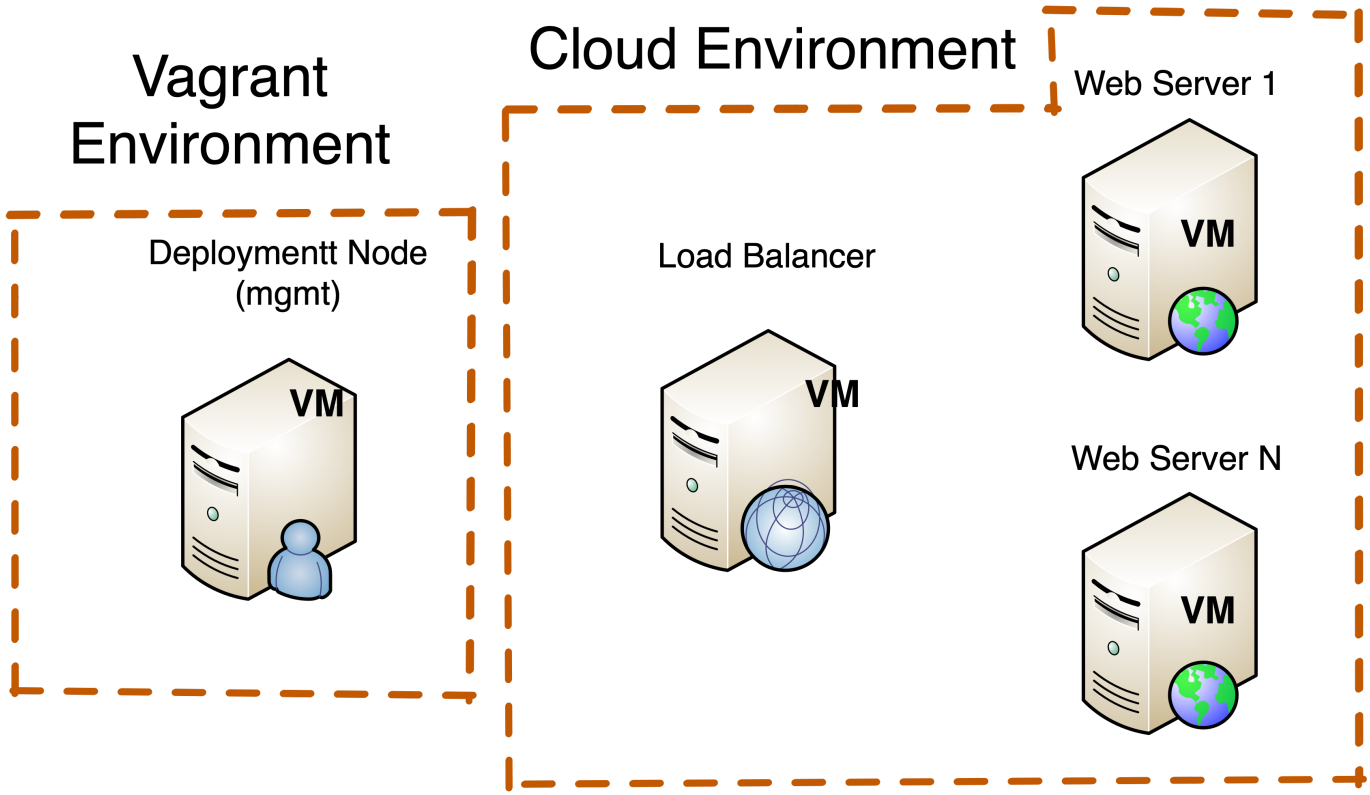
AGISIT Lab Guide 02

Deployment of a cloud-based infrastructure in a Private Cloud using automation tools

> **PLEASE NOTE:** The work in this Lab is complemented with some Tutorials in Moodle (that include a few questions that you are required to answer). The Tutorials are important for a better comprehension of the tools to be used. The evaluation of your answers in those Tutorials are part of the evaluation of this Lab. The Tutorials are the following:
>
> - Guide to SSH
> - What is YAML?
> - What is Infrastructure as Code (IaC)?
> - What is Ansible?
> - What is Terraform?

Objectives

The objective of the experiments in this Lab is exploring the deployment of a fully functioning privately addressed (inside IST network) front-end Load Balancer (implemented with **haproxy**) and several back-end web servers (implemented with **NGINX**). To deploy and configure the Private Cloud-based Infrastructure, we will use 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, install in those instances all of the required applications and packages, deploy the corresponding customized configuration files, and start the respective services.



Both **Terraform** and **Ansible** are outstanding Infrastructure as Code (IaC) products used in the "**DevOps/GitOps**" world that can be used to deploy repeatable environments that meet a vast range of

complexity requirements.

Working Methodology

The working Methodology from this lab onward will be as **Teamwork** for each Group, based on Coordination and/or division of Tasks among members of each Group.

Role Division

It is highly recommended to establish an Agreement among Group members to split “Roles” in the execution the the Lab Works and essentially on the Project, for example:

- **Infrastructure Resources:** one member of the Group is responsible for deploying the “network related” things, which may include a Load Balancer, firewall rules, and addresses and ports in configuration files, etc;
- **Compute Resources:** another member of the Group is responsible for deploying the servers, i.e., configuring all scripts and configuration files for the servers that will be created;
- **Application Configuration:** another member of the Group is responsible for Configuring the servers of the Infrastructure in terms of the applications that will be run in them, including the configurations of those applications;

The local Git repository for this Lab

Each member of the Group in their own computer, opens a Terminal, changes to the "Projects" directory previously created, where the respective Group Repository in the GIT platform (https://git.rnl.tecnico.ulisboa.pt) was cloned (hopefully). The repository of each Group is identified with the format `team-XXC` in which `C` represents the campus (**A** for Alameda, **T** for Taguspark) and `XX` is the number of the Group in Fenix System. For example, Group 3 from Alameda has the repository team-03A.

In that Group Repository you will find a Directory named `labs/vmcloud` that already contains the files for this Lab experiment.

```
├── README.md
├── Vagrantfile.docker
├── Vagrantfile.vbox
├── bootstrap-mgmt-docker.sh
├── bootstrap-mgmt.sh
└── labs
    ├── arch-arm64
    │   └── README.md
    ├── awscloud
    │   └── README.md
    ├── gcpcloud
    │   └── README.md
    ├── k8scloud
    │   └── README.md
    ├── project
    │   └── README.md
    ├── vmcloud
    │   ├── CREDENTIALS.sh
```

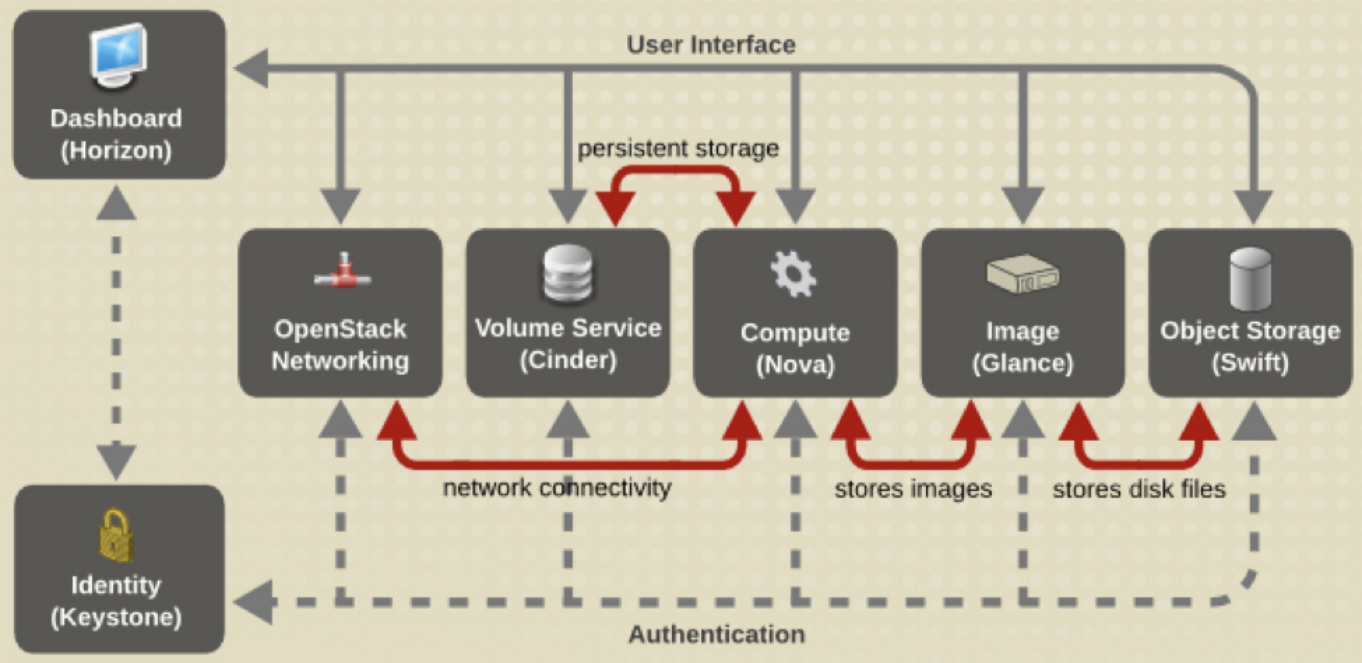
```
│   ├── README.md
│   ├── ansible.cfg
│   ├── myhosts
│   ├── outputs.tf
│   ├── templates
│   │   ├── default-site.j2
│   │   ├── haproxy.cfg.j2
│   │   ├── index.html.j2
│   │   └── nginx.conf.j2
│   ├── terraform-provider.tf
│   ├── terraform-vmcloud-networks.tf
│   ├── terraform-vmcloud-servers.tf
│   ├── terraform.tfvars
│   ├── versions.tf
│   └── vmcloud-site-servers-setup-all.yml
└── webfront
```

Part 1: Get Acquainted with the Private Cloud of IST (VMCloud)

A Private cloud is a Cloud Platform operated inside a private network. For that purpose we will use **VMCloud**, a Cloud Platform of Instituto Superior Técnico, based on the **OpenStack** framework.

OpenStack is an open-source framework that began as a joint project between NASA and Rackspace, originally intended to be an alternative for, but compatible with, the Amazon Elastic Compute Cloud (EC2). **OpenStack** has a a highly modular architecture offering virtualization of *compute*, *storage*, *networking*, and many other resources. Each component in **OpenStack** manages a different resource that can be virtualized for the end user.

Logically, the components of OpenStack can be divided into four main 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 virtualization hypervisor, with services and agents to handle virtual machines. The *Storage* tier manages block (Volumes; partitions) and object (containers; files) storage for the *Compute* instances.



- **Horizon** (dashboard): is the dashboard web interface component. The dashboard cannot do anything that the API cannot do. All the actions that are taken through the dashboard result in calls to the API to complete the task requested.
- **Keystone** (identity): is the identity management component. Besides authentication, Keystone manages *tenants* (aka, Projects), *users*, and *roles* and a catalog of services and endpoints for all the components. Everything in OpenStack must exist in a *tenant*. A *tenant* is a grouping of objects (such as users, instances, and networks) and objects cannot exist outside of a *tenant*. Another name for a tenant is project (on the command line, the term tenant is used, but in the dashboard web interface, the term used is project).
- **Glance** (image): is the image management component, a registry of preinstalled disk images to boot compute instances from.
- **Neutron** (networking): is the network management component. Neutron is an API frontend (and a set of agents) that manages the Software Defined Networking (SDN) infrastructure. Neutron allows creating virtual isolated networks that can be connected to virtual routers to create routes between the virtual networks. A virtual router can have an external gateway connected to it, and external access can be given to each instance by associating a *floating IP* on an external network. Neutron then puts all configuration in place to route the traffic sent to the *floating IP* address through these virtual network resources into a launched instance. This is also called Networking as a Service (NaaS).
- **Nova** (compute instances): is the instance management component. An instance can be launched once there is an image, a network, a *key pair*, and a *security group* available.
- **Cinder** (block storage): is the block storage management component. Volumes can be created and attached to instances. On the instance, the block device can be partitioned and a file system can be created and mounted. Cinder also handles snapshots of the instance.
- **Swift** (object storage): is the object storage management component. Object storage is a simple content-only storage system. Files are stored without the metadata that a block filesystem has. The objects are simply containers and files.

Some other components of interest are:

- **Ceilometer** (telemetry): is the telemetry component, used for collecting resource measurements and monitor the **OpenStack** Platform. Ceilometer was originally designed as a metering system for billing the users. When Ceilometer reads a meter, it is called a sample. The samples can also be used for alarms. Alarms are monitors that watch for a certain criterion to be met.
- **Heat** (orchestration): is the orchestration component. Orchestration is the process of launching multiple instances that are intended to work together. Orchestration uses templates (files), to define what will be launched.

1. Accessing the VMCloud

The **VMCloud** of Instituto Superior Te´cnico, is a Private Cloud Platform, available for internal users of the University.

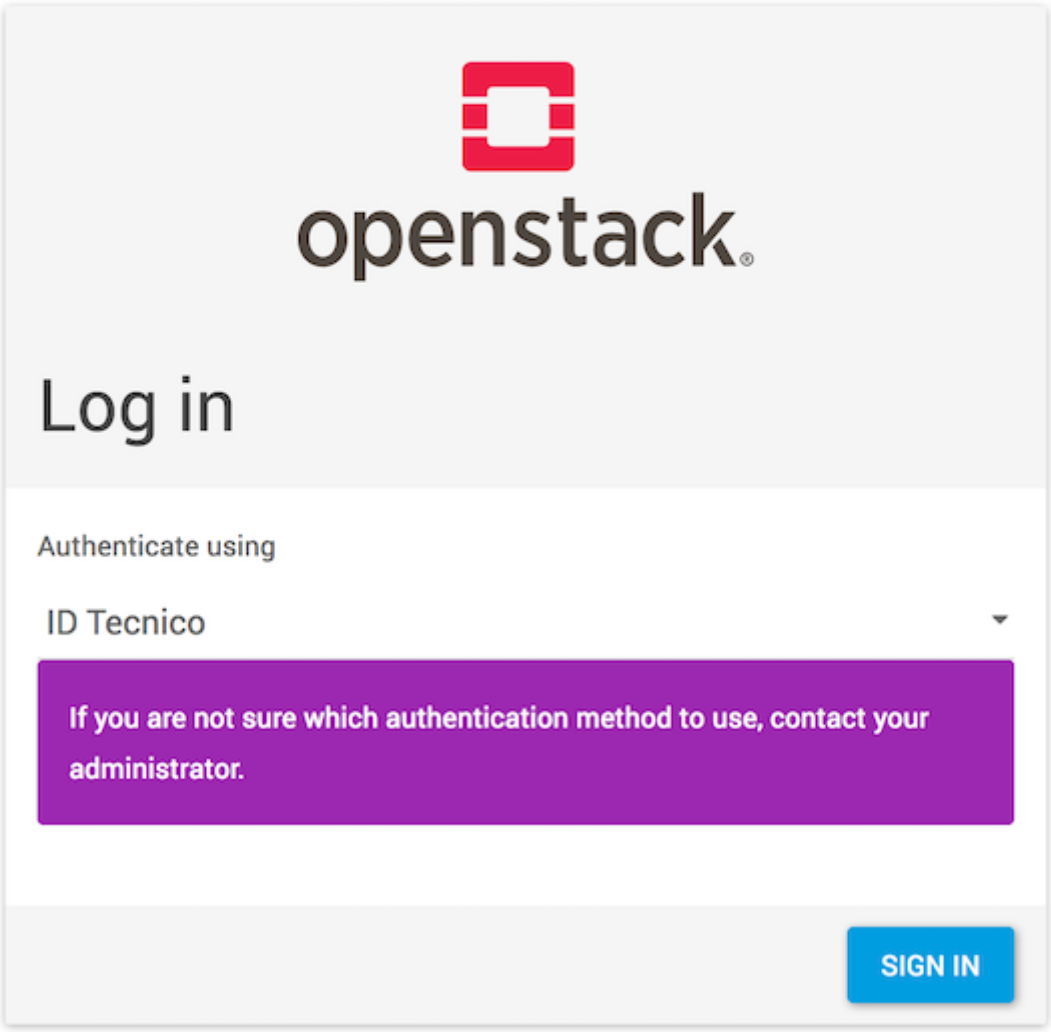
> **IMPORTANT:** As the VMCloud Platform is a Private Cloud system, the Compute instances and resources that you will use/create, can only be accessed with IPv4 addresses **from within the IST campi**. This means that, if you are accessing remotely, **YOU MUST ESTABLISH a VPN** connection to the IST 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/.

Each Working Group of students of the course will access their Tenant (Project) specifically created for the Group at the **VMCloud** Platform. The assigned Tenants and the corresponding Project IDs are as follows:

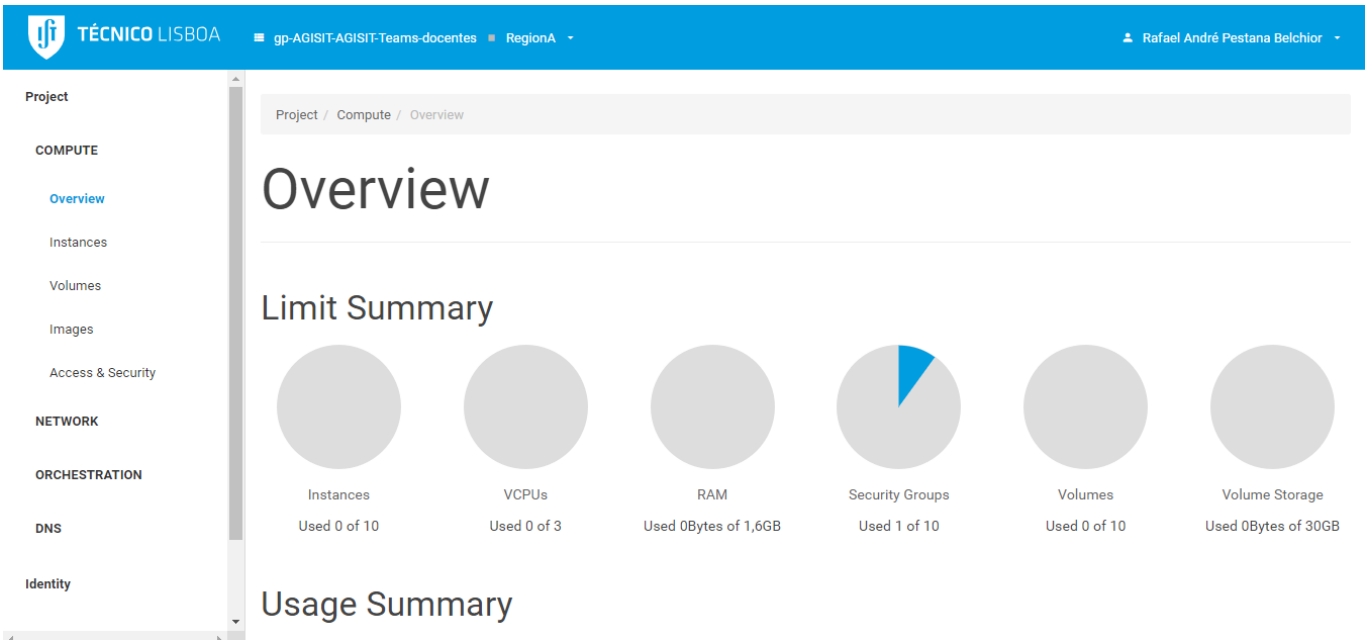
Group Name	Tenant Full Name	Project NAME
Group 01	gp-AGI-AGISIT-Teams-21-22-1	AGI-AGISIT-Teams-21-22-1
...	...	
Group 51	gp-AGI-AGISIT-Teams-21-22-51	AGI-AGISIT-Teams-21-22-51

> The documentation for the VMCloud can be foud here:
> https://iaas.projects.dsi.tecnico.ulisboa.pt/user/gettingstarted.html.
> It is worth to have a read for having a better understanding about this implementation of OpenStack.
> **NOTE ALSO:** Contrary to what is stated in the documentation, the AGISIT Projects/Tenants will not use their full name, e.g., "gp-AGI-AGISIT-Teams-21-22-XX", for generating their DNS records, but just "AGI-AGISIT-Teams-21-22-XX" as indicated in the above Table.

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



After a successfully login the Dashboard will present a menu on the left with the available management Tabs for your Project, the Compute instances, the Network, the Volumes and the Identity, similar to the following. Please note that you need to ENSURE that you have **Region T** selected (in the Top Bar of the Dashboard)



> Take a note of the Overview and Usage Summary of the resources, to be able to compare them with the situation after completing the next steps.

You can now navigate to the **Network** tab to see the already pre-assigned front-end network (coloured Orange) which is “routed” to the provider network, as illustrated here:

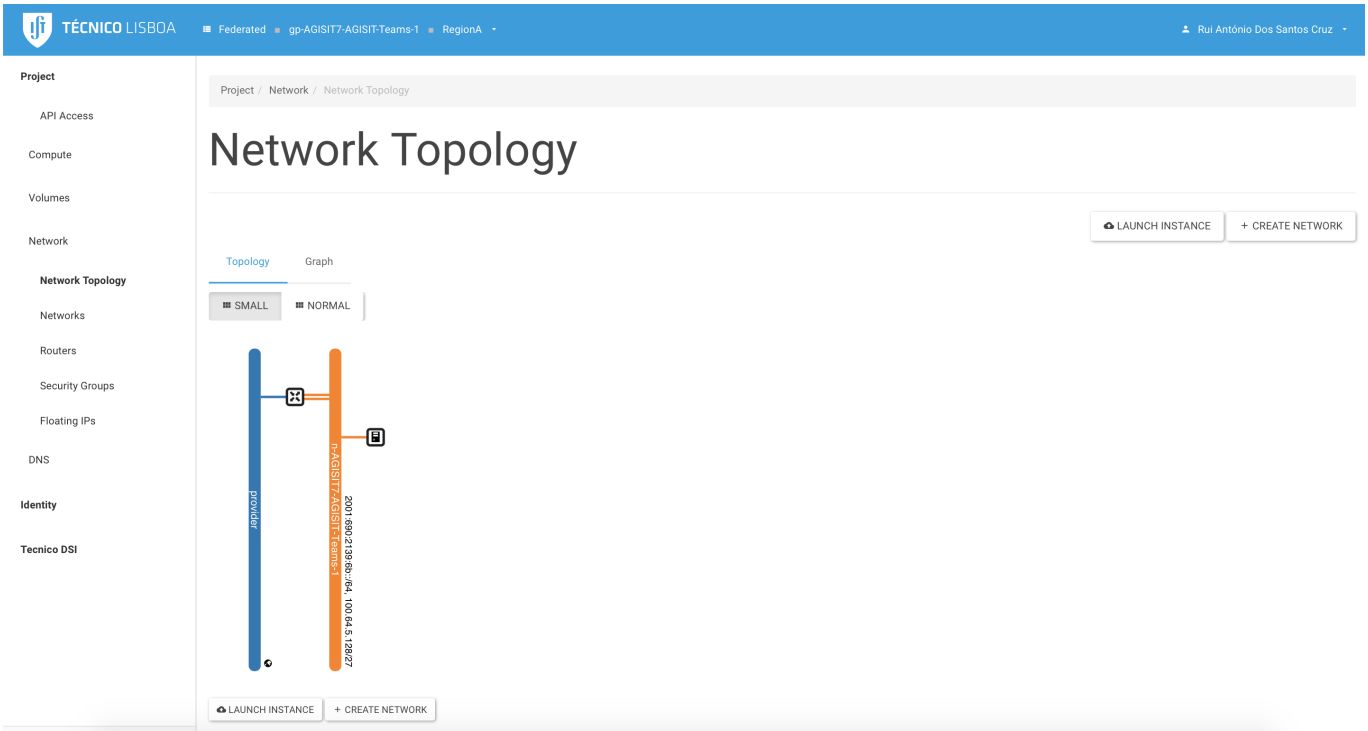
2. Create a compute instance from the Dashboard

As you already have pre-assigned networks, you can now create a compute instance (server) and connect it to the front-end network.

For that purpose select the tab ***Instances***, of the submenu **Compute**, of your Project, and in the right Panel select “Launch Instance”. In the form that pops-up fill the following:

1. In Details a VM name composed with your IST ID → Instance Name: vm-ist123456
2. In Source → Select Image as boot source. Click on the + button next to CirrOS
3. In Flavour → Click on the + button next to t1.nano
4. In Networks → verify that your instance is connected to the default network.
5. Finalize by clicking the button “Launch Instance” in the right bottom of the form.
6. Go the the Overview Tab and confirm that you have now some Resources being used.

You can now view the new topology you just created from the Network Topology tab, as you may observe:



3. Cleaning up Manually the Resources

Having finalized the experiment, delete all manual configurations you created in **VMCloud**. Start with the compute instances.

Do not forget also to **delete other associated resources** (for example Volumes)—**except default Networks**—previously used by your instance, and not anymore needed. In the end you should verify that no resources (except the “default” Security Group) are used.

Part 2: Create a VMCloud Infrastructure with Terraform

> **PLEASE NOTE:** The following instructions and steps **ARE NOT A RECIPE**, just a recommended Procedure. **Your job is to study the Process** to get insight of its purpose, **and modify whatever you will need to modify** in order to ensure that you will arrive to a correctly Provisioned Infrastructure, and a correctly Configured Web Service.

In the local Repository for your Projects, `team-XXC` you will find a `Vagrantfile.(docker/vbox)` and a `bootstrap-mgmt(-docker).sh` that will be used to create and configure the **Management Node** (`mgmt`)that will be used to manage your infrastructures. We will use a Ubuntu OS, named "ubuntu/ focal64", which corresponds to Ubuntu 20.04 LTS. The `mgmt` local node contains the Tools needed for Provisioning and Configuring the Infrastructures.

The `vmcloud` 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** (in practice, "identical" to the one you have created locally in previous Lab 01).

1. Preparing Credentials for the VMCloud

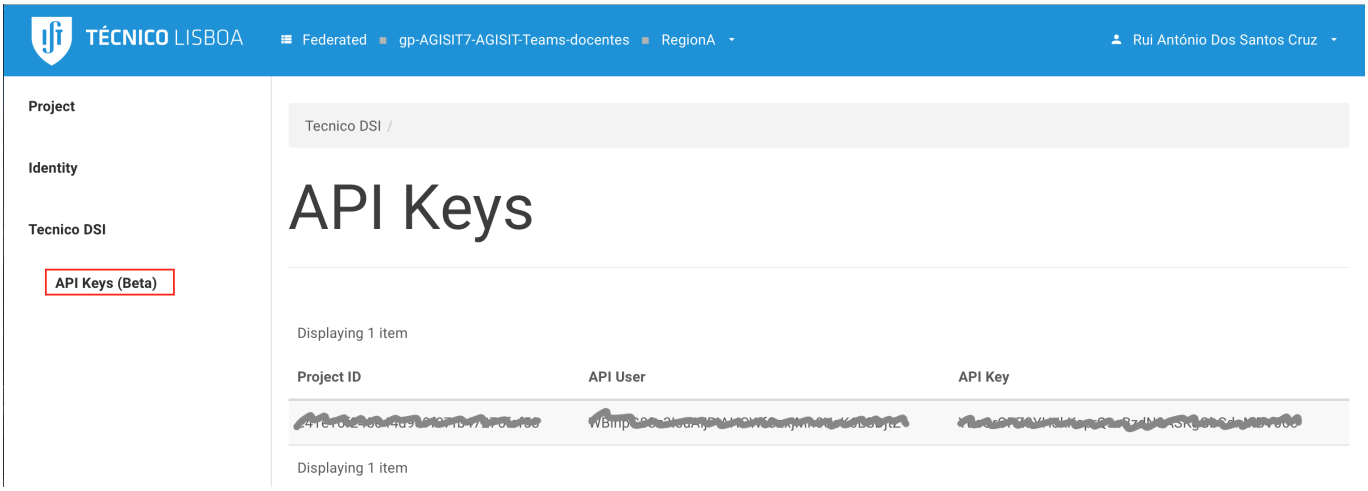
As you recall, each Group of the Lab has access to the respective Tenant/Project, specifically created for the Group in the **VMCLoud** Platform. Each member of the Group shares the resources allocated to the respective Project. This means that it is not advisable to have students of each Group interacting with the **VMCloud** at the same time, unless the members coordinate their work and tasks, as one may be destroying or disrupting 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.

Using a browser, enter the URL for the **VMCloud** Dashboard (the *Horizon*), and login with your IST ID credentials.

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 T**. If your Project is not listed, you can select it by going to **Identity** → **Projects** and select your project.

You can obtain the specific data for your Project by looking into the menu Tab **Tecnico DSI** → **API Keys** (and also the Project ID in Identity → Projects), as illustrated in the following figure:



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

> Please Note: these files are not pushed to the Repository (have a look at the `.gitignore`, as they contain sensitive information, such as the API keys credentials. So, you to create them locally.

The script called `CREDENTIALS.sh` 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="RegionT"
export OS_PROJECT_NAME="AGI-AGISIT-Teams-21-22-XX"
export OS_PROJECT_ID=XXXXXXXXXXXXXXXXX
export OS_USERNAME=XXXXXXXXXXXXXXXXX
export OS_PASSWORD=XXXXXXXXXXXXXXXXX
```

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 '**XX...X**' with the data (Keys and Group Name) for your Project that you have obtained from the **API Keys**.

You will need to do similarly to the `terraform.tfvars` files using the same information:

```
auth_url = "https://stackcontroller.al.dsi.tecnico.ulisboa.pt:5000/v3"
user_domain_name = "tecnico-apikeys"
region = "RegionT"
tenant_name = "AGI-AGISIT-Teams-21-22-XX"
unique_network_name = "n-AGI-AGISIT-Teams-21-22-XX"
tenant_id = "XXXXXXX"
user_name = "XXXXXXX"
password = "XXXXXXX"
ssh_key_public = "/home/vagrant/.ssh/id_rsa.pub"
ssh_key_private = "/home/vagrant/.ssh/id_rsa"
```

2. Create and share the Group the RSA SSH keypair

For this step you need to start your Management node (`mgmt`) with `vagrant up`. When ready, access the VM with `vagrant ssh`.

The tools you will work with, **need to have access via SSH** to the Compute instances that will be created.

As the Project can be worked independently by each member of teh Group, you will need to have a Group **SSH keypair**.

To generate the Group SSH 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 (`mgmt`) and run the command `ssh-keygen -t`, specifying the type of key (RSA) and its length with parameter `-b` as exemplified. Please hit **ENTER** to the prompts, and do not provide a password.

```
vagrant@mgmt:~$ ssh-keygen -t rsa -b 4096
```

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 `labs/vmcloud` folder:

```
vagrant@mgmt:~$ cp .ssh/id* labs/vmcloud/
```

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@mgmt**":

```
vagrant@mgmt:~$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
vagrant@mgmt:~$ cat .ssh/authorized_keys
```

3. In the Host system go to the `labs/vmcloud` folder and create a protected ZIP file containing the keypair (`id_rsa.pub` and `id_rsa`), and then delete those files from that folder. Then, provide the protected ZIP file containing those two SSH key files to the other members of the Group (telling them the password to open the archive, of course). After that, delete the ZIP file.
4. Each of the other members of the Group will save to the `vmcloud` folder the keypair received in the Zip file, and copy them from the current folder (`.`) to the `.ssh` folder, and then insert the public key to the `authorized_keys` file, after which you can remove the received files from the `vmcloud`:

```
vagrant@mgmt:~$ cp labs/vmcloud/id* .ssh/
vagrant@mgmt:~/.ssh$ cat id_rsa.pub >> authorized_keys
```

3. Initialize Terraform

We will use **Terraform** to **Provision the infrastructure** and later we will use **Ansible** to Configure and install in the created instances all of the required applications and packages, deploy configuration files, and start the correct services. All done without logging into any of those **VMCloud** nodes manually.

In the `mgmt` node go to the `vmcloud` 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@mgmt:~/labs/vmcloud$ terraform init
```

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

4. Provision the infrastructure with Terraform

After successful initiation, as you can observe by studying the `terraform-vmcloud-networks.tf` and `terraform-vmcloud-servers.tf` files, how your Infrastructure as Code is written (or, to be rigorous, **how the infrastructure is declared**).

You just need to make it live (i.e., create it in the **VMCloud**). **Terraform** will communicate with the **VMCloud** (OpenStack) APIs and make sure that the infrastructure will be always up-to-date with what you have declared in the configuration files.

But before making it live, you are supposed to create a **Plan**, i.e., simply compare your assumed **current state** declared in those configuration files, with the reality, using API calls which fetch the information 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:

```
vagrant@mgmt:~/labs/vmcloud$ terraform plan
```

To execute the **Plan** and create the infrastructure, run **Apply**:

```
vagrant@mgmt:~/labs/vmcloud$ terraform apply
```

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

5. Modify Ansible Inventory and the Management Node DNS resolution

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, in this case named `myhosts`, present in the same project directory, as well as edit the `mgmt` system `hosts` file with the **IP addresses** and names of the servers (**retrieved from the output of Terraform**):

```
vagrant@mgmt:~/labs/vmcloud$ terraform apply
```

6. 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@mgmt:~/labs/vmcloud$ ping balancer
vagrant@mgmt:~/labs/vmcloud$ ansible targets -m ping
```

7. Configuring VMCloud Instances with Ansible

The **Ansible** *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 in previous Lab.

> Due to recent changes in some of the tools used, as well as the Operating System of the Instances in **VMCloud**, you **MAY** need to modify some lines in the configuration files provided in your Group Repository. Please note also that the **Lab Guide IS NOT A RECIPE** to strictly follow. As already said, **it is of your Responsibility** to ensure that you successfully deploy the infrastructure and the Service (the Web Service). For that to happen, you need to make all the necessary adjustments in configuration files, in Playbooks, etc. **HINT 1:** Files you may need to verify having the adequate "variables" defined or adequate configurations, are:
>
> - The `haproxy.cfg.j2` template file
> - The `index.html.j2` template file
> - The `default-site.j2` template file
> - The `myhosts` inventory file
> - The `terraform.tfvars` file
> - The `terraform-vmcloud-servers.tf`file, to change the server instances desired -The `vmcloud-site-servers-setup-all.yaml` Playbook, to adapt for adequate "variables", etc.

So, having done some necessary changes, go ahead and play it:

```
vagrant@mgmt:~/labs/vmcloud$ ansible-playbook vmcloud-site-servers-setup-
all.yaml
```

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 (or not?).

In case you get errors in some tasks, try to figure out what are the causes, then correct the configuration files corresponding to the failed tasks, and play again the *Playbook*. Repeat the procedure until success.

> **HINT 2:** In case something is malfunctioning (in the Load Balancer or in Web servers) you can access them directly through the `mgmt` node with the command `ssh ubuntu@balancer` or `ssh ubuntu@webX` (X is the number of the server name), and there have a look at the configured files installed by Ansible, such as the **NGINX** default site file, the webpage `index.html`, the "haproxy" configuration, the `/etc/hosts` file, etc.
>
> - In case you need to check if the **NGINX** configuration in web servers is correct, use the command `sudo nginx -t`
> - To gracefully reload the the **NGINX** server use the command `sudo systemctl reload nginx`

> **HINT 3:** You may verify the "**facts**" of each server, using the following command (example for the balancer server): `ansible -m setup balancer`

8. Test the Deployed 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:

- 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).
- Hit the refresh button on the web browser. **Did something change?** If so, repeat the refresh several times and observe the changes.
- Open a second browser tab and navigate to the statistics page of the Load Balancer at http://ip-of-the-balancer/haproxy?stats.
- 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.

9. Finnish 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 `mgmt`Virtual Machine and verify the global state of all active Vagrant environments on the system.

> NOTE: From this Lab onward, you do not need to destroy the `mgmt` Virtual Machine, just **HALT** it, as it will be necessary for next Labs.