EE4221 Cloud Computing:

Mini Project Report

Abhimanyu Bhati, SID: 55991421

1. Background

The mini-project required us to develop a client server program to perform matrix multiplication using a fork-join multithreading framework. The program was to be tested locally first and after testing the robustness of the code, we were to deploy this program on an AWS instance (VM).

2. AWS Environment Infrastructure

After developing the client (*Client55991421.java*) and server (Server55991421.java) programs, results were obtained after testing the programs on the local host IP. Now, these programs were to be deployed on an AWS instance. Hence, an EC2 instance was chosen to host the programs. The client program runs on our local host while the AWS instance runs the server program. Before deployment, the AWS instance was to be configured along with the VPC and Security Group associated with it. After this set-up, java was installed on the created instance to be able to run the programs deployed. The following sequence of tasks were performed to deploy the program on our AWS instance.

A. <u>Configuration of VPC</u>

A VPC named *MiniProjectCloudComputing-vpc* was configured with the following settings

i. IPv4 CIDR block set to 10.0.0.0/16
ii. Number of AZs = 1
iii. Number of public subnets =1
iv. Number of private subnets =1
v. CIDR subnets blocks in *us-east-1a*
vi. Public Subnet CIDR block set to 10.0.0.0/24
vii. Private Subnet CIDR block set to 10.0.1.0/24
viii. NAT Gateways = in 1 AZ
ix. VPC endpoints to *None*
x. DNS hostnames and DNS resolution *enabled*

B. <u>Configuration of Security Group</u>

A VPC Security group named *MiniProjectCloudComputingx-sg* was created to be associated with the instance, to allow inbound SSH connections and also allow client-server communication using TCP over a given port (42210). The following settings were applied

i. Security Group Name: *MiniProjectCloudComputingx*
ii. Description: *Allow SSH and TCP access*
iii. Inbound Rules as follows:

Type	Protocol	Port Range	Source
SSH	TCP	22	Anywhere IPv4
Custom TCP	TCP	42210	Anywhere IPv4

iv. Outbound Rules are set to allow all traffic, however there is no outbound traffic pertaining to the EC2 instance.
v. VPC: *MiniProjectCloudComputing-vpc* configured in earlier

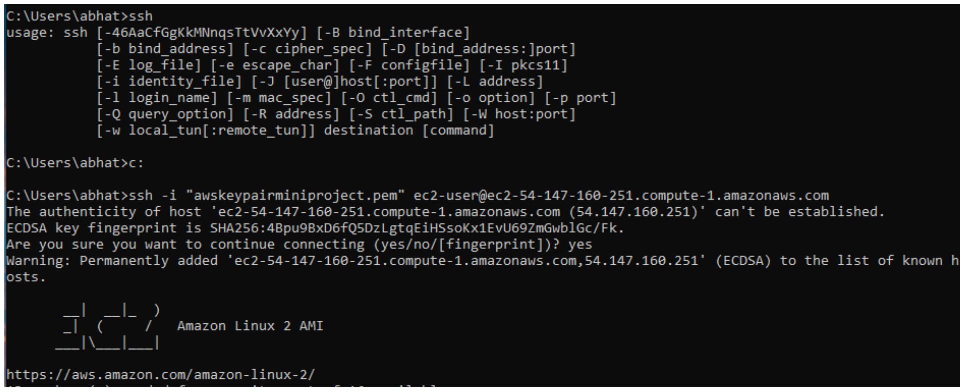
C. <u>Configuration of EC2 Instance</u>

An EC2 instance called *MiniProjectCloudComputingx*

i. Instance type: *t2.micro*
ii. OS: Amazon Linux 2 AMI (HVM) (64-bit x86 architecture)
iii. Key Pair: *awskeypairminiproject.pem*
iv. Storage: 1x8 GiB gp2 Root Volume
v. VPC: *MiniProjectCloudComputingx-vpc* created earlier
vi. Subnet: *MiniProjectCloudComputing-subnet-public1-us-east-1a*
vii. Auto-assign public IP: *Enable*
viii. Firewall: *MiniProjectCloudComputingx-sg* created earlier.
ix. Number of instances: 1

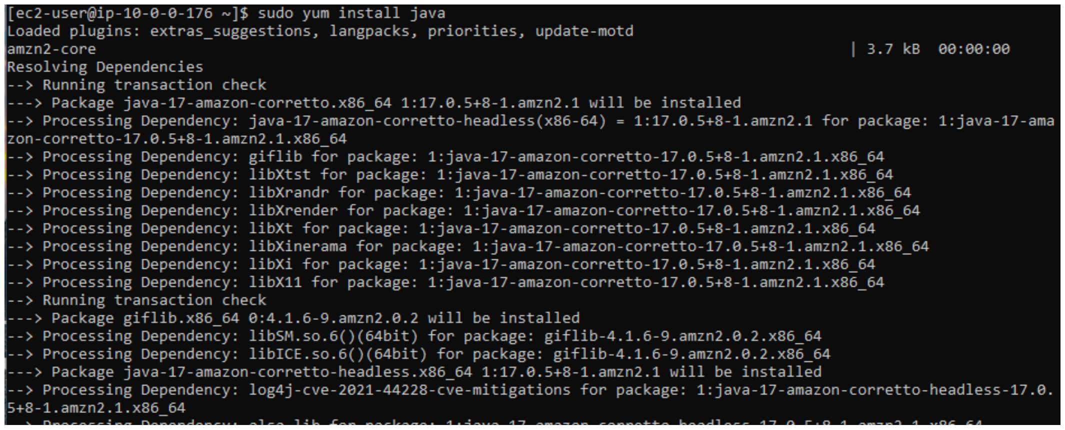
D. <u>Installation of Java on EC2 instance and uploading program</u>

i. Log into our instance through SSH



```
C:\Users\abhat>ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
           [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
           [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
           [-i identity_file] [-J [user@]host[:port]] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] destination [command]

C:\Users\abhat>c:

C:\Users\abhat>ssh -i "awskeypairminiproject.pem" ec2-user@ec2-54-147-160-251.compute-1.amazonaws.com
The authenticity of host 'ec2-54-147-160-251.compute-1.amazonaws.com (54.147.160.251)' can't be established.
ECDSA key fingerprint is SHA256:4Bpu9BxD6fQ5DzLgtqEiHSsoKx1EvU69ZmGwblGc/Fk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-147-160-251.compute-1.amazonaws.com,54.147.160.251' (ECDSA) to the list of known h
osts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
```

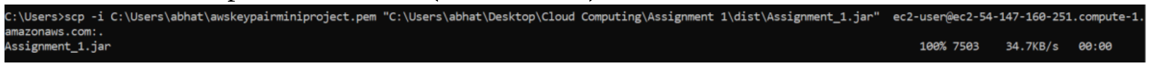
ii. Install Java using *sudo yum install java* Command



```
[ec2-user@ip-10-0-0-176 ~]$ sudo yum install java
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core                                                                   | 3.7 kB  00:00:00
Resolving Dependencies
--> Running transaction check
---> Package java-17-amazon-corretto.x86_64 1:17.0.5+8-1.amzn2.1 will be installed
--> Processing Dependency: java-17-amazon-corretto-headless(x86-64) = 1:17.0.5+8-1.amzn2.1 for package: 1:java-17-ama
zon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: giflib for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXtst for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXrandr for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXrender for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXt for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXinerama for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libXi for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Processing Dependency: libX11 for package: 1:java-17-amazon-corretto-17.0.5+8-1.amzn2.1.x86_64
--> Running transaction check
---> Package giflib.x86_64 0:4.1.6-9.amzn2.0.2 will be installed
--> Processing Dependency: libSM.so.6()(64bit) for package: giflib-4.1.6-9.amzn2.0.2.x86_64
--> Processing Dependency: libICE.so.6()(64bit) for package: giflib-4.1.6-9.amzn2.0.2.x86_64
---> Package java-17-amazon-corretto-headless.x86_64 1:17.0.5+8-1.amzn2.1 will be installed
--> Processing Dependency: log4j-cve-2021-44228-cve-mitigations for package: 1:java-17-amazon-corretto-headless-17.0.
5+8-1.amzn2.1.x86_64
```

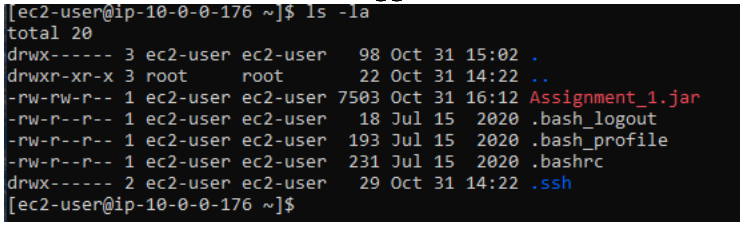
iii. Transferring the Java Program to the EC2 instance

The .jar called *Assignment_1.jar* file of all classes in the a2223.hw1.student package was created and uploaded to the EC2 instance using the following command on a separate terminal (on local PC) from that above.



```
C:\Users>scp -i C:\Users\abhat\awskeypairminiproject.pem "C:\Users\abhat\Desktop\Cloud Computing\Assignment 1\dist\Assignment_1.jar"  ec2-user@ec2-54-147-160-251.compute-1.
amazonaws.com:.
Assignment_1.jar                                                                                      100% 7503    34.7KB/s   00:00
```

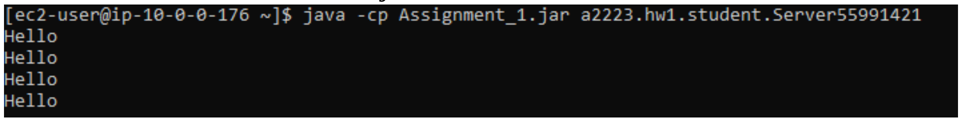
We can verify if the file was uploaded by running the following command on the SSH terminal which is logged into our EC2 instance



```
[ec2-user@ip-10-0-0-176 ~]$ ls -la
total 20
drwx------ 3 ec2-user ec2-user   98 Oct 31 15:02 .
drwxr-xr-x 3 root     root       22 Oct 31 14:22 ..
-rw-rw-r-- 1 ec2-user ec2-user 7503 Oct 31 16:12 Assignment_1.jar
-rw-r--r-- 1 ec2-user ec2-user   18 Jul 15  2020 .bash_logout
-rw-r--r-- 1 ec2-user ec2-user  193 Jul 15  2020 .bash_profile
-rw-r--r-- 1 ec2-user ec2-user  231 Jul 15  2020 .bashrc
drwx------ 2 ec2-user ec2-user   29 Oct 31 14:22 .ssh
[ec2-user@ip-10-0-0-176 ~]$
```

E. <u>Running Server Program on EC2 instance</u>

Run the file *Server55991421.java* on our EC2 instance via the SSH terminal:



```
[ec2-user@ip-10-0-0-176 ~]$ java -cp Assignment_1.jar a2223.hw1.student.Server55991421
Hello
Hello
Hello
Hello
```

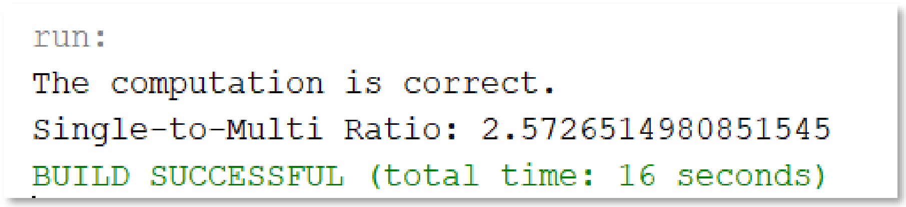
A simple print statement outputting "Hello" was placed in the main function of the server file to indicate that the server file is running. This server responds to any client trying to communicate with it on the public ipv4 address "54.147.160.251" on the port number 42210.

3. Results

The program was tested in 3 different ways described below and 4 concurrent client instances were opened to communicate with the server.

A. Local Client - Local Server (Local Computation)

When both client(s) and server are run locally, the following results are obtained

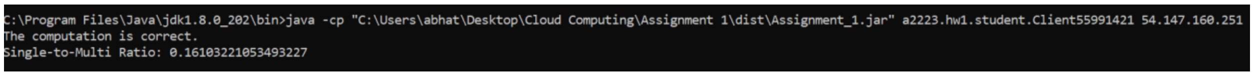


```
run:
The computation is correct.
Single-to-Multi Ratio: 2.5726514980851545
BUILD SUCCESSFUL (total time: 16 seconds)
```

The average single-to-multi ratio is 2.13 (approximate) which means that the fork-join framework approach is about 2.1 times faster than a single threaded approach which is expected.

B. Local Client - AWS EC2 Server (Mixed Computation)

The client program(s) were run through local PC terminal(s) using the following command and the results are observed

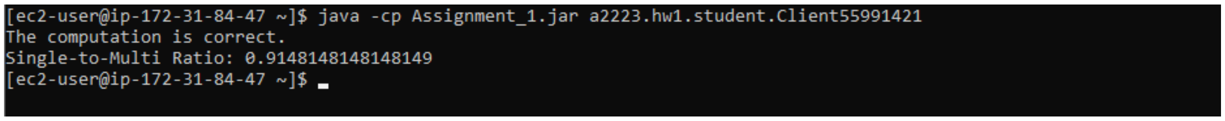


```
C:\Program Files\Java\jdk1.8.0_202\bin>java -cp "C:\Users\abhat\Desktop\Cloud Computing\Assignment 1\dist\Assignment_1.jar" a2223.hw1.student.Client55991421 54.147.160.251
The computation is correct.
Single-to-Multi Ratio: 0.16103221053493227
```

Here the average single to multi ratio is 0.149 (approximate). This is lower than the result obtained on local computation due to the connection establishment time and transmission time of the Matrix objects created from the Matrix class in the Client program running on the local PC in Hong Kong to the Server program running on the EC2 instance in N Virginia, US (us-east-1a availability zone) adding to the execution time hence reducing the ratio.

C. AWS EC2 Client - AWS EC2 Server (AWS Computation)

The client program was run on a **new** SSH terminal logged into the EC2 instance similar to the server program using the .jar file already uploaded earlier.



```
[ec2-user@ip-172-31-84-47 ~]$ java -cp Assignment_1.jar a2223.hw1.student.Client55991421
The computation is correct.
Single-to-Multi Ratio: 0.9148148148148149
[ec2-user@ip-172-31-84-47 ~]$
```

The average single-to-multi ratio is 0.79 which is better than the hybrid computation yet still not as fast as local computation. This is again likely due to the connection establishment bottleneck as well as the vCPUs being limited to 2 for our EC2 instance.

4. Conclusion

Through the processes described in this report, we were able to successfully deploy our Client Server program for matrix multiplication using the fork-join framework of multithreading on an AWS EC2 instance. We tested the same deployment in 3 different ways and obtained favorable results. We also tested multiple clients running concurrently to communicate with a single server. Hence, through this mini project I was able to learn more about configuring VPCs, Security Groups and Instances on AWS for our use case.