|  |  |  |
| --- | --- | --- |
| **Module Code**  **COMP5850M** | School of Computing  University of Leeds  **Coursework 1 - Report** | University of Leeds logo |

Full Name: Hiresh Darshan Joshi Username: sc20hdj

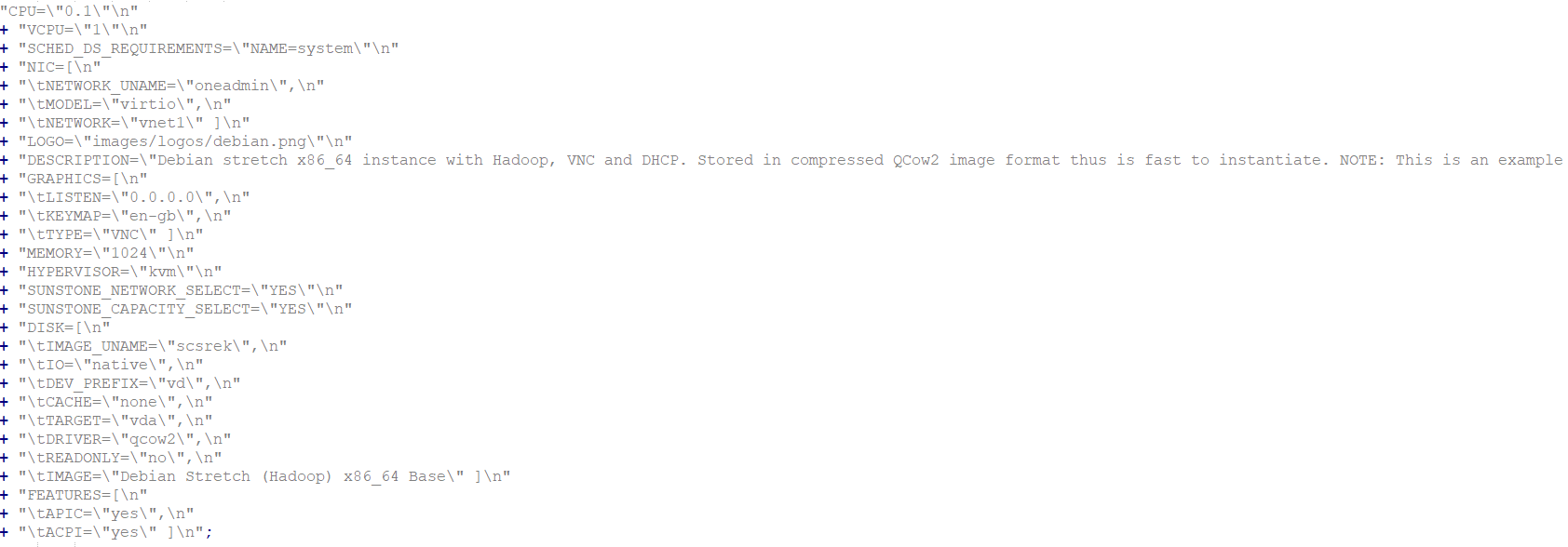
Coursework Title: VIM Deadline Date: 12/03/2021

**Part 1: VM Instantiation (10 marks)**

*Provide an explanation of the implementation of this task. The inclusion of the entire code is not required but you may include snippets if you wish.*

***VM template (2 marks)***

In order to instantiate any Virtual Machine, we need one specific Machine Image required. It may be an image of any guest OS or other variants. Open Nebula has list of 21 images out of which there is a provision for 19 Templates in the server. For my Virtual Machine I have used template of Debian Stretch (Hadoop) x86\_64 (VNC, DHCP) Instance. The template provides information about Graphics, key Description, CPU, Memory, Network Interface Card, Disk Capacity and few other characteristics. The template was available on Open Nebula and in order to use it I have converted it to a Java string as follows:-



*Fig 1.1*

***Information OpenNebula provides about the VM (2 marks)***

After Instantiating the VM, there are few aspects to be considered regarding the VM in order to perform scheduling and Migration on it. Some of the fundamental details that we can from VM are the VM ID which is a unique identifier to identify the VM on the server, along with the state at which the VM is running. We get all this information from ‘vm.info()’ which stores all the metadata regarding the Virtual Machine. Alongside, it is also important to know the host on which the Virtual Machine is running, to be precise ‘HostName’ and ‘HostID’. Open Nebula API prevails this feature being an open source platform by extracting all the information from ‘vm.xpath()’. Following is the snapshot of all the metadata that VM stores under vm.info():



*Fig 1.2*

Also in order to find the host related information I have added two lines, following is the snippet:

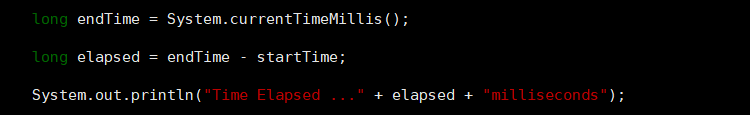


*Fig 1.3*

***Measure the time it takes to instantiate the VM (1 mark)***

In order to measure the time it takes to create a VM, you may first allocate the VM and use the following formula:-

Time Elapsed = End Time – Start Time

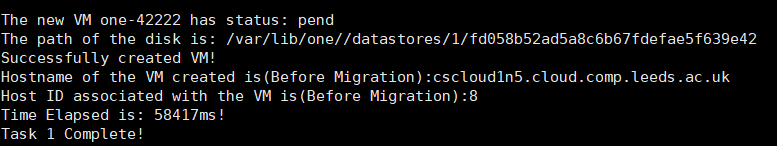


*Fig 1.4*

***Explain how you have obtained these measurements (1 mark)***

In order to calculate the time taken to instantiate a VM, I created an attribute StartTime after VM was allocated. From here, the system starts the timer, here I have recorded the live timer in milliseconds using ‘currentTimeMillis()’ function. After the timer starts, the VM begins its lifecycle from pending - > prolog -> boot -> running state and after it reaches the running state the timer stops and this entire time is recorded in milliseconds. Usually for a VM to come in a running state takes 1-2 minutes, but it is also dependent on the template you are using. For Debian Hadoop image it took me approximately 62067milliseconds.

***Evidence of successful run, e.g. screenshot (4 marks)***



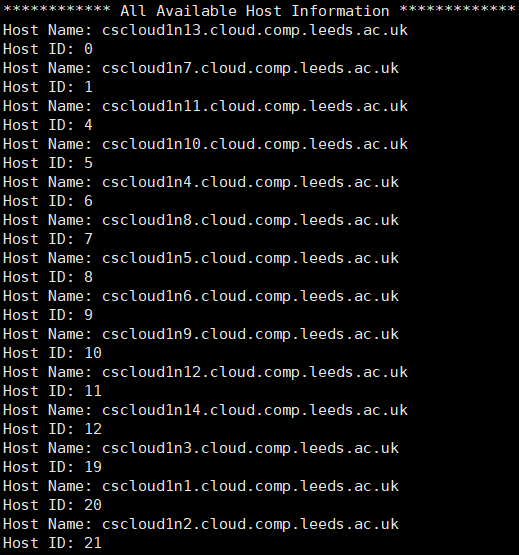
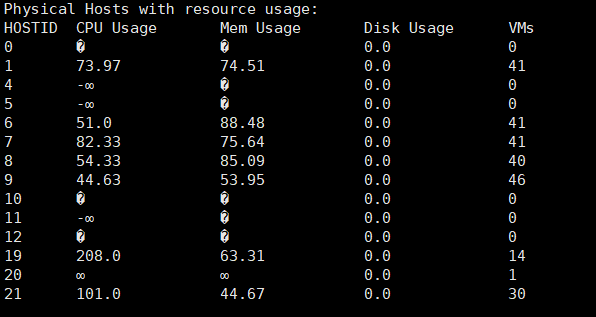
*Fig 1.5*

**Part 2: VM Migration (25 marks)**

*Provide an explanation of the implementation of this task. The inclusion of the entire code is not required but you may include snippets if you wish.*

***Requirements (4 marks)***

After creating a Virtual Machine, the first and foremost part is to check the associated Host Name and Host ID associated with it. While using any scheduling algorithm we must first prepare an algorithm of how to schedule it. Creating a Virtual machine in Open Nebula is easy, but there are lot of factors to be considered when it comes to Scheduling and Migration. In order to schedule and migrate , we must require the host information. There are very classes created by Open Nebula some of them are already provided in the open-source API such as ‘HostPool’, ’UserPool’,etc. HostPool provides details about all the physical host that are running on the server. In csgate1 server there are several host present, we can get a list of all this host using ‘HostPool’ class.

**

*Fig 2.1 Fig2.2*

In order to schedule, we must consider one of the following factors: CPU Usage, Memory, Disk usage, Number of VMs acquired by the host,etc. Open Nebula provides us with a privlige to check the Resource Usage of each and every host and decide on the scheduling parameter. In order to migrate and schedule, the primary requirement is to the Host present as well as the resources acquired by these hosts.

However, while displaying we can see two types of host and decided where to migrate:

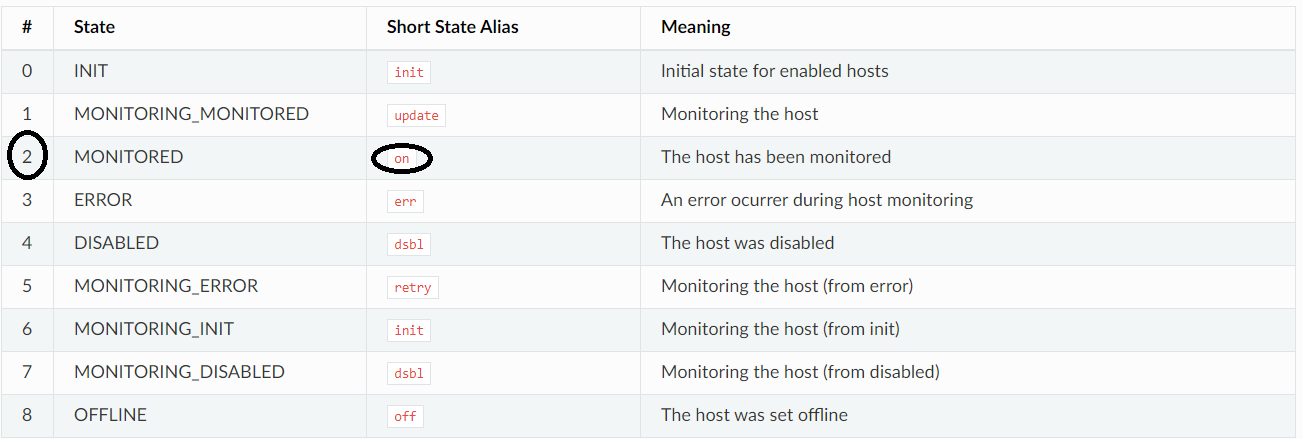
1. Powered-ON Hosts(presenting approximate resource usage)
2. Powered-OFF/Shutdown Hosts (Garbage values)

All these things are mandatory requirement in Open nebula before performing scheduling and migration.

***Solution Design (4 marks)***

While designing the scheduling algorithm as discussed in the above section, we need host details from HostPool class. We first go display the host details with resource usage. The scheduling algorithm that we are going to use is First Come First Serve (FCFS). For FCFS we are going to consider CPU Usage parameter and going to allocate the VM to the host having lowest CPU Usage. For this operation, we may sort the host list in ascending order from lowest to highest CPU usage.

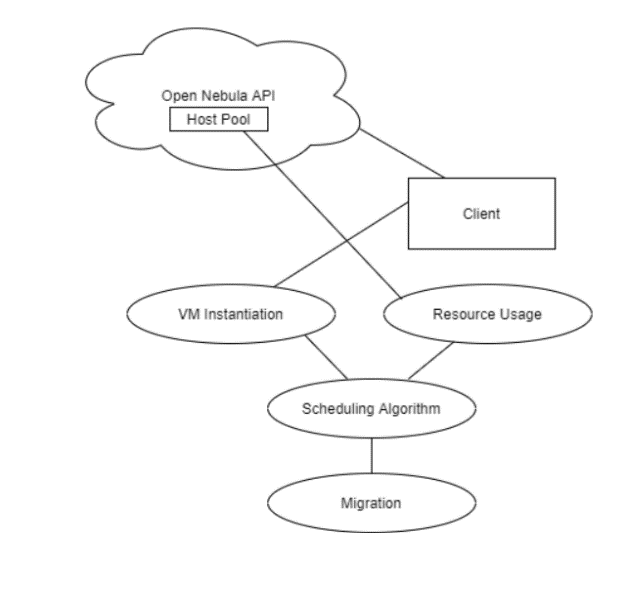
As mentioned previously HostPool provides information about all the available hosts on the server. For scheduling we must consider only those hosts that are POWERED-ON. Open Nebula API gives us the privilege to know the state of the Host. Following are the list of states that host can be in:



*Fig 2.3*

After selecting the host to migrate we can now deploy the VM to the chosen host for migration. I have made the following diagram to show a brief idea about the Scheduling a migration process in Open Nebula:

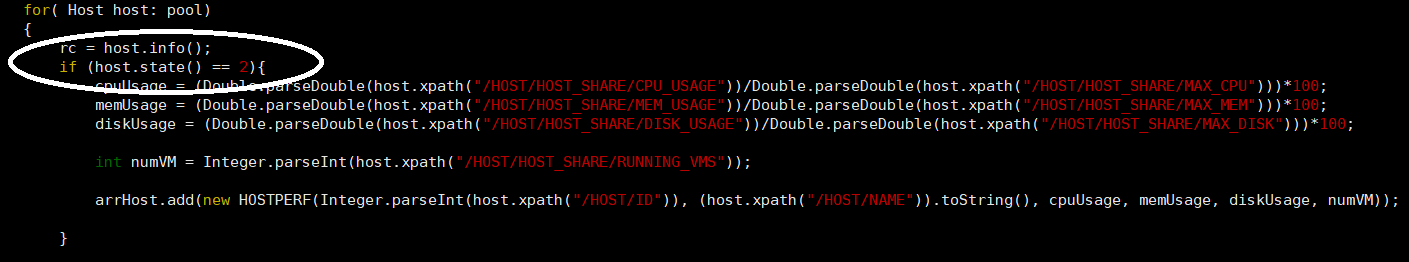
1. Client is a core entity here, in order to access the csgate1 server, we must create a client in order to authenticate to create a VM
2. Client requests the required information about the host associated by Retrieving information from the HostPool using Open Nebula API
3. After the client is authenticated the cycle goes in VM Instantiation phase:
   1. Specify the template of VM
   2. Retrieve VM related information such as VM ID, VM Name, Disk name, status of VM from ‘vm.info()’
   3. Retrieve host related information associated with VM from ‘vm.xpath()’
4. After creating the VM, we will schedule the VM to the decided host to Migrate as per the algorithm

**

*Fig 2.4*

***Implementation (4 marks)***

In order to implement the scheduling and migration, there are couple of parameters to be considered and changed. For the displayed host information I am only considering the POWERED-ON hosts. As per the table mentioned in the Design step we check only hosts that has state=2 that is ON state and display information regarding it. Refer the following block of code, I have tweaked the loop in order to display the specific result:



*Fig 2.5*

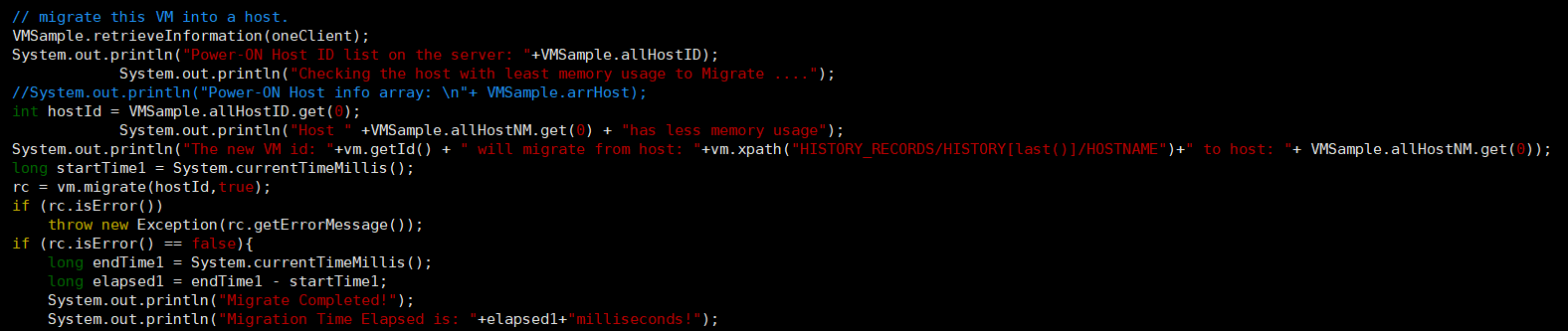
Also for the scheduling we are considering the FCFS policy so we will choose the first host having lowest CPU Usage. We therefore, sort the list in ascending order, by adding this



*Fig 2.6*

After identifying the host where to migrate, comes the migration part. I have created a sample object which retrieves information about the client as well as the VM. By displaying the table we can locate the Host to migrate. There are two types of migration in VM, one is migrate() and other one is liveMigrate(). Both plays a similar role but in liveMigrate the system is not shutdown. In order to migrate we must provide two paramters to the function ‘vm.migrate(hostID, boolean)’

Refer the following snippet to check how I have performed migration of my Virtual Machine:

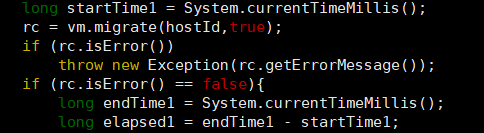


*Fig 2.7*

***Measure the time it takes to migrate the VM. (2 marks)***

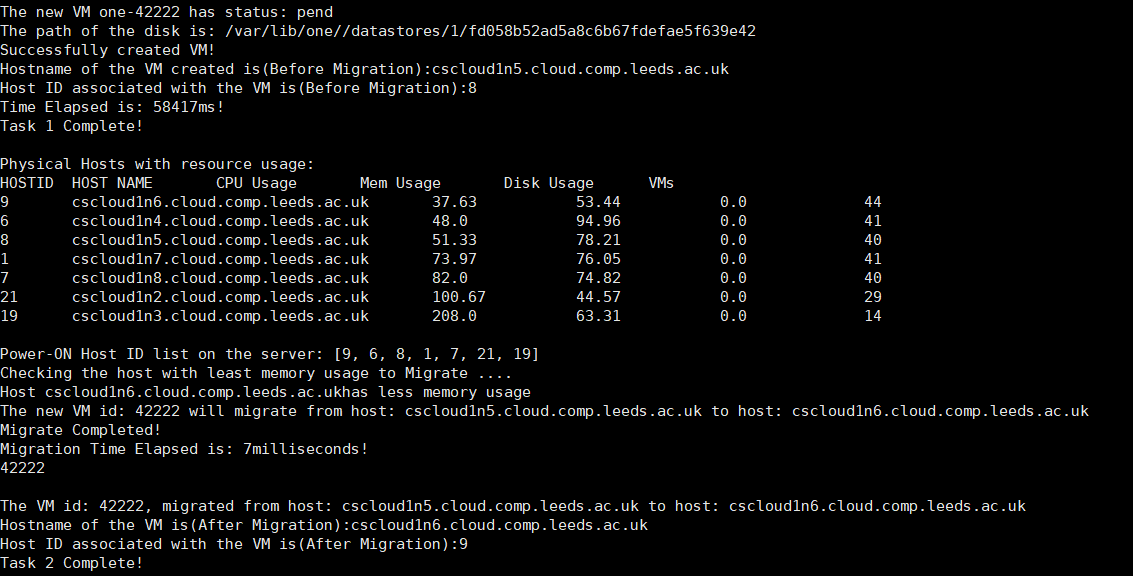
In order to calculate the time taken to migrate we start the time from the part where the migration process has begun. The formula to calculate the migration time is:

*Time Elapsed = End Time – Start Time.* The time is calculated in milliseconds using currentTimeMillis() function.

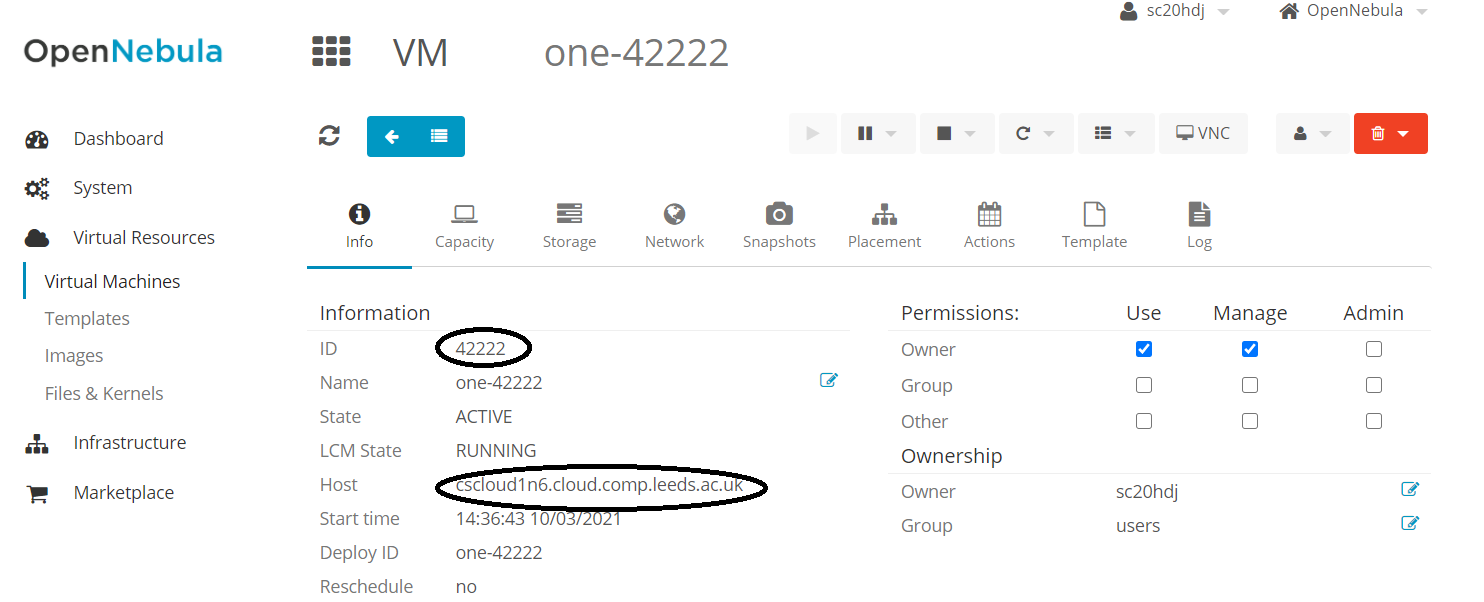


*Fig 2.8*

***Evidence of successful run, e.g. screenshot (8 marks****)*

**

*Fig 2.9*

**

*Fig 2.10*

***Discussion of the results (3 marks)***

The following output depicts that the VM has been successfully migrated. Above is the Task1 of the coursework. After instantiating the VM the physical hosts which are POWERED-ON are displayed with their resource usage respectively. In order to migrate to a new host we must consider any of the approach for scheduling. Here since I have taken CPU Usage into consideration, we can see that Host ID **‘9’** has the least CPU usage that is **37.63**. After knowing the destination host we begin the migration process. Earlier the VM was in ‘**cscloudln5.cloud.comp.leeds.ac.uk**’ and was destined to HOST ID ‘**8**’ which had CPU Usage of **51.33** which is greater than the decided host. After migration the VM belongs to host ‘**cscloudln6.cloud.comp.leeds.ac.uk**’ associated with HOST ID ‘**9**’.

The result also shows that it took 7 milliseconds to migrate the VM to the destination host.

**Part 3: VM Monitoring and Performance (15 marks)**

***Details of the application considered (2 marks)***

In order to install any application on VM we must first log in to the server using *ssh root@ip-address*

I am running a Debian Hadoop server so in order to install any application we must use OS specific commands to install it. For ex, installing an application on Linux will have different commands than installing it on Debian. For this part of task I am going to install ‘**Nginx**’ web server on the VM. Along with Nginx we also install ‘**stress**’ on the VM. With stress we can add the load on CPU, Memory, I/O disk and network.

***Design of the experiments (2 marks)***

1. ***Installing nginx-***

Because Nginx is available in Linux’s default repositories, it is possible to install it from these repositories using the ‘apt’ packaging system.

*apt install nginx*

Before testing Nginx, the firewall software needs to be adjusted to allow access to the service. Nginx registers itself as a service with ufw upon installation, making it straightforward to allow Nginx access.

ufw app list

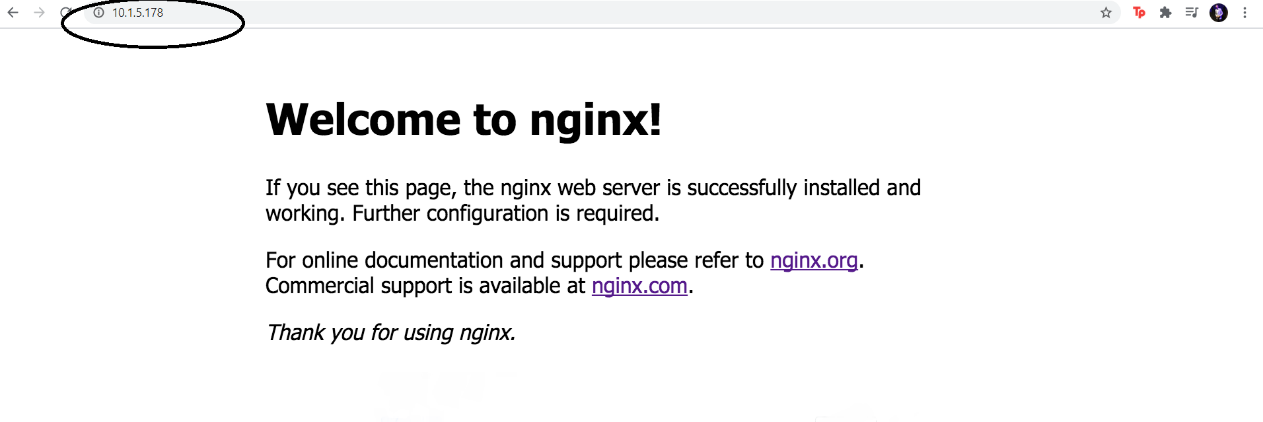
We can see listing of all application profiles

Here, I am trying to allow Nginx HTTP which by default opens port 80 and port 443 using the following command

ufw allow ‘Nginx HTTP

Now, the Nginx will start with the help of the command:

systemctl nginx service start



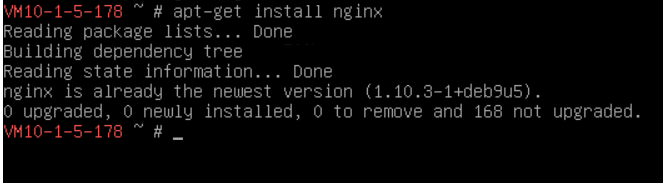
*Fig 3.1*

1. ***Installing stress-***

Stress can be installed on VM using following command: apt-get install stress

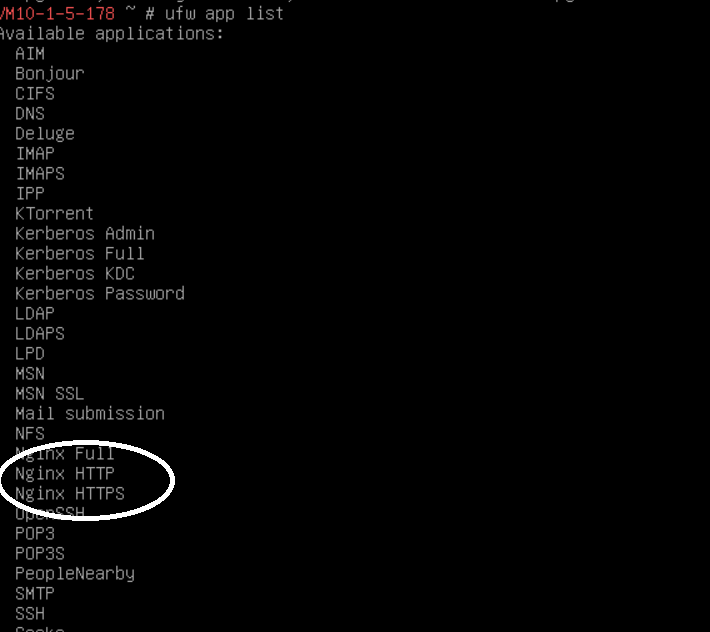
***Implementation of the experiments (2 marks)***

Installing nginx



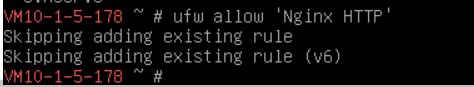
*Fig 3.2*

Checking available application list



*Fig 3.3*

Now we need to allow ‘Nginx HTTP’ to run on our VM



*Fig 3.4*

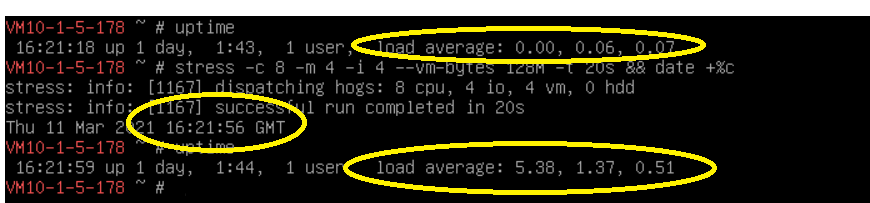
Nginx has successfully been created on the VM and we can access it on web through *http://ip-address*

Default port on which Nginx listen is 80, but we can also change the TCP port using netstat command.

After installing Nginx we run some stress on the VM using following command:

A load average of eight is imposed on the system by specifying eight CPU-bound processes, four I/O-bound process, and four memory allocator.

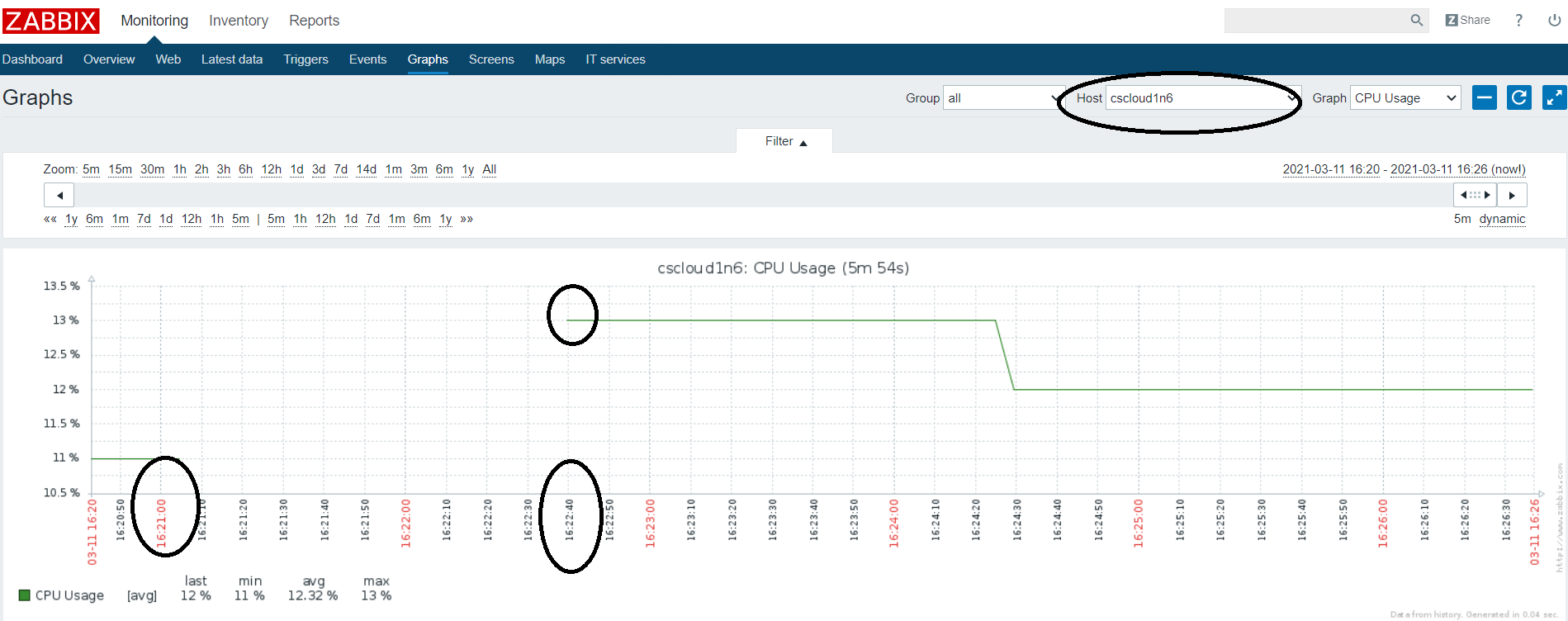
*stress –c 8 –m 4 –i 4 - -vm-bytes 128M –t 20s*

**

*Fig 3.5*

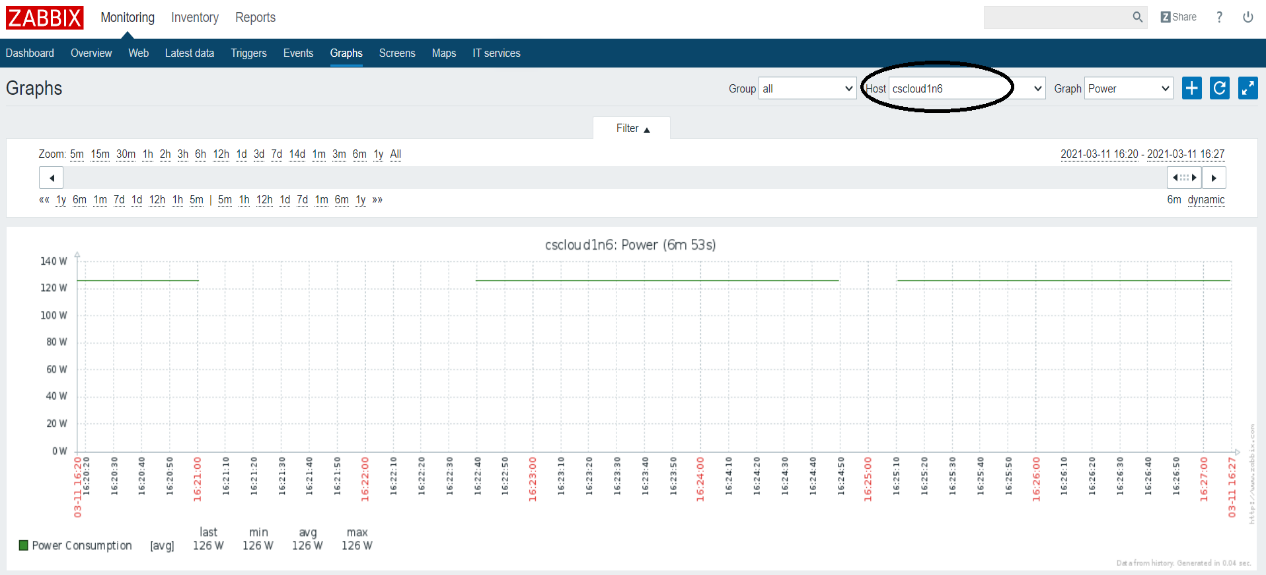
***Evidence of successful run, e.g. screenshot (5 marks)***

Monitoring CPU Usage on host ***cscloudln6***



*Fig 3.6*

Power consumption on host ***cscloudln6***



*Fig 3.7*

***Discussion of results (4 marks)***

After installing stress on the VM, the most important task was to register the VM to the host in Zabbix. By using **vm\_spot\_metrics** we achieved this task.

Installing stress on any Virtual Machine wouldn’t give any result until you apply some stress on it.

In the last step of installation after carrying out few test cases I applied some load on the VM in terms of CPU, memory and I/O disk. I recorded the load on VM by using default option ‘uptime’ in VM. The load average before stress was: **0.00, 0.06** and **0.07** for CPU, Memory and I/O respectively. After adding certain stress on it the load average was: **5.38, 1.37** and **0.51**. For reference, I have also printed the $timestamp at which this operation was performed so that we cross check with the metrics on Zabbix. The recorded timestamp was **11 March 2021 16:21:56 GMT.**

Monitoring the results, we can see graph of the results in *Fig3.6*. At timestamp **16:21:00hrs** the CPU load was at min **11%**. After some epochs, the load on CPU increased to **13%** at **16:22:40hrs** and was stagnant till **16:24:00hrs**. Talking about the CPU Usage metrics provided below for period of **5** minutes following were the calculations average load on CPU:

**Last load:** 12%

**Minimum load:** 11%

**Average load:** 12.32%

**Maximum load:** 13%

Other relevant metrics that can be found out from this experiment is about the Power Consumption on the VM. The power remains constant on the VM at 126W average for all the four metrics.

With this we can conclude that the stress imposed on VM was reflected on the associated host as well.

**Any other comments:**

References:

1. <https://www.tecmint.com/linux-cpu-load-stress-test-with-stress-ng-tool/>
2. <https://www.cyberciti.biz/faq/stress-test-linux-unix-server-with-stress-ng/>
3. <https://archives.opennebula.org/doc/4.4/oca/java/index.html?org/opennebula/client/host/HostPool.html>
4. <https://docs.opennebula.io/5.12/operation/references/host_states.html>
5. <https://docs.opennebula.io/5.12/operation/vm_management/vm_instances.html>
6. <https://docs.opennebula.io/5.12/operation/vm_management/vm_templates.html>
7. <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-18-04>
8. <https://collectd.org/documentation/manpages/collectd-nagios.1.shtml>
9. <https://www.howtogeek.com/410442/how-to-display-the-date-and-time-in-the-linux-terminal-and-use-it-in-bash-scripts/>
10. <https://docs.vonecloud.today/3.4/infrastructure_configuration/migrate_virtual_machines.html>