To run the code :

1. Clone the git repository “git clone git@github.tamu.edu:manamakhilbabu/689-18-b.git” or fork and clone this repository
2. Then move into the folder 689-18-b
3. In this folder, execute “make install” in Ubuntu based or linux based system.
4. You are ready to check the commands.
5. To run multiple commands one after the other in a pipe, insert the commands into the file multi_commands.sh leaving the first line and then run ./multi_commands.sh on kernel.

Timeline :

1. Part A : basic implementation : around 2 hr
2. Part B : basic implementation : around 3 hr
3. Part C : basic implementation : around 3 hr
4. Error check was performed multiple times after each part and going back to parts in between and hence most of the time was spent in error checking and code debugging - around 5 hr

Github URL :
https://github.tamu.edu/manamakhilbabu/689-18-b/tree/ad1d3c9b7e8d61bd982ab3031163f98225300d8e

Part solutions are not implemented in a conventional fashion where one part development and rigorous testing were done. Hence, the implementations available after each parts were first developed are very buggy. Although error checking and debugging were performed on the whole list, they are not extensively performed. Extensive error check was performed after completion of all the parts. Hence, the **final three commits (45, 46, 47) are to be tested for each part** simultaneously.

Regarding timestamp or timeline for completion of development of each version of part A, B and C is as follows:

1. Part A : commit 41 July 29 (but some files are missing which were identified and added later, so this can not be tested). For grading, please use commit 45

2. Part B : commit 41 July 29 (again development was done but a file was not added (common.py) which was used in testing on local machine). For grading please use commit 46
3. Part C : commit 43 July 30 (again the file common.py was missing and there were a lot of bugs unidentified). For grading please use commit 47

File Structure:

The project file structure is as follows :

- 689-18-b (parent)
    - setup.py
    - setup.cfg
    - Makefile
    - aggiestack
        - __init__.py
        - __main__.py
        - cli.py
        - hdwr-config.txt
        - image-config.txt
        - flavor-config.txt
        - aggiestack-log.txt
        - instance-config.txt
        - im2rack-config.txt
        - server-config.txt
        - commands
            - __init__.py
            - admin.py
            - show.py
            - config.py
            - server.py
            - common.py
            - placement.py
            - helpers.py

Design :

The entry point for any command starting with aggiestack is main function in cli.py and this code call the appropriate commands after analysing the command line arguments. Each command type has its own file for maintaining functions. Since admin_can_host_command was causing a circular dependency while importing it has been duplicated into common.py as well. helpers.py

contains all the functions which are called frequently and are not commands that are given by the user directly such as logging, printing error messages, parsing config files etc.

In addition to project 0, currently some extra config files have been added. For implementing part A and part B smoothly,

1. server-config.txt and instance-config.txt have been used. server-config.txt is similar in structure to hdwr-config.txt specified in the appendix of P1 but it keeps updating the server racks space availability and the server spaces availability as new instances are created or deleted.
2. Instance-config.txt is used to store primarily the server in which the a particular instance is running and also to have quick access for certain commands to the sizes in order to manage available rack space in case of deletion, evacuation or removal of instances, racks or servers respectively. It comes very handy, especially in case of evacuating a server in order to search the other currently available racks for replacing the instance.

For implementing part C, an other config file named im2rack-config.txt is being used. Implementation of part C goes as follows:

1. im2rack-config .txt will be parsed for getting images in racks.
2. While creating a new instances where the placement algorithm will be used, the task is divided into two scenarios. First, the algorithm checks if the image specified exists in any rack.
3. If it does not exist in any rack,
    a. All the usable racks are collected for the particular flavor by checking if there exists an available server which can host that instance.
    b. From these racks, hostable racks are collected by checking if the image can fit into a rack storage.
    c. If any rack can fit, then appropriate server in that rack is chosen.
    d. If any rack can not fit that image, then an error is given out.
4. If it exists in a few racks,
    a. Usable racks among them are collected if any and a server chosen from among them.
    b. If no usable rack is available, then all the servers are treated alike and step 3 is followed.