

TEXAS A&M UNIVERSITY

CSCE 689: CLOUD COMPUTING PROJECT I

# OpenStack v0.2

Aryan Sharma

UIN: 326006767 — [aryans@tamu.edu](mailto:aryans@tamu.edu)

Implemented in C++

April 19, 2018

## Contents

<b>1</b>	<b>Project Information</b>	<b>2</b>
<b>2</b>	<b>Steps to run the code</b>	<b>2</b>
<b>3</b>	<b>Command Line Interface Design</b>	<b>3</b>
<b>4</b>	<b>AggieStack v0.2 Design</b>	<b>6</b>
4.1	Part A design . . . . .	6
4.2	Part B design . . . . .	7
4.3	Part C design . . . . .	7
<b>5</b>	<b>Other design decisions</b>	<b>7</b>

# 1 Project Information

This project implements a simple version of OpenStack in two steps: P0 and P1. **The github link for the code is:**

<https://github.tamu.edu/aryans/689-18-a/tree/master/P1>

The code has been written in C++ and it is contained within `AggieStack.cpp`, `AggieStack.h`, `Main.cpp`, `MakeFile`. Following are the overall functions of the files:

1. `AggieStack.h`: It is the header file of the `AggieStack.cpp` file and contains the class definitions, global variables and the inline functions used.
2. `AggieStack.cpp`: Contains the class function implementations.
3. `Main.cpp`: It is the main driver of the `AggieStack v0.2`. It contains the CLI implementations and the `main()` function.
4. `MakeFile`: Contains the direction as to how to run the program.
5. `aggiestack-log-date.txt`: Contains the log updates.

The time spent to complete this project was three days for v0.1 and about seven days for v0.2. It was done over a month in short bursts. Note that not all v0.1 commands are not supported in the v0.2 versions due to changes in the config files. To get the v0.1 the code has to be switched back to a prior git commit. However the CLI is missing in that git version.

# 2 Steps to run the code

1. `make clean`: to remove all the object files and previous logs.
2. `make`: to compile the program. This would generate the executable file *Main*.
3. `./Main`: to run the program. This is a helper mode where it prints how to use this binary.
4. `./Main command.file`: to run the program with a text files containing the commands. Any command that begins with `#` are treated as comments and are ignored.
5. `./Main -I`: to run the program in an interactive mode. Commands can be issued now one by one and the output can be observed in the command line.
6. `./Main -help`: to get the instructions to run the program.

### 3 Command Line Interface Design

A list of commands can be run using a file. The method to run it has been described above as: `./Main commands.txt`. The command line interface is reached by `./Main -I` or `./Main -i`. Some initial instructions can be fetched by `./Main -help`. A screenshot below explains this.

```
Apple@aryans:P1$ make clean
rm AggieStack.o Main.o Main
rm -rf aggiestack-log-*
Apple@aryans:P1$
Apple@aryans:P1$
Apple@aryans:P1$ make
g++ --std=c++11 -g -c -o AggieStack.o AggieStack.cpp
g++ --std=c++11 -g -c -o Main.o Main.cpp
g++ --std=c++11 -g -Wall AggieStack.o Main.o -o Main
Apple@aryans:P1$
Apple@aryans:P1$ ./Main
----- Usage -----
To read input file, type: ./Main <command_file>
To go to Interactive Environment, type: ./Main -I
For help, type: ./Main -help
-----
Apple@aryans:P1$
Apple@aryans:P1$ ./Main -help
----- Usage -----
To read input file, type: ./Main <command_file>
To go to Interactive Environment, type: ./Main -I
-----
Apple@aryans:P1$
Apple@aryans:P1$
Apple@aryans:P1$ ./Main -I
----- AggieStack CLI -----
Version 0.2 for the AggieStack CLI
Developed by Aryan Sharma
-----

Entering Interactive Mode
Type command and hit ENTER to execute
Type "quit/q" to quit the program.

In [1]
COMMAND:
In [2]
COMMAND:
In [3]
COMMAND:
In [4] quit

----- Summary -----
Total number of Commands = 4
Total time elapsed = 0.353 ms
-----
Apple@aryans:P1$
Apple@aryans:P1$
```

Figure 1: How to run the program.

```

Apple@aryans:P1$
Apple@aryans:P1$
Apple@aryans:P1$
Apple@aryans:P1$ make
make: `Main' is up to date.
Apple@aryans:P1$
Apple@aryans:P1$
Apple@aryans:P1$ ./Main input-sample-1.txt
File read successfully
COMMAND: aggiestack config --hardware hwdr-config.txt
hwdr-config.txt read successfully: SUCCESS

COMMAND: aggiestack show hardware
Showing All Machines...
#1      Name: m1      Rack: r1      IP address: 128.0.0.1  Memory: 16      Number of D
#2      Name: m2      Rack: r1      IP address: 128.0.0.2  Memory: 16      Number of D
#3      Name: m3      Rack: r1      IP address: 128.0.0.3  Memory: 16      Number of D
#4      Name: m4      Rack: r2      IP address: 128.0.0.4  Memory: 16      Number of D
#5      Name: k1      Rack: r2      IP address: 128.1.1.0  Memory: 32      Number of D
#6      Name: k2      Rack: r2      IP address: 128.1.0.2  Memory: 32      Number of D
#7      Name: k3      Rack: r2      IP address: 128.1.3.0  Memory: 32      Number of D
#8      Name: calvin   Rack: r1      IP address: 128.129.4.4 Memory: 8        Number of D
#9      Name: hobbes   Rack: r1      IP address: 1.1.1.1    Memory: 16      Number of D
#10     Name: dora     Rack: r1      IP address: 1.1.1.2    Memory: 64      Number of D

COMMAND: aggiestack config --images image-config.txt
image-config.txt read successfully: SUCCESS

COMMAND: aggiestack show images
Showing Images...
#1      Name: linux-ubuntu   Size: 128      Path: /images/linux-ubuntu-v1.0.img
#2      Name: linux-sles     Size: 512      Path: /images/old-image.img
#3      Name: linux-ubuntu-16 Size: 2048     Path: /images/linux-ubuntu-16.img

COMMAND: aggiestack config --flavors flavor-config.txt
flavor-config.txt read successfully: SUCCESS

COMMAND: aggiestack show flavors
Showing Flavors...
#1      Type: small   RAM: 1  Disks: 1    vCPUs: 1
#2      Type: medium  RAM: 8  Disks: 2    vCPUs: 4
#3      Type: xlarge  RAM: 32 Disks: 4    vCPUs: 8

```

Figure 2: How to run the program using an input file

The commands supported has to be entered by the user in the CLI. Some commands that are run in interactive mode has an advanced suggestions mode. There have been some hardcoded commands other than the ones the user can enter based on the suggested syntax which can be invoked by typing `help` in the CLI. They appear in the form of option like 1, 2 etc. and may prompt the user to enter some specific detail to process their requests. This has been shown in Fig. 3. You can see that by typing `help`, a list was suggested. In the Fig.

3 option 2 was selected and the file entered was `hdwr-config.txt`. As we can see the command executed was:

```
aggiestack config --hardware hdwr-config.txt
```

In the end, it generates the summary of number of commands executed in the interactive mode and the time taken to execute them. The summary can be seen in Fig. 1.

```
Apple@aryans:P1$ ./Main -I
----- AggieStack CLI -----
Version 0.2 for the AggieStack CLI
Developed by Aryan Sharma
-----

Entering Interactive Mode
Type command and hit ENTER to execute
Type "quit/q" to quit the program.

In [1] help
COMMAND: help
----- Commands -----
1. aggiestack read all config
2. aggiestack config --hardware <hardware_config_file.txt>
3. aggiestack config --flavors <flavor_config_file.txt>
4. aggiestack config --images <image_config_file.txt>
5. aggiestack show hardware
6. aggiestack show images
7. aggiestack show flavors
8. aggiestack show racks
9. aggiestack server create --image IMAGE --flavor FLAVOR_NAME INSTANCE_NAME
10. aggiestack server delete INSTANCE_NAME
11. aggiestack server list
12. aggiestack server show imagecaches RACK_NAME
13. aggiestack server show hardware
14. aggiestack admin show instances
15. aggiestack admin show hardware
16. aggiestack admin evacuate RACK_NAME
17. aggiestack admin remove MACHINE
18. aggiestack admin add --mem MEMORY --disk NUM_DISKS --vcpus VCPUS --ip IP --rack RACK_NAME MACHINE
-----

In [2]
COMMAND:
In [3]
COMMAND:
In [4] 2
Enter hardware config file (relative/absolute path): hdwr-config.txt
COMMAND: aggiestack config --hardware hdwr-config.txt
hdwr-config.txt read successfully: SUCCESS

In [5] COMMAND:
In [6]
COMMAND:
In [7] q

----- Summary -----
```

Figure 3: Suggestion help mode in CLI

## 4 AggieStack v0.2 Design

The **AggieStack** has been designed as a **singleton class** which maintains a **private** list of the following:

1. Machine information
2. Available Machines while in process
3. Images that are available
4. Racks that are available for use

It has some categories of APIs which include **set** and **get**. Besides it has **read** comands to read the files, **show** comands to show the states, and some special functions like create, delete and evacuate etc. A single instance of class **AggieStack** is created and is maintained throughout the sessions to avoid any extra copy or corruption of any internal data. Also it makes the maintainance of the states quite easy.

Other classes are that of **Machine**, **Image**, **Flavor**, **Rack**, and **Instance**. The hierarchy in which they are connected is as follows. The singleton instance of **AggieStack** maintains the racks and also pointers to machines that are running. The **Rack** inturn has cached images information and the list of machines that are hosted on that rack. The **Machine** on the other hand has an id to the rack on which it is hosted and can provide that information through the **AggieStack**.

Besides, when the image, hardware and flavor config files are read, they create the **Image**, **Flavor** and as per the requirement the **Machine** objects from the class interface provided. They are maintained thorought the sessions except in the situation where they are either forced killed or have to be removed from the system. Each machines have the capability to host **Instance** class objects which are the user requested instances that they need to run. The information about the running instances are with the **Machine** themselves.

All of the classes have in addition some special functions that help them to update their states and interact with each other. Everything has been designed such that the **AggieStack**, **Racks**, **Machine**, **Instance** hierarchy is maintained.

### 4.1 Part A design

The part A design required the commands that create virtual servers. This was achieved by **AggieStack** by calling the **Machines**, deciding if they can host and then allocating the ressources to the instance. The states were internally updated and were reflected through **show** commands. If no physical machine has the required amount of ram, disks, and vcpu available, it indicates the error that it can't add the instance. The policy that was used here was to provide the instance that Machine which is found first to have available space. This was changed in part C.

## 4.2 Part B design

In this part the AggieStack admins evacuate the rack by invoking `evacuate` command. It migrates all the virtual machines in a server residing in the faulty rack to another rack. In case no rack has free space, it kills those instances. Also it removes the specified machine from its view of the datacenter through the `remove` command and add a new machine to the system so that it may receive new instances.

An interesting thing about this design was that many functions were re-used internally to make the code modular and reusable. This was kept in mind while designing the `remove_machine`, `can_host` commands.

## 4.3 Part C design

In this part the image files stored in the main storage server is copied to the storage unit for a rack, serving a number of servers in its cache. It holds 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 list of cached images is maintained by every rack and is sorted based in their available memory.

When an instance is requested, the image of the instance is searched across all racks. In case one of the rack has the image, it checks if any of its machines has the capability to host it. In case it cannot, it looks on other racks, otherwise it creates an instance there and returns. In the case that none of the racks have the cached image, it copies the image from the central storage, and creates an instance in a rack which has maximum amount of storage. The caches list is updated based on the storage of the particular rack. It's a queue kind of data structure where the oldest used image is dropped from the cache to accommodate a newer image. When all such search is exhausted, **AggieStack** return an error that it cannot host the instance due to memory issues. Please refer to the code and git commits for more details. Few of the examples have been shown in Fig. 4.

## 5 Other design decisions

The error check has been quite strict in the command line and therefore it prompts user to use the suggestion mode in the interactive mode. It logs the error in the log and prints on the console when an error happens. The `logger` function is responsible to log the results in the file `aggiestack-log.txt`. The file contains the timeline of the execution. An example is in Fig 5.



```

COMMAND: aggiestack server create --image linux-ubuntu --flavor small my-first-instance
Rack: r2
Instance created successfully

COMMAND: aggiestack server create --image linux-ubuntu --flavor medium my-second-instance
Cache hit in rack r2
Instance created successfully

COMMAND: aggiestack admin show instances
#0      Name: my-first-instance
        Machine Name: k1

#1      Name: my-second-instance
        Machine Name: k1

COMMAND: aggiestack admin evacuate r1
Machine calvin removed successfully
Machine dora removed successfully
Machine hobbes removed successfully
Machine m1 removed successfully
Machine m2 removed successfully
Machine m3 removed successfully

COMMAND: aggiestack admin add -mem 8 -disk 4 -vcpus 4 -ip 128.0.0.1 -rack r1 newmachine
Machine created successfully

COMMAND: aggiestack admin show instances
#0      Name: my-first-instance
        Machine Name: k1

#1      Name: my-second-instance
        Machine Name: k1

----- Summary -----
Total number of Commands = 0
Total time elapsed = 7.201 ms
-----
Apple@aryans:P1$ █

```

Figure 4: Examples

```

1 2018-04-19 16:54:41 AggieStack singleton initialised: SUCCESS
2 2018-04-19 16:54:41 set_machines_list(): SUCCESS
3 2018-04-19 16:54:41 set_available_machines(): SUCCESS
4 2018-04-19 16:54:41 set_machine_on_racks(): SUCCESS
5 2018-04-19 16:54:41 read_machine_config(): hdwr-config.txt read successfully: SUCCESS
6 2018-04-19 16:54:41 show_machines(): SUCCESS
7 2018-04-19 16:54:41 set_number_images() to 3: SUCCESS
8 2018-04-19 16:54:41 set_images(): SUCCESS
9 2018-04-19 16:54:41 read_image_config(): image-config.txt read successfully: SUCCESS
10 2018-04-19 16:54:41 show_images(): SUCCESS
11 2018-04-19 16:54:41 set_number_flavors() to 4: SUCCESS
12 2018-04-19 16:54:41 set_flavors(): SUCCESS
13 2018-04-19 16:54:41 read_flavor_config(): flavor-config.txt read successfully: SUCCESS
14 2018-04-19 16:54:41 show_flavors(): SUCCESS
15 2018-04-19 16:54:41 can_host() for machine k1 : SUCCESS
16 2018-04-19 16:54:41 Instance my-first-instance added successfully on new rack: SUCCESS
17 2018-04-19 16:54:41 Cache hit in rack r2
18 2018-04-19 16:54:41 can_host() for machine k1 : SUCCESS
19 2018-04-19 16:54:41 Instance my-second-instance added successfully: SUCCESS
20 2018-04-19 16:54:41 show_instances(): SUCCESS
21 2018-04-19 16:54:41 Machine calvin deleted successfully: SUCCESS
22 2018-04-19 16:54:41 Machine calvin removed successfully: SUCCESS
23 2018-04-19 16:54:41 Machine dora deleted successfully: SUCCESS
24 2018-04-19 16:54:41 Machine dora removed successfully: SUCCESS
25 2018-04-19 16:54:41 Machine hobbes deleted successfully: SUCCESS
26 2018-04-19 16:54:41 Machine hobbes removed successfully: SUCCESS
27 2018-04-19 16:54:41 Machine m1 deleted successfully: SUCCESS
28 2018-04-19 16:54:41 Machine m1 removed successfully: SUCCESS
29 2018-04-19 16:54:41 Machine m2 deleted successfully: SUCCESS
30 2018-04-19 16:54:41 Machine m2 removed successfully: SUCCESS
31 2018-04-19 16:54:41 Machine m3 deleted successfully: SUCCESS
32 2018-04-19 16:54:41 Machine m3 removed successfully: SUCCESS
33 2018-04-19 16:54:41 Rack r1 evacuated successfully: SUCCESS
34 2018-04-19 16:54:41 add_machine() for machine newmachine successfull: SUCCESS
35 2018-04-19 16:54:41 show_instances(): SUCCESS
36 2018-04-19 16:55:26 AggieStack singleton initialised: SUCCESS
37 2018-04-19 16:55:26 set_machines_list(): SUCCESS
38 2018-04-19 16:55:26 set_available_machines(): SUCCESS
39 2018-04-19 16:55:26 set_machine_on_racks(): SUCCESS
40 2018-04-19 16:55:26 read_machine_config(): hdwr-config.txt read successfully: SUCCESS
41 2018-04-19 16:55:26 show_machines(): SUCCESS
42 2018-04-19 16:55:26 set_number_images() to 3: SUCCESS
43 2018-04-19 16:55:26 set_images(): SUCCESS
44 2018-04-19 16:55:26 read_image_config(): image-config.txt read successfully: SUCCESS
45 2018-04-19 16:55:26 show_images(): SUCCESS
46 2018-04-19 16:55:26 set_number_flavors() to 4: SUCCESS
47 2018-04-19 16:55:26 set_flavors(): SUCCESS
aggiestack-log-2018-04-19.txt
"aggiestack-log-2018-04-19.txt" 170L, 10339C

```

Figure 5: Log file entry