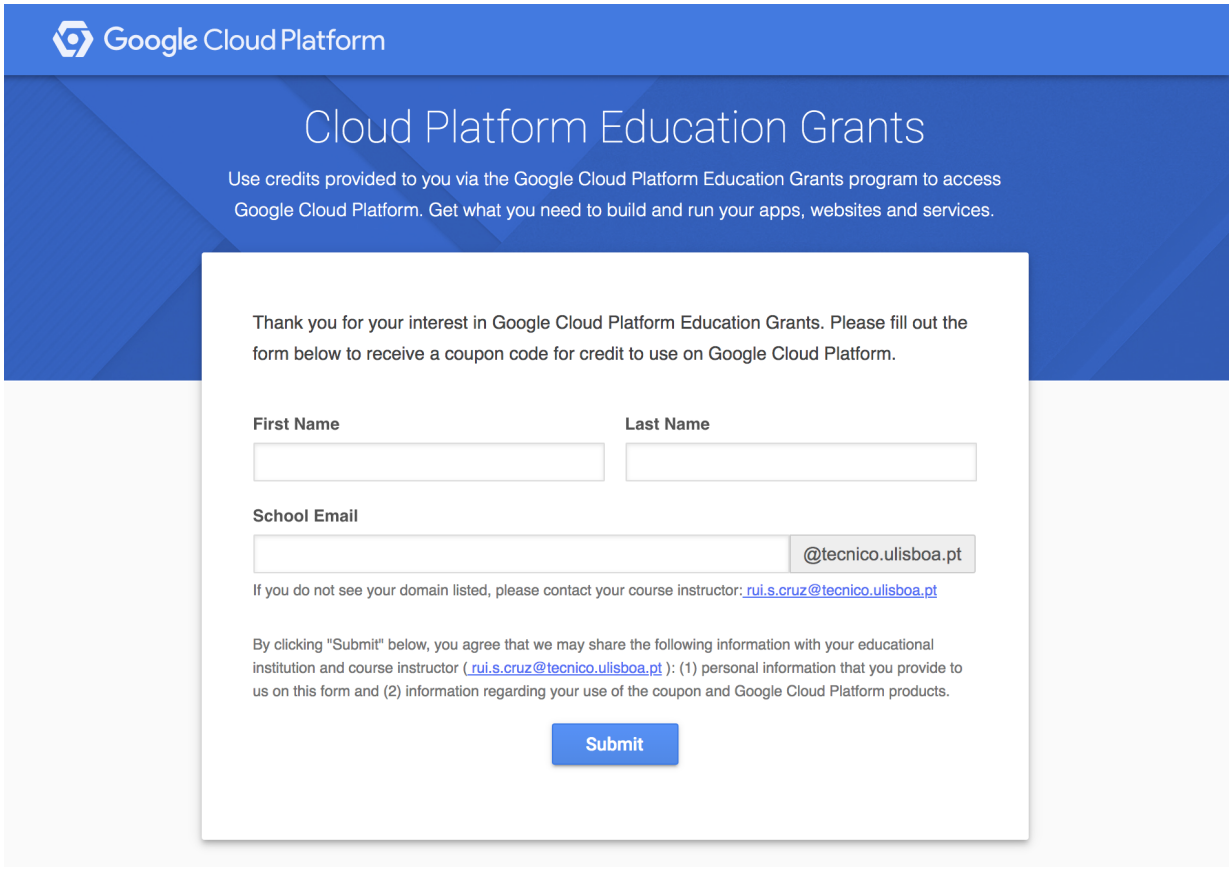

Figure 3: Google Cloud Platform - Request Coupon

4. To comply with previous requirement, grab the provided Coupon Code received in email and then **logout from any account** you may be using in any browser, and **sign in with the** `@tecnico.ulisboa.pt` **email account**;

   IMPORTANT: Be very carefull in these steps, as you may fail to redeem the Coupon in case you are not signed-in with the `@tecnico.ulisboa.pt` email account. Even if you succeed redeeming the Coupon to your private account, it will not allow you to access the Google Cloud Platform with the resources necessary for the Labs. In such case, THERE IS NO WAY TO CORRECT THE ERROR, nor to request another Coupon.

5. Open the the following URL `https://console.cloud.google.com/education`, and paste your Coupon, as illustrated in Figure 4;

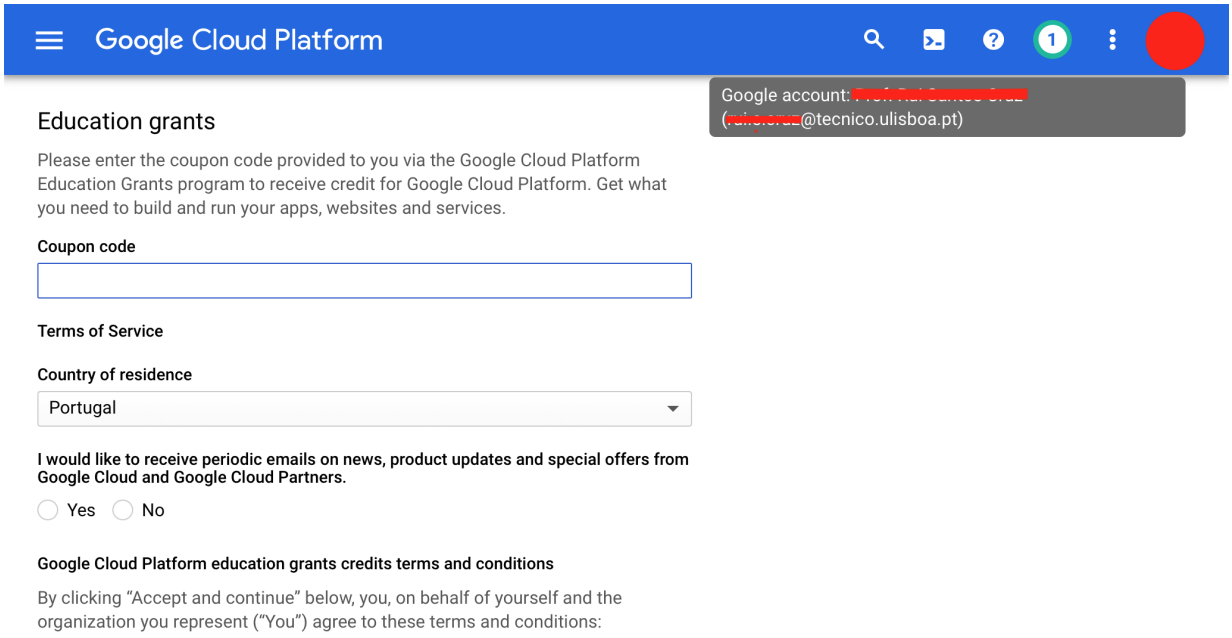

Figure 4: Google Cloud Platform - Redeem Coupon

6. After pasting the Coupon code, the page will confirm that it could redeem the Coupon to the Course "Management and Administration of IT", as illustrated in Figure 5, and you just need to Accept and then continue to the Google Cloud Platform Console; If you agree to the terms, click Accept and continue. The credit is added to your account for the course;

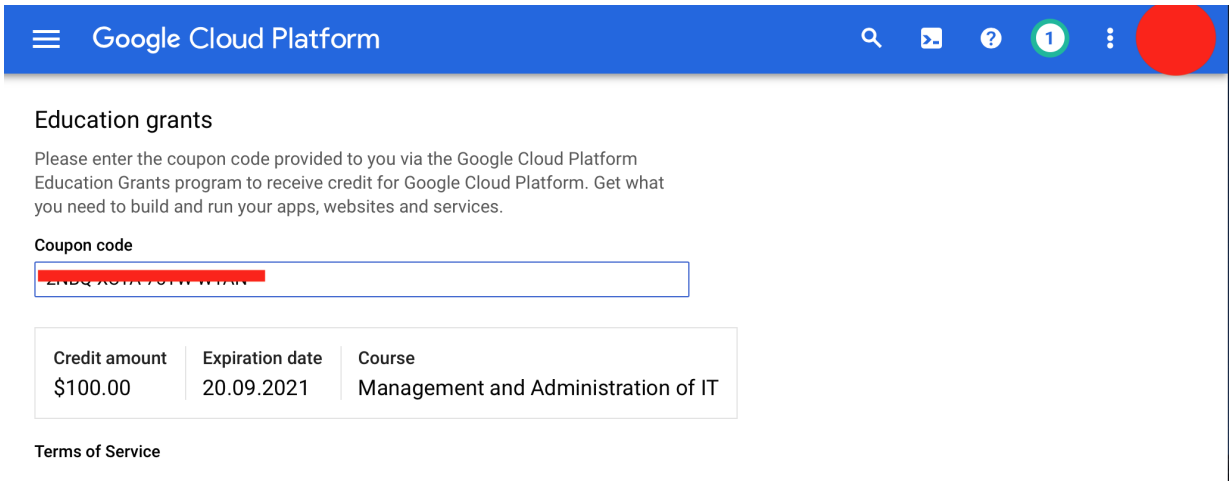

Figure 5: Google Cloud Platform - Accept Coupon

7. At the end of the specified duration for the Coupon (Expiration Date), any unused credit expires and the student's course Cloud Billing account is shut down automatically. A Coupon credit **can be used only for the course it was issued for**;

8. In each account you can create Projects (limited number) and share a Project with other **members of the team** (i.e., with email address matching the domain `@tecnico.ulisboa.pt`), if needed.

2.2 Creating a Project and Exploring the GCP Console

Your first task is to create a Project. For that purpose select on the top Menu Bar the Organization/Projects drop down button, that will open a window for selecting and/or creating a Project, as illustrated in Figure 6.

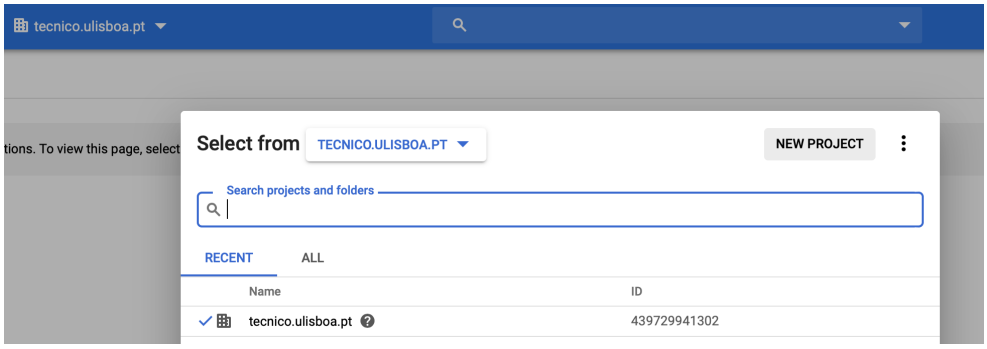

Figure 6: Google Cloud Platform - Create a Project

Select **NEW PROJECT** and give it a name, following the pattern **AGISIT-2021-AAAA-XXXXX**, where **AAAA** can be for example **website**, or **kube** and **AAAA** is your IST ID. The system will then take some time (a minute or more) to create the Project. When complete you will enter the Project Console (Dashboard).

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

Preliminary Notes

One nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is **not recommended** to apply this setup to a virtual machine (**nested virtualization**), although possible, as the configuration requires access to the Hypervisor environment (mandatory Virtualbox) in the host system.

Before proceeding you should verify if you have a "clean" environment, i.e., no Virtual Machine "instances" running (using precious resources in your system), or inconsistent instances in Vagrant and Virtualbox.

For that purpose run the `vagrant global-status` command and observe the results (as in the following example):

```
:~$ vagrant global-status
id       name      provider   state    directory
-------------------------------------------------------------------
28fb48a  mininet   virtualbox poweroff /Users/x/Projects/mininet
f0ccec2  web1      virtualbox running  /Users/x/Projects/multinode
f09c279  web2      virtualbox running  /Users/x/Projects/multinode
```

In the above example, you can observe that there are three Virtual Machines, being the first "mininet", which is **powered off**, but two "web" servers **still running**. It is **advisable to halt those VMs** if running, and then eventually **clean and destroy the VMs** if not needed anymore.

Note: Avoid copying text strings from the examples or configurations illustrated in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

3 Submitting the Results of the Experiments

The experiments that you will execute in this LAB will either produce results that you need to report or from which you will be asked questions about the execution of some procedures. In order to report the results you will achieve, proceed as follows:

3.1 General Procedure

On your system, be it Microsoft Windows, Linux or macOS, it is advisable to be working in a **Projects folder not located under the User Profile folder** (i.e., /Users/username/). That folder will contain the **Infrastructure folders** that will be managed by **Git** for versioning (i.e, will contain Repositories). You should also have a **work folder** to contain the necessary files for reporting the results of the experiments. These files may be screenshots, code snippets, text documents, system logs, etc.

It is advisable to give specific and good identifying names to those files, following what will be asked to report. For example, you may be asked to take several screenshots during the procedures, and so, rename those screenshots to the specific items being requested in the assignment. It is also advisable to collect results in a text file or document file while doing the experiment, so that you will have those results when you will need to submit them in the online assignment form.

The procedure for submission is quite simple:

1. This Lab assignment MUST be reported up to the **Report Due** date;
2. In the Moodle (`https://moodle.dei.tecnico.ulisboa.pt/login/index.php`) for the AGISIT course, you will find an **Assignment** for reporting the results from this specific Lab experiment.
3. The Assignment Link opens a **TSUGI CLOUD Web Form** containing the Questions to be answered;
4. The Assignment **Due Date** corresponds to the date it MUST be considered completed.
5. When you are prepared with the requested materials (screen-shots, command line outputs, developed code, etc.) you will submit the items into the respective Exercise Form question boxes of the Assignment;
6. At the end of the Exercise Form you may comment your submission, or respond to feedbacks received (quoting them);
7. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Moodle Assignment Link;

8. Please note that this process involves a **Peer Review** mechanism, in which you will be asked to provide **feedback** (anonymously) to the Reports of your classmates. So, after submitting your own assignment, get back to it to see the Reviews that the system have distributed automatically to you;

3.2 Specific Procedure

For this Lab Experiment you will need to submit the following items:

1. Interpret the content of *terraform-gcp-variables.tf* and *terraform-gcp-provider.tf* and their relashionship, and submit in the TSUGI Form where asked;
2. Capture a Screenshot of your GCP Billing tab, showing clearly the Promotional Credit you received, and submit in the TSUGI Form where asked;
3. Capture a Screenshot of your GCP Dashboard, showing clearly the Project Info data, and submit in the TSUGI Form where asked;
4. After successfully executing “terraform apply” provisioning the instances, run “terraform refresh” and post the output in the TSUGI Form where asked;
5. Post the output of running the Ansible Playbook for the initial configuration, in the TSUGI Form where asked;
6. Post the output of running the Ansible Playbook for the final configuration with 5 webservers, and submit in the TSUGI Form where asked;
7. Capture a Screenshot of the Load Balancer Statistics for a cluster of 11 webservers, and submit in the TSUGI Form where asked;
8. Capture a Screenshot of the Load Balancer Statistics for a cluster of 5 webservers, and submit in the TSUGI Form where asked;
9. Interpret the results from the Benchmarker tool and also from the statistics page of the Load Balancer comparing the two scenarios (11 servers and 5 servers for the same load), and submit in the TSUGI Form where asked;
10. Capture a Screenshot of the GCP Dashboard ACTIVITY after having completed the assignment (after terraform destroy), and submit in the TSUGI Form where asked;

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form you can find in Moodle for this course. No other type of submission will be considered.

4 Tools to Use

For this Lab experiment you will make use of **Terraform**, **Ansible** and also the **Google Cloud SDK**.

4.1 Terraform

Terraform (Figure 7) is a Provisioning and orchestration tool for building, changing, and versioning infrastructures safely and efficiently.

Figure 7: Terraform by Hashicorp

Terraform is strictly Declarative (i.e., the "desired state" for the Infrastructure is expressed in Configuration files). The Configuration files describe to **Terraform** the components that are needed to run, either for a single application or for an entire datacenter. **Terraform** codifies APIs into declarative objects in configuration files for the desired target environment, and if that environment changes (configuration drifts), it will be corrected the next time "Terraform Apply" is run.

For that purpose **Terraform** generates an **execution plan** describing what it will do to reach the desired state, and then executes the plan to build the described infrastructure. As the configuration changes, **Terraform** is able to determine what has changed and then create incremental execution plans to be applied.

The Infrastructure is described in Configuration files using a high-level syntax named HCL (Hashicorp Configuration Language) which is JSON-compatible. This allows a blueprint of a datacenter to be versioned and treated as would any other code. Additionally, the defined infrastructure can be shared and re-used. One of the great features of **Terraform** is that it supports multiple Providers, meaning that it is possible to create a multi-cloud infrastructure and manage it efficiently, avoiding being locked-in to one service[1]. The main parameters of the **Terraform** command are as follows:

```
Usage: terraform [--version] [--help] <command> [args]

apply          Builds or changes infrastructure
destroy        Destroy Terraform-managed infrastructure
init           Initialize a Terraform working directory
output         Read an output from a state file
plan           Generate and show an execution plan
refresh        Update local state file against real resources
show           Inspect Terraform state or plan
```

[1] Assuming that no provider exclusive feature is used.

```
validate      Validates the Terraform files
version       Prints the Terraform version
```

4.2 Ansible

Ansible is a Configuration Management tool for the automatic deployment of IT infrastructures, essentially used to install software and configure compute instances in a process that can be called "Full-Stack Automation" (see Figure 8). In our lab experiments it will be used to configure the compute instances previously deployed by **Terraform** with applications and services.

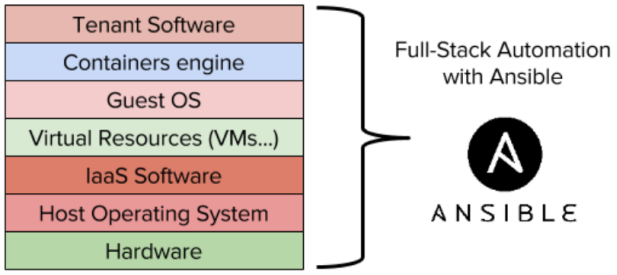

Figure 8: Full-Stack Automation in OpenStack with Ansible

The three bottom layers of Figure 8, form the core resources of the **VMCloud** Platform (OpenStack-based). Above them are the Service components made available to end users (virtual resources, Operating Systems, Container Engines). On top of the Stack, there are the Tenant-defined services needed to create the virtual systemss where the applications will reside.

Ansible is somewhat of a hybrid in terms of the way it performs its actions, between a "procedural" and "declarative" language as it performs *ad-hoc* commands that allow for procedural-style configuration (in the concept of idempotency, i.e., making multiple identical requests having the same effect as making a single request) or use most of its modules that perform declarative-style actions.

Ansible provides "modules" to manage every layer of the infrastructure. This is somehow true even for the networking hardware. It is not, however, the most efficient tool for creating just the infrastructure in a Cloud environments, and that is the reason we will use **Terraform** for that purpose.

4.3 Google Cloud SDK

Google Cloud SDK is a set of tools that you can use to manage resources and applications hosted on Google Cloud Platform. These include the **gcloud**, **gsutil**, and **bq** command line tools. A comprehensive guide to the **gcloud** CLI can be found in `https://cloud.google.com/sdk/gcloud/`.

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

5 Create a Google Cloud fully functional Web Service

The end result of of this lab experiment will be a fully functioning web infrastructure, composed by several web servers (implemented with ***nginx***) and one Load Balancer (implemented with ***haproxy***). We will use **Terraform** to deploy the infrastructure in Google Cloud, while use **Ansible** afterwards to install all of the required packages, deploy configuration files, and start the correct services in the instances of the infrastructure.

5.1 Step 1: Preparing the Deployment

In a temporary area of your system (Downloads, for example), clone the following repository:

```
https://git.rnl.tecnico.ulisboa.pt/AGISIT-20-21/gcpcloudfront.git
```

Go to the local Repository for your Projects, `projects-team-XX` and then to folder **gcpcloudfront** and copy the files of the repository you just cloned (except specific git hidden files/folders), and verify that you have the following file/folder structure in **gcpcloudfront** folder:

```
.
|__ gcpcloudfront
    |__ ansible-gcp-servers-setup-all.yml
    |__ ansible-load-credentials.sh
    |__ ansible.cfg
    |__ gcphosts
    |__ templates
    |   |__ haproxy.cfg.j2
    |   |__ index.html.j2
    |__ terraform-gcp-networks.tf
    |__ terraform-gcp-outputs.tf
    |__ terraform-gcp-provider.tf
    |__ terraform-gcp-servers.tf
    |__ terraform-gcp-variables.tf
```

The **gcploudfront** folder is where all the definition files for the Google Cloud infrastructure reside, allowing you to run the tasks that successively deploy the infrastructure and configure the compute instances of the Load Balanced Web site.

Before continuing, you need to modify the `bootstrap-osmgmt.sh` file to replace the method for installing the Google Cloud SDK, due to a recent change in the process, by eliminating or commenting the lines with **curl** and **bash** commands, and replacing with the **snap** command as follows:

```
# Install Google Cloud SDK non-interactive
sudo snap install google-cloud-sdk --classic
```

You can now start your Management node with `vagrant up --provision` in order to refresh the installation of all needed tools. When ready, confirm that the **osmgmt** VM is reachable by pinging its IP address (as defined in the Vagrantfile). After booting up, open the console to the Management node with `vagrant ssh`.

Due to eventual updating of more recent package versions, after entering the Console, you may find a notice of *** **System restart required** ***, due to some kernel module update. Just exit the Console and do `vagrant reload`.

5.2 Step 1: Generating the Google Cloud credentials

Start by creating the security keys using GCP Console. For that purpose you need to **ENABLE APIS AND SERVICES** for each Project, by choose **API & services** and next selecting the **Dashboard**, where you can see a button on the top menu for enabling, as illustrated in Figure 9.

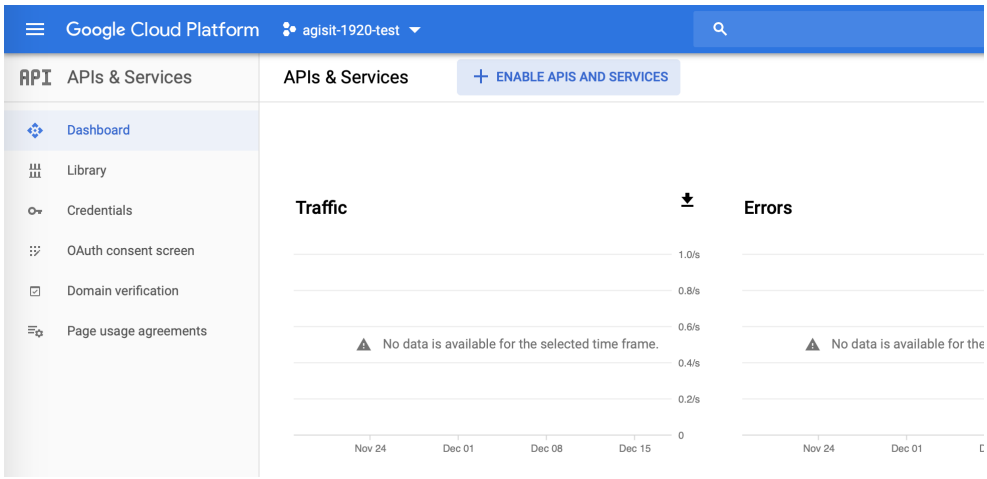

Figure 9: API & Services Dashboard

Selecting that button opens a new window for selecting the type of API (as in Figure 10). in that window search for Compute Engine API, select it and then click ENABLE.

When the API is enabled (it may take some time...) you can then access **API & services** and next **Credentials**, in order to create the necessary file.

You may then see that a **Service account** for the **Compute Engine default service account** is created, and so you need to select the check box of that **Service account** and then select on the right side Manage Service Account, which will open a new window, as illustrated in Figure 11.

There, in **Actions**, You select **Create key** and then the **JSON** checkbox, as illustrated in Figure 12, which will download to your computer a Credentials file.

Save the file to the **gcpcloudfront** folder.

The Credentials file has your keys to access the GCP service and should be kept safe, and not shared in a git repository.

AGISIT 20/21	LAB ASSIGNMENT	Number:	8
Cloud-based Infrastructures		Issue Date:	23 Nov 2020
IaaS (Infrastructure as a Service) - GCPCloud - Part 1		Report Due:	30 Nov 2020
Author: Prof. Rui Cruz		Due Date:	02 Dec 2020

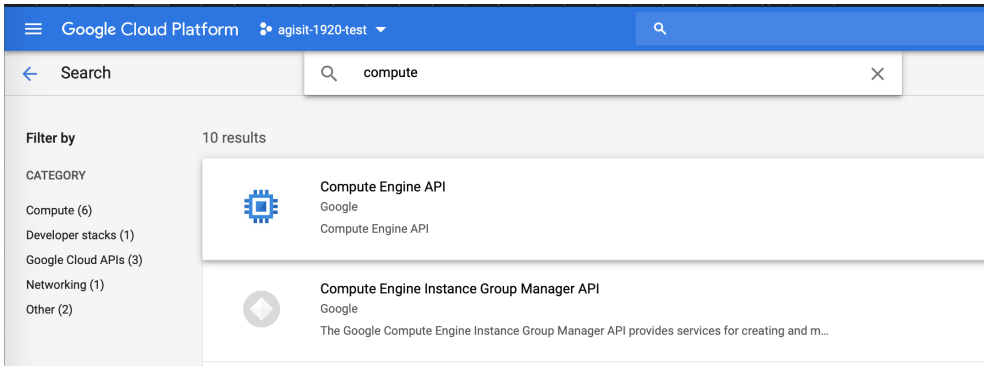

Figure 10: Selecting and Enabling the Compute Engine API

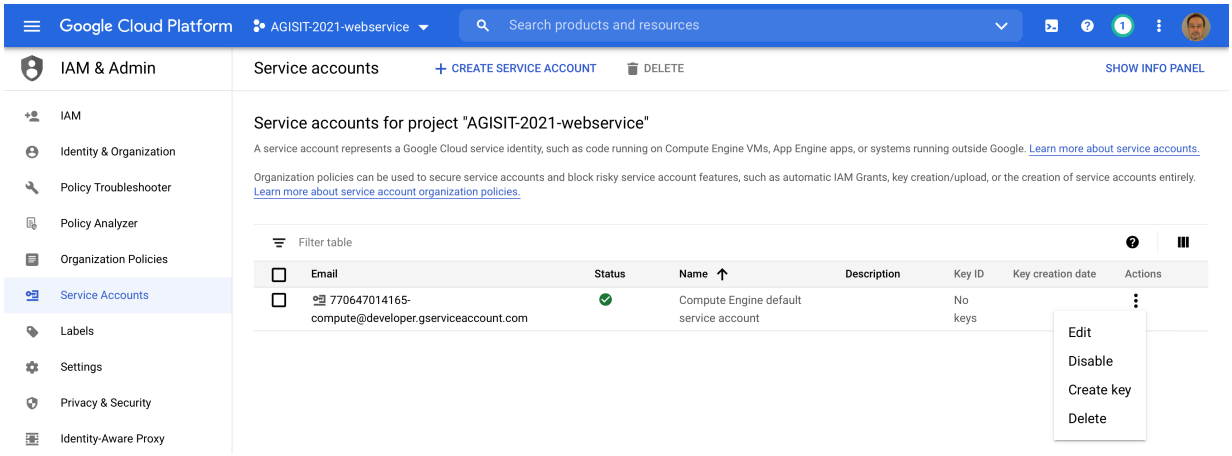

Figure 11: Manage Service Account screen

The next step is to authorize the Google Cloud SDK (client API) in order to allow controlling the GCP resources from interfaces (APIs) rather than the Dashboard.

This API can be invoked with the command **gcloud** on the console of your vagrant Management VM (**osmgmt**) in folder `gcpcloud-tenant`.

The API must be **authorized** before using it and so, in that `gcpcloud-tenant`, run the command, as illustrated hereunder, and **paste its output** (the whole string starting with https://) in a web browser in your host system. You will be requested to specify your `@tecnico.ulisboa.pt` email address in order to ensure that you can be authenticated to the GCP account.

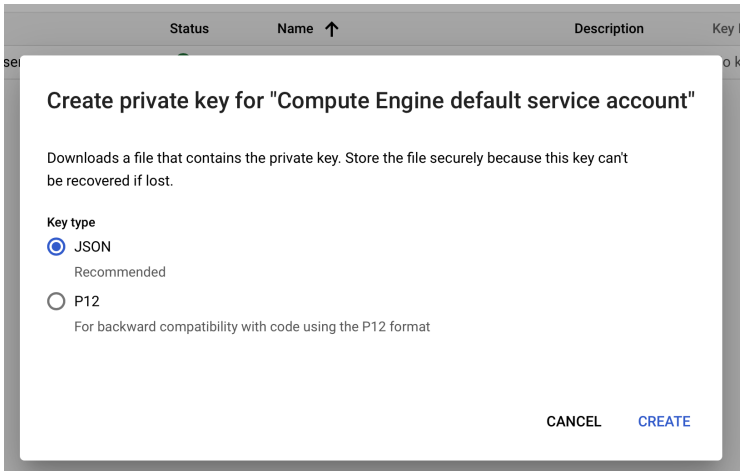

Figure 12: Creating the Service Account Key credentials

```
vagrant@osmgmt:~/gcpcloud-tenant$ gcloud auth login
Go to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?redirect_uri=urn%3Aietf%3
       Awg%3Aoauth%3A2.0%3Aoob&prompt=select_account&response_type=code&
       client_id=.....www.googleapis.com%2Fauth%2Faccounts.reauth&
       access_type=offline

Enter verification code:
```

Next, as in Figure 13, copy the response code, and go back to the **osmgmt** console and paste that code in the field **Enter verification code:**. This authorization is required for tasks such as connecting to the instances via SSH.

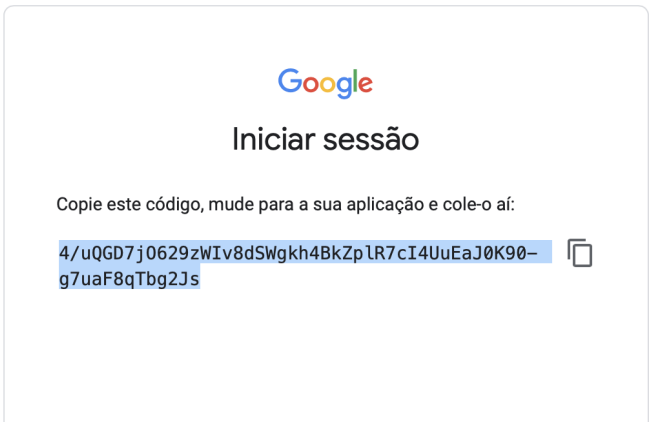

Figure 13: GCP auth verification code

5.3 Step 2: Provisioning the infrastructure

In this step-by-step approach you will be able to create the infrastructure in the GCP Public Cloud. Go to the `gcpcloud-tenant` folder, and there you have several **.tf** files.

Look into the **terraform-gcp-variables.tf** and **terraform-gcp-provider.tf** files and replace some of the TAGs the field values corresponding to your project, as well as the name of the Credential files from Google. Check `https://www.terraform.io/docs/providers/google/index.html` to find more information about how to configure these files.

The **terraform-gcp-variables.tf** defines variables that are used by other files. Edit this file and replace **XXXXX** with the value you gave for the ID of your project (the exact name of your project in the GCP Console), for example:

```
variable "GCP_PROJECT_NAME" {
    default = "agisit-2021-XX"
}
```

You also need to obtain the **image** names for your instances, to replace them in the **terraform-gcp-servers.tf** file. For that purpose, go to `https://cloud.google.com/compute/docs/images` and search for **Ubuntu** images, selecting the most recent Ubuntu Bionic.

With that step done, you can now initialize Terraform, in order to eventually satisfy some plugin requirements, and then validate the configuration files:

```
vagrant@osmgmt:~/gcpcloud-tenant$ terraform init

Initializing provider plugins...
- Checking for available provider plugins ...
- Downloading plugin for provider "google"...
...
Terraform has been successfully initialized!

vagrant@osmgmt:~/gcpcloud-tenant$ terraform validate
```

As you can observe by studying the files `terraform-gcp-servers.tf` and `terraform-gcp-networks.tf`, your infrastructure code is written (or, to be rigorous, the infrastructure is declared), by defining the desired resources and their characteristics.

You just need to make it live in the Google Cloud. **Terraform** "talks" to the Google Cloud APIs and makes sure that the planned infrastructure is always up-to-date with what you code in the configuration files.

Before making it live, you are supposed to create a **Plan**, i.e., a process that compares your assumed current state, with the one declared in those configuration files, using API calls that fetch the current state from the Google Cloud Platform. Once you are happy with the **Plan** output, you **Apply** it, and if any changes are scheduled, they are actually performed.

So, let's go and create the **Plan** with the following command,resulting in a long list something similar to:

```
vagrant@osmgmt:~/gcpcloud-tenant$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
    persisted to local or remote state storage.
------------------------------------------------------------------------
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
....
....

Plan: 6 to add, 0 to change, 0 to destroy.
```

To execute the **Plan** and create the infrastructure, run **terraform apply**, and a result similar to the following will be output:

```
vagrant@osmgmt:~/gcpcloud-tenant$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
.....
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.

Outputs:

balancer = 104.154.133.215
web1 = 35.194.18.167
web2 = 35.192.55.227
web3 = 35.224.58.218
web4 = 35.224.58.219
```

Once the command finishes, **Terraform** will create a new file **terraform.tfstate**. This file keeps the **current state** of the deployed infrastructure and it is a 1:1 mapping of your deployed infrastructure, meaning that if you change/remove resources form the configuration files, it will be removed from your infrastructure and if you add/change resources to the configuration files, when you run the command “apply” those changes will be reflected to the infrastructure.

Note the last lines of the apply command. They print out the public IP addresses of the instances created.

5.4 Step 3: Configuring GCP Instances with Ansible

Now that the infrastructure is created and deployed in Google Cloud, in order for Ansible to access the machines and configure them, there is the need to populate the INVENTORY file **gcphosts**, present in the same project directory and also edit the **osmgmt** system **hosts** file with the IP addresses (retrieved from the output of Terraform) and the names of the servers, obtaining something similar to the following:

```
vagrant@osmgmt:~/gcpcloud-tenant$ sudo nano /etc/hosts
...
vagrant@mgmt:~/gcpcloud-tenant$ cat /etc/hosts
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost   ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
127.0.1.1       osmgmt  osmgmt
100.68.20.102    balancer
100.68.20.101    web1
100.68.20.110    web2
100.68.20.107    webX

vagrant@osmgmt:~/gcpcloud-tenant$ cat gcphosts
# file: gcphosts
# for tenant hosts file

balancer  ansible_host=100.68.20.X ansible_user=ubuntu .....
web1      ansible_host=100.68.20.Y ansible_user=ubuntu .....
web2      ansible_host=100.68.20.Z ansible_user=ubuntu .....
webX      ansible_host=100.68.20.W ansible_user=ubuntu .....
...
```

5.5 Step 4: Test Communication with remote Instances

In order to ensure that you can reach the remote instances created and provisioned in GCP, you should execute two tests. The first one is just a simple PING using ICMP protocol. The other test is an Ansible ad-hoc “Ping-Pong“command. As for the following example, execute those tests to all the Instances that you provisioned with Terraform.

```
vagrant@osmgmt:~/gcpcloud-tenant$ ping balancer
PING balancer (100.64.6.228) 56(84) bytes of data.
64 bytes from balancer (100.64.6.228): icmp_seq=1 ttl=63 time=12.8 ms
...

vagrant@osmgmt:~/gcpcloud-tenant$ ansible balancer -m ping
balancer | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

5.6 Step 5: Configuring the Server Instances

The Playbook for this last step has three Plays. The first Play contains pre-configuration of the server instances, in order to insert the SSH key for "**password less**", and to allow remote operation without requiring direct user input.

The second Play includes the tasks to install the web server program "nginx" in all Web nodes and adapt the corresponding configuration files, ensuring in the end that the web service is duly started.

The third Play includes the tasks to install the "haproxy" program in the Load Balancer server, as well as the corresponding configuration file, ensuring that the Load Balancer service is duly started.

The Playbook to use for this step is the (*ansible-gcp-servers-setup-all.yml*), which is almost identical to the one you have already used in a previous Lab.

To use Ansible, we need to load the credentials generated in Google Cloud. For that, you need to edit the *ansible-load-credentials.sh* in order to insert the JSON file you have downloaded from GCP.

5.7 Step 6: Test Communication with remote Instances

In order to ensure that you can reach the remote instances created and provisioned in GCP, you should execute two tests. The first one is just a simple PING using ICMP protocol. The other test is an Ansible ad-hoc "Ping-Pong"command. As for the following example, execute those tests to all the Instances that you provisioned with Terraform.

```
vagrant@osmgmt:~/gcpcloud-tenant$ ping balancer
PING balancer (100.64.6.228) 56(84) bytes of data.
64 bytes from balancer (100.64.6.228): icmp_seq=1 ttl=63 time=12.8 ms
...

vagrant@osmgmt:~/gcpcloud-tenant$ ansible balancer -m ping
balancer | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

5.8 Step 7: Configure the Server Instances

The Playbook for this last step has three Plays. The first Play contains pre-configuration of the server instances, in order to insert the SSH key for "**password less**", and to allow remote operation without requiring direct user input.

The second Play includes the tasks to install the web server program "nginx" in all Web nodes and adapt the corresponding configuration files, ensuring in the end that the web service is duly started.

The third Play includes the tasks to install the "haproxy" program in the Load Balancer server, as well as the corresponding configuration file, ensuring that the Load Balancer service is duly started.

The Playbook to use for this step is the (*vmcloud-site-servers-setup-all.yml*), which is almost identical to the one you have already used i a previous Lab. So, go ahead and play it:

```
vagrant@osmgmt:~/gcpcloud-tenant$ ansible-playbook ansible-gcp-servers-
    setup-all.yml
```

The tasks in the three Plays will take some time, so be patient and wait a few minutes for them to complete. Analyze the returned information (quite large) to see what was performed.

5.9 Step 7: Test the created Infrastructure

If everything went smoothly, it means that you have now your Project Infrastructure created, with the servers running and providing services.

You can now use a web browser in whatever system to observe the Website, as the Project Infrastructure is available publicly from the Internet, and do the following:

1. Open a web browser, write the URL `http://ip-of-the-balancer` and hit return. You should be connected to the website, through the Load Balancer, and one of the web servers should have served your request (the name of the web server and its internal IP address are shown).

2. Hit the refresh button on the web browser. Did something change? If so, repeat the refresh several times and observe the changes.

3. Open a second browser tab and navigate to the statistics page of the Load Balancer at `http://ip-of-the-balancer/haproxy?stats`.

4. Going back to the first tab, Hit the refresh button on the web browser several times, and then go to the second tab and verify the changes in the counters.

You may also verify the configuration of each server, using the following command (example for the balancer server):

```
vagrant@osmgmt:~/gcpcloud-tenant$  ansible -m setup balancer
```

5.10 Now on Your Own

For this last experiment, you will modify the "running" infrastructure, to expand it (add some new webservers to the cluster).

1. Edit the *terraform-gcp-servers.tf* and other corresponding configuration files to expand the cluster of webservers to 11 instances.

2. In your browser navigate to `http://ip-of-balancer/haproxy?stats` and refresh the page. Capture a screenshot of the current state.

3. Run the Benchmarker tool for the following test: send 100,000 requests, with a concurrency of 10. Interpret the results from the benchmark tool.

   If you are in a Microsoft Windows host open a Git-Bash Terminal and run the Benchmarker tool with the command `sb -n 100000 -c 10 -u http://ip-of-balancer/` . If you are in a macOS or Linux host open a Terminal and run the command `ab -n 100000 -c 10 http://ip-of-balancer/`.

4. In your browser navigate to `http://ip-of-balancer/haproxy?stats` and refresh the page. Capture a screenshot of the state after the benchmark run.

5. Edit the *terraform-gcp-servers.tf* and other corresponding configuration files to expand the cluster of webservers to 5 instances.

6. In your browser navigate to `http://ip-of-balancer/haproxy?stats` and refresh the page. Capture a screenshot of the current state.

7. Run again the Benchmarker tool for the same test: send 100,000 requests, with a concurrency of 10. Interpret the results from the benchmark tool.

6 Finnishing the Experiments

To clean the environment (destroy created networks, security groups and instances) run **terraform destroy**. Make sure the infrastructure has been destroyed by checking with *terraform refresh* and checking it manually on the GCP Dashboard for your Project. Do not forget that **instances deployed and running are billed**, i.e., their usage is accounted **and discounted from you budget!**

When finished the experiment, Stop the Virtual Machine and verify the global state of all active Vagrant environments on the system, issuing the following commands:

```
:~$ vagrant halt
:~$ vagrant global-status
```

Confirm that the status of the VM is "powered off".