Assignment 1&2

**Code:**

**test.py**

import webapp2

class MainPage(webapp2.RequestHandler):

def get(self):

self.response.write("Hello World!")

print("Hello")

app = webapp2.WSGIApplication([('/', MainPage)], debug=True)

**app.yaml**

runtime: python312

api\_version: 1

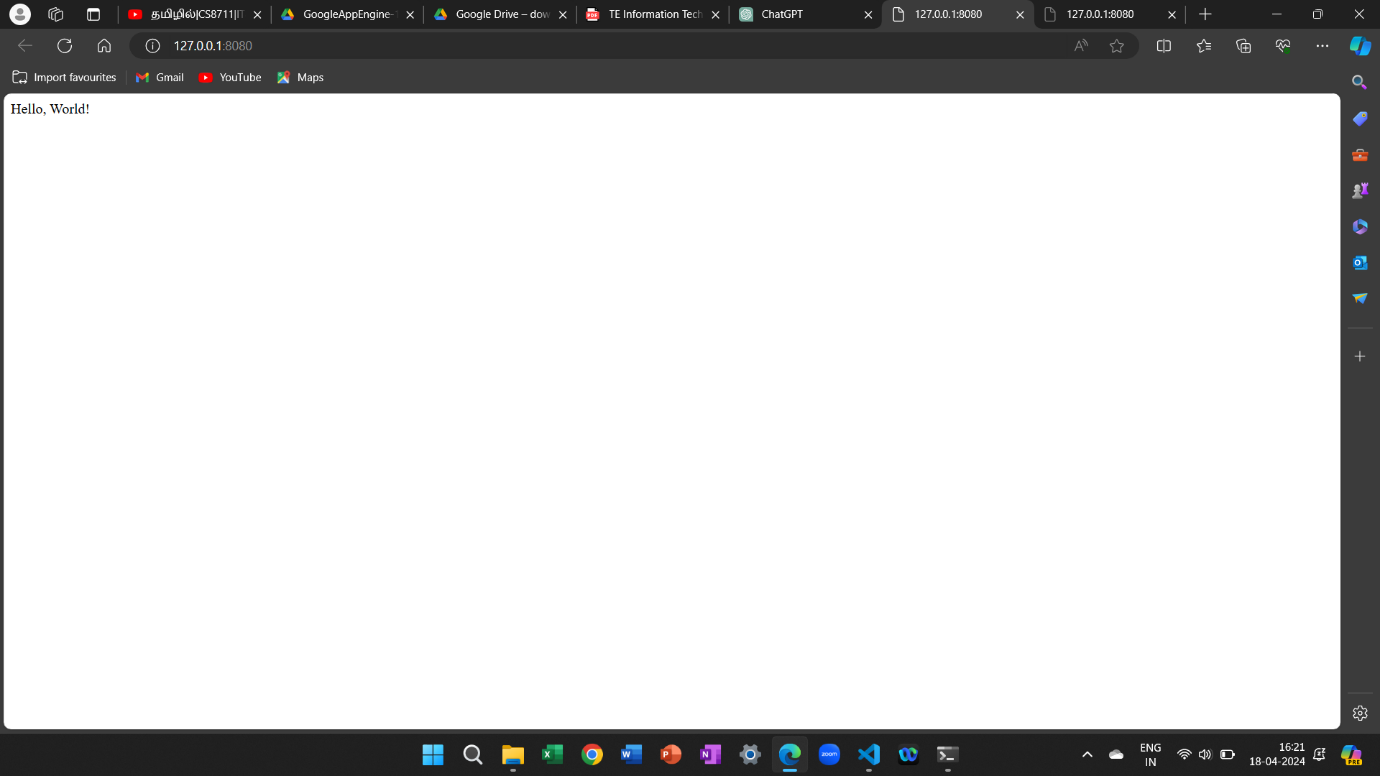
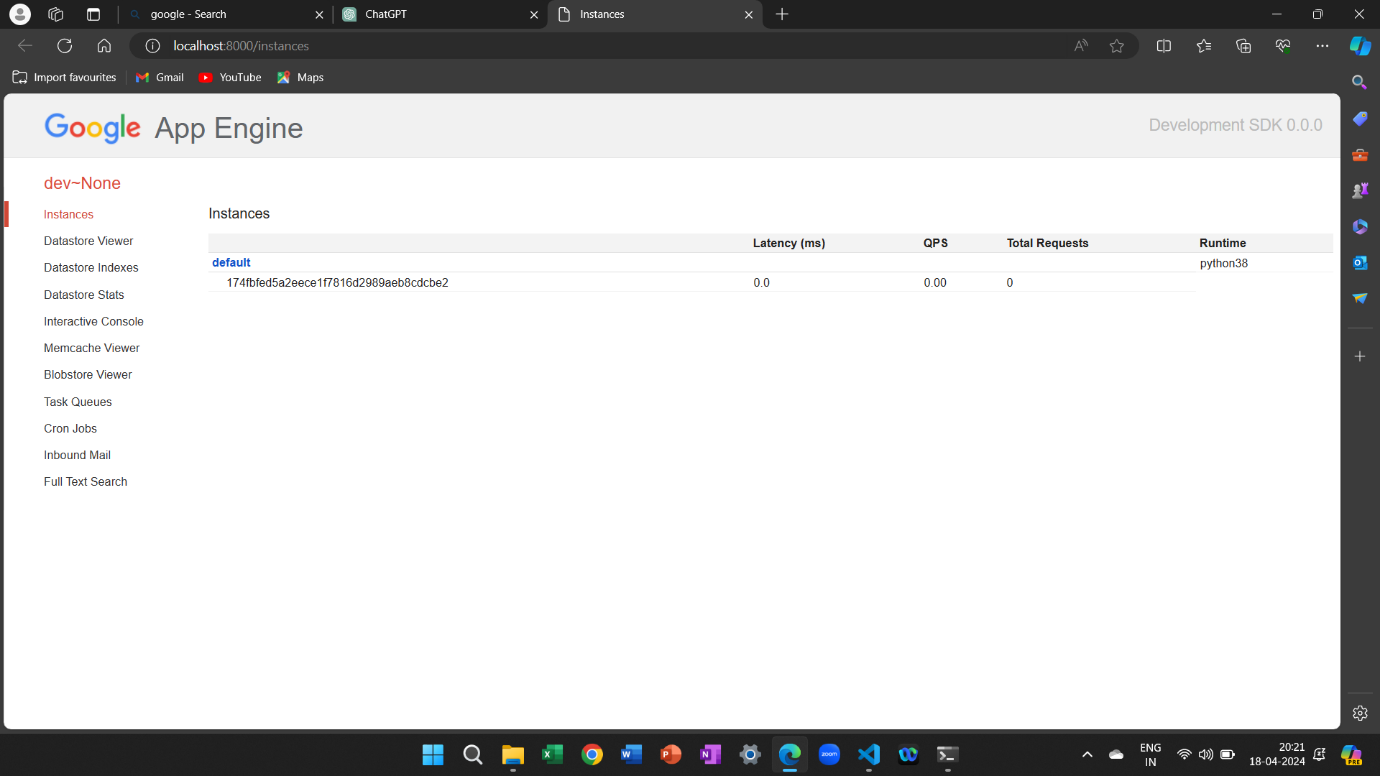
threadsafe: false

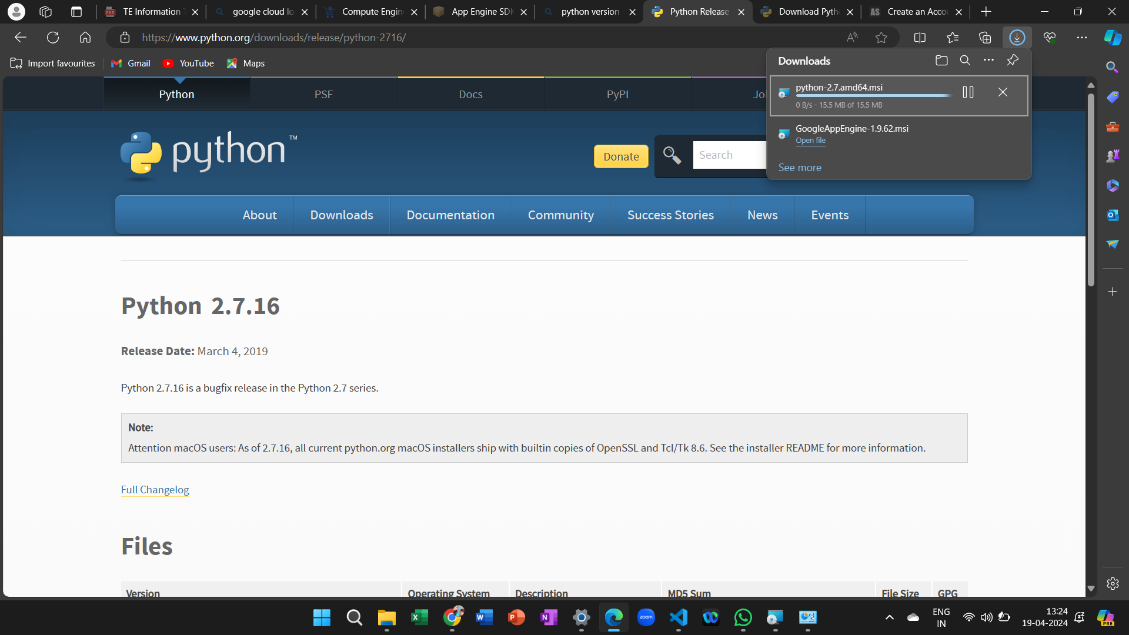
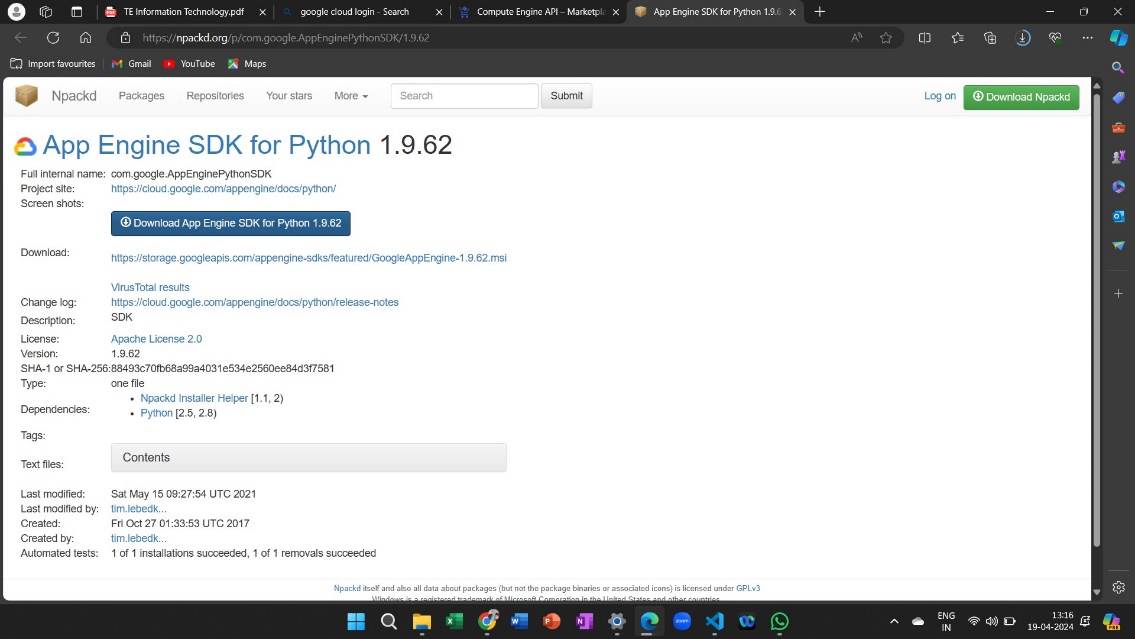
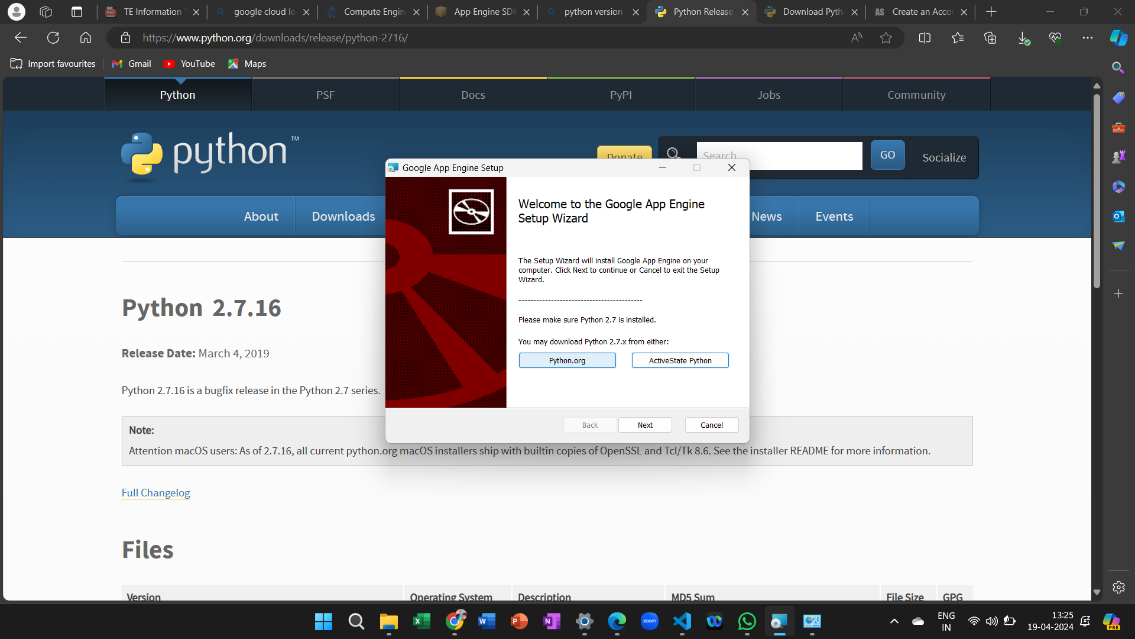
handlers:

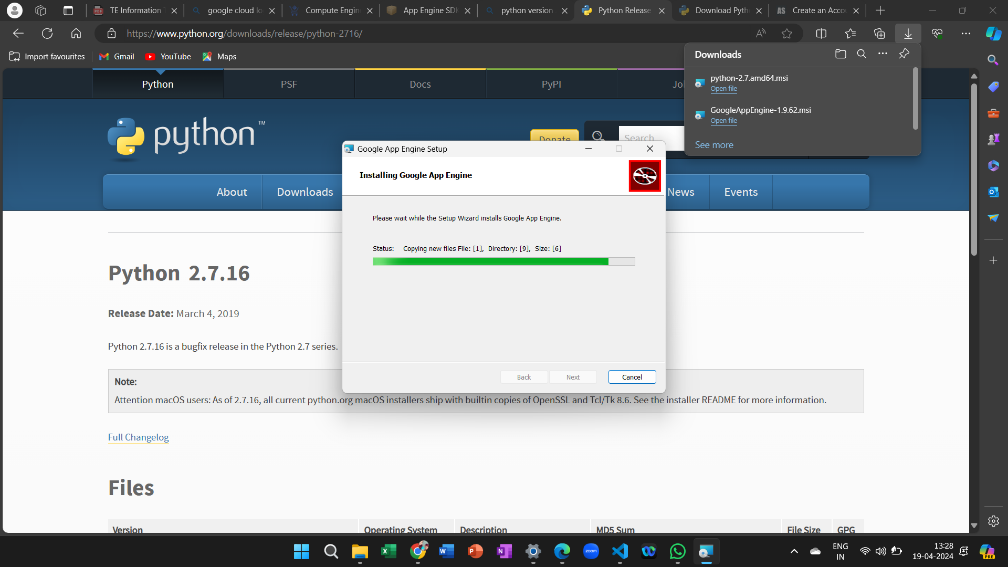
- url: /

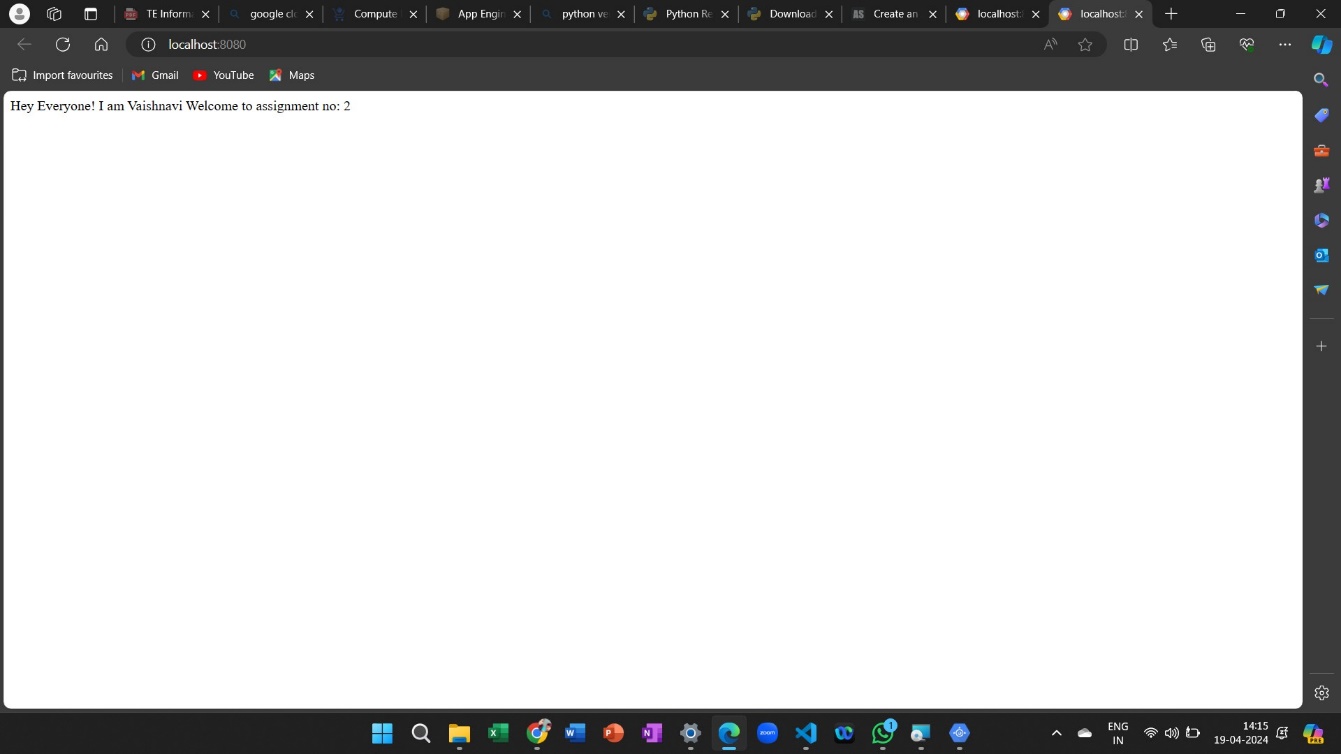
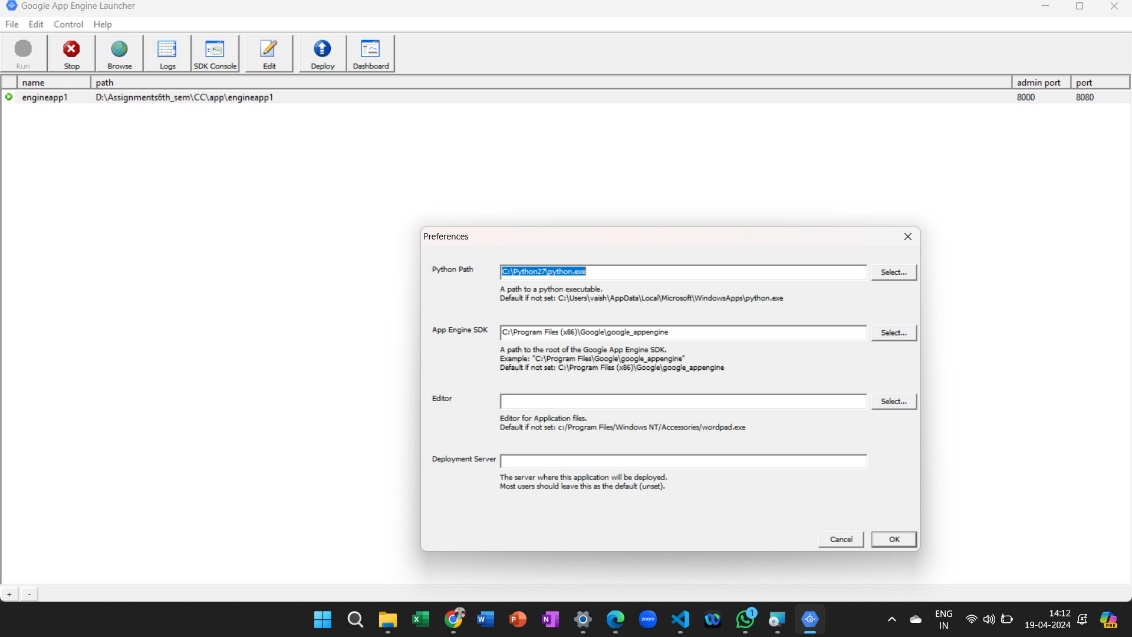
script: test.app

**Output:**



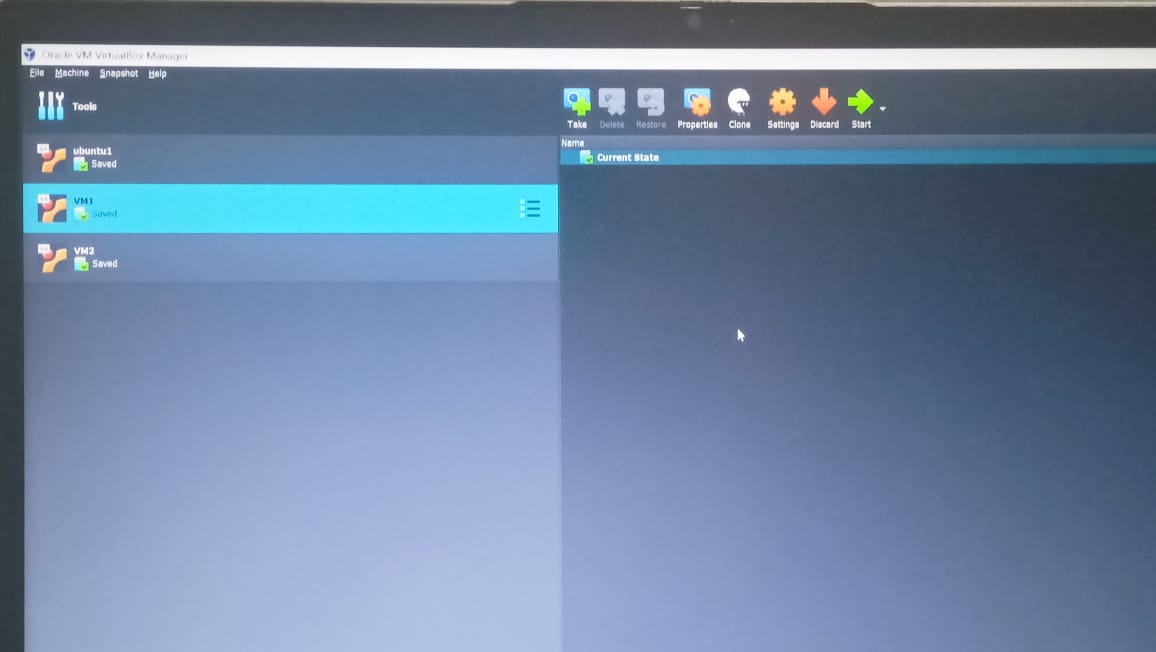
Procedure For installation of GAE: 

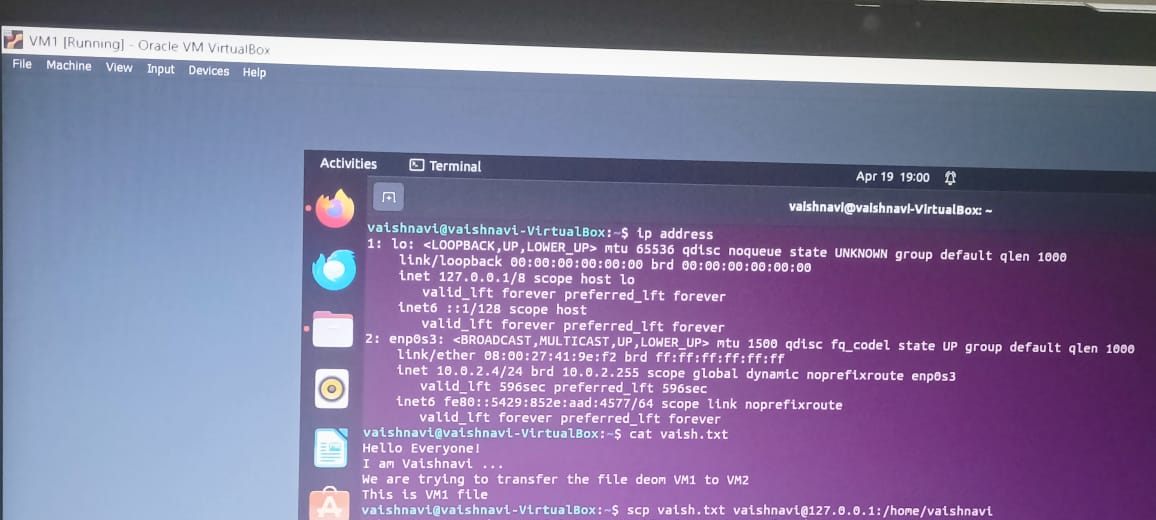


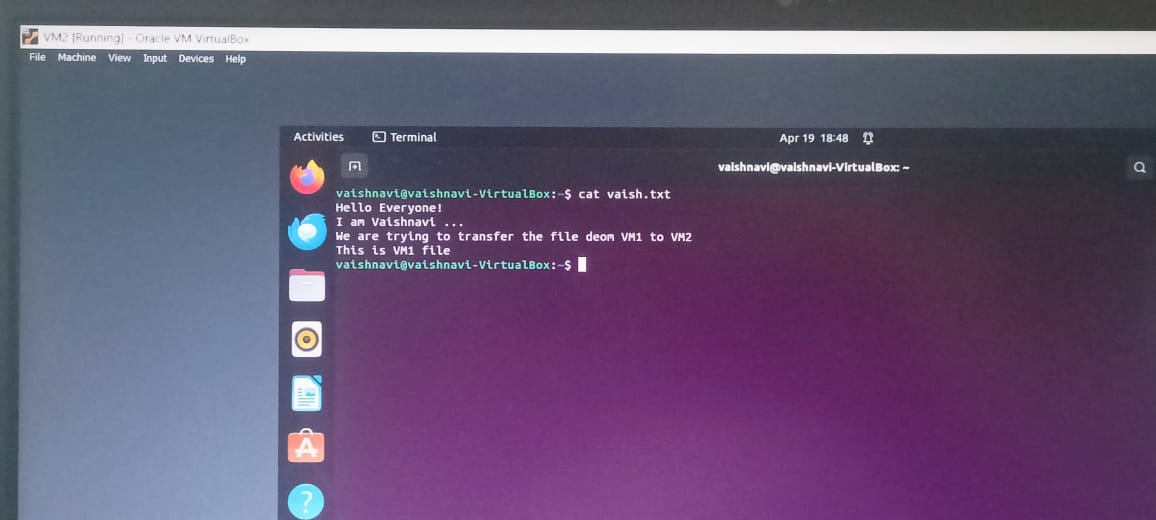


Assignment no.: 5

Procedure to transfer file from one Virtual Machine to another Virtual machine:

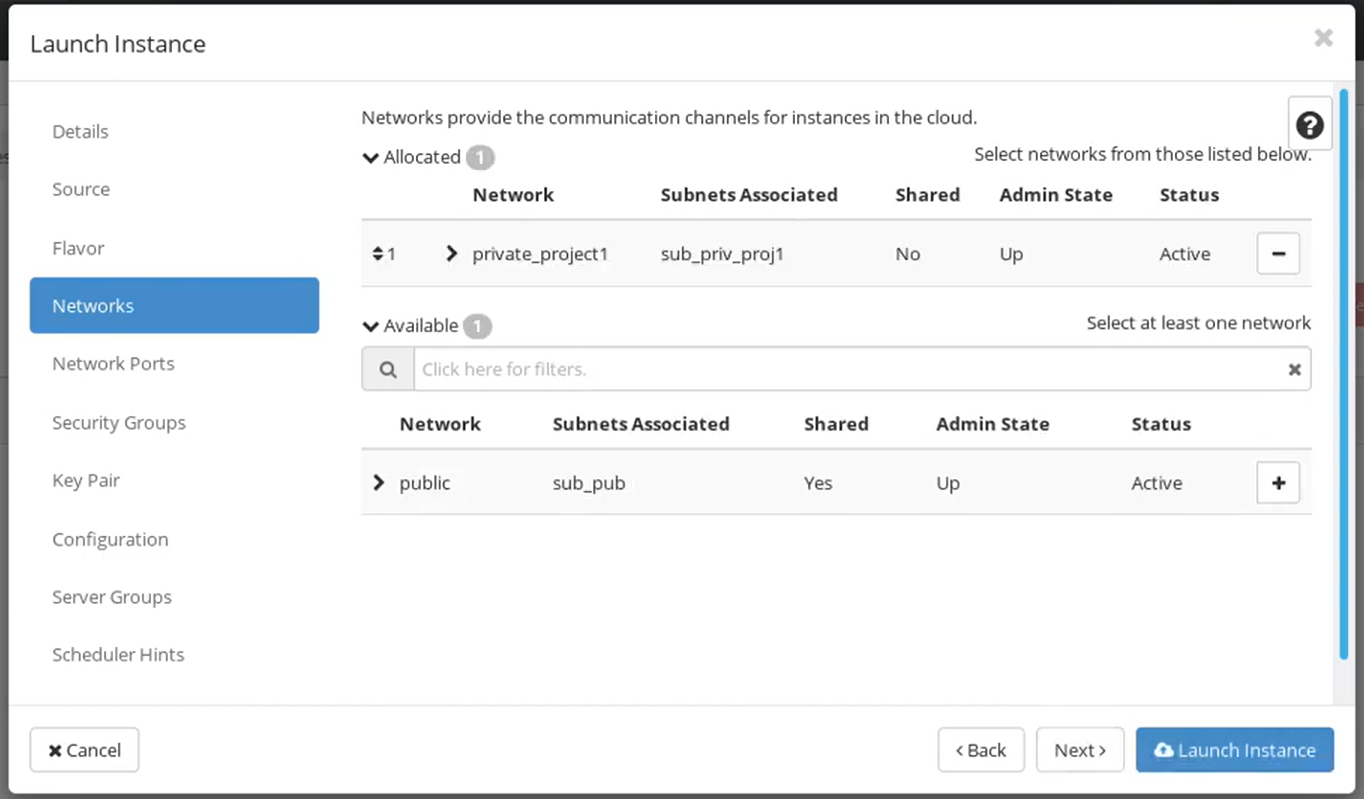
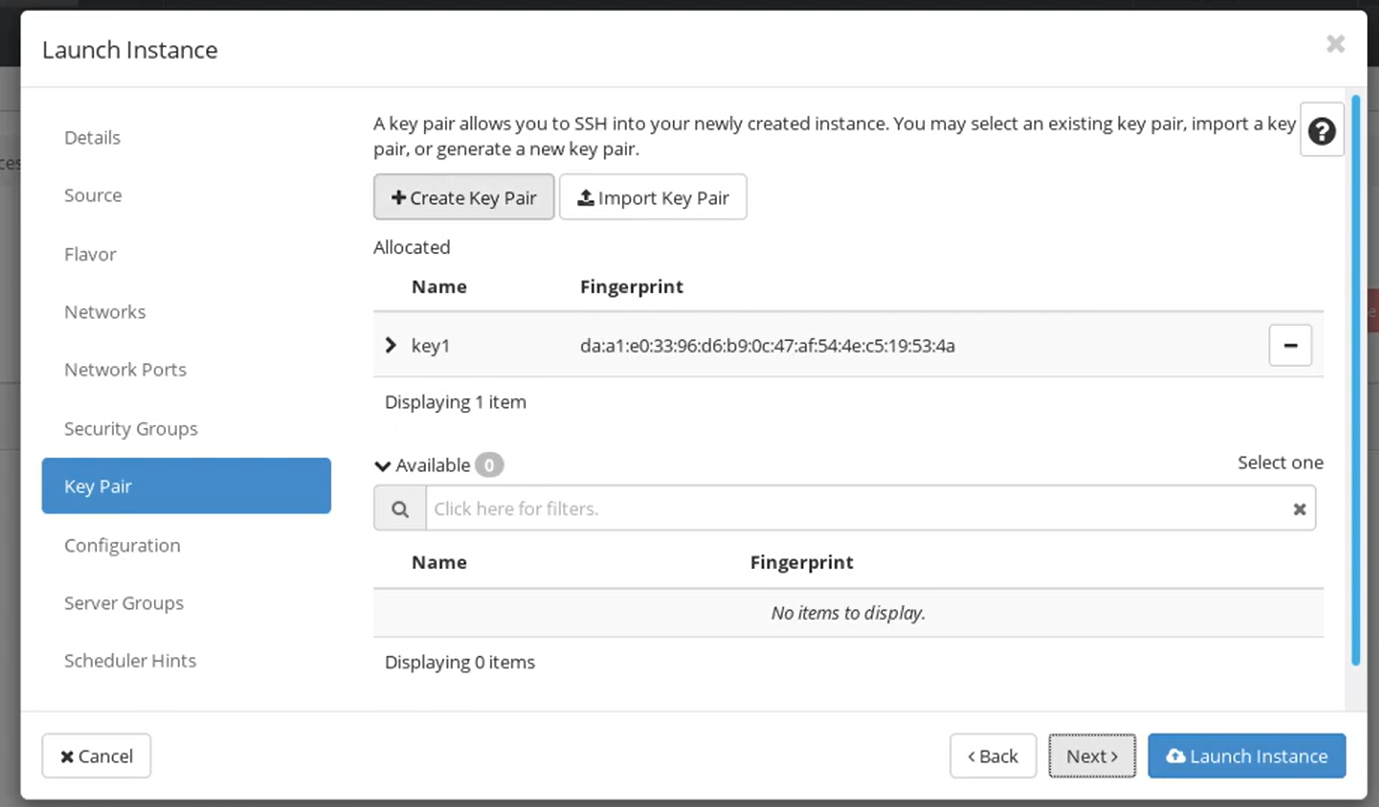
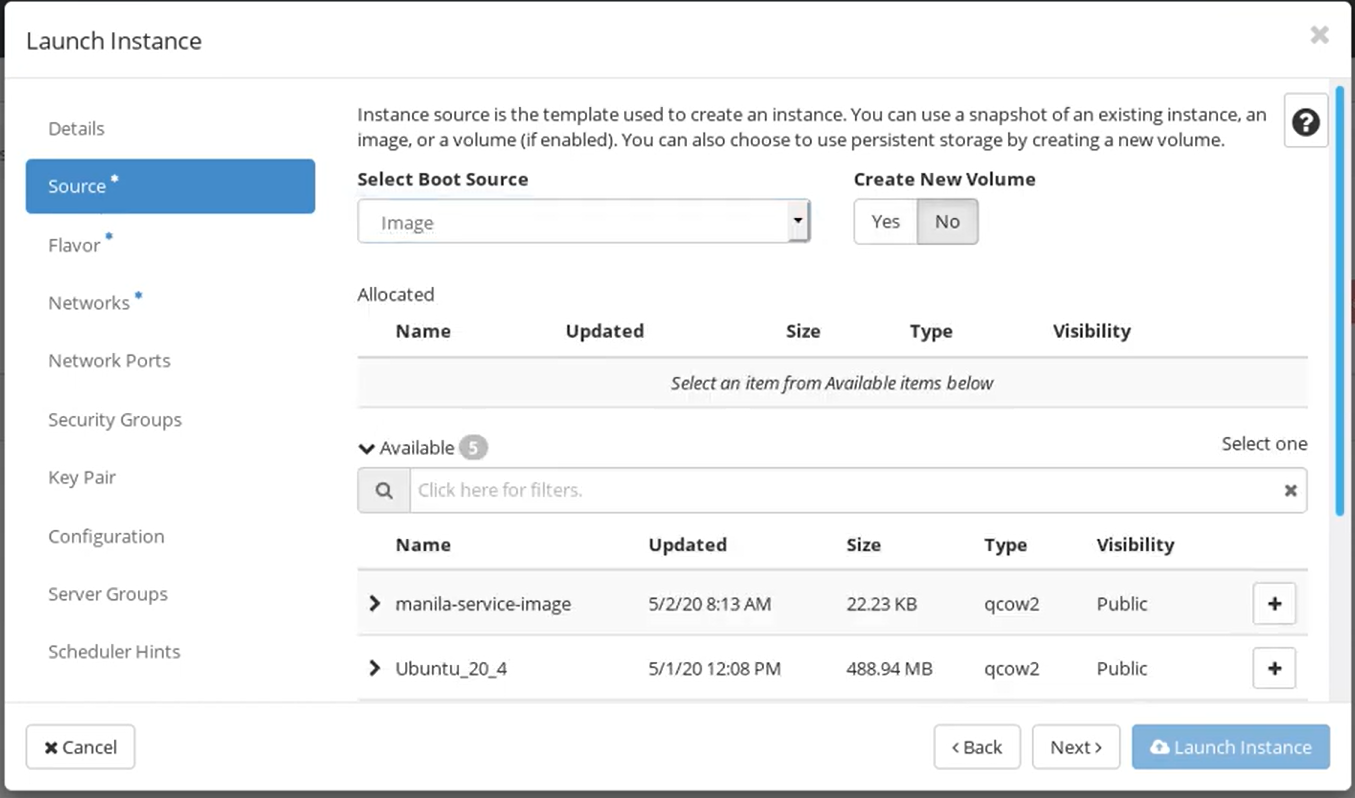
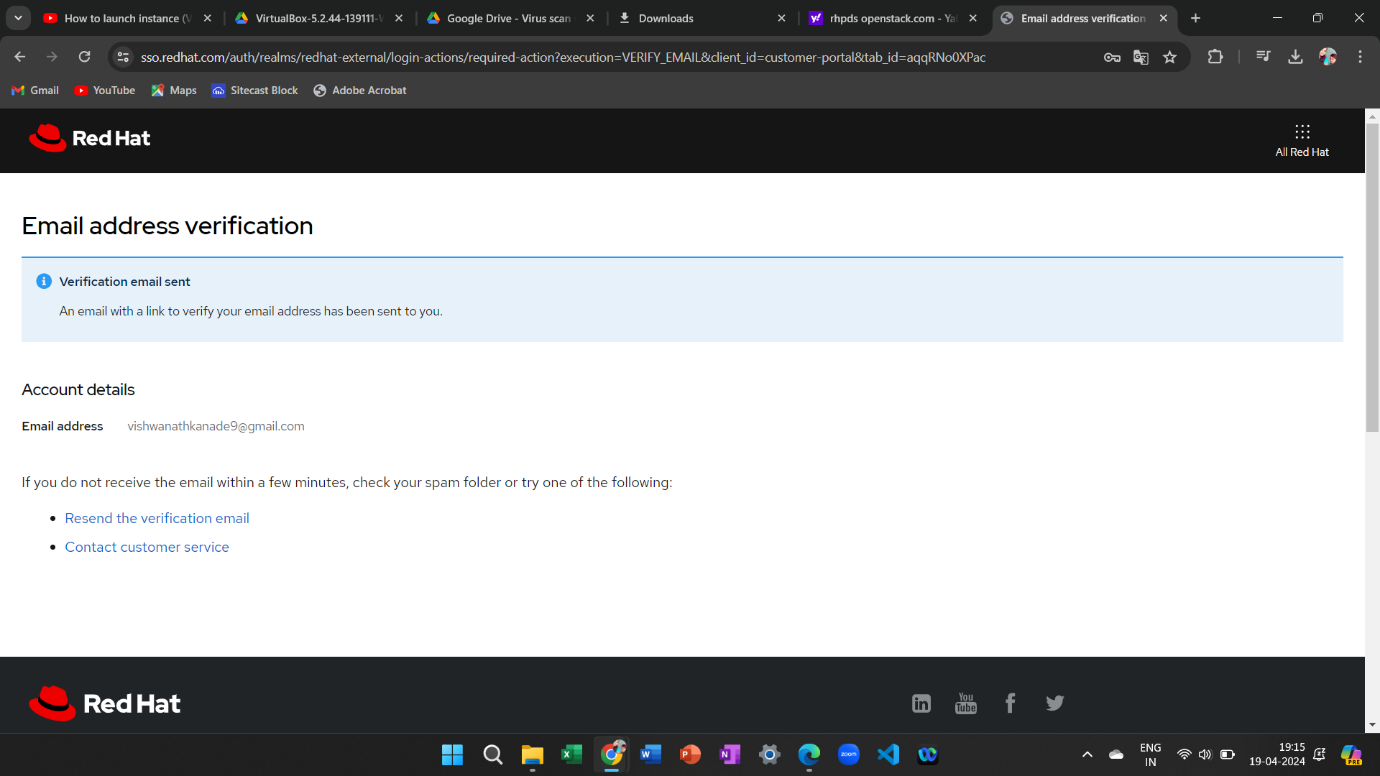
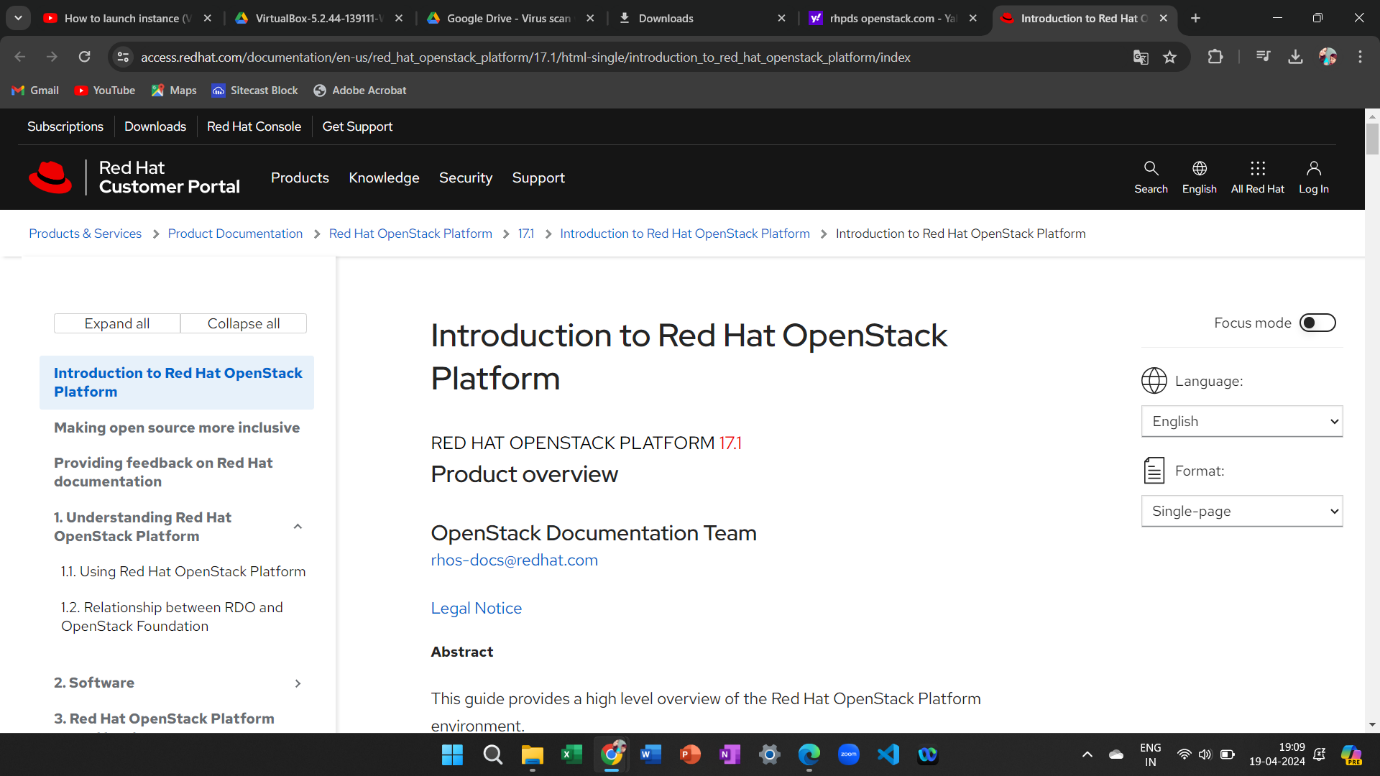


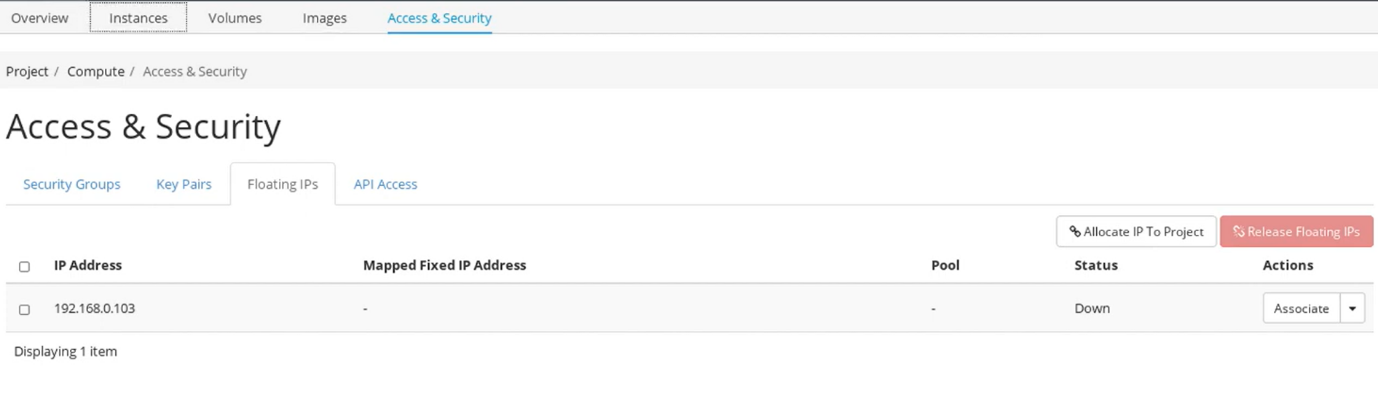


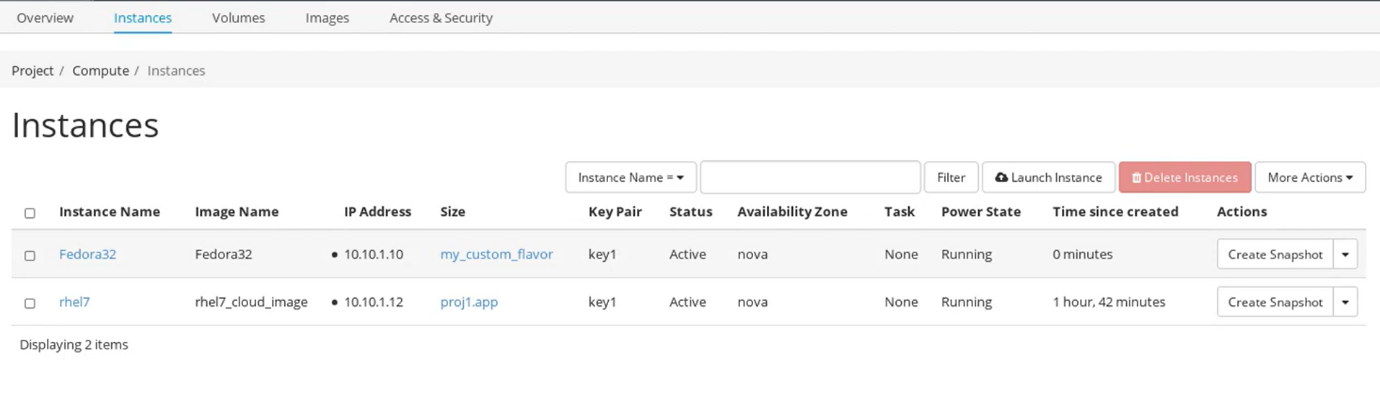


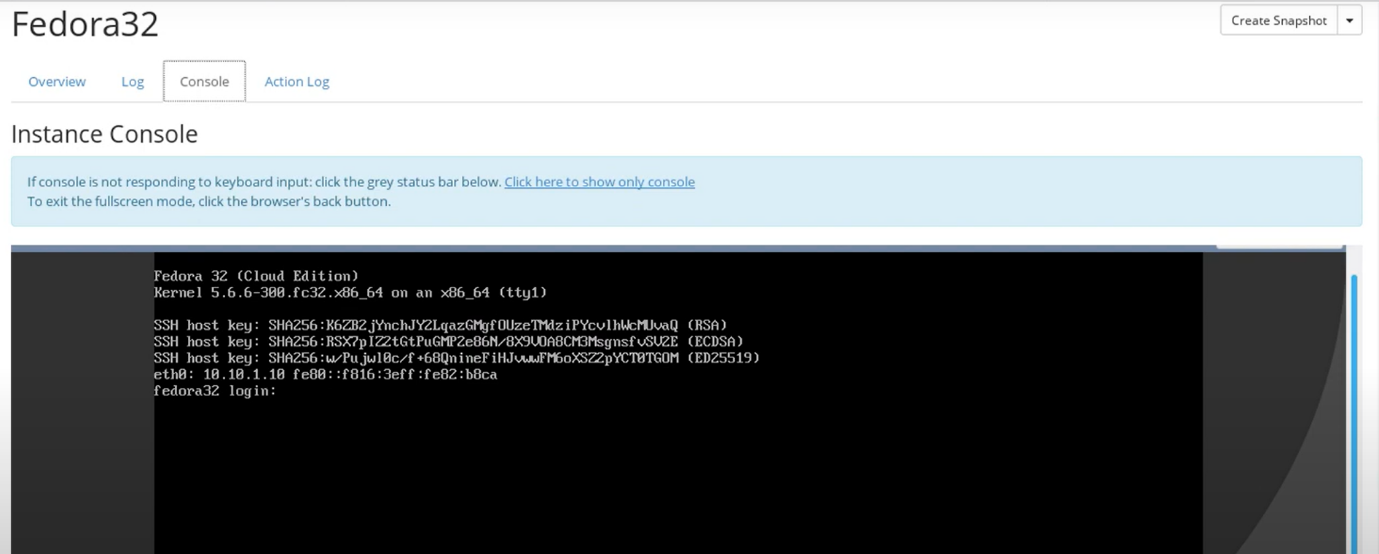
Assignment no.: 6

Procedure Creating Virtual Machine on OpenStack:









Assignment No.: 3&4

Code:

SJF\_Schedular.java

package <package\_name>;

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

//import utils.Constants;

//import utils.DatacenterCreator;

//import utils.GenerateMatrices;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

public class SJF\_Scheduler {

private static List<Cloudlet> cloudletList;

private static List<Vm> vmList;

private static Datacenter[] datacenter;

private static double[][] commMatrix;

private static double[][] execMatrix;

private static List<Vm> createVM(int userId, int vms) {

//Creates a container to store VMs. This list is passed to the broker later

LinkedList<Vm> list = new LinkedList<Vm>();

//VM Parameters

long size = 10000; //image size (MB)

int ram = 512; //vm memory (MB)

int mips = 250;

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create VMs

Vm[] vm = new Vm[vms];

for (int i = 0; i < vms; i++) {

vm[i] = new Vm(datacenter[i].getId(), userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());

list.add(vm[i]);

}

return list;

}

private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift) {

// Creates a container to store Cloudlets

LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

//cloudlet parameters

long fileSize = 300;

long outputSize = 300;

int pesNumber = 1;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet[] cloudlet = new Cloudlet[cloudlets];

for (int i = 0; i < cloudlets; i++) {

int dcId = (int) (Math.random() \* Constants.NO\_OF\_DATA\_CENTERS);

long length = (long) (1e3 \* (commMatrix[i][dcId] + execMatrix[i][dcId]));

cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

// setting the owner of these Cloudlets

cloudlet[i].setUserId(userId);

cloudlet[i].setVmId(dcId + 2);

list.add(cloudlet[i]);

}return list; }

public static void main(String[] args) {

Log.printLine("Starting SJF Scheduler...");

new GenerateMatrices();

execMatrix = GenerateMatrices.getExecMatrix();

commMatrix = GenerateMatrices.getCommMatrix();

try {

int num\_user = 1; // number of grid users

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

datacenter = new Datacenter[Constants.NO\_OF\_DATA\_CENTERS];

for (int i = 0; i < Constants.NO\_OF\_DATA\_CENTERS; i++) {

datacenter[i] = DatacenterCreator.createDatacenter("Datacenter\_" + i);

}

//Third step: Create Broker

SJFDatacenterBroker broker = createBroker("Broker\_0");

int brokerId = broker.getId();

//Fourth step: Create VMs and Cloudlets and send them to broker

vmList = createVM(brokerId, Constants.NO\_OF\_DATA\_CENTERS);

cloudletList = createCloudlet(brokerId, Constants.NO\_OF\_TASKS, 0);

broker.submitVmList(vmList);

broker.submitCloudletList(cloudletList);

// Fifth step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

List<Cloudlet> newList = broker.getCloudletReceivedList();

//newList.addAll(globalBroker.getBroker().getCloudletReceivedList());

CloudSim.stopSimulation();

printCloudletList(newList);

Log.printLine(SJF\_Scheduler.class.getName() + " finished!");

} catch (Exception e) {

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}}

private static SJFDatacenterBroker createBroker(String name) throws Exception {

return new SJFDatacenterBroker(name);

}

private static void printCloudletList(List<Cloudlet> list) {

int size = list.size();

Cloudlet cloudlet;

String indent = " ";

Log.printLine();

Log.printLine("========== OUTPUT ==========");

Log.printLine("Cloudlet ID" + indent + "STATUS" +

indent + "Data center ID" +

indent + "VM ID" +

indent + indent + "Time" +

indent + "Start Time" +

indent + "Finish Time" +

indent + "Waiting Time");

DecimalFormat dft = new DecimalFormat("###.##");

dft.setMinimumIntegerDigits(2);

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(indent + dft.format(cloudlet.getCloudletId()) + indent + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {

Log.print("SUCCESS");

Log.printLine(indent + indent + dft.format(cloudlet.getResourceId()) +

indent + indent + indent + dft.format(cloudlet.getVmId()) +

indent + indent + dft.format(cloudlet.getActualCPUTime()) +

indent + indent + dft.format(cloudlet.getExecStartTime()) +

indent + indent + indent + dft.format(cloudlet.getFinishTime())+

indent + indent + indent + dft.format(cloudlet.getWaitingTime()));}}

double makespan = calcMakespan(list);

Log.printLine("Makespan using SJF: " + makespan);

}

private static double calcMakespan(List<Cloudlet> list) {

double makespan = 0; double[] dcWorkingTime = new double[Constants.NO\_OF\_DATA\_CENTERS];

for (int i = 0; i < Constants.NO\_OF\_TASKS; i++) {

int dcId = list.get(i).getVmId() % Constants.NO\_OF\_DATA\_CENTERS;

if (dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];

dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];

makespan = Math.max(makespan, dcWorkingTime[dcId]); }

return makespan;

}

}

**Output:**

