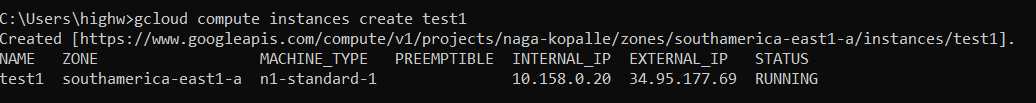
**Report**

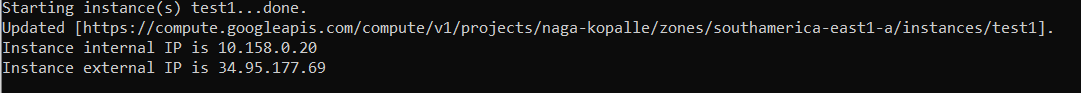
**Cloud test cases - Launching VMs, SSHing to VMs, Suspending Vms .**

**Please find the screen shots and the outputs .**

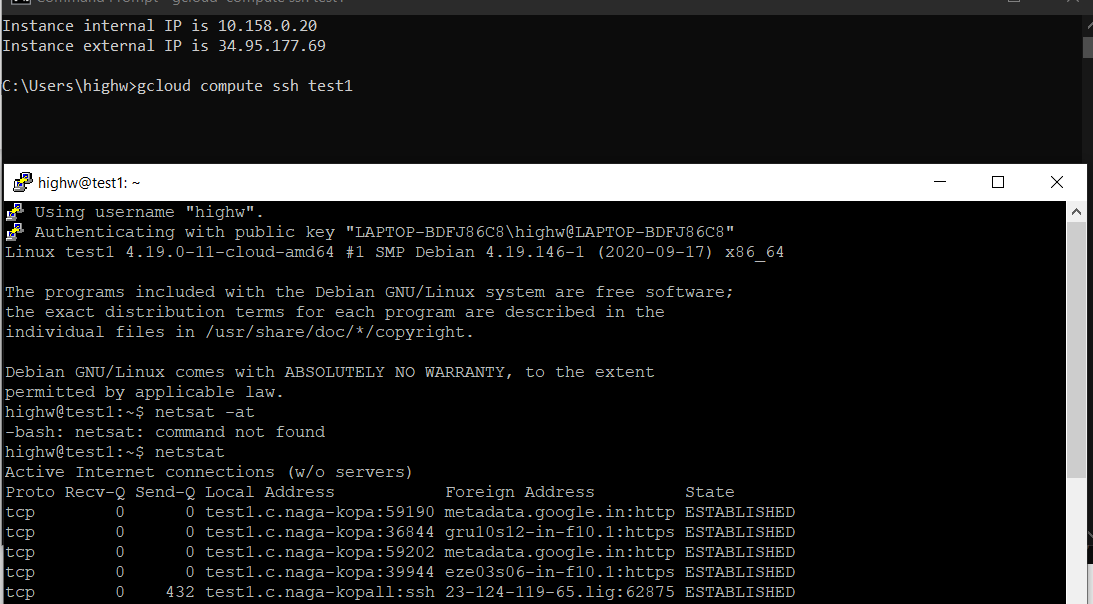
1. **Creating an instance**



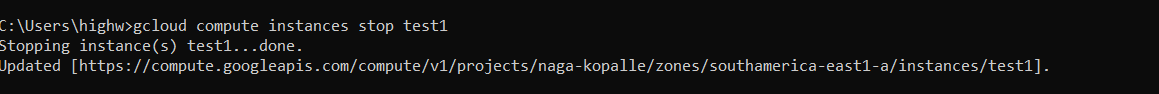
1. **Starting an instance**



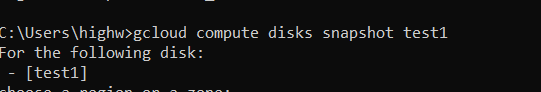
1. **SSHing into the vm instance**

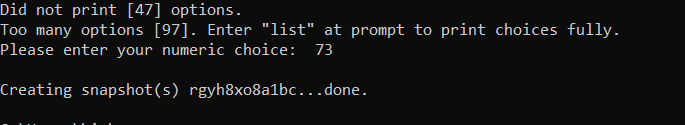


1. **Suspending an instance**

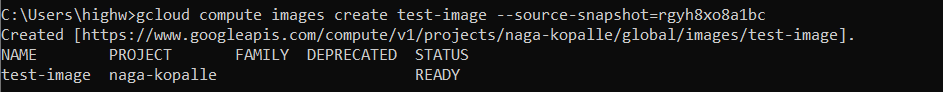


1. **Creating snapshot of disk**

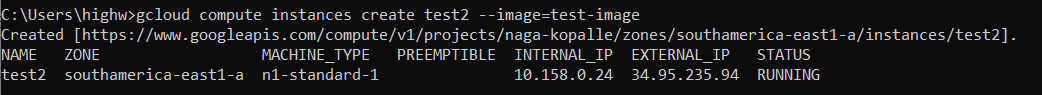




1. **Creating image from snapshot**



1. **Creating another vm instance from the image**



**Map reduce on the cloud**

**1. Firstly I have created two vms - master and keystore .**

**2. Installed all my dependencies in both vms and then created image – cloud-assign image from the snapshot of keystore disk.**

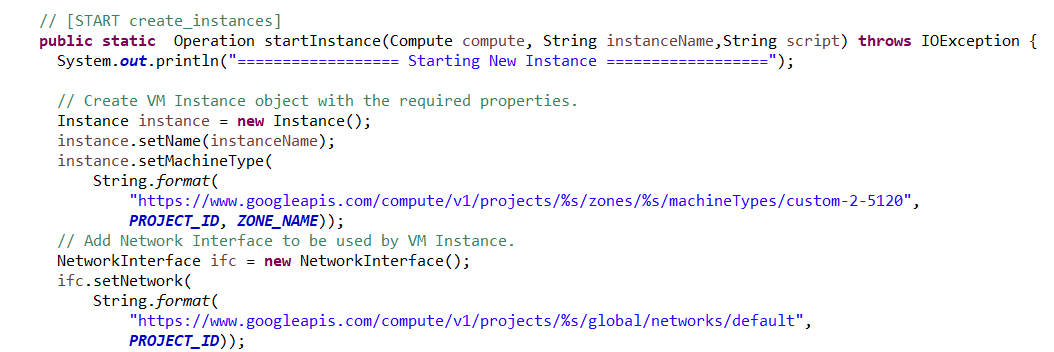
**3. I started keystore server on keystore vm and then I have started master http server on master vm .**

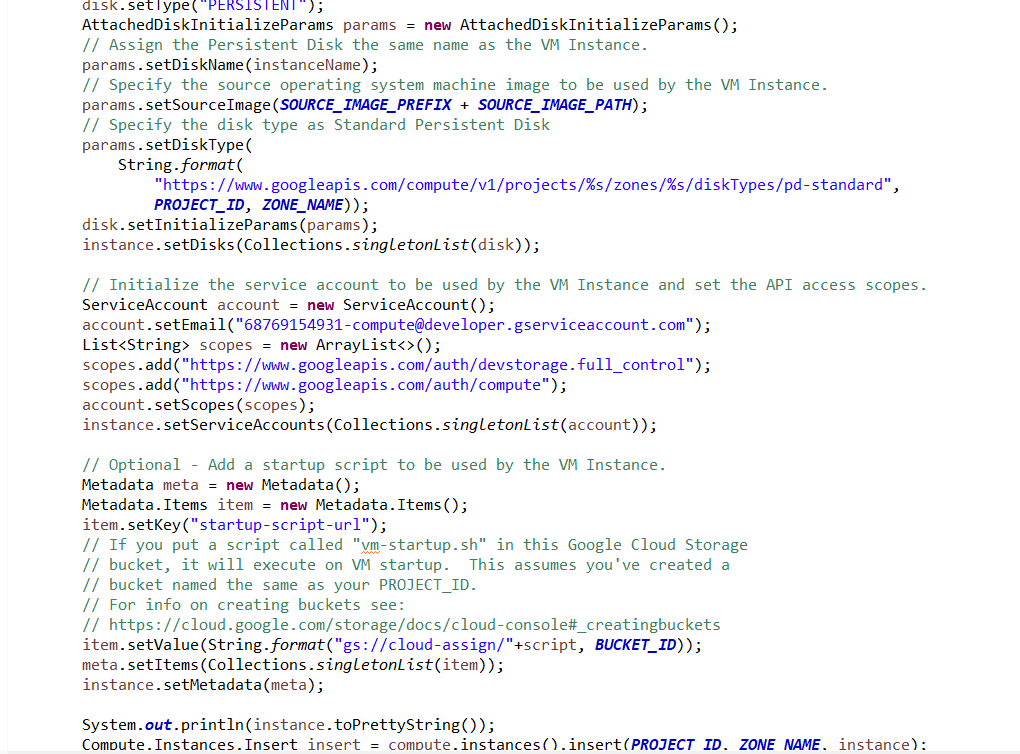
**4. This image is then used by master to spawn reducers and mappers .**

**5. I have use the JavaClient for creating and deleting new instances :**

**Below is the code snippet :**

**Creating instance : ComputeEngine.java**





**As you can see , I have specified the project id, zonename and my service account credentials , and my initial startup-script url for executing after vm start .**

**This is the out you get when you start an instance:**

**Text

Description automatically generated**

**But, for some reason the startup script isn’t running after vm has initialized, so we have manually execute our scripts on each of the vms …**

**So once the mapper job is started, it communicates with the master for getting it’s key , keystore address and port and the function. The way I have implemented is, you have a queue of mapper id’s when ever a mapper is created and when a mapper communicates with it on start , I poll the first mapper id in the queue and share it with the mapper which will be its unique id.**

**Once we run our map tasks ,for example for 1 mapper and word count :**

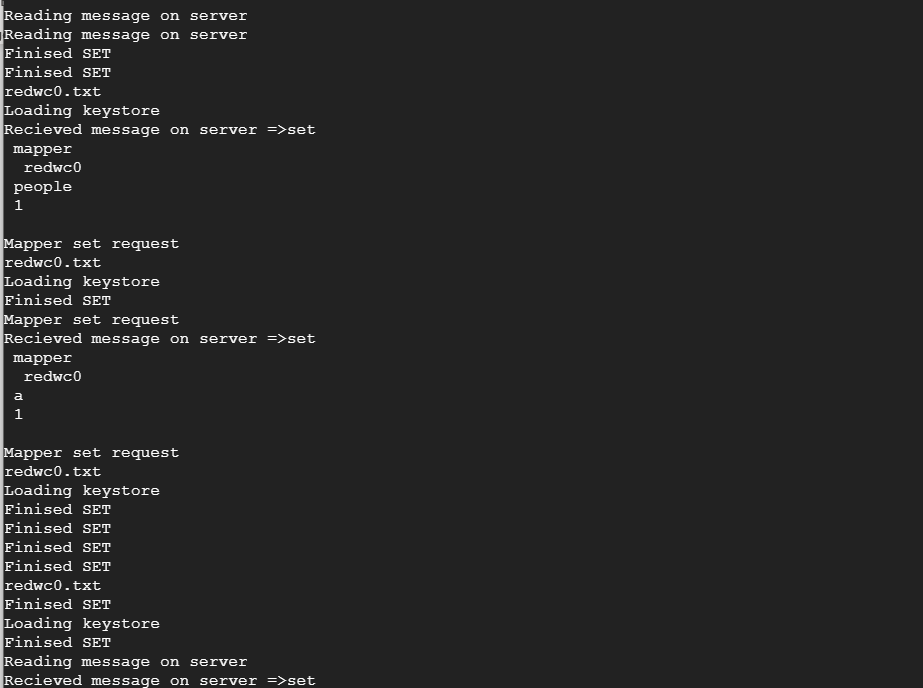
**We get the following output on mapper :**

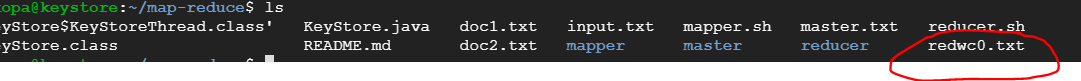
**A close up of a logo

Description automatically generated**

**As you can see , the mapper is communicating with keyvalue store to store the data .**

**Below is the output for keyvalue store**



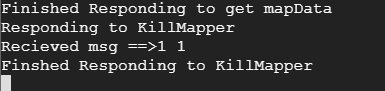


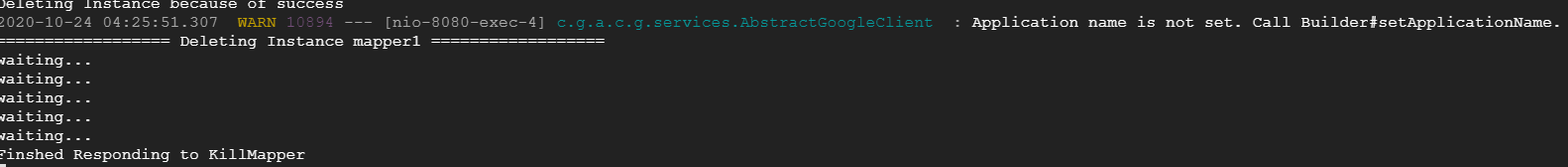
**Once a mapper is successfully executed , it communicates with the master , with its exit code based on which the master will either decide to just kill the current mapper vm if the mapper has successfully executed it’s task or**

**Delete the vm instance and spawn the mapper job again .The code snipped for that is here:**



**You can see the output here on the master machine:**





**The way I have implemented is when the mapper passes on its key and status code , I use that to identify the instance and kill it , if at all I have to restart the instance , then I use the same key to create the similar instance and place the mapper id in the queue again , so when it communicates back .. I can share it’s unique data .**

**The process is the same for reducers .**

**I have written various test cases for these get/post functions which can be found in Test folder .**

**Test1 – initiating the mapreduce task**

**Test2 ,Test3 – getting mapper / reducer data**

**Test4-Test8 – Various mapper/reducer kill scenarios .**

**Please Note: First check for the availability of particular vms before executing the test classes.There is some bug in the code which is preventing the reducers from spawning , otherwise redcuers would also work in seasmless fashion.**

**So the basic architecture flow is this :**

1. **Start Master**
2. **Start KeyStore**
3. **Communication b/n Master-keystore through sockets**
4. **Master receives request /mapred**
5. **Starts VM for Mappers , stores their ids**
6. **Mapper tasks are manually started on VMs, On start Mapper tasks Communicate with Master for their credentials,keystore credentials**
7. **Mappers perform Mapper tasks.**
8. **Mappers communicate with Master /killMapper indicating either success/failure of their job**
9. **Master accordingly kills the vms or kills & restarts the mapper jobs accordingly.**
10. **Master meanwhile continuously polls for checking mapper jobs status ( Something went wrong here)**
11. **Once done , Similar process repeated for reducer tasks ( Step 6 – Step 10)**

**How to run :**

**gcloud compute instances start master**

**gcloud compute instances start keystore**

**gcloud compute ssh master**

1. **cd map-reduce**
2. **sh master.sh**

**gcloud compute ssh keystore**

1. **cd map-reduce**
2. **sh keystore.sh**

**Then run Test1.java in Testfolder from local ..which will initiate the client request.**

**You can observe that mapper vm is created**

**Now for each of the mapper vms created**

**gcloud compute ssh mapper[ID]**

1. **cd map-reduce**
2. **sh mapper.sh**