TÉCNICO LISBOA

AGISIT 19/20	Project: Create a Lab	Group Number:	12
Container Technologies		Issue Date:	October 29, 2019
Container orchestration - Docker Swarm		Due Date:	December 31, 2019
Authors: 75834 - 83458 - 86437		Revision:	0.0

1 Introduction

The goal of this project is to enable the deployment of a **Docker Swarm** in an **Amazon Web Services Elastic Cloud Computing (AWS EC2)** environment. By the end of this guide we will have deployed the whole infrastructure on AWS with the help of **Terraform** and provisioned each instance resorting to **Ansible**. We will begin by making use of **Vagrant** to launch a virtual machine responsible for orchestrating the instances to be launched on the Amazon EC2 platform. Then, in order to create a **Docker Swarm**, we will launch a *manager* node and two *worker* nodes (although more textit-workers nodes can be easily created). The *manager* will be responsible for managing the nodes that belong to the Swarm, and both the *manager* and *workers* will be responsible for running replicas of a **nginx** webserver. Additionally, the *manager* node will run a **Prometheus** server and a **Grafana** dashboard in their respective docker containers. The **Prometheus** server is responsible for gathering metrics and monitoring all the nodes that are part of the Swarm and **Grafana** is a dashboard that will allow us to visualize and interpret those metrics.

1.1 Video presentation

A video walk-through of this guide is available at the following URL:

`https://www.youtube.com/watch?v=_7Pd9nr4FV8`

2 System Requirements

The instructions in this document are applicable to 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.

First of all, it is necessary to install the required software to deploy locally the virtual machine that will be used to manage the infrastructure on the cloud. For that purpose, we will have to install **Vagrant** with **Virtualbox**, and configure the environment accordingly.

Skip to the next section if you have already installed the necessary software and configured your environment.

AGISIT 19/20	Project: Create a Lab	Group Number:	12
Container Technologies		Issue Date:	October 29, 2019
Container orchestration - Docker Swarm		Due Date:	December 31, 2019
Authors: 75834 - 83458 - 86437		Revision:	0.0

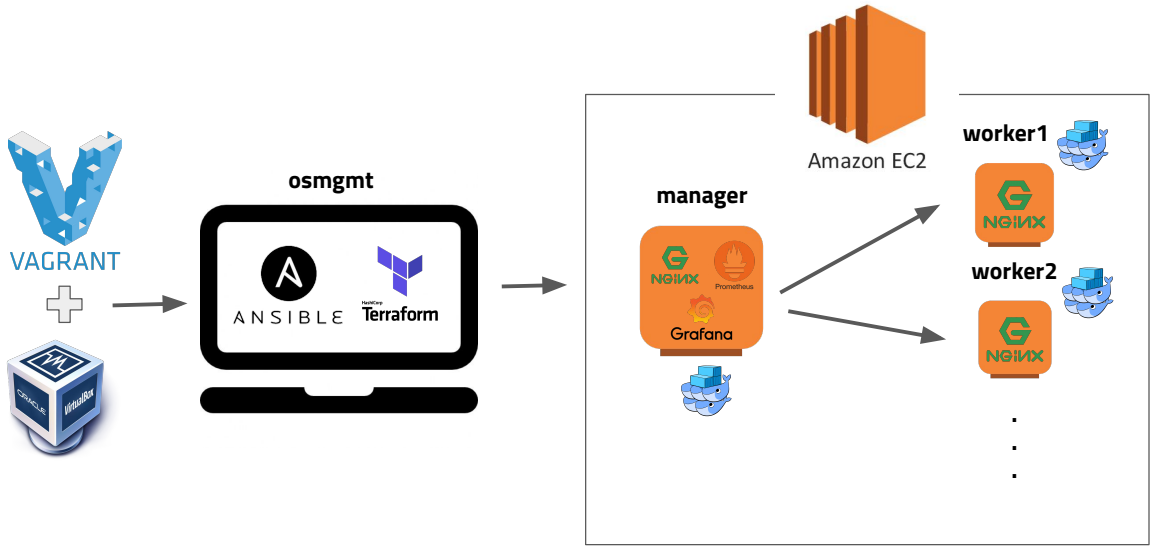

Figure 1: Deployment Architecture

2.1 Debian-based Linux distributions

For a Linux system, namely Debian-based distributions (e.g. Ubuntu), the standard package manager *apt-get* is already present. For other distributions, it is possible that different commands will have to be run (yet with the same purposes). To install the other packages open a Terminal and issue the following commands (answer `y` to the prompts, and several dependencies will also be installed).

- Git

```
$ sudo apt install git
```

- Oracle Virtualbox

  Start by editing the /etc/apt/sources.list file with this command:

```
$ sudo nano /etc/apt/sources.list
```

  Add the following line, and according to your distribution, replace ”mydist” with ”eoan”, ”bionic”, ”xenial”, etc:

```
deb [arch=amd64] https://download.virtualbox.org/virtualbox/debian
    mydist contrib
```

Then we need to download and register the Oracle public keys using the following commands:

```
$ wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc
    -O- | sudo apt-key add -
$ wget -q https://www.virtualbox.org/download/oracle_vbox.asc
    -O- | sudo apt-key add -
```

Now we can proceed to update the package lists and install Virtualbox:

```
$ sudo apt-get update
$ sudo apt-get install virtualbox-6.0
```

- Vagrant

  For Vagrant we need to get the latest version (not in the repositories) by downloading the package from the Vagrant website `http://downloads.vagrantup.com/` (adapt the command to the latest version):

```
$ wget https://releases.hashicorp.com/vagrant/2.2.6/vagrant_2.2.6
    _x86_64.deb
```

  Then issue the following command (from the downloads folder, adapt to your version):

```
$ dpkg -i vagrant_2.2.6_x86_64.deb
```

It is recommended that you restart before proceeding with the Lab experiment.

Afterwards, to add support for using the Virtualbox provider, install the *vagrant-vbguest* plugin with the following command in the Terminal:

```
$ vagrant plugin install vagrant-vbguest
```

2.2 Windows

The recommended Package Manager for Microsoft Windows is Chocolatey `https://chocolatey.org/`, similar to Linux *apt-get* or *yum*. Chocolatey was designed to be a decentralized framework for quickly installing Windows applications and tools, and is built on the NuGet infrastructure, currently using Windows PowerShell.

To install Chocolatey, open a privileged (i.e. administrative) Windows Command Prompt (cmd.exe) and paste the text string exemplified (at the command prompt) and press Enter. Please refer to the Chocolatey website for the available install options: `https://chocolatey.org/install`. You should have a string similar to the following:

```
C:\> @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe"
-NoProfile -InputFormat None -ExecutionPolicy Bypass -Command
"iex ((New-Object System.Net.WebClient). DownloadString('
https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;
%ALLUSERSPROFILE%\chocolatey\bin"
```

After Chocolatey installed, it is now time to install the other packages necessary for running the Lab experiment. Install the following programs using Chocolatey. Please note that for Git, the optional parameter also installs “Unix Tools” for Windows, such as “git-bash” that provides a command-line Terminal, and an SSH client:

- wget

```
C:\> choco install wget
```

- Git

```
C:\> choco install git.install --params "/GitAndUnixToolsOnPath"
```

- Oracle Virtualbox

```
C:\> choco install virtualbox
```

- Vagrant

```
C:\> choco install vagrant
```

After this point, and in order to settle everything, close the Console. Additionally, as an SSH (secure shell) client is not distributed with Windows, it is necessary to tell the system where to find the SSH client installed with Git:

- Open the Control Panel
- Go to System and Security
- Click on System, then on the Change Settings button
- Display the Advanced tab and click on Environment Variables...
- Look for the Path variable in the System variables list, select it then Edit...

At the end of the string, confirm if exists a path such as the following, otherwise add the path pointing to Git's bin folder:

```
C:\Program Files\Git\bin\;C:\Program Files\Git\usr\bin
```

You can now open a Terminal window using Git-Bash (that you should "pin to the taskbar" and configure its properties to "run as Administrator").

Afterwards, to add support for using the Virtualbox provider, install the *vagrant-vbguest* plugin with the following command in Git-Bash:

```
$ vagrant plugin install vagrant-vbguest
```

2.3 MacOS

The recommended Package Manager for macOS is Homebrew, similar the Linux *apt-get* or *yum*, `http://brew.sh`. Homebrew installs packages (binary or to be compiled) to their own directory and then symlinks their executables into */usr/local*. Homebrew provides Homebrew-Cask, implemented as a homebrew external command called *cask*. Homebrew-Cask extends Homebrew and brings its elegance, simplicity, and speed to the installation and management of GUI macOS applications such as Google Chrome.

As a first step you may need to install Command Line Tools for XCode from Apple (free from the App Store), in order for Homebrew to be able to compile applications from their source code. Macs do not have any of the developer's 'Command Line tools'

installed by default, so we need to install them before we can get anywhere. If you do not have XCode installed then open Terminal and at the shell prompt type the following:

```
$ xcode -select  --install
```

To install Homebrew, at the shell prompt use the script (starting at the ruby invocation) from the box below:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent
    .com/Homebrew/install/master/install)"
```

The shell prompt will tell you what it is about to do, and ask you if you want to proceed: press Enter to do so. The shell prompt may then ask for a password: this is the password to the Admin account on your computer. Type your password, and press Enter. When it's done, the shell prompt will say that the installation was successful, and ask you to run brew doctor. Do as it suggests:

```
$ brew doctor
```

This will make Homebrew inspect your system and make sure that everything is set up correctly. If the shell prompt informs of any issues, you will need to fix them, and then run **brew doctor** again to verify that all was correctly fixed. When everything is set up correctly, you will see the message `Your system is ready to brew`, and so, move on. It is now time to install the other packages for the Lab experiment environment. Open Terminal and install the following programs using Homebrew:

- wget

  ```
  $ brew install wget
  ```

- Git

  ```
  $ brew install git
  ```

- Oracle Virtualbox

  ```
  $ brew cask install virtualbox
  ```

- Vagrant

```
$ brew cask install vagrant
```

For each command the Terminal will start displaying lots of information, as Homebrew is keeping you updated on what it is doing. This flow of information will be a guide to let you know whether or not the computer is still working, and so, do not interrupt it.

Afterwards, to add support for using the Virtualbox provider, install the *vagrant-vbguest* plugin with the following command in the Terminal:

```
$ vagrant plugin install vagrant-vbguest
```

Preliminary Notes

Make sure you have one of the Operating Systems recommended in the section above (or know how to setup your own preferred Operating System accordingly) before starting this experiment. Also, do not forget to **setup your environment** if you haven't already.

It is not recommended to apply this setup to a virtual machine (nested virtualization), although possible, as the configuration requires access to a hypervisor environment (recommended 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.

It is **advisable to halt VMs** if running, and then **clean and destroy VMs from previous Lab experiments not related with this Lab**, as we may use in this Lab the same machine names.

Vagrant is used to create the deployment virtual environment, i.e., a virtual machine with **Ansible** and **Terraform** in it to act as the Orchestration and Configuration Management Node.

3 How to do it?

3.1 Getting the required files

Firstly, get the ZIP file with the support files. Unzip it into a directory with the name *dockeraws*.

Verify that the contents of the *dockeraws* directory are similar to the following:

```
.
|-- bootstrap-mgmt.sh
|-- aws-tenant
|.. |-- templates
|.. |.. |-- prometheus.yml
|.. |.. |-- daemon.json
|.. |-- ansible.cfg
|.. |-- aws-docker-swarm-destroy.yml
|.. |-- aws-docker-swarm-setup.yml
|.. |-- aws-nodes-setup.yml
|.. |-- inventory
|.. |-- terraform-aws-networks.tf
|.. |-- terraform-aws-servers.tf
|.. |-- terraform-provider.tf
|.. |-- terraform.tfvars
|-- Vagrantfile
```

Check that no virtual machines are currently running so that we are able to start with a clean environment:

```
$ vagrant global-status

id       name   provider   state   directory
-----------------------------------------------
abbbde4  osmgmt virtualbox poweroff /home/....
```

3.2 Setting up the orchestrator virtual machine

After performing the previous checks and ensuring that we are inside the *dockeraws* directory, we are now ready to launch the local virtual machine that will act as the orchestrator. The virtual machine, from here onwards referred to as *osmgmt*, will be launched as an **Ubuntu 18.04.03 LTS** instance and assigned the default IP of `192.168.56.10`. All of these configurations, among others, can be checked and modified by editing the *Vagrantfile*. However, for the purpose of this guide, **we recommend leaving the file as is**.

Inside the *dockeraws* directory, run *vagrant up*. After a while, you should observe that the *osmgmt* machine is running by executing the *vagrant status* command.

```
$ vagrant up
...
$ vagrant status
Current machine states:
osmgmt                    running (virtualbox)
```

To ensure the minimum amount of communication is possible between the host and the *osmgmt* machine, perform a *ping* request.

```
$ ping 192.168.56.10
PING 192.168.56.10 (192.168.56.10) 56(84) bytes of data.
64 bytes from 192.168.56.10: icmp_seq=1 ttl=64 time=0.292 ms
64 bytes from 192.168.56.10: icmp_seq=2 ttl=64 time=0.252 ms
```

If the *ping* request is successful we are ready to ssh into the *osmgmt* machine.

```
$ vagrant ssh osmgmt
```

The *osmgmt* was also provisioned resorting to the *bootstrap-mgmt.sh* file used to install and configure the necessary tools such as **Terraform** and **Ansible**.

3.3 Setting up an account at Amazon Web Services

The **Amazon Web Services** platform is the public cloud where our infrastructure will be deployed. You will need an account in order to have access to the credentials needed to use the services available.

Since there are various ways for an user to configure their AWS account and security credentials, we will not go through the process of explaining every possible method in this guide. We will assume that the reader **is enrolled in the AGISIT course this semester**, and as such we will explain how to proceed in that case.

To have access to the services, you should have received an e-mail from the instructors with an invitation to join **AWS Educate** and join a *Classroom*. Follow the link provided in the e-mail and follow the instructions in the website. You'll soon receive another e-mail to verify your e-mail address. After verifying your e-mail address, you'll receive an e-mail confirming that your application to **AWS Educate** has been approved, and requesting you to login and set your password.

3.4 Amazon EC2 Deployment using Terraform

Terraform will be used to deploy and manage the virtual machines that will be launched in the chosen cloud platform which in this guide will correspond to the *Amazon Web Services* platform.

3.4.1 Terraform configuration files

Firstly, we need to provide **Terraform** the necessary *credentials* so that it is able to perform calls to the AWS API. For that, first login to your **AWS Educate** account at `https://www.awseducate.com/student/s/classrooms`.

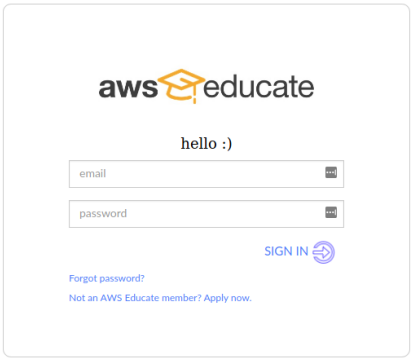

Figure 2: AWS Educate login page

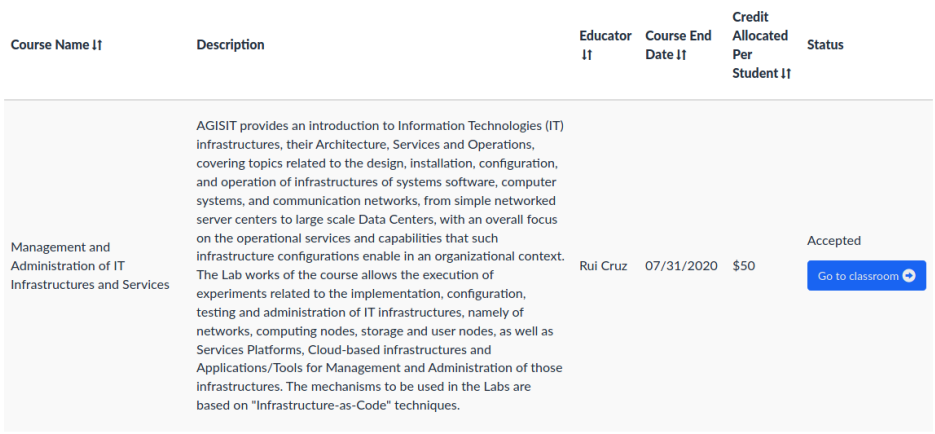

Figure 3: AWS Classrooms page

You'll be taken to the page with your classrooms. Click on the button *Go to classroom* for the AGISIT course. Click on *Continue* at the prompt to accept the conditions, and be taken to your workbench.

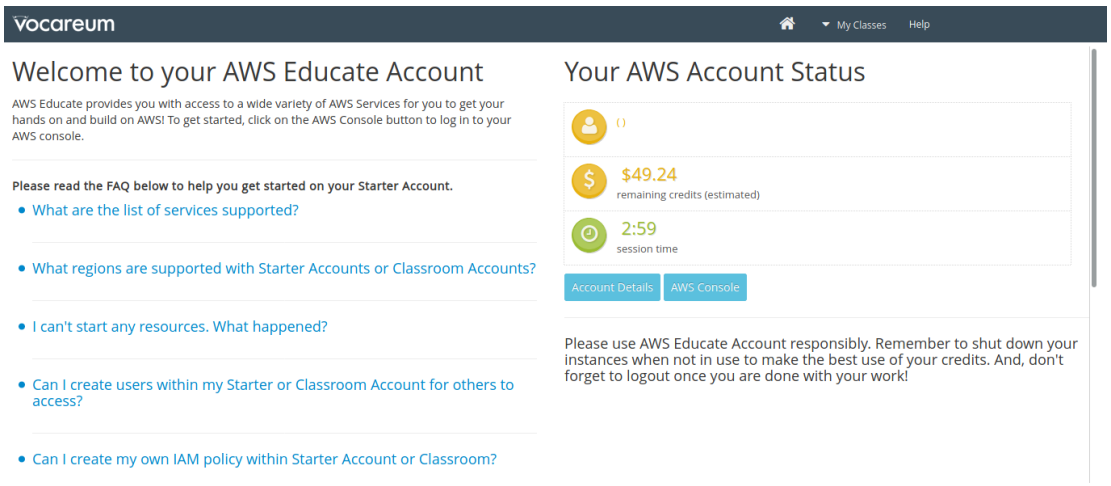

Figure 4: Workbench page

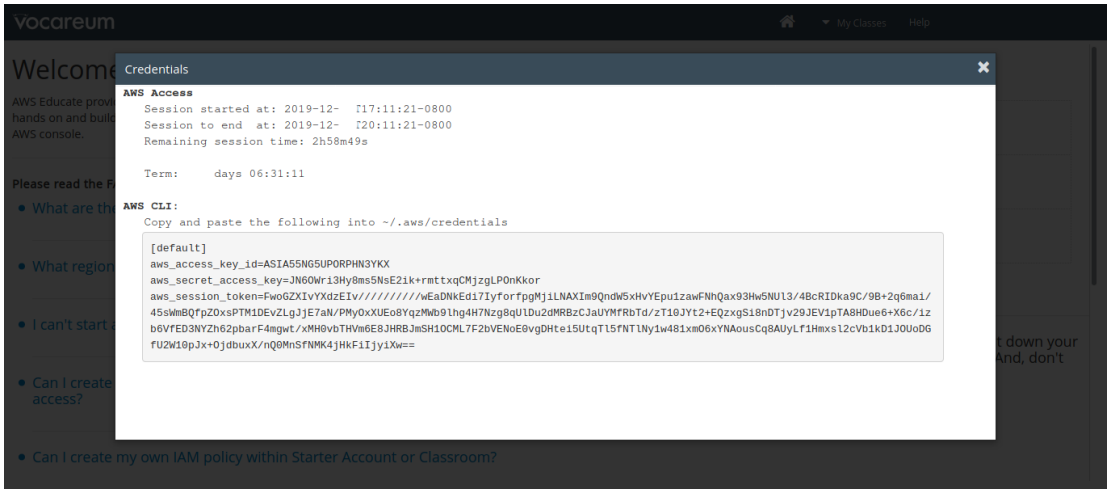

Figure 5: AWS credentials page

Next, in your workbench, click on the button *Account Details*, under Your AWS Account Status. On the box that appears, click on the button *Show* next to AWS CLI.

Here you'll see the credentials you'll need in order to deploy the infrastructure with **Terraform**. These credentials are **temporary**, so if you take more than a few hours to complete the lab, you'll need to replace the credentials with new ones, obtained from this same source. If your Terraform commands seem to hang, it's possible that the cause is the expiration of the credentials.

Edit the **terraform.tfvars** file and replace the *access_key*, *secret_key* and *token* vari-

ables with the respective AWS credentials.

```
# terraform.tfvars
access_key          = "AWS ACCESS KEY ID"
secret_key          = "AWS SECRET ACCESS KEY"
token               = "AWS SESSION TOKEN"
...
```

Before the Amazon EC2 instances can be launched on the AWS platform let us walk through some of the **Terraform** configuration files that will enable us to do just that. The **terraform-aws-servers.tf** file declares two types of AWS resources: *aws_key_pair* and *aws_instance*. The *aws_key_pair* resource will be used to provide access to the EC2 instances by registering the supplied SSH public key into AWS. Additionally, there are two separate *aws_instance* resources declared. One corresponds to the node responsible for managing the Docker Swarm and the other corresponds to the worker nodes who will join it. For each *aws_instance* resource the following parameters are defined:

- ami: The *Amazon Machine Image* used in the instance. In this guide we will be using "ami-00068cd7555f543d5", which is an *AMI* that runs a *4.14 Linux kernel* and is optimized for performance on the **Amazon EC2** platform.
- instance_type: The **Amazon EC2** instance type.
- key_name: The key pair resource that will be used to access the instance.
- subnet_id: The **VPC**'s *subnet* ID where the machine will be launched in.
- vpn_security_group_ids: The list of **AWS Security Groups** which the instance will belong to.
- private_ip: The private IP address assigned to the instance.

The IP defined in the *private_ip* field has to belong to the range of IPs available in the **VPC (Virtual Private Cloud)** and **subnets** we are going to use.

Note: This Lab guide and the support files assume that the **default VPC** and the **default subnets** of said VPC have not been changed or removed in the region `us-east-1`. To know how to restore them, visit the following URL: `https://docs.aws.amazon.com/vpc/latest/userguide/default-vpc.html`.

Currently this file is configured so as to launch one manager and two workers. We can define the number of worker nodes to deploy by changing the *count* variable located in the corresponding resource block.

The **terraform-aws-networks.tf** file declares an *aws_default_vpc* resource, an *aws_subnet* data source, and an *aws_security_group* resource.

The *aws_default_vpc* resource captures the information of the default **VPC** of the region defined in **terraform-provider.tf**. When deleted, it will not delete the default **VPC**.

The *aws_subnet* data source reads the information of the **subnet** inside a given **VPC** that corresponds to the parameters given. It does not modify any resources. In this Lab, it reads the information of the **subnet** with the CIDR block `172.31.32.0/20`, corresponding to the **subnet** with the private IPs we want.

A *security group* acts as a virtual firewall for your instance to control inbound and outbound traffic of the **VPC**. In order for the instances to communicate with each other as well as with devices outside the network, rules have to be created. As all incoming traffic to the instances is blocked by default, we need to specifically define which traffic we are going to allow. If we inspect the file we can observe that several ports are opened for incoming traffic such as ports 22 and 80, for SSH and HTTP connections respectively, among others which will be used for monitoring tools.

3.4.2 Create an SSH key pair

The next step is to generate an ssh key pair so that we are able to remotely access the virtual machines. Say yes to every prompt (by pressing ENTER) and *make sure not to* assign any passphrase nor change the default path of where the keys will be stored. **Terraform** is already configured so that the generated keys are imported into AWS.

```
vagrant@osmgmt:$ ssh-keygen -t rsa -b 2048
```

3.4.3 Deploying the infrastructure with Terraform

Now move to the *aws-tenant* directory and run *terraform init* so as to initialize the **Terraform** working directory.

```
vagrant@osmgmt:$ cd aws-tenant
vagrant@osmgmt:~/aws-tenant$ terraform init
...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" ...
...
Terraform has been successfully initialized!
...
```

Afterwards run *terraform plan* in order to create a terraform *execution plan*. This plan will be used to determine what actions have to be performed so as to achieve the desired state declared in the Terraform files.

```
vagrant@osmgmt:~/aws-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
 <= read (data resources)
Terraform will perform the following actions:
 <= data "aws_subnet" "agisit-subnet"  {
...
Plan: 6 to add, 0 to change, 0 to destroy.
...
```

If no errors are reported, then we are ready to deploy the instances on AWS by running the *terraform apply* command and typing ”yes” when prompted to do so.

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

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:

 <= data "aws_subnet" "agisit-subnet"  {
...
  + resource "aws_default_vpc" "default-vpc" {
...
  + resource "aws_instance" "manager" {
...
```

```
  + resource "aws_instance" "worker" {
...
  + resource "aws_instance" "worker" {
...
  + resource "aws_key_pair" "keypair" {
...
  + resource "aws_security_group" "agisit-security-group" {
...
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
...
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

ssh_swarm_keypair = osmgmt_swarm
swarm_manager_public_ip = [
  "34.236.244.78",
]

swarm_worker_public_ip = [
  "worker1 = 3.94.29.135",
  "worker2 = 34.200.249.101",
]

terraform-provider = Connected with AWS at region us-east-1
```

Notice that in the output the *public IP addresses* assigned to each instance are displayed.

3.5 Docker Swarm deployment using Ansible

Ansible is a tool that can be used to provision, configure and deploy applications on remote machines. We will be using **Ansible** to install the necessary dependencies for building the Docker Swarm as well as deploying services on it.

3.5.1 Populate hosts files

The next step is to prepare the **Ansible** configuration files so that we can provision the instances, and edit the **osmgmt** hosts file, in case we want to directly SSH into the machines. Let us start by editing the **inventory** file and replacing the value of the *ansible_host* variable associated with each machine with the corresponding public IP address assigned to it.

```
vagrant@osmgmt:~/aws-tenant$ nano inventory
```

The end result should look similar to the following:

```
# file: inventory
# for tenant hosts file

manager     ansible_host=34.236.244.78 ...
worker1     ansible_host=3.94.29.135 ...
worker2     ansible_host=34.200.249.101 ...

# The local deployment host VM
[osmgmt]
localhost           ansible_connection=local

# Contains the swarm worker nodes
[workers]
worker1
worker2

# Contains the swarm manager nodes

[managers]
manager
```

```
[targets]
manager
worker1
worker2
```

You can edit the hosts file with the following command:

```
vagrant@osmgmt:~/aws-tenant$ sudo nano /etc/hosts
```

The file should have these lines added (with the IP addresses of the output):

```
...
34.236.244.78    manager
3.94.29.135      worker1
34.200.249.101   worker2
...
```

3.5.2 Establish an SSH Trust

Next, we shall establish a ”password less” but secure access to the infrastructure nodes, using the *ssh-keyscan* command for all nodes, and piping the output to the file **known_hosts**.

```
vagrant@osmgmt:~/aws-tenant$ ssh-keyscan manager worker1 worker2
    ... workerX >> ~/.ssh/known_hosts
```

Afterwards, we need to confirm that we are able to manage all the EC2 instances from the *osmgmt* node using **Ansible**. We can do that by performing the following command:

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

```
    "ping": "pong"
}
manager | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
worker1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
worker2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

If all pings succeed then we are ready to start provisioning the instances.

3.5.3 Playbook execution

If everything went well so far, we are now ready to start provisioning the instances. First, we need to run the **aws-nodes-setup.yml** playbook which is responsible for:

- Updating the package lists of all nodes
- Upgrading the packages of all nodes
- Installing and configuring Docker on both the manager and the workers
- Installing other packages necessary for Ansible to run the other playbooks successfully

Execute the following command in order to run the playbook:

```
vagrant@osmgmt:~/aws-tenant$ ansible-playbook aws-nodes-setup.yml
```

Next we need to build the cluster and deploy the services we want to be replicated. For that we run the **aws-docker-swarm-setup.yml** which is responsible for:

- Enabling Prometheus compatible Docker metrics
- Starting a Docker Swarm in the manager node
- Adding all worker nodes to the Swarm
- Starting a Prometheus and a Grafana container on the manager node
- Starting node-exporter and cAdvisor services on all nodes part of the Swarm
- Start a nginx service with three replicas on the Swarm

```
vagrant@osmgmt:~/aws-tenant$ ansible-playbook aws-docker-swarm-setup.yml
```

To confirm that everything went as expected, we need to ssh into the manager node and verify what nodes are in the Swarm and what services are running.

```
vagrant@osmgmt:~/aws-tenant$ ssh -i ~/.ssh/id_rsa ec2-user@manager

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___| ___|___|

[ec2-user@PUBLIC_IP_OF_MANAGER ~]$ docker node ls
...
[ec2-user@PUBLIC_IP_OF_MANAGER ~]$ docker service ls
...
```

If all nodes we launched are in the Swarm and the *cadvisor*, *node-exporter* and **nginx** services are running then we can move on to configure the tools we are going to use to monitor the Docker Swarm.

3.6 Swarm monitoring tools

3.6.1 Prometheus

Firstly, we are going to access the **Prometheus** server dashboard to see if the connection to all the nodes is up. Open a web browser and access the dashboard through the following url:

`http://PUBLIC_IP_OF_MANAGER:9090/targets`

A web page similar to the one displayed in Figure 6 should appear.

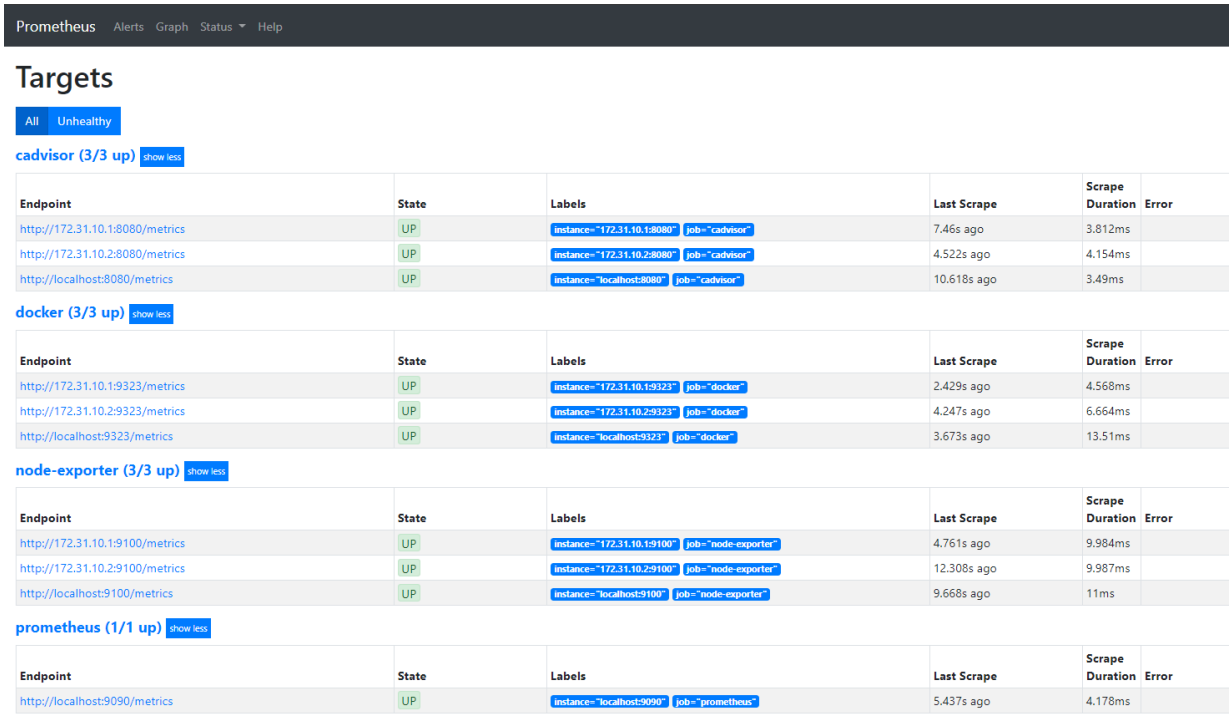

Figure 6: Prometheus server dashboard

If all the nodes and services have the *State* showing as *UP* then it implies that the swarm manager node can reach all the monitoring services.

3.6.2 Grafana

Grafana is a dashboard that will allow us to query and visualize the recorded metrics from the monitoring services we have deployed. As an initial configuration, access the following url:

`http://PUBLIC_IP_OF_MANAGER:3000`

A login page will show up similar to the one depicted in Figure 7. The default credentials are **admin** for both the *username* and the *password*.

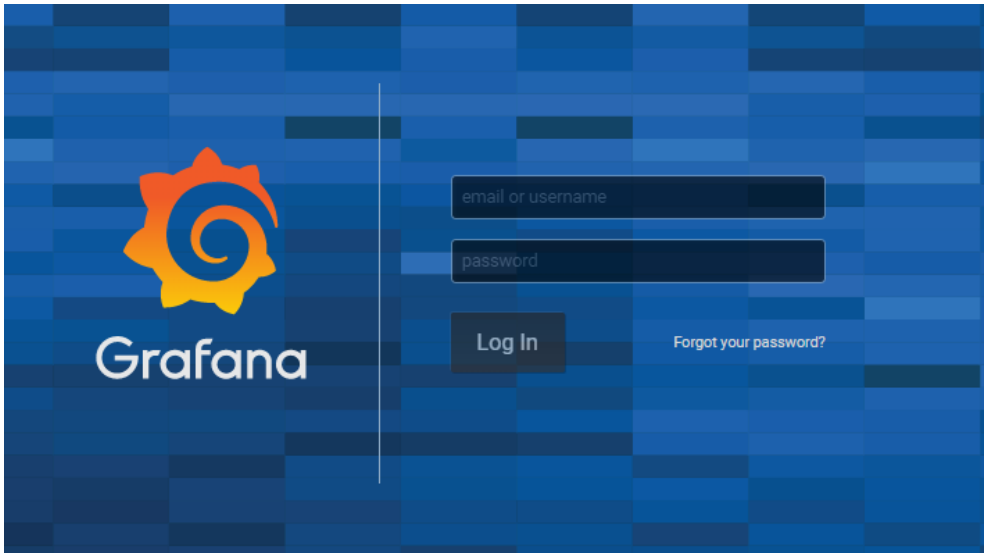

Figure 7: Grafana login page

Now that we are logged in, let us import a dashboard configuration that will allow us to see useful metrics for each node. Before we can do that, we first need to add a new data source. Select ”add a new data source” and then select ”Prometheus”. A form similar to the one shown in Figure 8 should appear. Fill the ”URL” field with the url for the Prometheus server using the private IP of the swarm manager:

```
http://PRIVATE\_IP\_OF\_MANAGER:9090
```

For the ”Scrape interval” field let us input **15s** which means that the data will be updated every 15 seconds. Now press the ”Save & Test” button and the data source should now be added.

The next step is to import a dashboard so that we can visualize all the data we are scraping. There are a number of custom dashboards created by community members at:

```
https://grafana.com/grafana/dashboards
```

We will select one that works with **Prometheus**, **node-exporter** and **cAdvisor**.

Import a new dashboard and in the ”ID” field write **11074** as shown in Figure 9. Then select the data source we have just created and press the ”Import” button. We should be taken to a page where we can keep track and visualize several metrics and personalize them as we like. It might be a good idea to take a moment and explore the several features of **Grafana**.

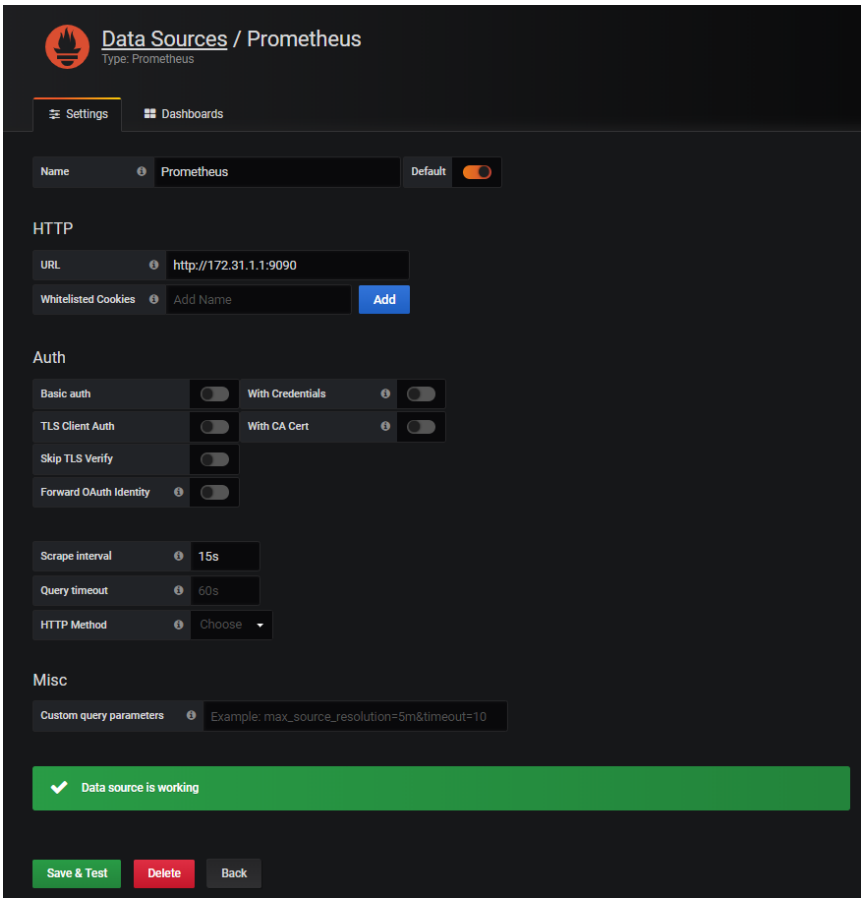

Figure 8: Prometheus data source form page

3.6.3 Nginx webserver

The **nginx** webserver uses a fully-fledged **nginx** container, with 3 replicas running in the Swarm. The nginx container was built using the Official **nginx** container available in Docker Hub and the configuration files required to create a simple webpage that shows the hostname and IP address of the container that replied to the request.

To see the webpage, access the following url:

```
http://PUBLIC_IP_OF_MANAGER
```

The **Docker Swarm** performs the load balancing automatically, so that every replica answers the requests made. However, the hostname and IP address shown in the page don't change, even if two requests are answered by different replicas in different nodes.

That happens because the replicas are completely equal to one another, and the hostname and IP address shown are not the ones of the node of the replica, but of the container that is being replicated.

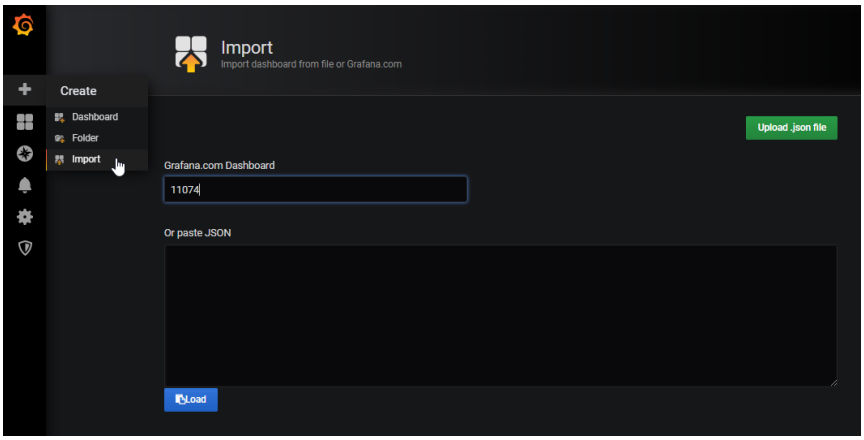

Figure 9: Prometheus data source form page

Hello!

URI = /

My hostname is 180ce739019c

My address is 10.255.0.8:80

Figure 10: Nginx webpage showing IP address and hostname of container

3.7 Finishing up

In order to shutdown the whole infrastructure we start by running the **aws-docker-swarm-destroy.yml** playbook so as to gracefully destroy the Docker Swarm.

```
vagrant@osmgmt:~/aws-tenant$ ansible-playbook aws-docker-swarm-destroy.yml
```

Then we can proceed to terminate the instances deployed on Amazon EC2 with the help of **Terraform**. Answer ”yes” to the prompt when requested to do so.

```
vagrant@osmgmt:~/aws-tenant$ terraform destroy
```

Finally, exit the ssh session on the *osmgmt* machine and stop it:

```
vagrant@osmgmt:~/aws-tenant$ exit
$ vagrant halt osmgmt
```

Verify the global state of all active Vagrant environments on the system with the command `vagrant global-status`. Confirm that the status of the VM is “powered off”.