**VENNELA G**

**20BDS0146**

**PRINCIPLES OF CLOUD COMPUTING**

**LAB ASSESSMENT-3**

# Build and deploy a Flask application using Docker on Ubuntu 18.04. The sample application displays about your short profile upon calling a python method profile() and displays information about your home town upon calling a python method hometown() on the browser.

**App.py:**

import os

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def main():

return "Welcome!"

@app.route('/profile')

def hello():

return 'Vennela G - 20BDS0146'

@app.route('/hometown')

def home():

return 'I Live in AP'

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8080)

**Dockerfile:**

FROM python:3.10

RUN python3 -m pip install flask

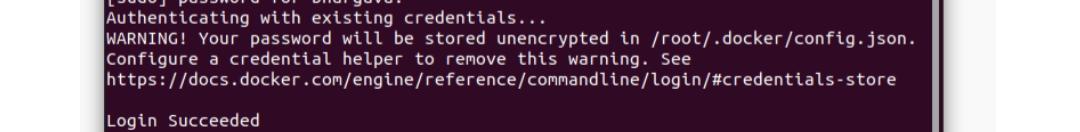
RUN mkdir /app

WORKDIR /app

COPY app.py /opt/app.py

ENV FLASK\_APP=/opt/app.py

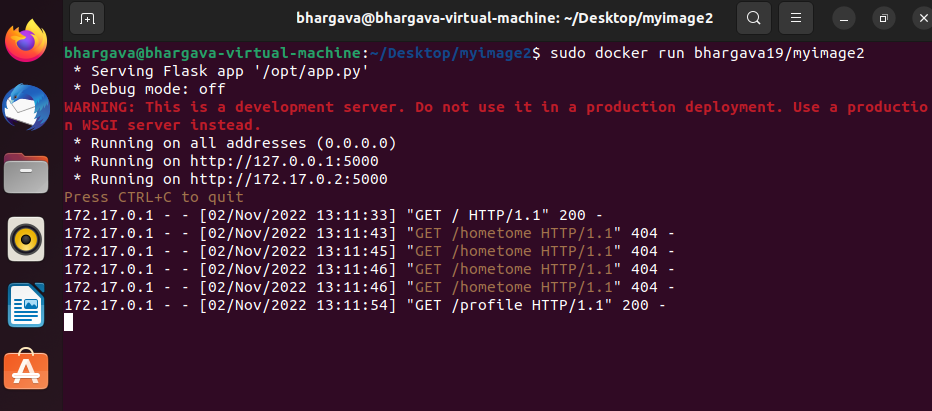
CMD ["flask", "run", "--host=0.0.0.0"]





Text

Description automatically generated



package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.DatacenterCharacteristics;

import org.cloudbus.cloudsim.Host;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.Pe;

import org.cloudbus.cloudsim.Storage;

import org.cloudbus.cloudsim.UtilizationModel;

import org.cloudbus.cloudsim.UtilizationModelFull;

import org.cloudbus.cloudsim.Vm;

import org.cloudbus.cloudsim.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerTimeShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

public class CloudSimExample1 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList;

/\*\* The vmlist. \*/

private static List<Vm> vmlist;

/\*\*

\* Creates main() to run this example.

\*

\* @param args the args

\*/

public static void main(String[] args) {

Log.printLine("Starting CloudSimExample1...");

try {

// First step: Initialize the CloudSim package. It should be called

// before creating any entities.

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

Calendar calendar = Calendar.getInstance();

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

// Initialize the CloudSim library

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

// Second step: Create Datacenters

// Datacenters are the resource providers in CloudSim. We need at

// list one of them to run a CloudSim simulation

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

// Third step: Create Broker

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

// Fourth step: Create one virtual machine

vmlist = new ArrayList<Vm>();

// VM description

// VM description

int vmid1 = 1,vmid2 = 2;

int mips = 500;

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

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

long bw = 1000;

int pesNumber = 1; // number of cpus

String vmm = "Xen"; // VMM name

// create VM

Vm vm1 = new Vm(vmid1, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

Vm vm2 = new Vm(vmid2, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

// add the VM to the vmList

vmlist.add(vm1);

vmlist.add(vm2);

// submit vm list to the broker

broker.submitVmList(vmlist);

// Fifth step: Create one Cloudlet

cloudletList = new ArrayList<Cloudlet>();

// Cloudlet properties

int id = 0;

long length = 400000;

long fileSize = 300;

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet1.setUserId(brokerId);

cloudlet1.setVmId(vmid1);

id++;

Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet2.setUserId(brokerId);

cloudlet2.setVmId(vmid1);

id++;

Cloudlet cloudlet3 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet3.setUserId(brokerId);

cloudlet3.setVmId(vmid2);

id++;

Cloudlet cloudlet4 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet4.setUserId(brokerId);

cloudlet4.setVmId(vmid2);

// add the cloudlet to the list

cloudletList.add(cloudlet1);

cloudletList.add(cloudlet2);

cloudletList.add(cloudlet3);

cloudletList.add(cloudlet4);

// submit cloudlet list to the broker

broker.submitCloudletList(cloudletList);

// Sixth step: Starts the simulation

CloudSim.startSimulation();

CloudSim.stopSimulation();

//Final step: Print results when simulation is over

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

printCloudletList(newList);

// Print the debt of each user to each datacenter

datacenter0.printDebts();

Log.printLine("CloudSimExample1 finished!");

} catch (Exception e) {

e.printStackTrace();

Log.printLine("Unwanted errors happen");

}

}

/\*\*

\* Creates the datacenter.

\*

\* @param name the name

\*

\* @return the datacenter

\*/

private static Datacenter createDatacenter(String name) {

// Here are the steps needed to create a PowerDatacenter:

// 1. We need to create a list to store

// our machine

List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.

// In this example, it will have only one core.

List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into a list.

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

// 4. Create Host with its id and list of PEs and add them to the list

// of machines

int hostId = 0;

int ram = 2048; // host memory (MB)

long storage = 1000000; // host storage

int bw = 10000;

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

)

); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

String vmm = "Xen";

double time\_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.001; // the cost of using storage in this

// resource

double costPerBw = 0.0; // the cost of using bw in this resource

LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding SAN

// devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, cost, costPerMem,

costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

// We strongly encourage users to develop their own broker policies, to

// submit vms and cloudlets according

// to the specific rules of the simulated scenario

/\*\*

\* Creates the broker.

\*

\* @return the datacenter broker

\*/

private static DatacenterBroker createBroker() {

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker("Broker");

} catch (Exception e) {

e.printStackTrace();

return null;

}

return broker;

}

/\*\*

\* Prints the Cloudlet objects.

\*

\* @param list list of Cloudlets

\*/

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 + "Time" + indent

+ "Start Time" + indent + "Finish Time");

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

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

cloudlet = list.get(i);

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

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

Log.print("SUCCESS");

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

+ indent + indent + indent + cloudlet.getVmId()

+ indent + indent

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

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

+ indent + indent

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

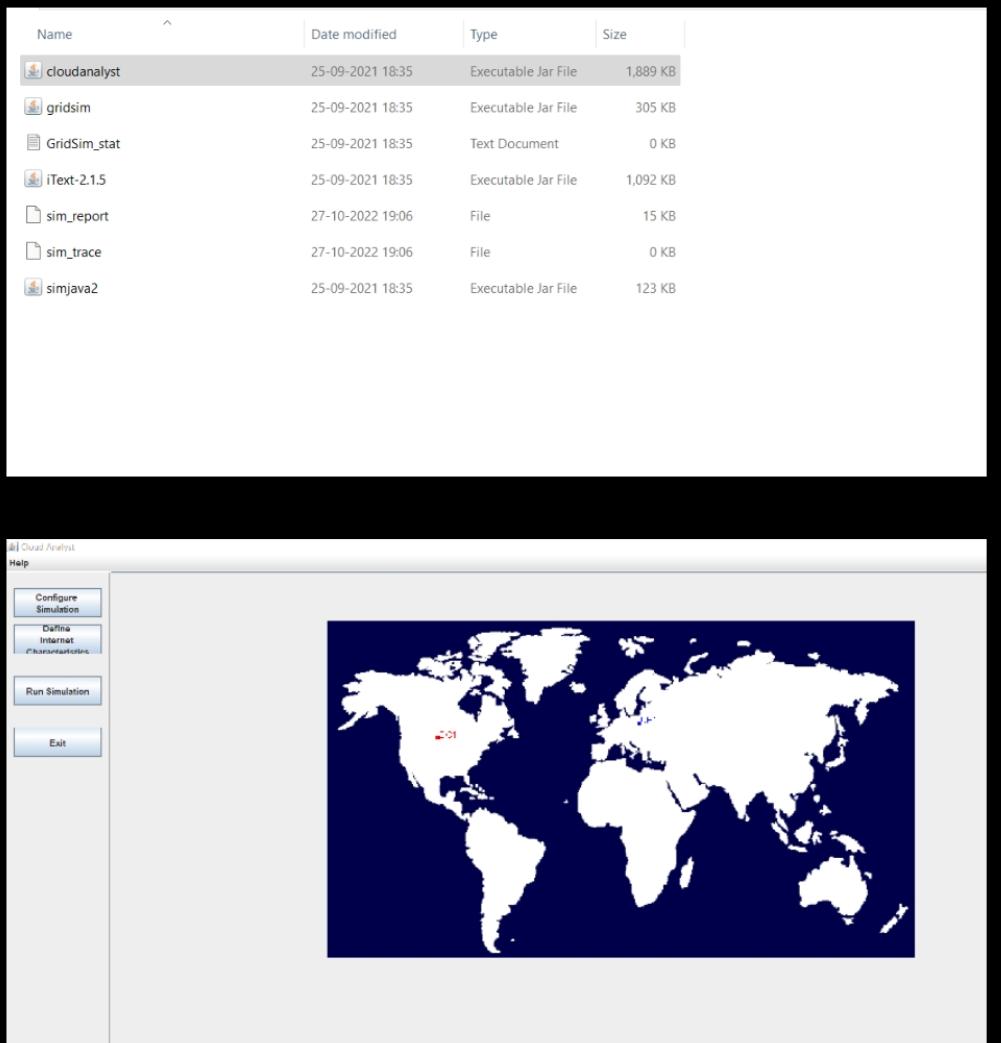
}

}

}

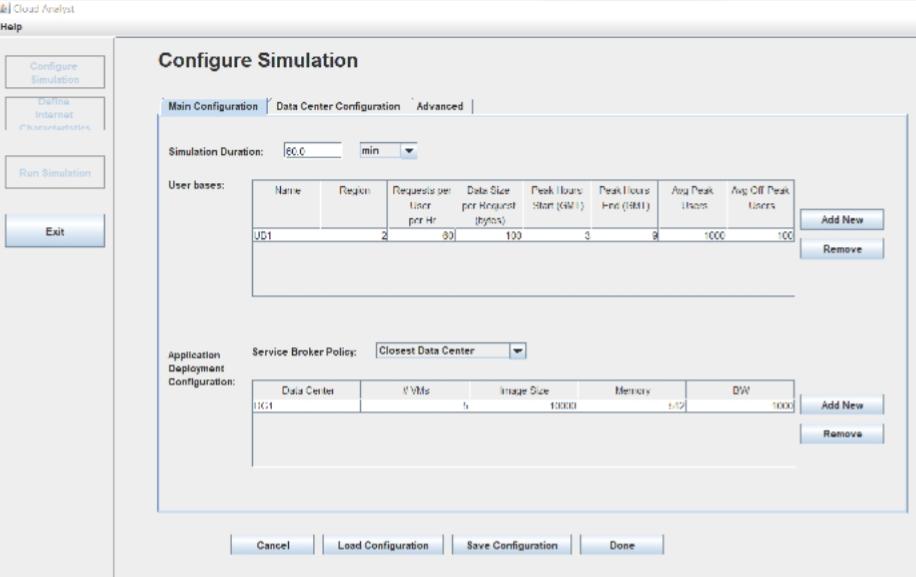
}

Open the Cloud Analyst executable file from the downloaded Github repository:

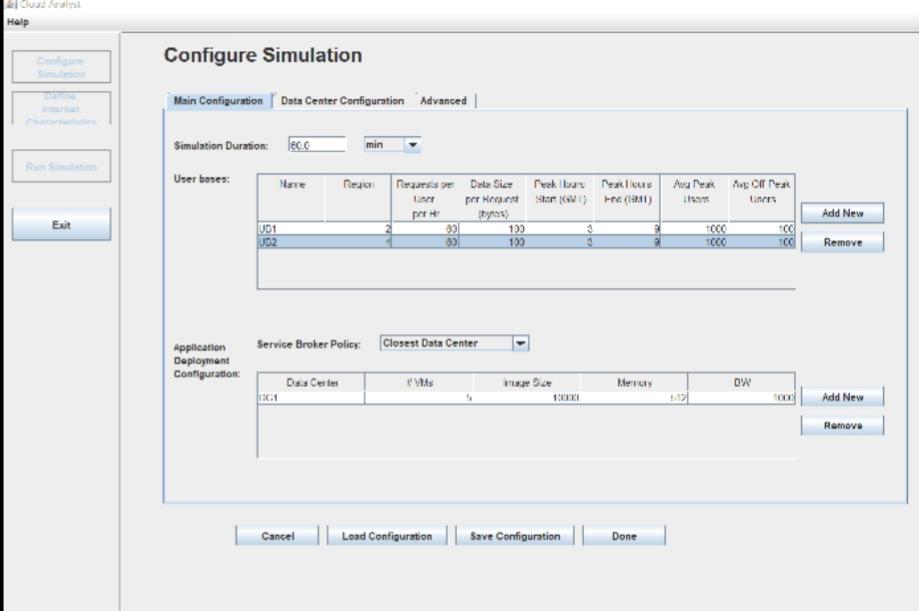


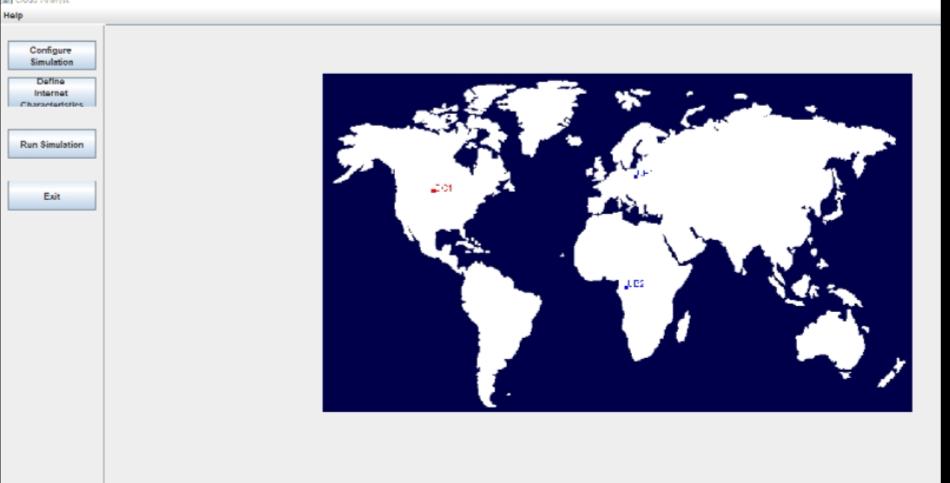
Add a new User base at a different region for understanding the difference provided by the Load Balancing Algorithms.

- Click on Configure Simulation:

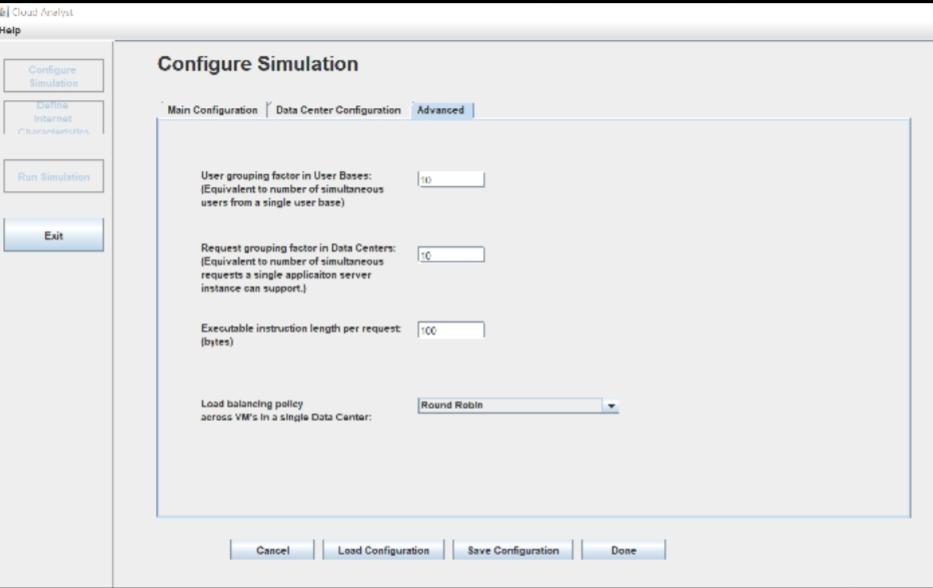


- Add new User base and change the region along with any other property if wanted

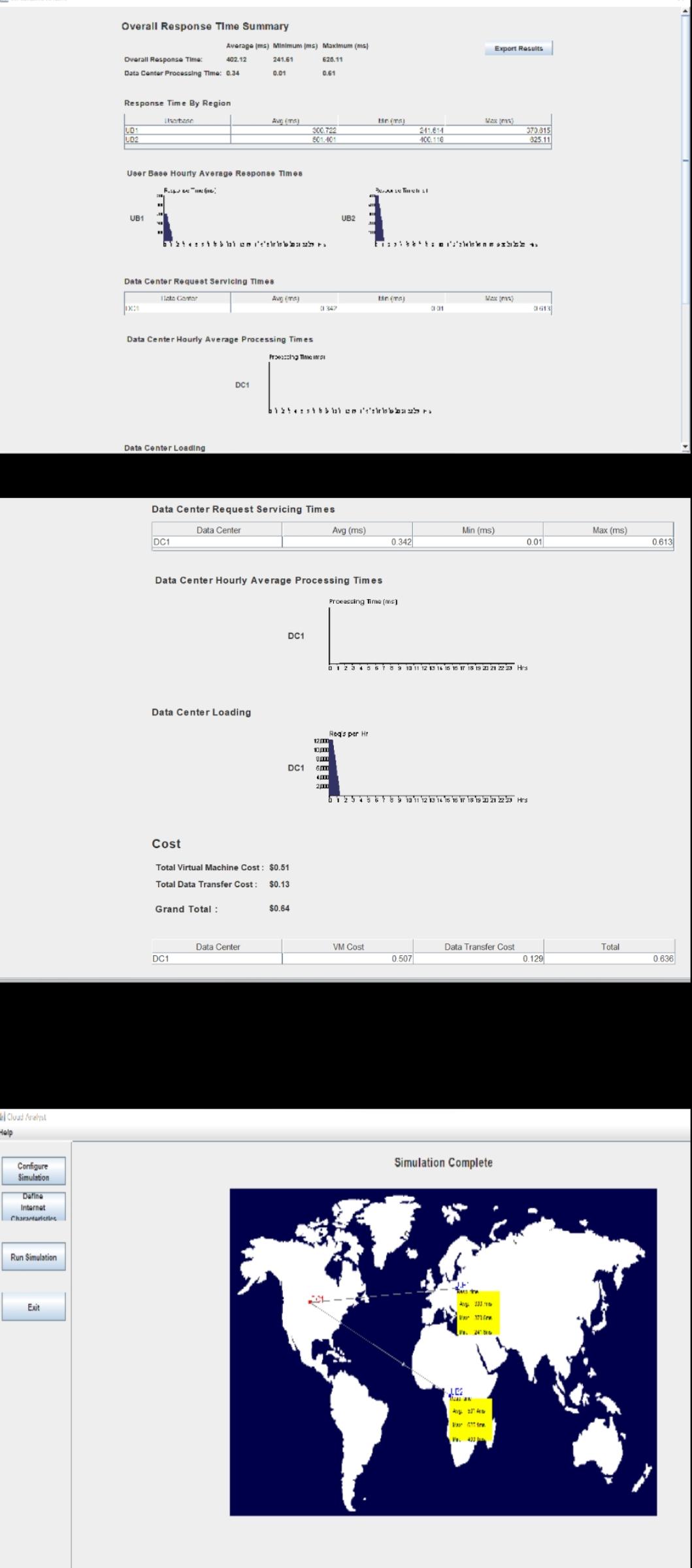




Now click on configure simulation > click on Advanced Tab > select the Load Balancing algorithm to Round Robin:



Click on Run Simulation and we observe the simulation result:



Now click on the Configure Simulation > Advanced Tab > and then in the Load Balancing Algorithm select ‘Equally Spread Current Execution Load’ algorithm and click done



We observe difference in the response time for each of the user base due to change in the load balancing algorithm.

