Report Project1 Aggiestack

Clarification

This is the first time that I use Python for programming. I am a C/C++ programmer, the shell language I use is cshell. I worked on system software as well as lower level device driver, but has limited experience on application level development.

Installation and Prerequisite

The code is written using Python with only 1 dependency module.

1. install python module prettytable.

   ```
   $ sudo python -m pip install prettytable
   ```

2. python version.

   ```
   $ python --version
   Python 2.7.11+
   ```

Time to complete

The whole project took me around a 6 full days to finish. The time took for the project are sparse, so the time estimation may not be accurate, please refer to the git trace for more detials. Here I provide the route map for finishing the whole project. The route map of implementing the project contains 5 parts.

1. Understanding the background and requirements about the P0 and implementing the first three config conmands with designed data structure. This part took me around 1 day to finish.

2. Implement the P0 part II as well as the P1 partA for creating, deleting and listing the instances. This took me around 1 day to finish.

3. Re-design the code for P1 part C adding the **Rack** concept and the new policy for creating instance with the imagecache to work with the new architecture. This took me around 2 days to finish.

4. Implement P1 partB for the health monitor service and the 3 new commands. This took me around 1 days to finish.

5. Refining the error handling for each command and adding logging and process script frile and misc and writing the report. This took me around 1 day to finish.

Github tree and Timestamp

Github tree for the Project1. The last commit before submitting is **commit 924954ff39b9f8a3d2298f851ab84b43a7b085b3**
The Github tree contains two folder, 0.1 is the unfinished code for P0 and P1 partA without full error handling and logging and process from srcipt file feature. 0.2 is the final version of the code with all the functionality and requirements.

Design and Implementation

Overview

The whole software of Aggiestack is very similar like a Database command shell. It can accept user input of aggiestack command. Also it can accept a filename as command line arguement to process the commands in the input file. This is useful for batch processing and also useful for debug and testing. See the detailed run cases below.
The main **idea** of implementing the software is to store all the hardware, image, flavor information in a in-memory data structure inside the program and keep track of the free resouces when creating and deleting the instances and do other things such like evacuation of a rack. The design philosophy are similar like a memory allocator, but with speicial policy (for partC requirement). For the actuall implementation, I use OOP design in Python. The main class are Hardware, Image, Flavor, Instance and Rack. The resource information are stored in the class and the class provide method to manipulate with the resource information. See below for the detailed data structure design.
Data Structure

1. Hardware
   The Hardware class maintain two lists (use dictionary in python), one rack list and one machine list. The machine name/rack name as key and a dictionary for detailed information such as name, mem, vcpus, etc as value. The class provide many basic methods such as insert/delete rack/machine, get a machine information, listing the hardware information and etc.
   The program keep track of two Hardware entity one for configured hardware information, the other for the free resource information for creating and deleting instances.

2. Rack
   The Rack class maintain a list of image using as the information of image cache in the Rack storage server. This class is used for the allocating policy (creating instance). The image cache information can be retrived from the Rack class using rack name.

3. Image
   Image class is similar like Hardware, it maintains a list (using dictionary) of image records.

4. Flavor
   Flavor class is also similar like Hardware, it maintains a list (using dictionary) of flavor records. Which used to retrive the detailed flavor information.

5. Instance
   Instance class keep track of all the instances created. It keep the information of instance name, which rack/machine it sits on and also the image and flavor information. This class is used when creating and removing instances and also migrating/listing instances.

Design policy for PartC

The policy for allocating/creating instances in PartC's new architecture is as follow.
The whole create process is a exhaustive search algorithm with certain priorities. First, it will find whether there are rack that contains the image for the new instance, if exist, try to create a instance on that rack using first fit algorithm for finding a physical machine. If no machine can server the instance. Add the rack into a unavail_rack list. Second, if the rack which contains the image is out of resources, find a rack with the maximum space in the rack storage server (for storing the image cache). If no space for holding the new image, remove the oldest image in the image cache. This is done by mainting a **LRU list** for the image cache. Then, try to create a server on that rack. If no machine can fit for that instance, put that rack into the unavail_rack list. Third, keep looping until all the rack is checked, if all of the rack is unavailable, reporting error message that not enough resources available.

Design for PartB

The main part of PartB is the "evacuate" command. This command is mainly done by a server_migrate() function. This funtion use two functions server_create() and server_delete() for creating and removing instance with some special logic. When do the migration of the instance in the sick rack. It will first get all the instance in that rack. Then for each instacne, it will first try to create a new instance in all other racks using the old instance's name with "_tmp". If create succeed, it will then delete the instance in the sick rack and then rename the instance back to it's orignal name. If not succeed, which means there is not enough resources in the other racks, it will terminate the migrate process and print error to the user to remove some instances manually

and try the command again. After successfully migrate all the instances in the sick rack, the program will clear all the records of the machine under the sick rack, meaning no instance will be create under the machines in the sick rack.
For the remove machine command, it will automatically delete all the instances in the machine if there contains any.
For the add machine command, it will first do legitimate check to see whether the rack specified is exist, if not, it will throw error to user since we suppose that the new added machine must sit all the rack that initial configured in the hardware configuration file.

Error handling

The design of the error handling is pretty user friendly. It behaves like a practical linux software. For input commands that are far from the API provided, it will print a help information which contains all the support commands for the user. For commands that are close but not exactly correct, for example, missing a flag or flag argument it will report the specific error and show specific information of that command like what's the correct format and help user to correct their input commands. For errors such as wrong filename, duplicated instance name when create a new instance, instance name/image/flavor not found in the configuration. It will report the specific error to the user.
All the error information are print to stderr using the erpint() wrapper. The program usage information is print to the stdout. All the output information for the show commnad print to stdout.

Run case

Running as command interpreter shell

```
python aggiestack.py
```

Running with script file

```
python aggiestack.py stript_filename
```

Debug and testing