# Project 1 – Setup (Project 0)
# Getting ready for AggieStack
# Due: 4/27/18 11:59 pm CT

By the end of the semester, you will have learned about OpenStack, a free and open-source software platform for cloud computing. We will release the complete specification of Project 1 next week; in which you implement some of the OpenStack functionality. Project 0 gives you the opportunity to do the necessary setup for Project 1.

## 1. Get ready to code: set up your TAMU GitHub repository

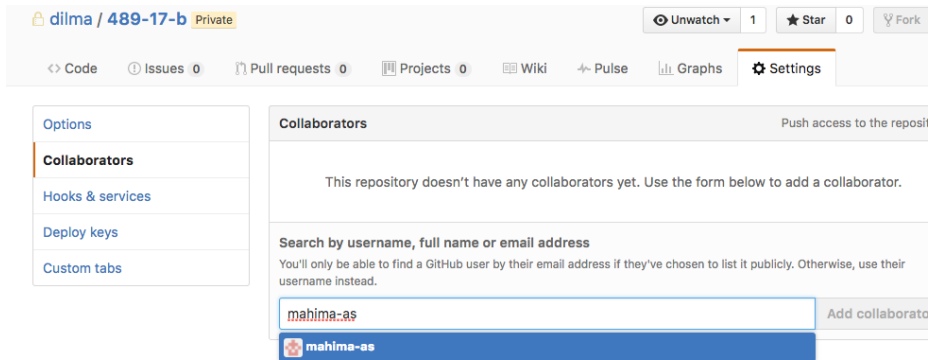You need to make your code available through a repository hosted at github.tamu.edu:

- Create a <u>private</u> repository named 689-18-a. The figure below is from last semester when we asked for "489-17-b".



then



- Add the following as collaborators
  - "dilma", "mahima-as", "rahul-bhagat", "cameron-smith", "k-godfrey22"

You will need to create a directory called "P1" to check in your code for Project 1.

Your repository needs to reflect your software development, i.e., every time you make a change to your code, you commit a version to the repository. If the modification is writing new code (for example, when you just started with an empty file), you should be committing new versions of our code often, for instance after defining a function or adding some lines of code. If your change is a code fix, you should document in the commit message what you fixed it. A reasonable rule of thumb is that for short work sessions (when you code for an hour or less), you commit when you stop the work session. For longer sessions (writing code for many hours), you may want to commit a few times, depending on how many code changes you are doing.

## 2. Context for the project

We have not even discussed what cloud services are yet, but we can expect that those services (whatever they do) are available to customers through interfaces (probably through web pages, web services, and apps available for smartphones). You can assume that someone else is responsible for implementing beautiful user interfaces and for handling user authentication.

Your job is to implement the AggieStack CLI (command line interface) as specified in Sections 4 and 5.

You may feel uncomfortable working on a project that involves terms that you don't properly understand yet, such as "virtual server", "instance", "images" and "flavors". This lack of familiarity is not uncommon in the initial phase of a software development project, as you are learning about your customer's domain, so aim at implementing the requested functionality even if you feel uncomfortable. By the time you go back to work on this code, later on, you will have a better understanding of the domain.

Please read this document to the end. If you would like more information before starting, you can watch the video available on eCampus->Projects and Homeworks->Project 1.

## 3. Implementation environment

You can implement this project using whatever language you prefer. I would suggest using Python because it is a language used in many cloud management stacks. If you never coded in

Python, you may want to use the beginning of the semester to learn and try it out. If you are using it for the first time, indicate in your report as you will get some bonus points.

Feel free to use libraries to parse arguments from the command line. If you use libraries or code ideas you find on places such as StackOverflow, make sure to indicate that in your report. Reusing software from the web is ok as long as you cite all your sources. Reusing code from colleagues or code "stolen" or "purchased" is not ok.

## 4. The AggieStack CLI – Part I

The first version of AggieStack is very primitive. The AggieStack 0.1 CLI (command line interface) implements the functionality in the table below. Your program read the command from the standard output and generates the expected output in the standard output. Error messages go to standard error. You also should append information about the execution to a file named aggiestack-log.txt in the current directory.

| Command | Description |
|---|---|
| aggiestack config --hardware hdwr-config.txt | Reads the configuration file describing the hardware available for the cloud. See Appendix 1 for details on the configuration file format. **Output**: none if successful, error message if the file does not exist. **Log**: adds command to the log and success/failure |
| aggiestack config --images image-config.txt | Reads the configuration file listing images available in the storage server. See Appendix 2 for details on the configuration file format. |
| aggiestack config --flavors flavor-config.txt | Reads the configuration file listing the flavor for instances available for the users. A flavor specifies a virtual machine configuration: the amount of ram, number of disks, and number of vcpus. See Appendix 3 for details on the configuration file format. |
| aggiestack show hardware | Output is the information about the hardware hosting the cloud in the format specified in Appendix 1. |
| aggiestack show images | Output the list of images available for the user to choose when creating virtual machines. |

| | |
|---|---|
| aggiestack show flavors | Output the list of flavors available for the user to choose when creating virtual machines. |
| aggiestack show all | Output has "show" for hardware, images, flavors. You already implemented the commands you need for this. |

Your implementation needs to print error messages for invalid input, e.g., non-existing configuration file, flavor, instance.

All commands generate output and update the aggiestack-log.txt log as follows:

- **Output**: none if successful, error message if there is one
- **Log**: adds command to the log and SUCCESS or FAILURE

By "updating" a log we mean that you append new information to it. If there is already a log file in the current directory, you add information to it instead of starting from an empty file.

## 5. The AggieStack CLI – Part II

Once your Part I API working, the next step is to add capability in your code to represent *instances* (in some platforms, the same entity may be called "guests" or "virtual machines"). Think of "instances" as virtual servers that customers can get from your cloud service. These virtual services run on top of the hardware that you have available, they use as software an "image" and how much resources they use are dictated by the instance "flavor" (or type).

| Command | Description |
|---|---|
| aggiestack admin show hardware | For each physical server in your cloud infrastructure, it lists how much memory, disks, and vcpus the physical server has currently available. |
| aggiestack admin can_host <machine name> <flavor> | Output is "yes" if physical server <machine name> has *currently* enough memory, disks, and vcpus to host a new virtual server of type <flavor> |

As with the previous commands, you need to generate the appropriate output for errors and log information on each command.

Notice that, as specified in this document, your code reads arguments provided on the command line. In order to test a lot of commands, we will have a script that invokes your program multiple times, each time to execute a single command line.

## 4. What to turn in for Project 1

Submit on eCampus a zip file containing your code and a short write-up with the following information:

- how much time you spent implementing this project
- show to run your code (providing a makefile to build the code if necessary

**Rubric for Project 0:**

Principles guiding the grading:

- If failed to adhere to GitHub way of working: -100 points
- No instructions on how to build or run your code: -100 points

Specific submission and rubric information will be released with the complete Project1 specification.

## Appendix 1 – Hardware configuration file

The list of physical machines is provided in a configuration file in the following format:

<number of machines>

<list of machine spec one per line>

The specification of each machine has the following format:

<name> <ip> <mem> <num-disks> <num-vcpus>

where <mem> is the amount of RAM in GB, <num-disks> is the number of virtual disks the machine can accommodate localy, and <num-vcpus> is the number of vcpus (virtual cores) that he hardware can accommodate.

On eCampus->Projects and Homeworks->Project 1, you will find a sample input file hdwr-config.txt illustrating the expected format.

## Appendix 2 – Image configuration file

The image configuration file specifies the number of images available to be used when starting virtual machines, providing for each one the name of the image and the path in the storage server to the file containing the image. The file image-config.txt (available on eCampus) is an example of a configuration file in the expected format.

## Appendix 3 – Flavor configuration file

A "flavor" describes a type of instance that users can request. It specifies the amount of RAM in GB, the number of disks, the number of vcpus. The file flavor-config.txt is an example of the expected format (like the other sample configuration files, it is available on eCampus).