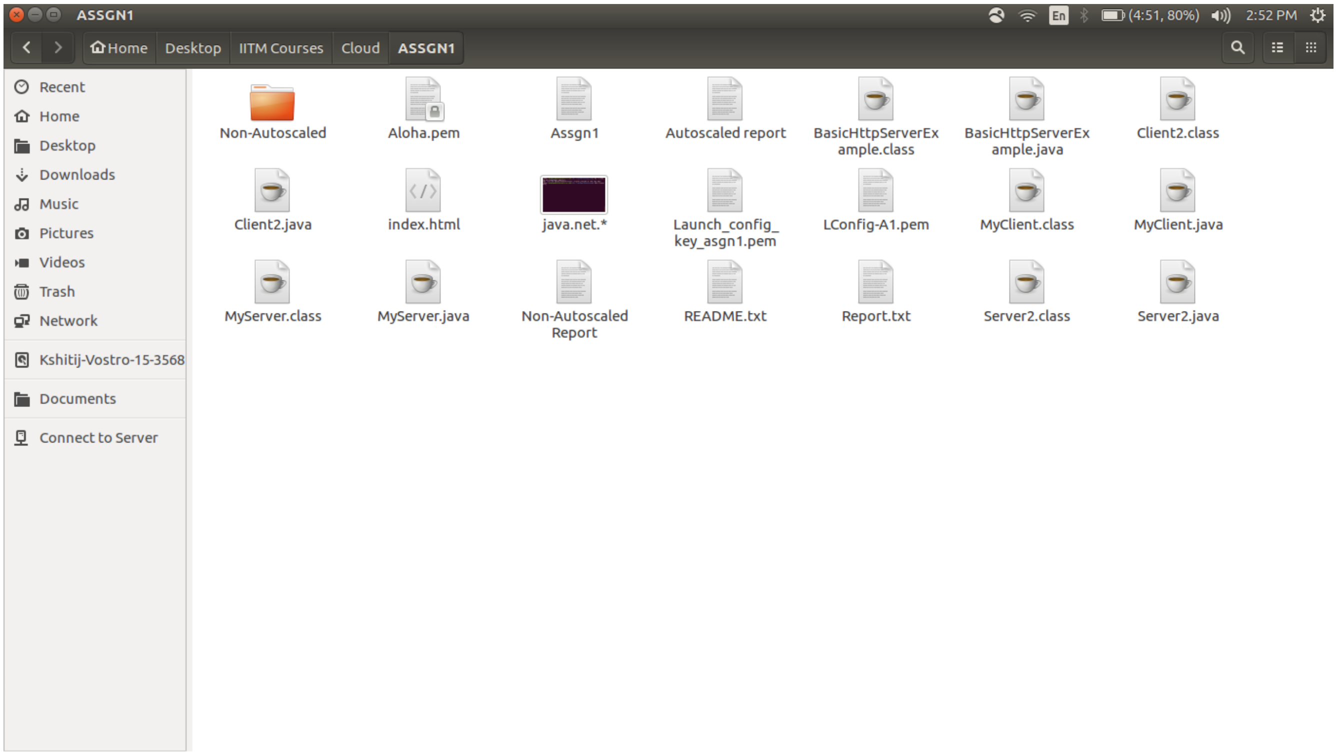
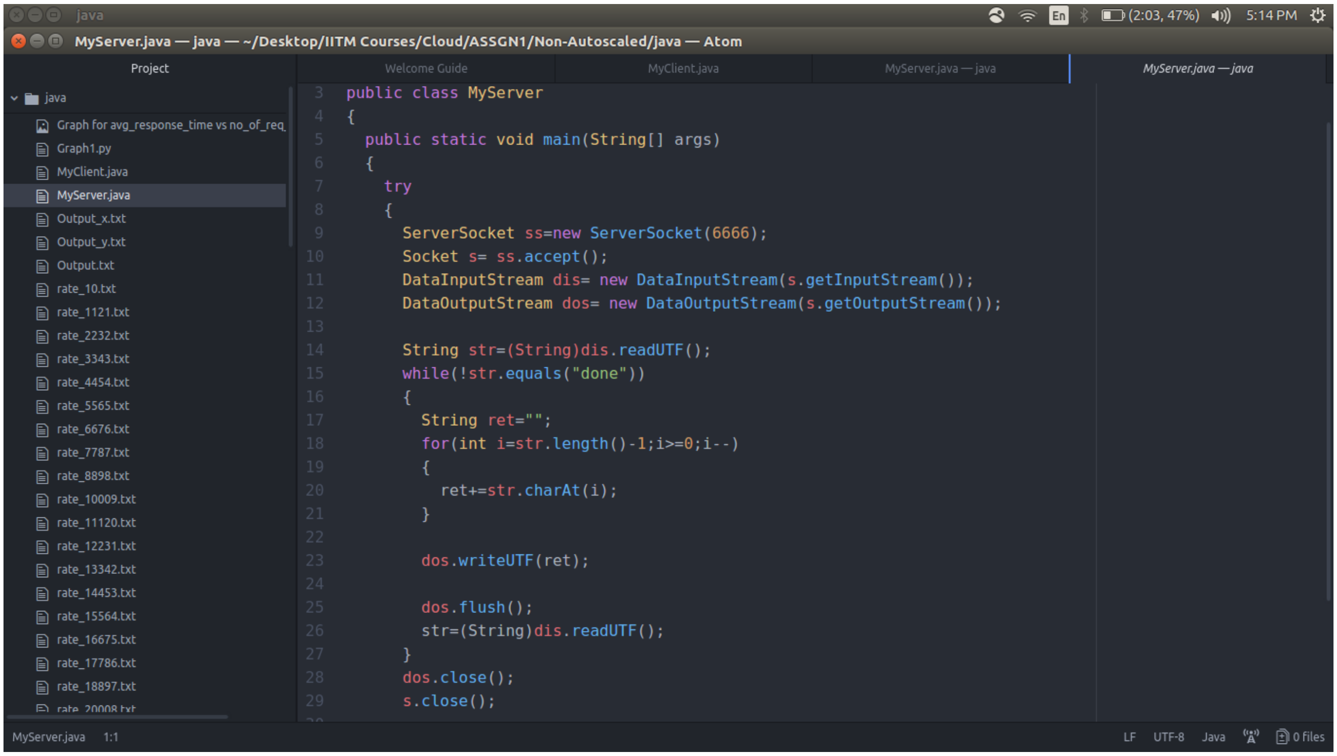
I have completed all aspects of this exercise, including all rate_x.txt files and a graph of average_response_time vs no_of_req_per_sec
My steps were:

1) I made an instance, whose key was Aloha.pem and I would run my client and server , both, on this instance.



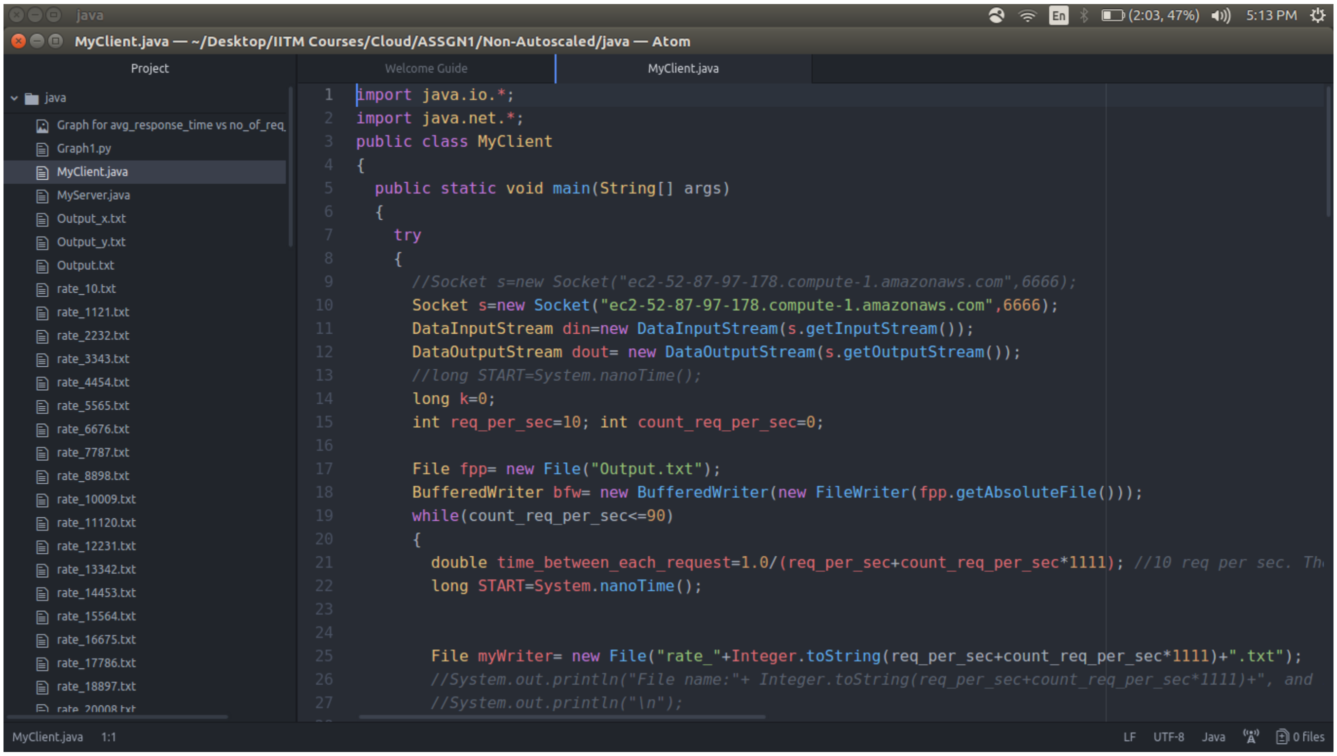
Note that my client is not on my machine, but it also on my cloud instance.
This allows me to reach request rates of 100000 requests per second, and doesnt make a difference, as we anyway want to plot a graph of response time of cloud server, which is the same.

2) My MyServer.java and Client.java codes are:-
MyServer.java-



```java
public class MyServer
{
  public static void main(String[] args)
  {
    try
    {
      ServerSocket ss=new ServerSocket(6666);
      Socket s= ss.accept();
      DataInputStream dis= new DataInputStream(s.getInputStream());
      DataOutputStream dos= new DataOutputStream(s.getOutputStream());

      String str=(String)dis.readUTF();
      while(!str.equals("done"))
      {
        String ret="";
        for(int i=str.length()-1;i>=0;i--)
        {
          ret+=str.charAt(i);
        }

        dos.writeUTF(ret);

        dos.flush();
        str=(String)dis.readUTF();
      }
      dos.close();
      s.close();
```

MyCLient.java-



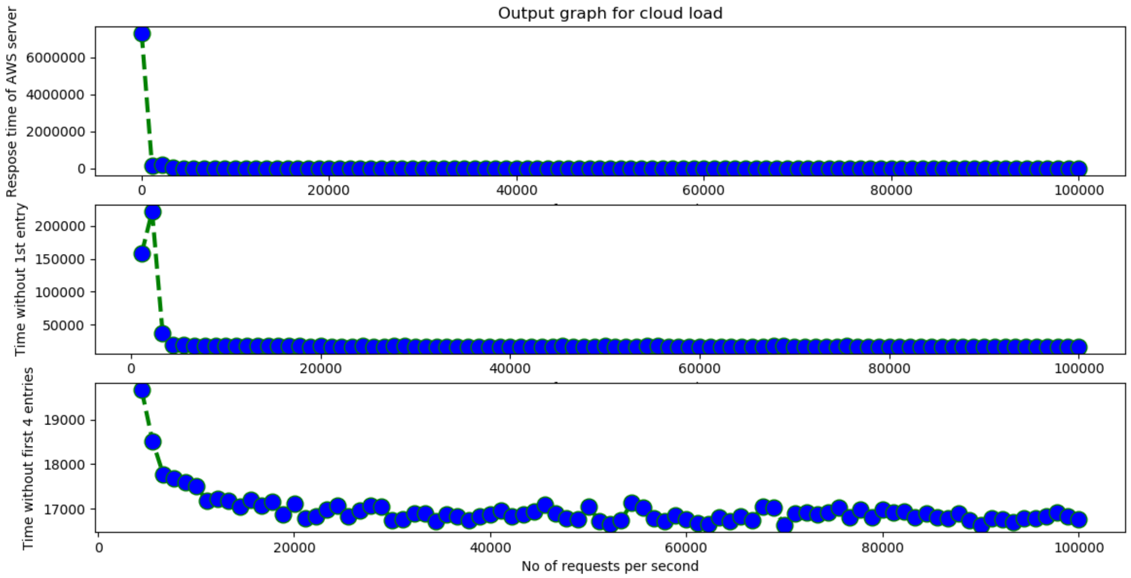
```java
import java.io.*;
import java.net.*;
public class MyClient
{
  public static void main(String[] args)
  {
    try
    {
      //Socket s=new Socket("ec2-52-87-97-178.compute-1.amazonaws.com",6666);
      Socket s=new Socket("ec2-52-87-97-178.compute-1.amazonaws.com",6666);
      DataInputStream din=new DataInputStream(s.getInputStream());
      DataOutputStream dout= new DataOutputStream(s.getOutputStream());
      //long START=System.nanoTime();
      long k=0;
      int req_per_sec=10; int count_req_per_sec=0;

      File fpp= new File("Output.txt");
      BufferedWriter bfw= new BufferedWriter(new FileWriter(fpp.getAbsoluteFile()));
      while(count_req_per_sec<=90)
      {
        double time_between_each_request=1.0/(req_per_sec+count_req_per_sec*1111); //10 req per sec. Th
        long START=System.nanoTime();


        File myWriter= new File("rate_"+Integer.toString(req_per_sec+count_req_per_sec*1111)+".txt");
        //System.out.println("File name:"+ Integer.toString(req_per_sec+count_req_per_sec*1111)+", and
        //System.out.println("\n");
```

Note that the DNS name of my instance is mentioned, that corresponds to Aloha.pem; and both have port 6666 open, on which client will ping server.

If you observe my code, you will observe that I have a variable count_req_per_sec, which starts from 0, and req_per_sec which starts from 10.
Now what I am doing is:
In a while loop, I am doing till count_req_per_sec becomes <=90-
Make file named-
"rate_"+Integer.toString(req_per_sec+count_req_per_sec*1111)+".txt"
For each (req_per_sec+count_req_per_sec*1111) # of requests per second, I calculate the time it takes to invert the string: “Put a coin to your witcher”, by making sure that each request has constant time between each other, so that (req_per_sec+count_req_per_sec*1111) # of requests can be met in 1 second.

I am writing these counts to the created file. Thus, at last, I will get rate_100000.txt, with 100000 entries, which are the response times for the 100000 requests in nano seconds.

3) I am able to get all the files. In Output.txt, I have the average of all the rate_x.txt files. Thus, they total 100000 in count.
When you see the graph:-



At the start, each response takes a lot of time.

Then, it progressively decreases, but does so, while fluctuating increases and decreases.
This may possibly be due to high initial time due to direction from memory, but later lesser time due to cache access.

I think I have done all I can for Non-Autoscaling part.
Thanks for reading :)