Oscar Reyes
P4 Part C Report
Cloud Computing

I am using all the 3 late days that I have left for this project as specified in Piazza.

After reading all the requirements for Part C, I believe a good and experienced programmer would be easily able to implement all this in about 8 hours.

In addition, because now we are going to deal with racks our API changes. We have to take into account that we have different racks and each rack holds servers. Then each server can hold multiple virtual servers. Each virtual server can have their own flavor and also their own image. This implies that we have to keep track of the unallocated resources in each server. This encompasses the memory, disks, and cores. Each virtual server requires a set amount of resources that will be allocated by the servers. I am basing a lot of this report on my implementation as I created racks, physical servers and virtual servers.

In code, the way I would implement this would be using classes to create objects. I would keep track of the racks in and list of racks. Within the rack object there will be another list to keep track of the physical servers. Each physical server would then also keep track of the virtual servers with a list. The virtual server would keep track of its image information and also its flavor size. In addition, the image would also be cached in the rack storage that is connected to all servers in order to make the image easier to access. In case another virtual server needs to run the same image.

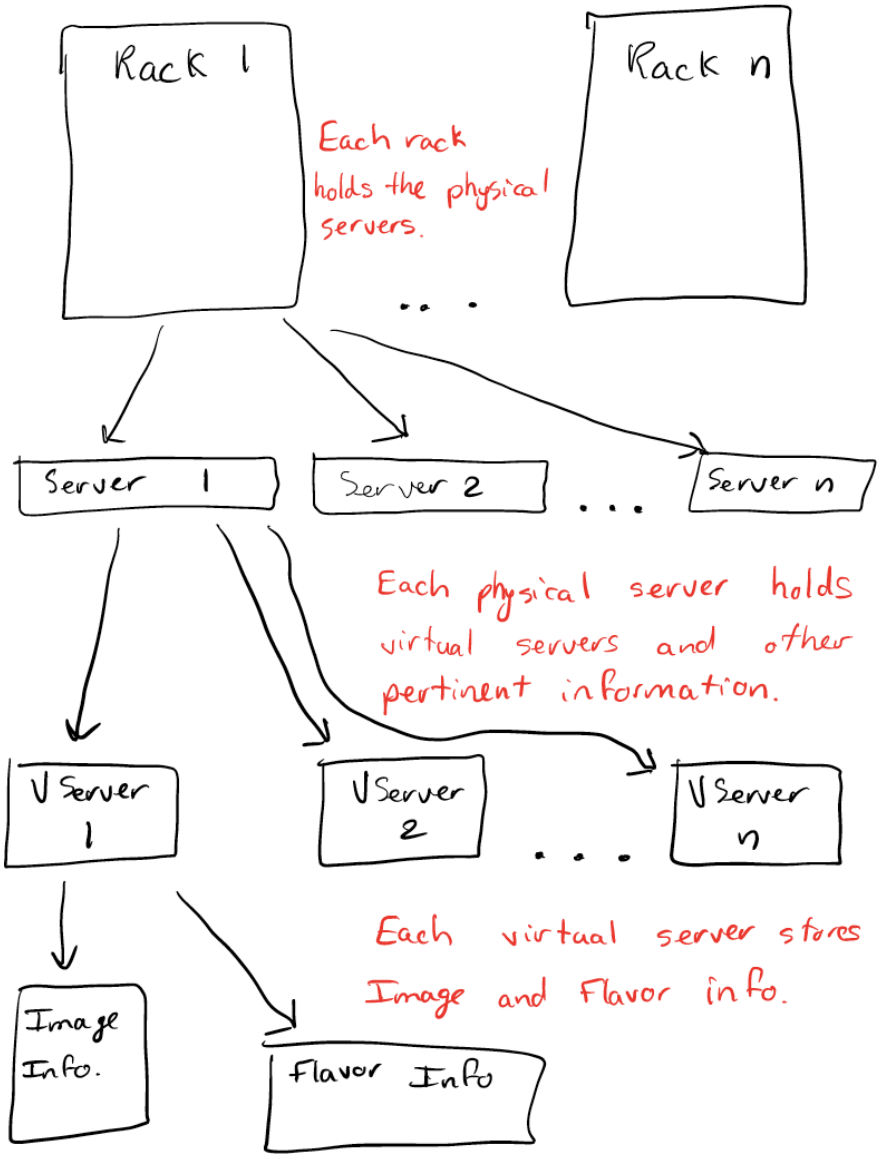
Furthermore, when saving the configurations for flavor and images. A list of objects would be created to keep track of each kind of object. We would create a list of images and a list of flavors.

A few changes would have to be made to the functions we already have. Now we also have to show the rack information in the show hardware function. We also now have to take into account the image cache in each of the racks in order to facilitate the deployment of an image. In my code, the cache is not something I implemented as it was not mentioned anywhere else before part C. If an image is not cached we will have to find a rack with enough space to cache it, otherwise we will have to remove one from a rack and put in the new one that is needed.

To delete an instance, we now have more overhead as we have to look for the rack in which the instance is stored in. A similar thing would happen for the command in which we list the running instances, there will be more overhead as we will have to go through all racks and all servers to get the proper information. Otherwise things stay the same as before.

To show all the cached images, one simply would go through all racks, iterate through them and print out the cached imaged information.

In addition, for testing I would allocate at least a whole day to make sure that there are no bugs. In this time, I would try and break the design to see how the program crashes, what makes it crash and find ways to properly handle errors and correct bugs.
The following is a rough sketch of how I believe the design of the system should be.



I have also attached a UML diagram of how things would be roughly designed. Please see next page. It is loosely based on my implementation of the code as I did take into account racks in my design due to the input file we were given, I assumed for part B we had to do so.