TÉCNICO LISBOA

AGISIT 20/21	LAB ASSIGNMENT	Number:	07
Cloud-based Infrastructures		Issue Date:	16 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 3		Report Due:	23 Nov 2020
Author: Prof. Rui Cruz		Due Date:	24 Nov 2020

1 Introduction

This lab experiment is the third of a series exploring deployment of a Private Cloud-based Infrastructure, but now using as Automation Engines **Terraform**, an *Infrastructure Provisioning* 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.

1.1 VMCloud - OpenStack-based Private Cloud of IST

As a recall of the concepts, a private cloud is a cloud platform operated inside a private network. **OpenStack** is an open-source cloud platform that offers virtualisation of *compute*, *storage*, *networking*, and many other resources. Each component in **OpenStack** manages a different resource that can be virtualised for the end user. Separating each of the resources that can be virtualised into separate components, makes the **OpenStack** architecture very modular (see Figure 1).

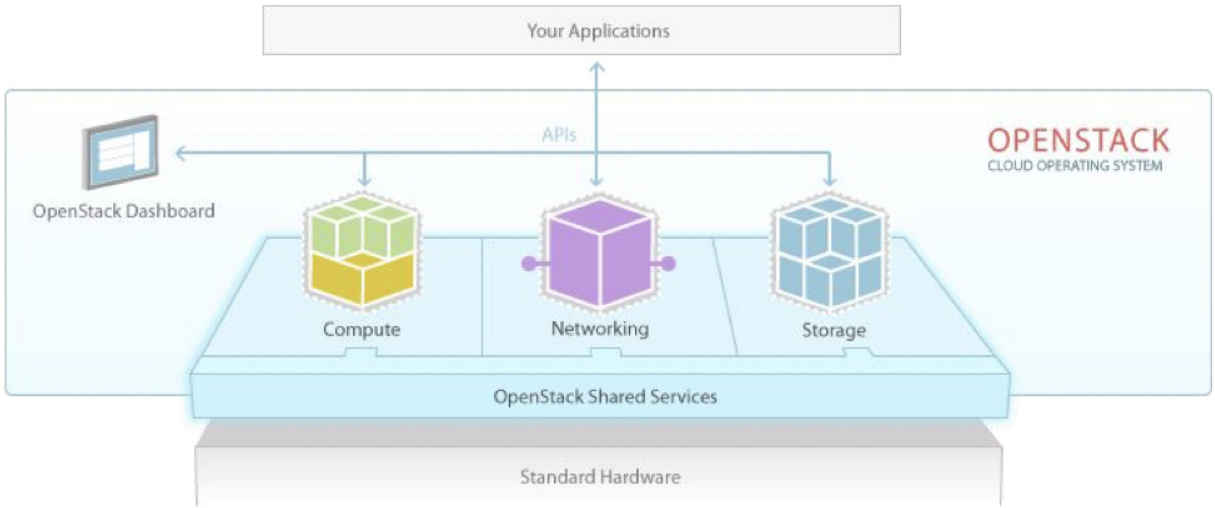

Figure 1: OpenStack component groups

Logically, the key components of **OpenStack** can be divided into four groups: **Control**, **Networking**, **Compute** and **Storage**. The **Control** tier runs the Application Programming Interfaces (API) services, web interface, database, and message bus. The **Networking** tier runs network service agents for networking. The **Compute** tier is the virtualisation hypervisor, with services and agents to handle virtual servers (Machines and Containers). The **Storage** tier manages *block* (Volumes; partitions) and *object*

(containers; files) storage for the **Compute** instances. All of the components use a database and/or a message bus.

VMCloud is an implementation of an OpenStack-based Private Cloud for providing cloud services to the comunity of Instituto Superior Técnico, Universidade de Lisboa.

The **VMCloud** implementation, due to its nature as Academic environment, imposes some "restrictions" in the access to the system and to some OpenStack modules, as well as to broader networking configurations, and to Operating System images for the compute instances.

The access to **VMCloud** Tenants/Projects can only be done when users are connected through **Eduroam** or externally through a **VPN** to the **IST intranet**.

1.2 Terraform

Terraform (Figure 2) is a Provisioning and orchestration tool for building, changing, and versioning infrastructures safely and efficiently.

Figure 2: 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:

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

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

1.3 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 3). In our lab experiments it will be used to configure the compute instances previously deployed by **Terraform** with applications and services.

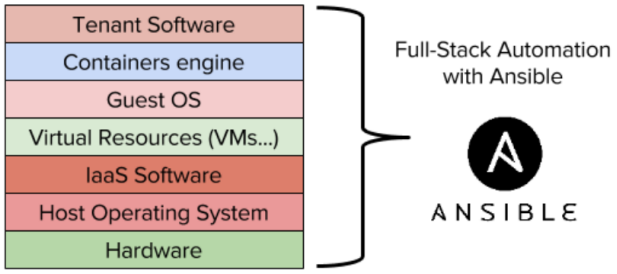

Figure 3: Full-Stack Automation in OpenStack with Ansible

The three bottom layers of Figure 3, 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.

AGISIT 20/21	LAB ASSIGNMENT	Number:	07
Cloud-based Infrastructures		Issue Date:	16 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 3		Report Due:	23 Nov 2020
Author: Prof. Rui Cruz		Due Date:	24 Nov 2020

Preliminary Notes

IMPORTANT: As the **VMCloud** Platform is a Private Cloud system, the Compute instances that you will create, can only be accesed with IPv4 addresses from within the IST campi. This means that, **YOU MUST ESTABLISH a VPN** connection to the IST (if you are accessing from outside) in order to complete this Lab.
For more information on how to establish a VPN, please go to:
`https://si.tecnico.ulisboa.pt/en/servicos/redes-e-conetividade/vpn/`.

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.

2 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:

2.1 General Procedure

On your system, be it Microfoft 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;

2.2 Specific Procedure

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

1. Capture a screenshot of the VMCloud Dashboard showing the **Overview of your Project** after Provisioning the Infrastructure, and submit in the TSUGI Form where asked;
2. Capture a screenshot of the VMCloud Dashboard showing the **Network Topology** of your Project **after creating the instances**, and submit in the TSUGI Form where asked;
3. Post the content of terraform apply log (removing the content of the ssh keys), and submit in the TSUGI Form where asked;
4. Post the content of the “myhosts“file, modified after provisioning the Infrastructure, in the TSUGI Form where asked;
5. Post the content of the /etc/hosts file of the Management node, in the TSUGI Form where asked;
6. Post the output of execution of the Ansible Playbook, and submit in the TSUGI Form where asked;
7. Post the output of ”git log –pretty=oneline” of your repository, and submit in the TSUGI Form where asked;
8. Capture a Screenshot of your browser showing the page of URL of the Balancer http://ip-of-the-balancer/haproxy?stats, and submit in the TSUGI Form where asked;
9. Capture a Screenshot of your browser showing the page of URL http://ip-of-the-balancer, to submit in the TSUGI Form where asked;
10. Capture a screenshot of the VMCloud Dashboard Compute Overview after having executed 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.

3 Working Methodology

The working Methodology will be in Group, based on Coordination and/or division of Tasks among members of each Group.

1. **Commit Changes to the GIT Repository:** Each member of the Group in their own computer, opens a Bash Terminal, changes to the Projects directory cloned from the Remote Repository, and in there executes the following:

   - **First Task:** Commit changes on the local GIT repository with the modified Configuration Files used for Provisioning with Terraform in Part 2 of this Lab:
     - Update the file **.gitignore** to contain the following additions:

```
# General
**/.git/*
# Vagrant specific
.vagrant
# Ansible specific sentitive files
CREDENTIALS.sh
# Terraform specific
# Local .terraform directories
**/.terraform/*
# Exclude all .tfvars files, containing sentitive data.
# These should not be part of version control.
*.tfvars
```

     - If not yet done, add the modified files to the local repository with

```
git add --all
```

     - Commit the changes with:

```
git commit -m "Added modified Conf. Files for provisioning"
```

   - **Second Task:** Each member of a Group, will push their local Repository to the Remote with a Tag (for this part of the Lab) of the form **vmcloud-prov-istXXXXX**, replacing **istXXXXX** with the corresponding IST-ID:

```
$ git tag vmcloud-prov-istXXXXX
$ git push origin vmcloud-prov-istXXXXX
```

2. **Execute the Provisioning and Configuration of the Web System (as explained in Section 4 after which, updates the Remote Repository:** After completing the Configuration steps of this Part 3 of the Lab, push again the local Repository to the Remote (after having committed the changes) with a Tag of the form **vmcloud-conf-istXXXXX**, replacing **istXXXXX** with the corresponding IST-ID::

```
$ git tag vmcloud-conf-istXXXXX
$ git push origin vmcloud-conf-istXXXXX
```

4 Create a fully functional VMCloud Web Service

In Part 2 of this Lab series of experiments you learned how to provision an infrastructure in the VMCloud of IST, using **Terraform**.

In this Lab experiment you will complete the process in two steps, starting by re-creating the base infrastructure with **Terraform**, and then Configuring the software and services in the instances with **Ansible**.

IMPORTANT: As the **VMCloud** Platform is a Private Cloud system, the Compute instances that you will create, can only be accesed with IPv4 addresses from within the IST campi. **YOU MUST ESTABLISH a VPN** connection to the IST (if you are working remotely) in order to complete this Lab.
For more information on how to establish a VPN, please go to:
`https://si.tecnico.ulisboa.pt/en/servicos/redes-e-conetividade/vpn/`.

4.1 Step 1: Preparing the Deployment

In the local Repository for your Projects, `projects-team-XX` verify that you have the following file/folder structure in **projects-team-XX** folder:

```
.
|__ README.md
|__ Vagrantfile
|__ awscloudfront
|   |__ README.md
|__ bootstrap-osmgmt.sh
|__ gcpcloudfront
|   |__ README.md
|__ vmcloudfront
    |__ CREDENTIALS.sh
    |__ README.md
    |__ ansible.cfg
    |__ myhosts
    |__ outputs.tf
    |__ templates
    |   |__ default-site.j2
    |   |__ haproxy.cfg.j2
    |   |__ index.html.j2
    |   |__ nginx.conf.j2
    |__ terraform-provider.tf
    |__ terraform-vmcloud-networks.tf
    |__ terraform-vmcloud-servers.tf
    |__ terraform.tfvars
    |__ vmcloud-site-servers-setup-all.yml
```

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

You can now start your Management node with `vagrant up`. 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`.

4.2 Step 2: Provision the infrastructure with Terraform

Use **Terraform** to Provision the infrastructure, as you did in part 2 of this Lab. In the Management node go to the `vmcloud-tenant` folder, and there you will see several **Terraform** configuration files, with extensions **tf**, as well as YAML files related with **Ansible**.

Confirm that the `terraformr.tfvars` file has the correct configuration for your project in **VMCloud**. If all the data is correct, you can now start the process with the following:

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

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.
    com...
- Downloading plugin for provider "openstack" (1.12.0)...
....
* provider.openstack: version = "~> 1.12"

Terraform has been successfully initialized!
```

In case you get some Error about Providers versions, follow the on-screen recommendations.

After successful initiation, as you can check up your configuration files, to adapt them to your needs, as explained previously, when you have divided your work with your colleagues. For example, you could have split the `terraform-vmcloud-servers.tf` in two, being one for the Load Balancer only and another for the Web servers, such that:

- One group member would have only a `terraform-vmcloud-balancer.tf`
- Other member would have only a `terraform-vmcloud-web.tf`

Independently of your division of tasks in the Group, you can Provision the Infrastructure, staring by creating the **Plan** with the following command, resulting in something similar to:

```
vagrant@osmgmt:~/vmcloud-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

Terraform will perform the following actions:
......
```

To execute the **Plan** and create the infrastructure, run **terraform apply**

```
vagrant@osmgmt:~/vmcloud-tenant$ terraform apply
....
Plan: X to add, 0 to change, 0 to destroy.

.....
Apply complete! Resources: X added, 0 changed, 0 destroyed.

Outputs:

balancer_IP = 100.XXX.XXX.YYY
web1_IP = 100.XXX.XXX.YXY
web2_IP = 100.XXX.XXX.XYX
....
```

The output of **Terraform** can be viewed again with the command **terraform output**.

4.3 Step 3: Configuring VMCloud Instances with Ansible

Now that the infrastructure is created and deployed in VMCloud, in order for Ansible to access the machines and configure them, there is the need to populate the INVENTORY file **myhosts**, 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:~/vmcloud-tenant$ sudo nano /etc/hosts
...
vagrant@mgmt:~/vmcloud-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:~/vmcloud-tenant$ cat myhosts
# file: myhosts
# 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 .....
...
```

4.4 Step 4: Test Communication with remote Instances

In order to ensure that you can reach the remote instances created and provisioned in VMCloud, 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:~/vmcloud-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:~/vmcloud-tenant$ ansible balancer -m ping
balancer | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

4.5 Step 5: 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.

Due to recent changes in some of the tools used, as well as the Operating System of the Instances in VMCloud, you need to modify some lines in the configuration files provided in the Renmote Repository that you have cloned.

The changes are as follows:

- In the haproxy.cfg.j2 template file, change line 56, inthe “hostvars“ to the following variables:

```
<p>({{ ansible_default_ipv4.address }})</p>
```

- In the index.html.j2 template file, change line 19 to the following variables:

```
{{ hostvars[host]['ansible_default_ipv4']['address'] }}
```

- In the Ansible Playbook, disable the tasks:
  write our nginx.conf and **write our /etc/nginx/sites-available/default**.
- In the Ansible Playbook modify the task **deploy website content** in the **dest:** line, as follows:

```
dest: /var/www/html/index.html
```

Having done those changes, go ahead and play it:

```
vagrant@osmgmt:~/vmcloud-tenant$ ansible-playbook vmcloud-site-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.

4.6 Step 4: Test the created Infrastructure

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

You can now use a web browser in whatever system to observe the Web site, as the Infrastructure is available from the IST network (or from the Internet if you establish first a VPN to IST), 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 web site, through the Load Balancer, and one of the web servers should have served your request (the name of the web server and its 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 verify the configuration of each server, using the following command (example for the balancer server):

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

5 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 VMCloud Horizon.

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”.