Project4 Part B

Nathaniel Leake, 424003778
Cloud Computing, Fall 2017
Time to complete Task: **4hrs**

Compiling & Executing

Compile by navigating to the directory containing the code and typing

- `javac AggieStack.java`

To run the compiled project, type

- `java AggieStack.java`

GitHub Link

Here is the last commit for my Early Bird submission to Part B:

- https://github.tamu.edu/nateleake/489-17-c/tree/0cdfb968581d232b6e6fde7806787f411ca4846e

Design choices

//Copied from Part A

I have never used Python before, but the instructions document said we could use any language we wanted so I did it in Java. Each command is implemented in its own file named "CommandX" where X is the name of the command. Images, Flavors, Instances, Machines, and Racks are each represented as their own classes to make the project modular and easy to modify. The actual instances of these objects are stored in maps within the AggieStack code, but the only computation handled in AggieStack is the function to locate which Machine to place a new virtual Instance on.

For the evacuate command, all virtual instances are moved to machines outside of the affected rack. The rack is not removed from the system, but new instances will not consider it as a potential location to host themselves and thus it remains empty until custom "unevacuate" command is called. If there is not enough room in the cloud to fully evacuate the rack, it is evacuated as much as possible and the flag is still set to prevent new instances from using that rack. The `aggiestack admin remove MACHINE` command behaves in a similar fashion – all virtual instances are migrated elsewhere. Unlike the evacuate command, the remove command actually deletes the machine from the cloud system. If there is not enough space in the cloud, the operation is cancelled and an error message is displayed. Finally, the add command works as specified to add a machine to the system. I spent a lot of time on this particular command to ensure reliable error checking and user-friendliness (like all of the other commands in my implementation, the user can specify arguments in arbitrary order as long as they specify their flags properly). Error checking on this command includes: IP-validation, duplicate machine names, missing arguments, conflicting arguments, number format, and negative values for memory, disks, and VCPUs.

Main Ideas

As with all parts of this project, my focus while designing these commands was to maximize correctness & speed of execution of the program. Secondarily, I spent a lot of time ensuring user-friendliness of the system, because ease-of-use and understandability are tremendous aid to anyone trying to diagnose or upgrade the system.