Cloud Computing Project 1:
AggieStack – Implementing OpenStack Functionalities
Detailed Report

– Submitted By: Sameer Kumar Behera UIN: 526004296

Task Specifications:

The task given is to build an AggieStack CLI which implements several OpenStack Functionalities. The program should read the command from the standard input and generate the expected output in the standard output. The error messages should go to standard error. You also should append information about the execution to a file named aggiestack-log.txt in the directory.

The Tasks are sub-divided as follows:

• **Part A -** You need to implement commands that create virtual servers (also called virtual instances). The code should be responsible for doing the resource management for the data center. The system needs to keep an inventory of which physical servers exist in the datacenter and how much space on these physical servers is available for fulfilling requests for new virtual servers. The code will decide where to allocate the virtual instances. We are not asked to come up with optimal placements of new instances on the virtual server.

• **Part B -** If a physical server is showing signs of hardware or software problems, your code should move the virtual servers hosted on the "sick" physical server to other healthy physical servers.

• **Part C -** Racks have their own storage server to hold images. The hardware configuration specification is changed to allow for the description of how much storage there is local to each rack. There is no need to implement this part, only to design the solution.

Implementation Environment:

- **Python Version 2.7**

As requested, Python has been used as it is the language used in many cloud management stacks. However, this is the first time I have used python to build an application. My preferable language is C++.

Timing Specifications:

Approximately 52 hours taken to accomplish the task which included:

- Around 4 hours of thorough analysis of the task required.
- Around 12 hours of research on the proper implementation & error resolution of the tasks.
- Around 32 hours in programming the complete project adhering to Github Working Environment.
- Around 4 hours in the comprehensive preparation of the report.

Installation Guidelines:

- Fork the following repository: https://github.tamu.edu/Sameer-TAMU/689-18-a.git
- Now clone the forked repository into a folder on your local machine.
- Enter the Following Directory: $ cd 689-18-a/P1
- Install Python and MongoDB
    - Instructions to Install MongoDB:
    - https://docs.mongodb.com/manual/administration/install-community/

    - Instructions to Install Python
    - Install Python 2.7.13 from: https://www.python.org/downloads/
    - Run Python and Install pymongo from: http://api.mongodb.com/python/current/installation.html
- Command to Run the Project: $ python aggie_cmd.py
- A Command Line appears saying, 'Waiting For Command: '
- Now enter the aggiestack commands as described in the report.

Standard Input Format:

- The Input Commands should be given one-by-one until the termination of the program.

Github Latest TimeLine to Test all the Commands:
https://github.tamu.edu/Sameer-TAMU/689-18-a/commit/162c03741bc86a863da4fee016de4e9cc2df2570

Part A - Implementation: Virtual Server Creation/Instantiation
Commands Implemented:

- **aggiestack server create --image IMAGE –flavor FLAVOR_NAME INSTANCE_NAME**
- Description: Creates an instance named INSTANCE_NAME to be boot from IMAGE and configured as indicated in FLAVOR_NAME.
- Time Taken: 4 Hours
- Design Idea: Find the given FLAVOR_NAME and its details from the List of Flavors. Find the Appropriate Machine and its details using First Come First Serve, i.e. Basic FCFS Algorithm from the List of 'Running' or 'Active' Machines (i.e. Avoids Machines in the Sick List). Then Create an Instance in the Machine (i.e. Add an Instance in the Instance List with the Instance Name, Image Name, Rack Name, Machine Name, Flavor Name, Flavor Size, Flavor Disks & Flavor VCpus) and also Update the Free Memory, Disk and VCpus of the Specific Machine.

- **aggiestack server delete INSTANCE_NAME**
- Description: Deletes instance named INSTANCE_NAME.
- Time Taken: 1 Hour
- Design Idea: Delete the given Instance from the Instance List and also Update the Free Memory, Disk and VCpus of the Machine freed.

- **aggiestack server list**
- Description: List all instances running (name, image, flavor)
- Time Taken: 30 Mins
- Design Idea: List all the Running Instances with the Image and Flavor from the Instances List.

- **aggiestack admin show hardware**
- Description: For each physical machine in the cloud, it lists how much memory, disks, and vcpus are currently free and the rack used by the machine (Added a functionality as per my usage).
- Time Taken: 2 Hours
- Design Idea: List all the 'Running' or 'Active' Physical Machines with their Available Size, Disk and VCpus (i.e. Avoiding the Machines that are in Sick List) from the Hardware Machines List.

- **aggiestack admin show instances**
- Description: It shows the physical server for each instance currently running.
- Time Taken: 30 Mins
- Design Idea: List all the Instances along with their Machine and Rack from Instances List.

Other Commands Implemented:

- **aggiestack config --hardware hdwr-config.txt**
- Description: Reads the configuration file describing the hardware available for the cloud.
- Time Taken: 5 Hours
- Design Idea: Delete any old configuration. Then read the hdwr-config.txt file line by line. Insert the Rack and Storage Capacity (In MB) into the Hardware Machine List. Also insert the Machine, Rack, IP, Memory (In GB), Disk and VCpus into the Hardware Machine List.

- **aggiestack config --images image-config.txt**
- Description: Reads the configuration file listing images available in the storage server.
- Time Taken: 1 Hour
- Design Idea: Delete any old configuration. Then read the image-config.txt file line by line. Insert the Image, Image Size (In MB) and Image Path into the Images List.

- **aggiestack config --flavors flavor-config.txt**
- Description: Reads the configuration file listing the flavor for instances available for the users.
- Time Taken: 30 Mins
- Design Idea: Delete any old configuration. Then read the flavor-config.txt file line by line. Insert the Flavor, Flavor Size (In GB), Flavor Disks and Flavor VCpus into the Flavors List.

- **aggiestack show hardware**
- Description: Output is the information about the hardware hosting the cloud in the format specified as per Appendix 1.
- Time Taken: 1 Hour
- Design Idea: List all the Racks with their Storage Capacity and all the Physical Machines with their Memory, Disks and VCpus from the Hardware Machines List Configuration.

- **aggiestack show images**
- Description: Output is the list of images available for the user to choose when creating virtual machines or instances in the format specified as per Appendix 2.
- Time Taken: 30 Mins
- Design Idea: List all the images available for the user along with their Image Size and Path.

- **aggiestack show flavors**
- Description: Output is the list of flavors available for the user to choose when creating virtual
- Machines or instances in the format specified as per Appendix 3.
- Time Taken: 30 Mins
- Design Idea: List all the flavors available for the user along with their size, Disks and Vcpus.

- **aggiestack show all**
- Description: Output is the whole list of hardware, images and flavors details.
- Time Taken: 30 Mins
- Design Idea: List all the physical machines, images and flavors along with all other details.

- **aggiestack admin can_host <machine name> <flavor>**
- Description: Output is “yes” if physical server <machine name> has currently enough memory, disks, and vcpus to host a new virtual server of type <flavor>, else is “no”.
- Time Taken: 2 Hours
- Design Idea: If the Memory Size, No of Disks and No of VCpus of the given Machine is greater than the Flavor Size, Flavor Disks and Flavor VCpus of the given Flavor then the Machine can host the Virtual Server.

- **quit**
- Description: Exits out of the Command Line interface

Part B - Implementation: Evacuating Sick Physical Servers

Commands Implemented:

- **aggiestack admin evacuate RACK_NAME**

- Description: The health monitor components detected an unusual number of errors in the rack storage unit. To free the rack for preventive maintenance, the AggieStack admins want to evacuate the rack by invoking the command. This code should migrate all the virtual machines in a server residing in RACK_NAME to another rack. Also, Servers on the evacuated rack won't be available for hosting new instances.
- Time Taken: 7 Hours
- Design Idea: From the given Rack, all the physical machines in the Rack are obtained using the list of Physical Machines. Then the Rack and all its Machine are added into a Sick List. All the Machines of that rack, whether in use or not in use are present in the Sick List using which the no of 'active' machines can be found out. Then, the Existing Instances which have the Flavor along with their Flavor Size , Flavor Disk and Flavor VCpus are compared with the Memory Size, Disks and VCpus of the 'Running' or 'Active' Physical Machines of a Different Rack on the basis of First Come First Serve, i.e. FCFS Algorithm and if the latter is greater than the former, then the instances are assigned to the new machine of the new rack as per the algorithm and migrate to it. The list of instances gets updated along with the list of the hardware machines.
  If there is not sufficient available memory, disk and vcpus in the new machines of other racks or if there are no more available racks, then the instance doesn't migrate to another machine and is forced to be removed from the machine until the whole evacuated rack, i.e. the sick rack has been fixed.

- **aggiestack admin remove MACHINE**

- Description: The command should remove the specified machine from its view of the datacenter. This implies that the machine does not appear in any output and that it does not host any new instance. It needs to make sure that no instance is created in that MACHINE
- Time Taken: 2 Hours
- Design Idea: From the given Machine, its Rack is found out by using the List of Physical Machines. Then that particular Machine is added to the Sick List. As the machine gets added to the Sick List, thus the machine becomes an 'inactive' machine, doesn't appear in the 'active' machines and thus no new instance can be created on it.

- **aggiestack admin release RACKNAME**

- Description: The command can be used after the complete fix of the Rack, so that all the machines of the rack can be used to host new instances.
- Time Taken: 2 Hours
- Design Idea: The given Rack along with all its machines is removed from the Sick List after complete fix of the rack. Once removed from the Sick List, the rack with all its machines are available to host new instances.

- **aggiestack admin add –mem MEM –disk NUM_DISKS –vcpus VCPUs –ip IP –rack RACK_NAME MACHINE**
- Description: Add the machine to the system so that it may receive new instances. This is usually invoked when a "sick" server was fixed, and it is ready to work again.
- Time Taken: 2 Hours
- Design Idea: The given memory, disks, vcpus, ip and machine for a specific rack is added to the list of physical machines. Also, this new machine is 'active' and ready to host new instances.

Part C – Design Document Explanation: Rack Storage Server

Task Specifications:
With the new data center architecture, image files stored in the main storage server can be copied to the storage unit for a rack, serving a number of servers. The rack storage server can be a cache of images, i.e., hold copies of the image files that are made available to the servers hosted in the rack. Once the image is copied to a rack, the servers in the rack have in-rack access to the images without having to go to the external storage server.
The code now needs to manage the image cache at each rack. When creating a new instance, The placement algorithm should prioritize servers in a rack where the required image is already cached. If no rack caches the image, then choose the ones with more free space in the rack storage unit. If no rack has enough space, then choose a rack to remove an image from the cache such as there is space for the new cached image. Freedom is provided to choose whatever scheme to decide which image to evict from the cache.

Given Commands for Changes in Functionality:

Total Expected Time For Development: 16 Hours
Total Allocated Time For Testing: 4 Hours

- **aggiestack config --hardware hdwr-config.txt**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack config --images image-config.txt**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack config --flavors flavor-config.txt**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack show hardware**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack show images**

- Changes in the Command Functionality:
  We need to fetch data from a newly built list of imagecaches which gets built on successful hosting of an instance and creation of an imagecache fetches data from a list of images using image, image size, imagepath from the image-config.txt & the rack from the list.
- Time for Implementation: 1 Hour

- **aggiestack server create --image IMAGE --flavor FLAVOR_NAME INSTANCE_NAME**

- Changes in the Command Functionality:
  While creating a new instance with the existing functionality of the command:

  - and if the IMAGE is new or loaded for the first time (i.e. IMAGE is Not Found in the list of imagecaches) and there is enough Storage Capacity in the Racks (i.e. image size is lesser than the storage capacity in any of the active racks) then:
    - arrange the racks and respective machines as per decreasing order of their storage capacity and then assign the instance to a machine and the image to the rack of the machine using the same First Come First Serve, i.e. FCFS algorithm used in the previous functionality.
    - Just like the instance list in the previous functionality, the list of imagecaches is also built with the image, image size, image path, instance and rack.
    - the storage capacity of the rack gets updated as <Original Storage Capacity – Image Size> and can be easily reflected in the list of hardware.

  - and if the IMAGE is new or loaded for the first time (i.e. IMAGE is Not Found in the list of imagecaches) but there is not enough Storage Capacity in any the Racks (i.e. image size is greater than the remaining storage capacity of every active rack) then:
    - arrange the racks and respective machines as per decreasing order of their remaining storage capacity and then keep on deleting the instances from the list of instances & the images from the list of imagecaches and hereby increasing the remaining storage capacity of a rack till the image size becomes lesser than the maximum remaining storage capacity of that rack after which the new instance to a machine and a new image to that rack gets created by again selecting a machine of that rack using the same First Come First Serve, i.e. FCFS algorithm used in the previous functionality. (NOTE: Another different approach can be used to delete instances which according to system history have been kept idle or the images haven't been used for a long time).
    - Just like the instance list in the previous functionality, the list of imagecaches is also built with the image, image size, image path, instance and rack.
    - the Storage Capacity of the Rack gets Updated as <Original Storage Capacity – Image Size> and can be easily reflected in the list of hardware.

- and if the IMAGE is already loaded (i.e. IMAGE is Present in the list of imagecaches) then:
  - find out the rack of the image from the list of imagecaches and thus only select the respective machines of that particular rack and then assign the instance to a machine of that rack using the same First Come First Serve, i.e. FCFS algorithm used in the previous functionality.
  - even though the instance list is built again, however this time no list of imagecaches needs to be built.
  - There is no change in the Storage Capacity of the Rack

- Time for Implementation: 13 Hours

- **aggiestack server delete INSTANCE_NAME**

- Changes in the Command Functionality:
  We need to check presence of the INSTANCE_NAME in the list of imagecaches
  - and if the instance is present then:
    - the instance gets deleted and the physical machines also get freed up as per the previous functionality.
    - however, the image also gets deleted from the list of imagecaches and the rack storage capacity of the rack also gets freed up when the image size is removed.

  - and if the instance is absent then:
    - the instance gets deleted and the physical machines also get freed up as per the previous functionality.
    - however, there is no change in the rack storage capacity of the rack.
- Time for Implementation: 1 Hour

- **aggiestack server list**

- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack admin show imagecaches RACK_NAME**

- New Command Functionality:
  The command lists the names of the images present in the specified rack and the amount available in the rack storage to host more image files.
- Design Idea: The list of imagecaches can be used to show the names of the images present in a specific rack. The list of hardwares can be used to show the remaining rack storage capacity of the particular rack.
- Time for Implementation: 1 hour

- **aggiestack admin show hardware**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

- **aggiestack admin show instances**
- Changes in the Command Functionality: No Changes Needed as Already Implemented
- Time for Implementation: N/A

REFERENCES:

- https://stackoverflow.com/questions/21672155/how-to-check-collections-insert-is-successfully-inserted-or-not-in-meteor
- https://stackoverflow.com/questions/5025399/python-2-7-try-and-except-valueerror
- https://docs.mongodb.com/manual/reference/method/db.collection.insert/#writeresult
- https://docs.mongodb.com/manual/reference/glossary/#term-cursor
- https://docs.mongodb.com/manual/reference/method/db.collection.distinct/
- https://docs.mongodb.com/manual/reference/method/db.collection.find/
- https://docs.mongodb.com/manual/reference/method/db.collection.remove/
- https://stackoverflow.com/questions/4706499/how-do-you-append-to-a-file
- https://wiki.python.org/moin/ForLoop
- https://docs.python.org/3/tutorial/errors.html
- https://stackoverflow.com/questions/43605512/python-difference-between-valueerror-and-exception
- https://docs.python.org/2/library/exceptions.html
- https://stackoverflow.com/questions/3277503/in-python-how-do-i-read-a-file-line-by-line-into-a-list
- https://stackoverflow.com/questions/8009882/how-to-a-read-large-file-line-by-line-in-python