TÉCNICO LISBOA

AGISIT 20/21	LAB ASSIGNMENT	Number:	06
Cloud-based Infrastructures		Issue Date:	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		Report Due:	16 Nov 2020
Author: Prof. Rui Cruz		Due Date:	17 Nov 2020

1 Introduction

This lab experiment is the second 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).

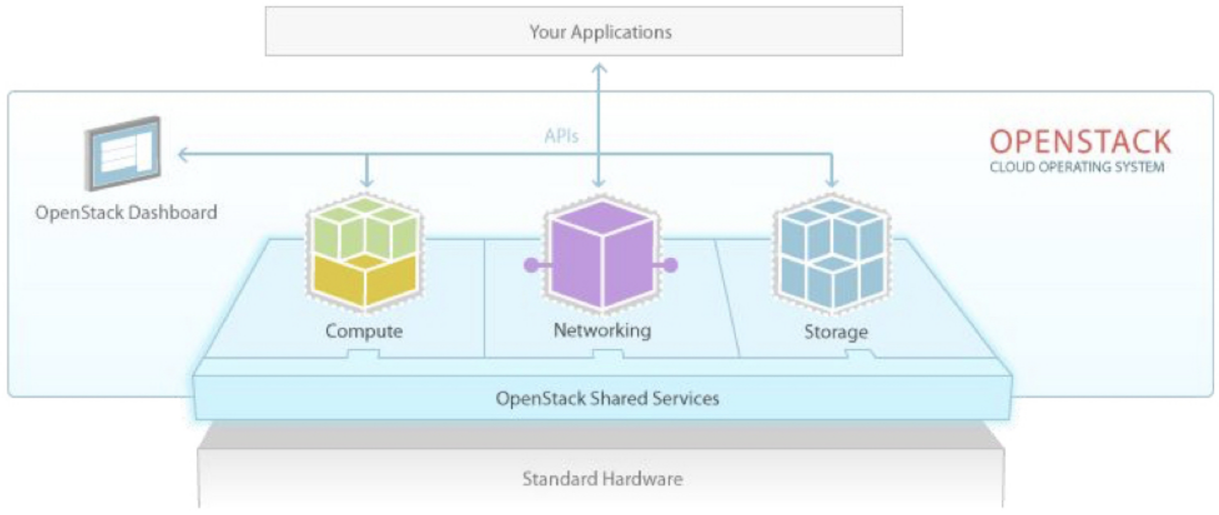


Figure 1: OpenStack component groups

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*

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	06
Cloud-based Infrastructures		**Issue Date:**	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		**Report Due:**	16 Nov 2020
Author: Prof. Rui Cruz		**Due Date:**	17 Nov 2020

(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.

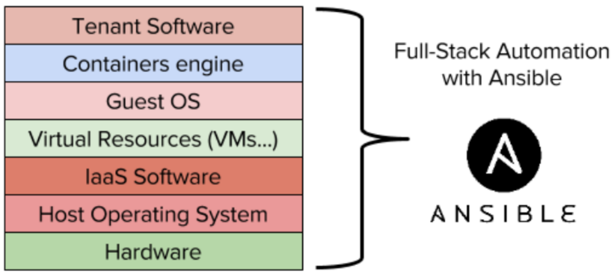


Figure 3: Full-Stack Automation in OpenStack with Ansible

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:**	06
Cloud-based Infrastructures		**Issue Date:**	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		**Report Due:**	16 Nov 2020
Author: Prof. Rui Cruz		**Due Date:**	17 Nov 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:**	06
Cloud-based Infrastructures		**Issue Date:**	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		**Report Due:**	16 Nov 2020
Author: Prof. Rui Cruz		**Due Date:**	17 Nov 2020

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 create a **Projects folder** **not located under the User Profile folder** (i.e., /Users/username/), that will then 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;

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	06
Cloud-based Infrastructures		**Issue Date:**	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		**Report Due:**	16 Nov 2020
Author: Prof. Rui Cruz		**Due Date:**	17 Nov 2020

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. Interpret the content of the file *terraform-vmcloud-servers.tf*, and submit in the TSUGI Form where asked;
2. Interpret the content of the file *terraform-vmcloud-networks.tf*, and submit in the TSUGI Form where asked;
3. Interpret the content of the file *outputs.tf*, and submit in the TSUGI Form where asked;
4. Post the content of the file *versions.tf* in the TSUGI Form where asked;
5. Post the content of the file *terraform.tfstate* in the TSUGI Form where asked;
6. After executing the command *terraform apply* interpret the results, and submit in the TSUGI Form where asked;
7. Capture a screenshot of the VMCloud Dashboard showing the **Network Topology** of your Project **after Privioningg the instances** with Terraform Apply, and submit in the TSUGI Form where asked;
8. Post the content of **terraform apply** log (removing the content of the ssh keys) in the TSUGI Form where asked;
9. Post the content of **terraform destroy** log (removing the content of the ssh keys) in the TSUGI Form where asked;
10. Capture a screenshot of the VMCloud Dashboard Cpmpute 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.

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	06
Cloud-based Infrastructures		**Issue Date:**	9 Nov 2020
IaaS (Infrastructure as a Service) - VMCloud - part 2		**Report Due:**	16 Nov 2020
Author: Prof. Rui Cruz		**Due Date:**	17 Nov 2020

3 Working Methodology

The working Methodology from this lab onward will be in Group, based on Coordination and/or division of Tasks among members of each Group. It is therefore mandatory to consider the following:

1. **Prepare Git (if not yet done:** each member of the Group in their own computer, opens a Bash Terminal, changes to the Projects directory previously created, and in there executes the following

   - **First Task:** Configure GIT:

```
$ cd Projects
# Config GIT with your IST ID
$ git config --global user.name "ist1234567"
# Your email address of IST
$ git config --global user.emai ist.user@tecnico.ulisboa.pt
# 'editorname' is your preferred code editor:
# can be atom, code (VS-Code), etc.
$ git config --global core.editor "editorname --wait"
# specify how to treat end of lines
# on Windows they are \r\n (carriage return - line feed)
$ git config --global core.autocrlf true
# on macOs and linux it is just \n (line feed)
$ git config --global core.autocrlf input
```

   - **Second Task:** Each member of a Group, needs to **create a local Repository**, cloned from the already existing Group Repository in the **GIT** platform (`https://git.rnl.tecnico.ulisboa.pt`). For that, you access the GIT Platformm and open the Group Repository named **projects-team-XX** (where **XX** is the Group number) to copy the URL for cloning; Then, on the Projects directory execute the command as follows (with the copied URL):

```
$ git clone https://git.rnl.tecnico.ulisboa.pt/AGISIT-20-21/
    projects-team-XX.git
```

   - **Third Task:** After cloning, initialize the repository:

```
$ git init
# the dot (.) means all files and folders in that clooned
    repository
$ git add .
# Now commit the 'changes'
$ git commit -m "initial commit of cloned repo by yourname"
```

   - **Fourth Task:** Try to push a small change to the Group repository, by editing the README.md file:

```
# put a line on the README.md with your name, and then save and:
$ git add README.md
# Now commit the 'changes'
$ git commit -m "Add my name to README"
$ git push origin
```

2. **Role Division:** Agreement among Group members to split "Roles", for example:
   - **Infra Resources:** one member of the Group is responsible for deploying the "network related" things, which may include a Load Balancer;
   - **Web Resources:** another member of the Group is responsible for deploying the **web** servers;
   - **Application Configuration:** another member of the Group is responsible for Configuring the all the servers of the Infrastructure;
3. **Common Task:** On a second phase the Group should ensure to use the same RSA keypair in each individual local Management server (osmgmt) (instructions on part 3 of the Lab).

4 Create a VMCloud Infrastructure with Terraform

In Part 1 of this Lab series of experiments you learned how to provision an infrastructure manually, using the **OpenStack** dashboard of the VMCloud of IST.

In this Lab experiment you will automate the processes in two steps, starting by creating the infrastructure with **Terraform**, and then Configuring the software and services in the instances with **Ansible**.

4.1 Step 1: Preparing the Deployment

In the local Repository for your Projects, `projects-team-XX` you will find a `Vagrantfile` and a `bootstrap-osmgmt.sh` that will be used to create and configure the Management Node that will be used to manage your infrastructures. We will use a Ubuntu OS, named "ubuntu/bionic64", which corresponds to Ubuntu 18.04 LTS. The Management node contains the Tools needed for Provisioning and Configuring the Infrastructures.

Verify that you end up with 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, in several steps, the tasks that successively deploy the infrastructure and configure the corresponding compute instances with the software that sets up a Load Balanced Web site.

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). You can go now 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: Preparing Credentials for the VMCloud

As you recall, each Working Group of the Lab has access to the respective Tenant/Project, specifically created for the Group in the VMCLoud Platform. Each Group member shares the resources allocated to the respective Project. This means that it is not advisable to have students of each group interacting with the VMCloud at the same time without coordinating their work, as one may be “destroying” the work of the other without noticing.

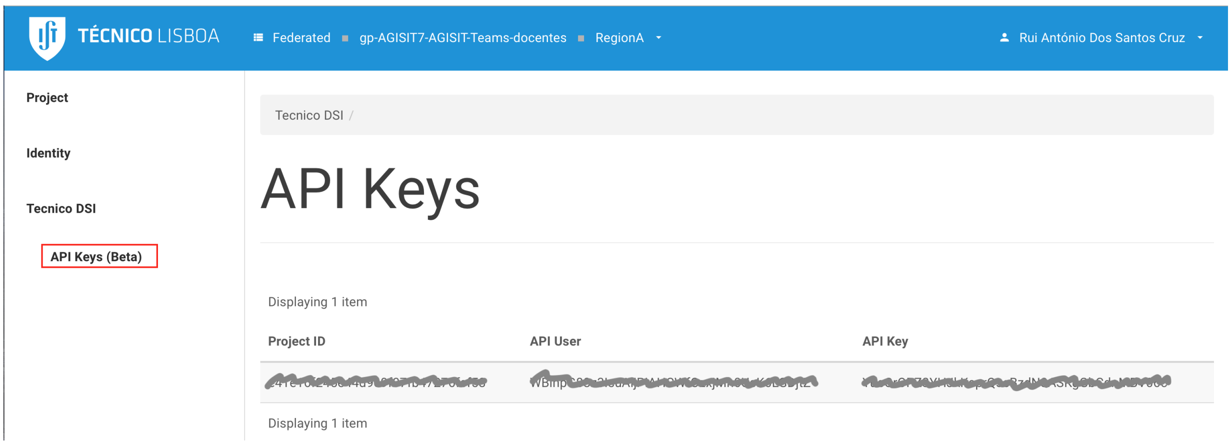
In order to access and use **VMCloud** via API calls, you need to authenticate against the OpenStack **Identity service** (*Keystone*), which returns a *Token* and a *Service Catalog*. The catalog contains the endpoints for all services the Tenant has access to.

The assigned Projects for each Group are as follows:

Group Name	Tenant Name
Group 1	gp-AGISIT7-AGISIT-Teams-1
...	...
Group 45	gp-AGISIT7-AGISIT-Teams-45

Using a browser, enter the URL for the VM Cloud dashboard (the *Horizon*), and login with your **IST ID** credentials:

https://vmcloud.tecnico.ulisboa.pt


TÉCNICO LISBOA Federated gp-AGISIT7-AGISIT-Teams-docentes RegionA Rui António Dos Santos Cruz

Project
Identity
Tecnico DSI
API Keys (Beta)

Tecnico DSI /

API Keys

Displaying 1 item

Project ID	API User	API Key
[redacted]	[redacted]	[redacted]

Displaying 1 item

Figure 4: VMCloud Tenant Dashboard - API Keys

After a successfully login, the Horizon Dashboard will present a menu on the left with the available management Tabs for your Project, the Compute instances, the Network, the Object Store, the Identity, etc. Check that the project assigned to your Group is listed on the bar at the top of the page, and that you are in **Region A**.

You can obtain the specific data for your Project by looking into **Tecnico DSI** → **API Keys(Beta)** (and also the Project ID in **Identity** → **Projects**), as in Figure 4,.

You need to populate two files in your Repository (the scripts called `CREDENTIALS.sh` and `terraform.tfvars`), with the necessary variables that will be loaded in the environment (for Ansible) or by Terraform.

The script called `CREDENTIALS` defines the environment variables for Ansible, and is similar to the following (you should carefully verify ALL the variables values):

```
export OS_AUTH_URL=https://stackcontroller.al.dsi.tecnico.ulisboa.pt
   :5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_USER_DOMAIN_NAME="tecnico-apikeys"
export OS_REGION_NAME="RegionA"
export OS_PROJECT_NAME="gp-AGISIT7-AGISIT-Teams-XX"
export OS_PROJECT_ID=XXXXXX
export OS_USERNAME=XXXXX
export OS_PASSWORD=XXXXX
```

The variables that you need to set are `OS_AUTH_URL`, `OS_PROJECT_NAME`, `OS_PROJECT_ID`, `OS_USERNAME` and `OS_PASSWORD`, replacing the respective values 'XXX' with the data (Keys and Group Name) for your Project.

You also need to populate the file `terraform.tfvars` with the same information:

```
auth_url            = "https://stackcontroller.al.dsi.tecnico.ulisboa.pt
   :5000/v3"
user_domain_name    = "tecnico-apikeys"
region              = "RegionA"
tenant_name         = "gp-AGISIT7-AGISIT-Teams-XX"
unique_network_name = "n-AGISIT7-AGISIT-Teams-XX"
tenant_id           = "XXXXX"
user_name           = "XXXXX"
password            = "XXXXX"
ssh_key_public      = "/home/vagrant/.ssh/id_rsa.pub"
ssh_key_private     = "/home/vagrant/.ssh/id_rsa"
```

4.3 Step 3: Resetting the Private Cloud Environment of the Tenant

Your first task is to delete all previous manual configurations you might have created in the VMCloud in previous Lab experiment. Start with the compute instances, then the network(s) you might have created, **except the "provider" and the "n-AGISIT-AGISIT-Teams-XX" and the router**, which are pre-provisioned for each Project by the Platform administrator. Do not forget also to delete security groups and/or rules you have created, as well as any key pairs (for SSH). In the end you should verify that no resources (except the "default" Security Group are used.

4.4 Step 4: Create and share the Group the RSA SSH keypair

For precaution, you should ensure that no other keypairs are still deployed in your Project (from previous experiment attempts).

To generate the Group Keypair, **agree among yourselves the member of the Group** that will be **responsible to create and distribute the Keypair**. For that purpose, the chosen member goes to the home directory of the Management node and runs the command `ssh-keygen -t`, specifying the type of key (**RSA**) and its lenght with parameter **-b** as exemplified. Please hit **ENTER** to the prompts, and do not enter a password.

```
vagrant@osmgmt:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
```

That member of the Group responsible for sharing the SSH keypair, will then need to copy the corresponding files and share them with the other members of the group. For that purpose do the following commands:

1. Copy the public and private keys from the **.ssh** folder to the **vmcloud-tenant** folder:

```
vagrant@osmgmt:~$ cp .ssh/id* vmcloud-tenant/
```

2. Insert into the `authorized_keys` file the generated PUBLIC key verifying that you have a second entry starting with "ssh-rsa" and ending with "vagrant@osmgmt":

```
vagrant@osmgmt:~$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
vagrant@osmgmt:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8......WhQ== vagrant insecure
    public key
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD4......M9ENJ vagrant@osmgmt
```

3. Move to the **vmcloud-tenant** folder and create there a **pub-key.txt** file into which you will inset the PUBLIC key:

```
vagrant@osmgmt:~$ cd vmcloud-tenant
vagrant@osmgmt:~/vmcloud-tenant$ touch pub-key.txt
vagrant@osmgmt:~/vmcloud-tenant$ cat ../.ssh/id_rsa.pub >> pub-key.
   txt
```

That Group member will then provide those **two SSH key files**, together with the **pub-key.txt** file in a ZIP protected archive to the other members of the Group (telling them the password to open the archive, of course). After sending those files to the other members of the Group, you can remove them (delete) from the **vmcloudfront** folder.

4.5 Step 5: Inserting the shared RSA SSH keypair

Each of **the other members of the Group** will save to the **vmcloudfront** folder the received files, and will proceed as follows:

1. Copy the public and private keys from the the current folder (**.**) to the **.ssh** folder:

```
vagrant@osmgmt:~/vmcloud-tenant$ cp id* ../.ssh/
```

2. Insert the content of the pub-key.txt into the `authorized_keys` file:

```
vagrant@osmgmt:~$ cd .ssh/
vagrant@osmgmt:~/.ssh$ cat ../vmcloud-tenant/pub-key.txt >>
    authorized_keys
```

After this procedure, you can remove the received files (delete) from the **vmcloud-front** folder.

4.6 Step 6: Provision the infrastructure with Terraform

We will start in this lab experiment the deployment of fully functioning privately (inside IST network) addressed front-end Load Balancer (implemented with ***haproxy***) and with two web servers (implemented with ***nginx***).

We will use **Terraform** to Provision the infrastructure and later (in part 3 of this lab) we will use **Ansible** to install in the created instances all of the required applications and packages, deploy configuration files, and start the correct services on each of those instances. All without logging into any of those VMCloud nodes manually (see Figure 5).

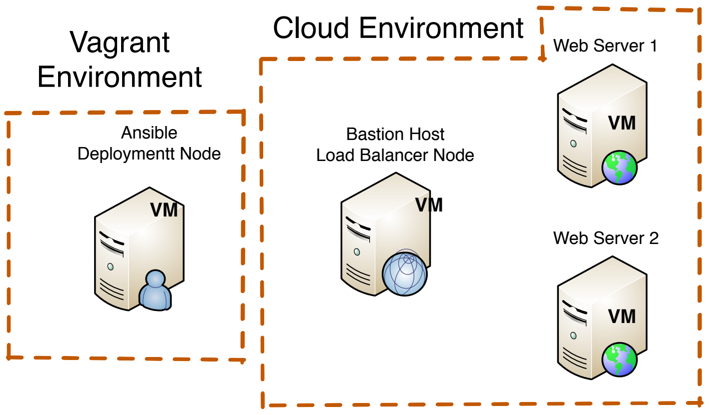


Figure 5: Environment for Cloud Deployment with Ansible

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**.

Look into the `terraform-provider.tf` file and study it. **Terraform** allows for different service providers, such as other Openstack-based systems, Microsoft Azure, Amazon AWS, or Google Cloud. This is the place to modify their configuration.

Check `https://www.terraform.io/docs/providers/` to find more about how to configure Terraform. Now, initialize Terraform as follows, in order to eventually satisfy plugin requirements:

```
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 observe by studying the `terraform-vmcloud-networks.tf` and `terraform-vmcloud-servers.tf` files, your infrastructure code is written (or, to be rigorous, the infrastructure is declared).

You just need to make it live (i.e., create it in the VMCloud). Terraform talks to the VMCloud (OpenStack) APIs and makes sure that the infrastructure is always up-to-date with what you code in the configuration files.

But before making it live, you are supposed to create a **Plan**, i.e., simply compare your assumed current state, stored in those configuration files with the reality, using API calls which fetch it from the VMCloud. Once you are happy with the **Plan**, you call **Apply** and if any changes are scheduled, they are actually performed.

So, let's go and create 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:

  + openstack_compute_instance_v2.balancer
      id:                          <computed>
      access_ip_v4:                <computed>
      access_ip_v6:                <computed>
      all_metadata.%:                  <computed>
      availability_zone:               <computed>
      flavor_id:                       "0"
      flavor_name:                     <computed>
      force_delete:                    "false"
      image_id:                        <computed>
      image_name:                      "Ubuntu-18.04"
      key_pair:                        "cloud-key"
      name:                            "balancer"
      network.#:                       "1"
      network.0.access_network:        "false"
      network.0.fixed_ip_v4:           <computed>
      network.0.fixed_ip_v6:           <computed>
      network.0.floating_ip:           <computed>
      network.0.mac:                   <computed>
      network.0.name:                  "n-AGISIT7-AGISIT-Teams-XXXX"
      network.0.port:                  <computed>
      network.0.uuid:                  <computed>
      power_state:                     "active"
      region:                          <computed>
      security_groups.#:               "2"
      security_groups.3814588639:      "default"
      security_groups.981515365:       "sec_ingr"
      stop_before_destroy:             "false"

  + openstack_compute_instance_v2.dbase
      id:                              <computed>
....
  + openstack_compute_instance_v2.web1
      id:                              <computed>
....
  + openstack_compute_instance_v2.web2
      id:                              <computed>
....
  + openstack_compute_keypair_v2.rc-cloud-key
      id:                              <computed>
....
  + openstack_compute_secgroup_v2.sec_ingr
      id:                              <computed>

Plan: 8 to add, 0 to change, 0 to destroy.
------------------------------------------------------------------------
Note: You didn't specify an "-out" parameter to save this plan, so
   Terraform cannot guarantee that those actions will be performed if
\textbf{terraform apply} is subsequently run.
```

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

```
vagrant@osmgmt:~/vmcloud-tenant$ terraform apply
....
Plan: 6 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

openstack_compute_keypair_v2.rc-cloud-key: Creating...
openstack_compute_secgroup_v2.sec_ingr: Creating...
openstack_compute_keypair_v2.rc-cloud-key: Creation complete after 1s [id
   =rc-cloud-key]
openstack_compute_secgroup_v2.sec_ingr: Creation complete after 6s [id=6
   d1e6770-a88b-4c81-81a1-37458ecdcd96]
openstack_compute_instance_v2.balancer: Creating...
openstack_compute_instance_v2.dbase: Creating...
openstack_compute_instance_v2.web2: Creating...
openstack_compute_instance_v2.web1: Creating...
...
openstack_compute_instance_v2.dbase: Creation complete after 58s
openstack_compute_instance_v2.web2: Creation complete after 59s
openstack_compute_instance_v2.balancer: Creation complete after 1m0s
openstack_compute_instance_v2.web1: Still creating... [1m0s elapsed]
openstack_compute_instance_v2.web1: Creation complete after 1m1s

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

balancer_IP = 100.68.20.102
dbase_IP = 100.68.20.107
private_key = -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA+DTXmiIIH+...
-----END RSA PRIVATE KEY-----

public_key = ssh-rsa AAAAB3NzaC1yc2EA.... vagrant@osmgmt

terraform-provider = Connected with VM Cloud at var.auth_url
web1_IP = 100.68.20.101
web2_IP = 100.68.20.110
```

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

5 Finnishing the Experiments

To clean the environment (destroy created networks, security groups and instances) run **terraform destroy**. Make sure the infrastructure you have created (only your systems, not your colleagues) has been destroyed by checking with *terraform refresh* and checking it manually on the VMCloud Horizon.

When finished the experiment, Stop the Management node 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”.