

PRACTICAL 1

AIM:

Introduction to CloudSim tool.

Introduce Installation steps of CloudSim on NetBeans.

THEORY:

CloudSim is a simulation toolkit that supports the modeling and simulation of the core functionality of cloud, like job/task queue, processing of events, creation of cloud entities(datacenter, datacenter brokers, etc), communication between different entities, implementation of broker policies, etc. This toolkit allows to:

- Test application services in a repeatable and controllable environment.
- Tune the system bottlenecks before deploying apps in an actual cloud.
- Experiment with different workload mix and resource performance scenarios on simulated infrastructure for developing and testing adaptive application provisioning techniques

Core features of CloudSim are:

- The Support of modeling and simulation of large scale computing environment as federated cloud data centers, virtualized server hosts, with customizable policies for provisioning host resources to virtual machines and energy-aware computational resources
- It is a self-contained platform for modeling cloud's service brokers, provisioning, and allocation policies.
- It supports the simulation of network connections among simulated system elements.
- Support for simulation of federated cloud environment, that inter-networks resources from both private and public domains.
- Availability of a virtualization engine that aids in the creation and management of multiple independent and co-hosted virtual services on a data center node.
- Flexibility to switch between space shared and time shared allocation of processing cores to virtualized services.

INSTALLATION-STEPS FOR NETBEANS:

You need to have a setup file of the NetBeans JAVA into your setup.

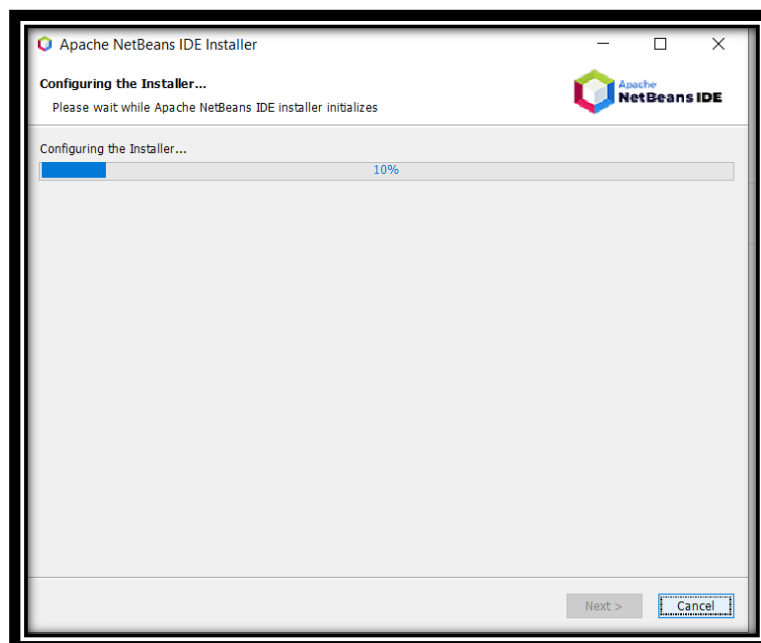
If you didn't have the setup you can download from the following link:
https://netbeans.org/images_www/v6/download/community/8.2



You can download any type of setup as per your requirements from the above mention web page.

Right-click on the setup or you can Double-Click on the setup by using the mouse.

Click on the next option.

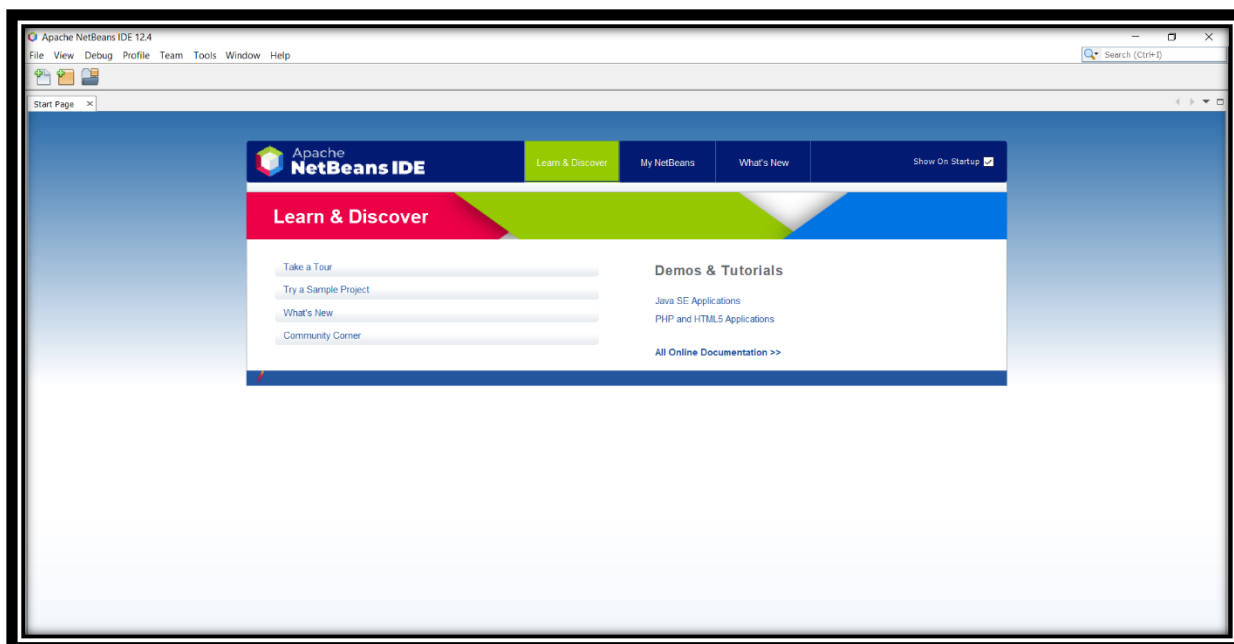


Click on the “Install” button.

Wait for the while till the time the setup is properly Installed into the Computer

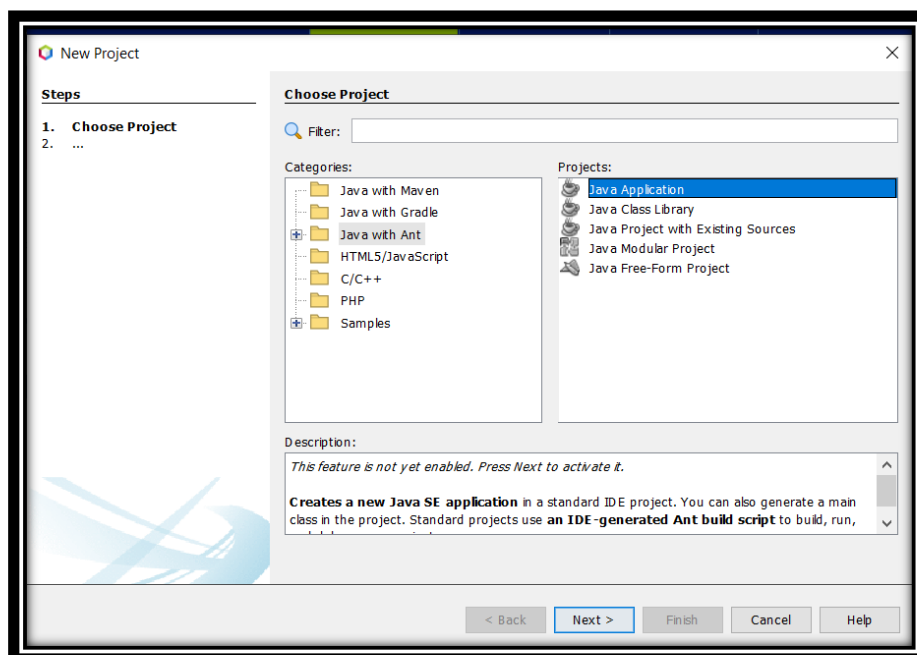
After complication of the setup you can click on the “Finish” button or you can also register the Software, for Further Assistance because it is a Free Software.

Now you can start the NetBeans for further use.



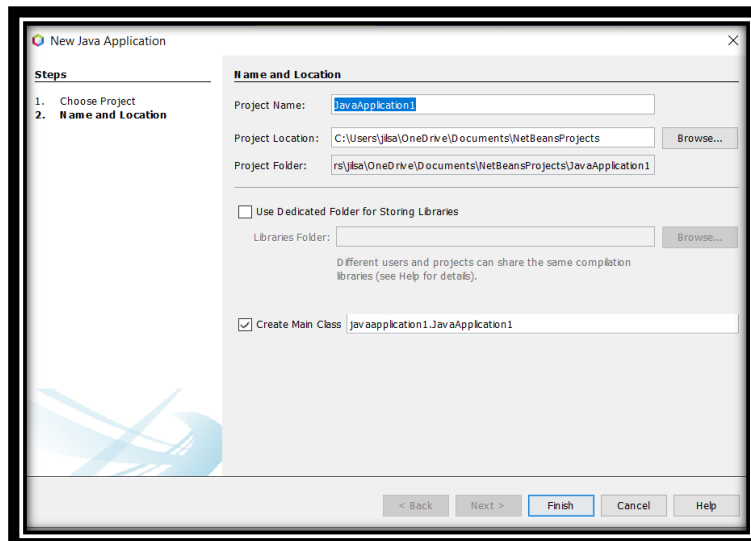
INSTALLATION-STEPS FOR CLOUDSIM:

Open Netbeans, Go to file->>new project.

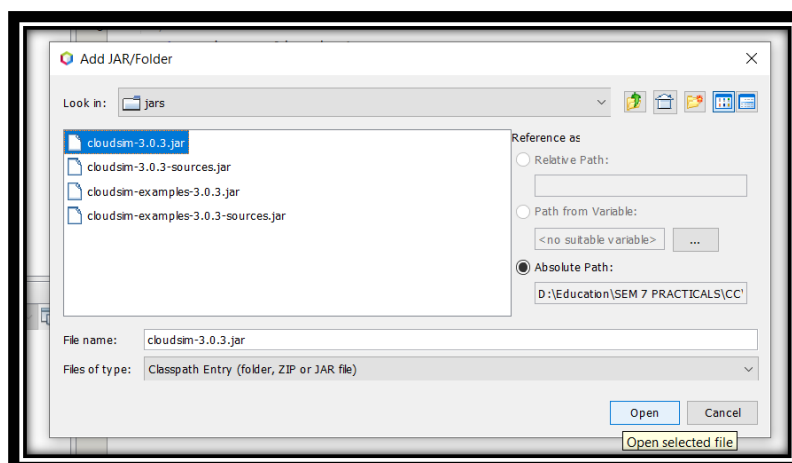
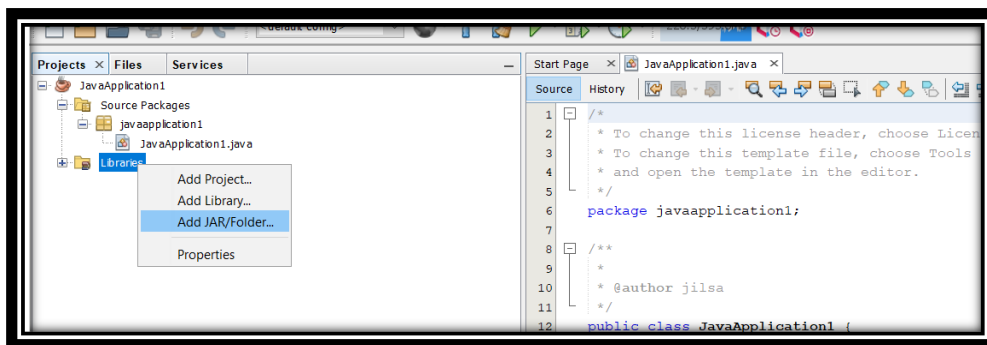


select “Java with Ant” folder then select first option Java Application, Press next

Now give a name to the project as you wish.



Go to library, right click on it, a menu will come, click on “Add jars/Folders”



That’s how cloudsim can be installed in Netbeans.

CONCLUSION:

In this practical, we learnt about Netbeans and Cloudsim. We installed both the tools in our system.

PRACTICAL 2

AIM:

Understanding of basic CloudSim Examples.

- (1) Write a program in cloudsim using NetBeans IDE to create a datacenter with one host and run four cloudlet on it.
- (2) Write a program in cloudsim using NetBeans IDE to create a datacenter with three hosts and run three cloudlets on it.
- (3) Write a program in cloudsim using NetBeans IDE to create a five datacenters with five hosts and run cloudlets of five users on them.

CODE:

1st:

```
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 Prac2_A {

    /**
     * @param args the command line arguments
     */

    private static List<Cloudlet> cloudletList;

    private static List<Vm> vmList;

    public static void main(String[] args)
    {
        Log.println("Starting Cloudsim ...");

        try
        {
            int num_user = 1;

            Calendar calendar = Calendar.getInstance();

            boolean trace_flag = false;

            CloudSim.init(num_user, calendar, trace_flag);

            Datacenter datacenter0 = createDatacenter("Datacenter_0");
```

```
DatacenterBroker broker = createBroker("Broker");  
int brokerId = broker.getId();  
  
vmList = new ArrayList<Vm>();  
  
int vmid = 0;  
int mips = 1000;  
long size = 10000;  
int ram = 512;  
long bw = 1000;  
int pesNumber = 1;  
String vmm = "Xen";  
  
Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new  
CloudletSchedulerTimeShared());  
  
vmList.add(vm);  
  
broker.submitVmList(vmList);  
  
cloudletList = new ArrayList<Cloudlet>();  
  
int id = 0;  
long length = 400000;  
long fileSize = 300;  
long outputSize = 300;  
UtilizationModel utilizationModel = new UtilizationModelFull();  
  
Cloudlet cloudlet0 = new Cloudlet(id++, length, pesNumber, fileSize, outputSize,  
utilizationModel, utilizationModel, utilizationModel);
```

```
cloudlet0.setUserId(brokerId);  
cloudlet0.setVmId(vmid);  
cloudletList.add(cloudlet0);
```

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

```
cloudlet1.setUserId(brokerId);  
cloudlet1.setVmId(vmid);  
cloudletList.add(cloudlet1);
```

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

```
cloudlet2.setUserId(brokerId);  
cloudlet2.setVmId(vmid);  
cloudletList.add(cloudlet2);
```

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

```
cloudlet3.setUserId(brokerId);  
cloudlet3.setVmId(vmid);  
cloudletList.add(cloudlet3);
```

```
broker.submitCloudletList(cloudletList);
```

```
CloudSim.startSimulation();
```

```
CloudSim.stopSimulation();
```

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



```
        Log.println("CloudSim example finished!");
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
        Log.println("Unwanted error occurred");
    }
}

public static Datacenter createDatacenter(String name)
{
    List<Host> hostList = new ArrayList<Host>();

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

    int mips = 1000;

    peList.add(new Pe(0, new PeProvisionerSimple(mips)));

    int hostId = 0;
    int ram = 2048;
    long storage = 1000000;
    int bw = 10000;

    hostList.add(new Host(hostId, new RamProvisionerSimple(ram), new
    BwProvisionerSimple(bw), storage, peList, new VmSchedulerTimeShared(peList)));

    String arch = "x86";
    String os = "Linux";
    String vmm = "Xen";
```

```
double time_zone = 10.0;
double cost = 3.0;
double costPerMem = 0.05;
double costPerStorage = 0.001;
double costPerBw = 0.0;
```

```
LinkedList<Storage> storageList = new LinkedList<Storage>();
```

```
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os,
vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
```

```
Datacenter datacenter = null;
try
{
    datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
}
catch(Exception ex) {
    ex.printStackTrace();
}

return datacenter;
}
```

```
private static DatacenterBroker createBroker(String name) {
    DatacenterBroker broker = null;
    try
    {
        broker = new DatacenterBroker(name);
    }
}
```

```
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
        return broker;
    }
```

```
private static void printCloudletList(List<Cloudlet> list) {
    int size = list.size();
    Cloudlet cloudlet;

    String indent = "  ";
    Log.println();
    Log.println("===== OUTPUT =====");
    Log.println("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");

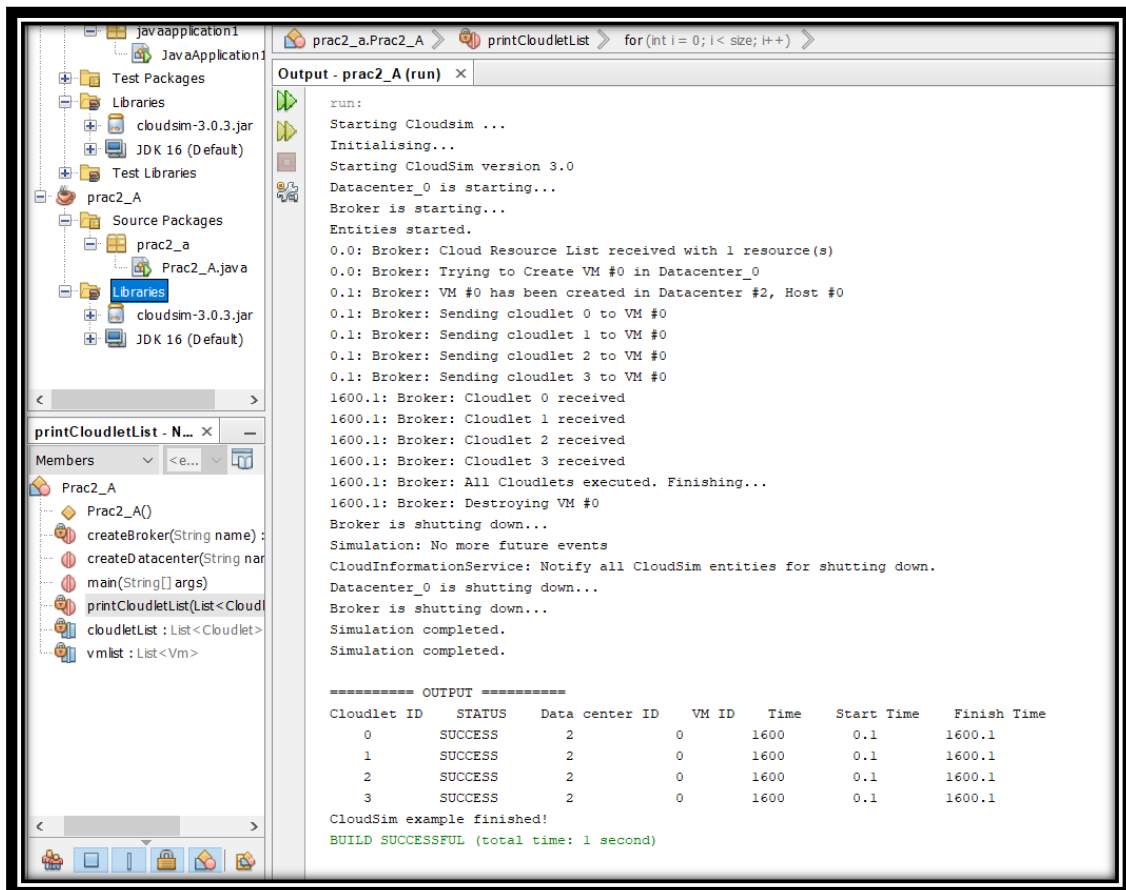
            Log.println(indent + indent + cloudlet.getResourceId()
                + indent + indent + indent + cloudlet.getVmId()
                + indent + indent

```

```

    + dft.format(cloudlet.getActualCPUTime()) +
indent
+ indent +
dft.format(cloudlet.getExecStartTime())
+ indent + indent
+ dft.format(cloudlet.getFinishTime());
}
}
}
```

OUTPUT 1:



2nd:

```
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 Prac2_B {

    /**
     * @param args the command line arguments
     */

    private static List<Cloudlet> cloudletList;
```

```
private static List<Vm> vmlist;

public static void main(String[] args) {
    // TODO code application logic here
    Log.println("Starting CloudSim ...");

    try
    {
        int num_user = 1;
        Calendar calendar = Calendar.getInstance();
        boolean trace_flag = false;

        CloudSim.init(num_user, calendar, trace_flag);

        Datacenter datacenter0 = createDatacenter("Datacenter_0");

        DatacenterBroker broker = createBroker("broker1");
        int brokerId = broker.getId();

        vmlist = new ArrayList<Vm>();

        int vmid = 0;
        int mips = 250;
        long size = 10000;
        int ram = 512;
        //int ram = 1024;
        long bw = 1000;
        int pesNumber = 1;
        String vmm = "Xen";
```

```
//Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new  
CloudletSchedulerTimeShared());
```

```
Vm vm0 = new Vm(vmid++, brokerId, mips, pesNumber, ram, bw, size, vmm, new  
CloudletSchedulerTimeShared());
```

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

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

```
//vmList.add(vm);
```

```
vmList.add(vm0);
```

```
vmList.add(vm1);
```

```
vmList.add(vm2);
```

```
broker.submitVmList(vmList);
```

```
cloudletList = new ArrayList<Cloudlet>();
```

```
int id = 0;
```

```
long length = 400000;
```

```
long fileSize = 300;
```

```
long outputSize = 300;
```

```
UtilizationModel utilizationModel = new UtilizationModelFull();
```

```
Cloudlet cloudlet0 = new Cloudlet(id++, length, pesNumber, fileSize, outputSize,  
utilizationModel, utilizationModel, utilizationModel);
```

```
cloudlet0.setUserId(brokerId);
```

```
cloudlet0.setVmId(0);
```

```
cloudletList.add(cloudlet0);
```

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

```
cloudlet1.setUserId(brokerId);
```

```
cloudlet1.setVmId(1);  
cloudletList.add(cloudlet1);
```

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

```
cloudlet2.setUserId(brokerId);  
cloudlet2.setVmId(2);  
cloudletList.add(cloudlet2);
```

```
broker.submitCloudletList(cloudletList);
```

```
CloudSim.startSimulation();
```

```
CloudSim.stopSimulation();
```

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

```
Log.println("Cloudsim finished");  
}  
catch(Exception ex)  
{  
    ex.printStackTrace();  
    Log.println("Unwanted error occurred");  
}  
}
```

```
public static Datacenter createDatacenter(String name)  
{  
    List<Host> hostList = new ArrayList<Host>();
```



```
List<Pe> peList0 = new ArrayList<Pe>();
List<Pe> peList1 = new ArrayList<Pe>();
List<Pe> peList2 = new ArrayList<Pe>();

int mips = 1000;

peList0.add(new Pe(0, new PeProvisionerSimple(mips)));
peList1.add(new Pe(0, new PeProvisionerSimple(mips)));
peList2.add(new Pe(0, new PeProvisionerSimple(mips)));

int hostId = 0;
int ram = 2048;
long storage = 1000000;
int bw = 10000;

hostList.add(new Host(hostId++, new RamProvisionerSimple(ram), new
BwProvisionerSimple(bw), storage, peList0, new VmSchedulerTimeShared(peList0)));

hostList.add(new Host(hostId++, new RamProvisionerSimple(ram), new
BwProvisionerSimple(bw), storage, peList1, new VmSchedulerTimeShared(peList1)));

hostList.add(new Host(hostId++, new RamProvisionerSimple(ram), new
BwProvisionerSimple(bw), storage, peList2, new VmSchedulerTimeShared(peList2)));

String arch = "x86";
String os = "Linux";
String vmm = "Xen";
double time_zone = 10.0;
double cost = 3.0;
double costPerMem = 0.05;
double costPerStorage = 0.001;
double costPerBw = 0.0;
```

```
LinkedList<Storage> storageList = new LinkedList<Storage>();

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os,
vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

Datacenter datacenter = null;

try
{
    datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
}
catch(Exception ex)
{
    ex.printStackTrace();
    return null;
}

return datacenter;
}

public static DatacenterBroker createBroker(String name)
{
    DatacenterBroker broker = null;

    try
    {
        broker = new DatacenterBroker(name);
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
        return null;
    }
}
```

```

        return broker;
    }

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

        int size = list.size();

        Cloudlet cloudlet;

        String indent = "  ";

        Log.println();

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

        Log.println("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.println(indent + indent + cloudlet.getResourceId()
                    + indent + indent + indent + cloudlet.getVmId()
                    + indent + indent
                    + dft.format(cloudlet.getActualCPUTime()) +
indent
                    + indent + indent + indent + indent + indent +
dft.format(cloudlet.getExecStartTime())
                    + indent + indent
                    + dft.format(cloudlet.getFinishTime()));
            }
        }
    }

```

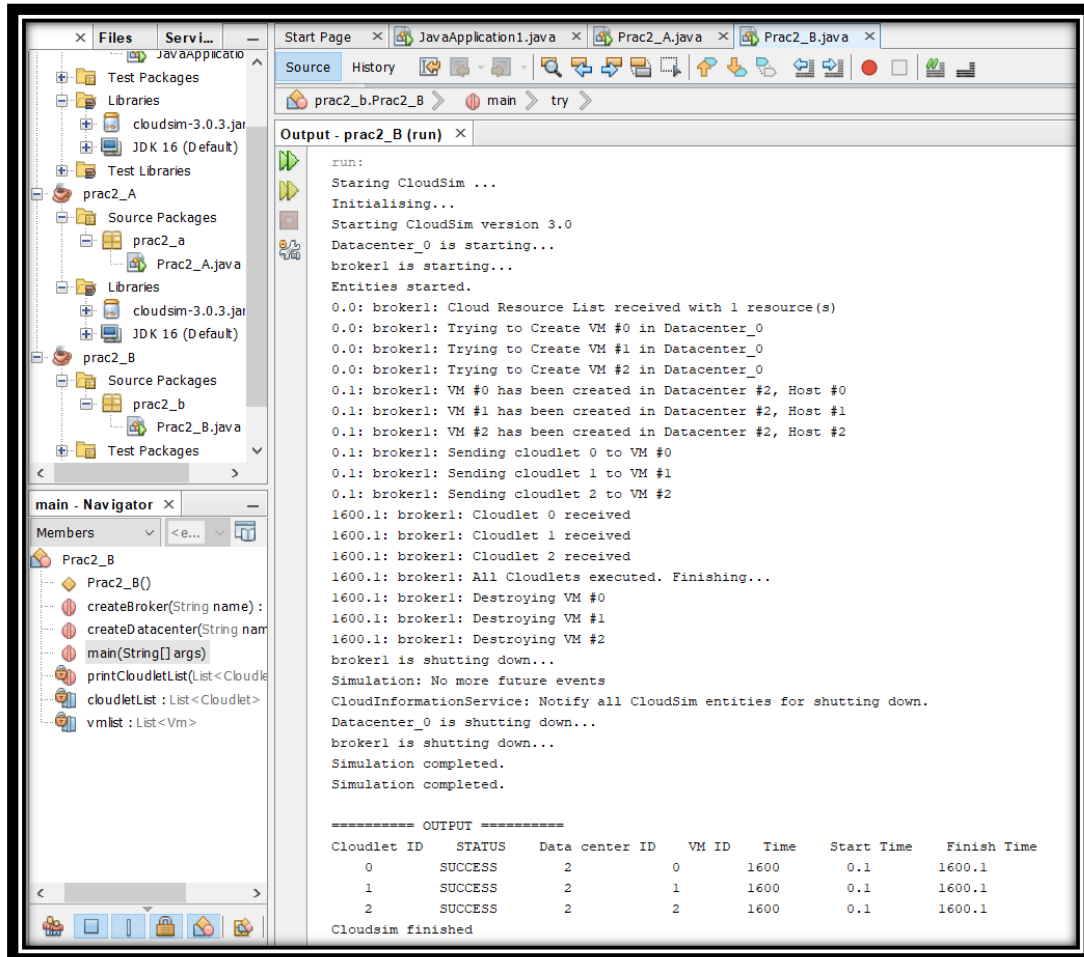
```

    }

}

}

```

OUTPUT 2:

3rd:

```
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.VmSchedulerSpaceShared;
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 Prac2_C {

    /**
     * @param args the command line arguments
     */

    /** The cloudlet lists. */
    private static List<Cloudlet> cloudletList1;
    private static List<Cloudlet> cloudletList2;
    private static List<Cloudlet> cloudletList3;
    private static List<Cloudlet> cloudletList4;
    private static List<Cloudlet> cloudletList5;
```

```
/** The vmlists. */

private static List<Vm> vmlist1;
private static List<Vm> vmlist2;
private static List<Vm> vmlist3;
private static List<Vm> vmlist4;
private static List<Vm> vmlist5;


public static void main(String[] args) {
    // TODO code application logic here
    Log.println("Starting CloudSim ...");

    try {
        // First step: Initialize the CloudSim package. It should be called
        // before creating any entities.

        int num_user = 5; // 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

        @SuppressWarnings("unused")
        Datacenter datacenter0 = createDatacenter("Datacenter_0");

        @SuppressWarnings("unused")
        Datacenter datacenter1 = createDatacenter("Datacenter_1");

        @SuppressWarnings("unused")
```

```
Datacenter datacenter2 = createDatacenter("Datacenter_2");
@SuppressWarnings("unused")
Datacenter datacenter3 = createDatacenter("Datacenter_3");
@SuppressWarnings("unused")
Datacenter datacenter4 = createDatacenter("Datacenter_4");

//Third step: Create Brokers
DatacenterBroker broker1 = createBroker(1);
int brokerId1 = broker1.getId();

DatacenterBroker broker2 = createBroker(2);
int brokerId2 = broker2.getId();

DatacenterBroker broker3 = createBroker(3);
int brokerId3 = broker3.getId();
DatacenterBroker broker4 = createBroker(4);
int brokerId4 = broker4.getId();
DatacenterBroker broker5 = createBroker(5);
int brokerId5 = broker5.getId();

//Fourth step: Create one virtual machine for each broker/user
vmList1 = new ArrayList<Vm>();
vmList2 = new ArrayList<Vm>();
vmList3 = new ArrayList<Vm>();
vmList4 = new ArrayList<Vm>();
vmList5 = new ArrayList<Vm>();

//VM description
int vmid = 0;
//int mips = 250;
```

```
int mips = 100;

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 two VMs: the first one belongs to user1

Vm vm1 = new Vm(vmid++, brokerId1, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());


//the second VM: this one belongs to user2

Vm vm2 = new Vm(vmid++, brokerId2, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());


//the third VM: this one belongs to user3

Vm vm3 = new Vm(vmid++, brokerId3, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());


//the forth VM: this one belongs to user4

Vm vm4 = new Vm(vmid++, brokerId4, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());


//the fifth VM: this one belongs to user5

Vm vm5 = new Vm(vmid++, brokerId5, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());


//add the VMs to the vmlists

vmlist1.add(vm1);

vmlist2.add(vm2);

vmlist3.add(vm3);

vmlist4.add(vm4);

vmlist5.add(vm5);
```



```
//submit vm list to the broker  
broker1.submitVmList(vmlist1);  
broker2.submitVmList(vmlist2);  
broker3.submitVmList(vmlist3);  
broker4.submitVmList(vmlist4);  
broker5.submitVmList(vmlist5);
```

```
//Fifth step: Create two Cloudlets  
cloudletList1 = new ArrayList<Cloudlet>();  
cloudletList2 = new ArrayList<Cloudlet>();  
cloudletList3 = new ArrayList<Cloudlet>();  
cloudletList4 = new ArrayList<Cloudlet>();  
cloudletList5 = new ArrayList<Cloudlet>();
```

```
//Cloudlet properties  
int id = 0;  
long length = 40000;  
long fileSize = 300;  
long outputSize = 300;  
UtilizationModel utilizationModel = new UtilizationModelFull();
```

```
Cloudlet cloudlet1 = new Cloudlet(id++, length, pesNumber, fileSize, outputSize,  
utilizationModel, utilizationModel, utilizationModel);  
cloudlet1.setUserId(brokerId1);
```

```
Cloudlet cloudlet2 = new Cloudlet(id++, length, pesNumber, fileSize, outputSize,  
utilizationModel, utilizationModel, utilizationModel);  
cloudlet2.setUserId(brokerId2);
```

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

```
cloudlet3.setUserId(brokerId3);
```

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

```
cloudlet4.setUserId(brokerId4);
```

```
Cloudlet cloudlet5 = new Cloudlet(id++, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
```

```
cloudlet5.setUserId(brokerId5);
```

```
//add the cloudlets to the lists: each cloudlet belongs to one user
```

```
cloudletList1.add(cloudlet1);
```

```
cloudletList2.add(cloudlet2);
```

```
cloudletList3.add(cloudlet3);
```

```
cloudletList4.add(cloudlet4);
```

```
cloudletList5.add(cloudlet5);
```

```
//submit cloudlet list to the brokers
```

```
broker1.submitCloudletList(cloudletList1);
```

```
broker2.submitCloudletList(cloudletList2);
```

```
broker3.submitCloudletList(cloudletList3);
```

```
broker4.submitCloudletList(cloudletList4);
```

```
broker5.submitCloudletList(cloudletList5);
```

```
// Sixth step: Starts the simulation
```

```
CloudSim.startSimulation();
```

```
// Final step: Print results when simulation is over
```

```
List<Cloudlet> newList1 = broker1.getCloudletReceivedList();
```

```
List<Cloudlet> newList2 = broker2.getCloudletReceivedList();
```

```
List<Cloudlet> newList3 = broker3.getCloudletReceivedList();
```

```
List<Cloudlet> newList4 = broker4.getCloudletReceivedList();
List<Cloudlet> newList5 = broker5.getCloudletReceivedList();

CloudSim.stopSimulation();

Log.print("=====> User "+brokerId1+" ");
printCloudletList(newList1);

Log.print("=====> User "+brokerId2+" ");
printCloudletList(newList2);

Log.print("=====> User "+brokerId3+" ");
printCloudletList(newList3);
Log.print("=====> User "+brokerId4+" ");
printCloudletList(newList4);
Log.print("=====> User "+brokerId5+" ");
printCloudletList(newList5);

Log.println("CloudSimExample finished!");
}
catch (Exception e) {
    e.printStackTrace();
    Log.println("The simulation has been terminated due to an unexpected error");
}
}

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> peList0 = new ArrayList<Pe>();
List<Pe> peList1 = new ArrayList<Pe>();
List<Pe> peList2 = new ArrayList<Pe>();
List<Pe> peList3 = new ArrayList<Pe>();
List<Pe> peList4 = new ArrayList<Pe>();

int mips=1000;

// 3. Create PEs and add these into a list.
peList0.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe
id and MIPS Rating
peList1.add(new Pe(1, new PeProvisionerSimple(mips))); // need to store Pe id and
MIPS Rating
peList2.add(new Pe(2, new PeProvisionerSimple(mips))); // need to store Pe id and
MIPS Rating
peList3.add(new Pe(3, new PeProvisionerSimple(mips))); // need to store Pe id and
MIPS Rating
peList4.add(new Pe(4, 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;
```

//in this example, the VMAllocationPolicy in use is SpaceShared. It means that only one VM

//is allowed to run on each Pe. As each Host has only one Pe, only one VM can run on each Host.

```
    hostList.add(
        new Host(
            hostId++,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList0,
            new VmSchedulerSpaceShared(peList0)
        )
    ); // This is our first machine

    hostList.add(
        new Host(
            hostId++,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList1,
            new VmSchedulerSpaceShared(peList1)
        )
    ); // This is our second machine

    hostList.add(
        new Host(
            hostId++,
```

```
        new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList2,
        new VmSchedulerSpaceShared(peList2)
    )
); // This is our third machine
hostList.add(
    new Host(
        hostId++,
        new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList3,
        new VmSchedulerSpaceShared(peList3)
    )
); // This is our forth machine
hostList.add(
    new Host(
        hostId++,
        new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList4,
        new VmSchedulerSpaceShared(peList4)
    )
); // This is our fifth 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

```
private static DatacenterBroker createBroker(int id){
```

```
    DatacenterBroker broker = null;
```

```
    try {
```

```
        broker = new DatacenterBroker("Broker"+id);
```

```
    } 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.println();
```

```
    Log.println("===== OUTPUT =====");
```

```
    Log.println("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.println( 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()));
    }
}

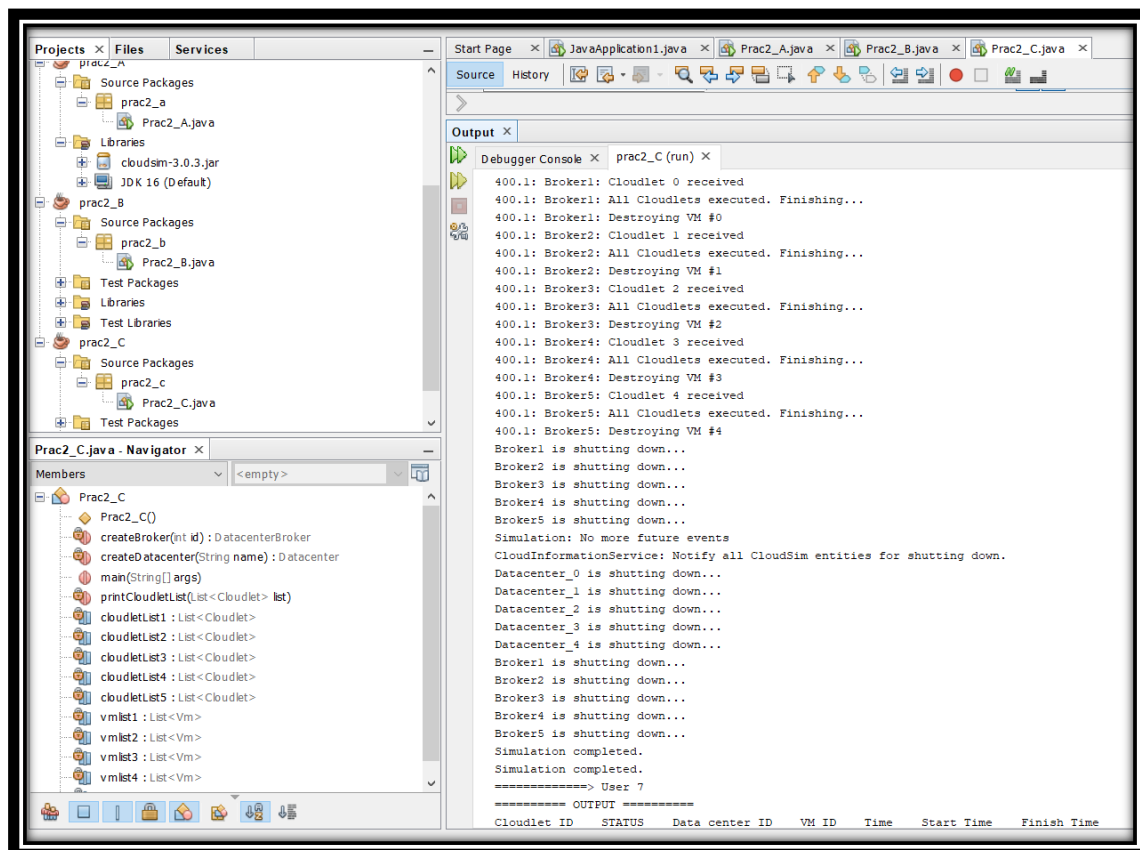
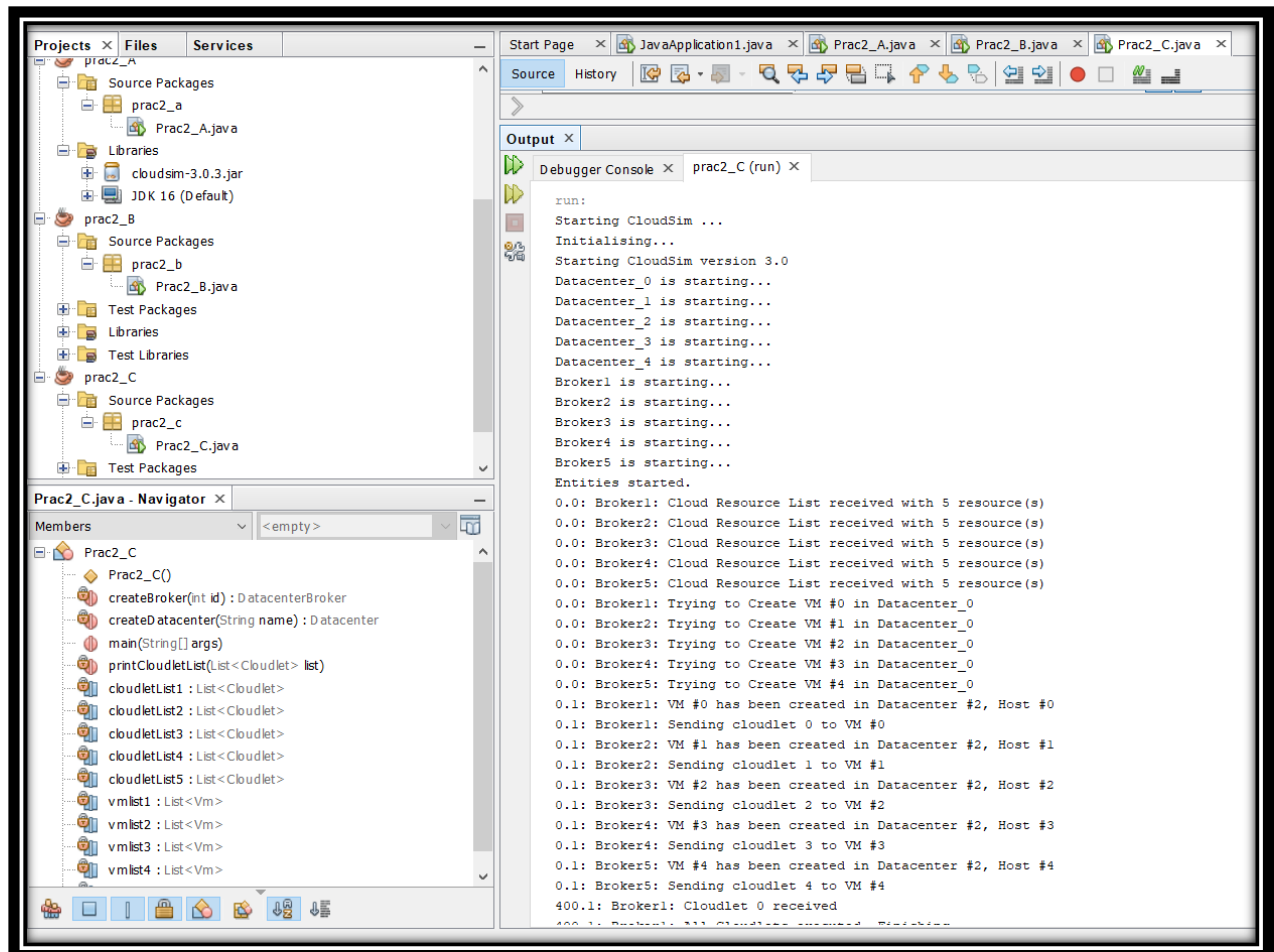
}

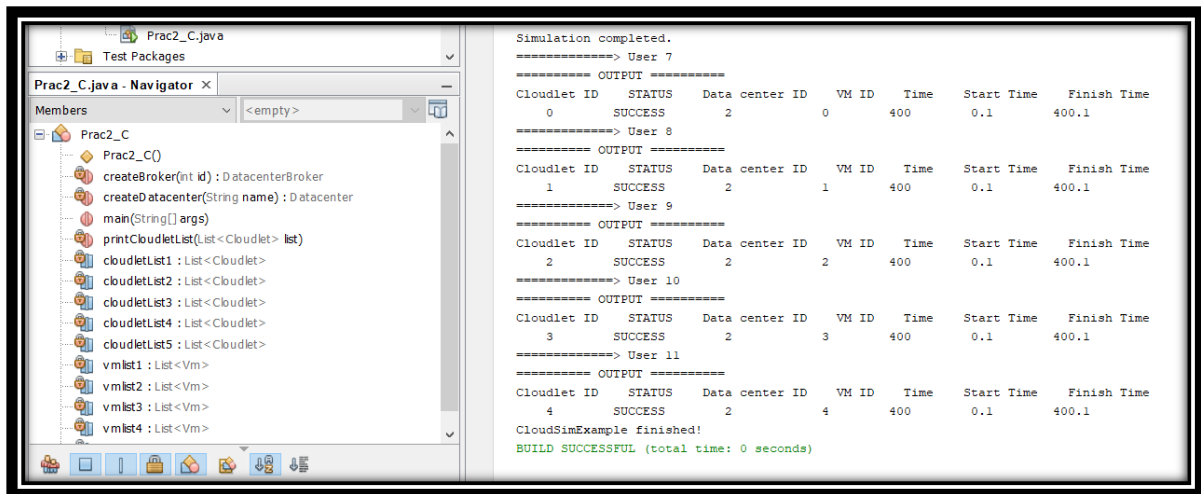
}

}

```

OUTPUT 3:





CONCLUSION:

In this practical, we learnt about cloudsims architecture and implemented several different scenarios using different number of datacenters, hosts and cloudlets.

PRACTICAL 3

AIM:

Understanding Network Examples (Any 2 Examples) in cloudsim

CODE:

1st:

```
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.NetworkTopology;

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;

/**
 * A simple example showing how to create
 * a datacenter with one host and a network
 * topology and and run one cloudlet on it.
 */
public class NetworkExample1 {

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vm list. */
    private static List<Vm> vmList;

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {

        Log.println("Starting NetworkExample1...");

        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
int vmid = 0;
int mips = 250;
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(vmid, brokerId, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerTimeShared());

//add the VM to the vmList
vmList.add(vm1);
```

```
//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 = 40000;
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);

//add the cloudlet to the list
cloudletList.add(cloudlet1);

//submit cloudlet list to the broker
broker.submitCloudletList(cloudletList);

//Sixth step: configure network
//load the network topology file
NetworkTopology.buildNetworkTopology("topology.brite");

//maps CloudSim entities to BRITE entities
```

```
//PowerDatacenter will correspond to BRITE node 0
int briteNode=0;
NetworkTopology.mapNode(datacenter0.getId(),briteNode);

//Broker will correspond to BRITE node 3
briteNode=3;
NetworkTopology.mapNode(broker.getId(),briteNode);

// Seventh step: Starts the simulation
CloudSim.startSimulation();

// Final step: Print results when simulation is over
List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.println("NetworkExample1 finished!");
}
catch (Exception e) {
    e.printStackTrace();
    Log.println("The simulation has been terminated due to an
unexpected error");
}
}

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

```
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.println();
        Log.println("===== OUTPUT =====");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent +
                    "Data center ID" + indent + "VM ID" + indent + "Time" +
indent + "Start Time" + indent + "Finish Time");

        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

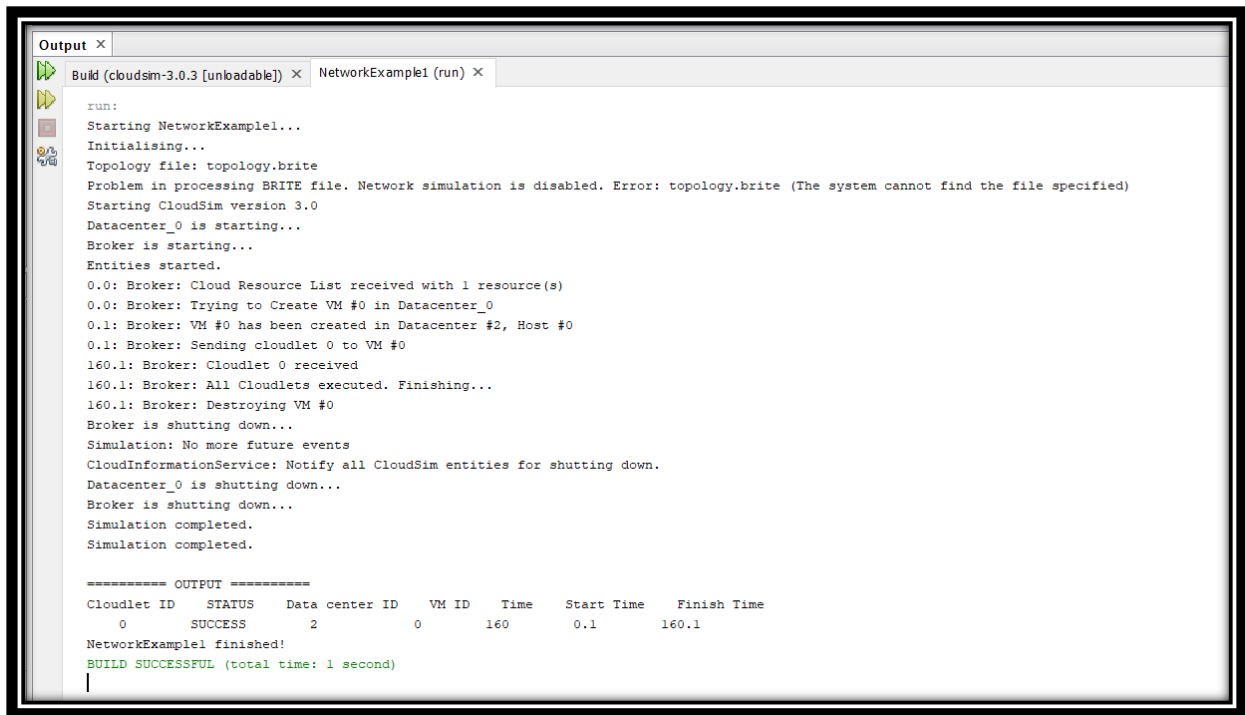
            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                Log.print("SUCCESS");

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

                Log.println( 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()));
            }
        }
    }
}

```

OUTPUT 1:



```

Output x
Build (cloudsim-3.0.3 [unloadable]) x NetworkExample1 (run) x
run:
Starting NetworkExample1...
Initialising...
Topology file: topology.brite
Problem in processing BRITE file. Network simulation is disabled. Error: topology.brite (The system cannot find the file specified)
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
160.1: Broker: Cloudlet 0 received
160.1: Broker: All Cloudlets executed. Finishing...
160.1: Broker: Destroying VM #0
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

===== OUTPUT =====
Cloudlet ID   STATUS   Data center ID   VM ID   Time   Start Time   Finish Time
0            SUCCESS   2                0       160    0.1          160.1
NetworkExample1 finished!
BUILD SUCCESSFUL (total time: 1 second)

```

2nd:

```

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.NetworkTopology;

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;
```

```
/**
```

```
 * A simple example showing how to create
 * two datacenters with one host and a
 * network topology each and run two cloudlets
 * on them.
```

```
*/
```

```
public class NetworkExample2 {
```

```
    /** The cloudlet list. */
```

```
    private static List<Cloudlet> cloudletList;
```

```
    /** The vmList. */
```

```
    private static List<Vm> vmList;
```

```
    /**
```

```
     * Creates main() to run this example
```

```
     */
```

```
    public static void main(String[] args) {
```

```
        Log.println("Starting NetworkExample2...");
```

```
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");  
    Datacenter datacenter1 = createDatacenter("Datacenter_1");  
  
    //Third step: Create Broker  
    DatacenterBroker broker = createBroker();  
    int brokerId = broker.getId();  
  
    //Fourth step: Create one virtual machine  
    vmlist = new ArrayList<Vm>();  
  
    //VM description  
    int vmid = 0;  
    int mips = 250;  
    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 two VMs

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

//the second VM will have twice the priority of VM1 and so will
receive twice CPU time

vmid++;

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

//add the VMs to the vmList

vmList.add(vm1);
vmList.add(vm2);

//submit vm list to the broker

broker.submitVmList(vmList);

//Fifth step: Create two Cloudlets

cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties

int id = 0;

long length = 40000;

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);

id++;

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

cloudlet2.setUserId(brokerId);

//add the cloudlets to the list
cloudletList.add(cloudlet1);
cloudletList.add(cloudlet2);

//submit cloudlet list to the broker
broker.submitCloudletList(cloudletList);

//bind the cloudlets to the vms. This way, the broker
// will submit the bound cloudlets only to the specific VM
broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());
broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

//Sixth step: configure network
//load the network topology file
NetworkTopology.buildNetworkTopology("topology.brite");

//maps CloudSim entities to BRITE entities
//Datacenter0 will correspond to BRITE node 0
int briteNode=0;
```

```
NetworkTopology.mapNode(datacenter0.getId(),briteNode);

//Datacenter1 will correspond to BRITE node 2
briteNode=2;
NetworkTopology.mapNode(datacenter1.getId(),briteNode);

//Broker will correspond to BRITE node 3
briteNode=3;
NetworkTopology.mapNode(broker.getId(),briteNode);


// Sixth step: Starts the simulation
CloudSim.startSimulation();


// Final step: Print results when simulation is over
List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.println("NetworkExample2 finished!");
}
catch (Exception e) {
    e.printStackTrace();
    Log.println("The simulation has been terminated due to an
unexpected error");
}
}
```

```

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;

    //in this example, the VMAllocationPolicy in use is Time Shared with
priorities. It means that VMs
    //receive time shares according to their priority.
    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

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.println();

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

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

        "Data center ID" + indent + "VM ID" + indent + "Time" +
indent + "Start Time" + indent + "Finish Time");

    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");

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

            Log.println( 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()));

        }

    }
}

```

```

    }
}

```

OUTPUT 2:

```

Output x
Build (cloudsim-3.0.3 [unloadable]) x NetworkExample2 (run) x
run:
Starting NetworkExample2...
Initialising...
Topology file: topology.brite
Problem in processing BRITE file. Network simulation is disabled. Error: topology.brite (The system cannot find the file specified)
Starting CloudSim version 3.0
Datacenter_0 is starting...
Datacenter_1 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 2 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
160.1: Broker: Cloudlet 0 received
160.1: Broker: Cloudlet 1 received
160.1: Broker: All Cloudlets executed. Finishing...
160.1: Broker: Destroying VM #0
160.1: Broker: Destroying VM #1
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

===== OUTPUT =====
Cloudlet ID   STATUS   Data center ID   VM ID   Time   Start Time   Finish Time
0            SUCCESS   2                0       160    0.1          160.1
1            SUCCESS   2                1       160    0.1          160.1
NetworkExample2 finished!
BUILD SUCCESSFUL (total time: 0 seconds)

```

CONCLUSION:

In this practical, we learnt about network examples of cloudsim and we run the first two examples and analysed their code.

PRACTICAL 4

AIM:

Install a Cloud Analyst and Integrate with Eclipse/Netbeans. Monitor the performance of an Existing Algorithms. (Perform Load Balancing).

THEORY:

CloudAnalyst:

- Cloud Analyst is a tool developed at the University of Melbourne whose goal is to support evaluation of social networks tools according to geographic distribution of users and data centers.
- In this tool, communities of users and data centers supporting the social networks are characterized and, based on their location; parameters such as user experience while using the social network application and load on the data center are obtained/logged.

PRACTICAL:

Download CloudAnalyst from

<http://www.cloudbus.org/cloudsim/CloudAnalyst.zip>

Extract Files from the Zip file which will give following folder structure.

Name	Date modified	Type	Size
.settings	16-08-2021 09:33	File folder	
classes	16-08-2021 09:33	File folder	
config	16-08-2021 09:33	File folder	
jars	16-08-2021 09:33	File folder	
javadoc	16-08-2021 09:33	File folder	
resources	16-08-2021 09:33	File folder	
source	16-08-2021 09:33	File folder	
test	05-08-2010 09:40	File folder	
.classpath	05-08-2010 10:23	CLASSPATH File	1 KB
.project	25-11-2009 05:14	PROJECT File	1 KB
readme.txt	05-08-2010 11:02	Text Document	1 KB
run.bat	05-08-2010 11:00	Windows Batch File	1 KB

If you want to Run from Command line then type the following command in cmd.

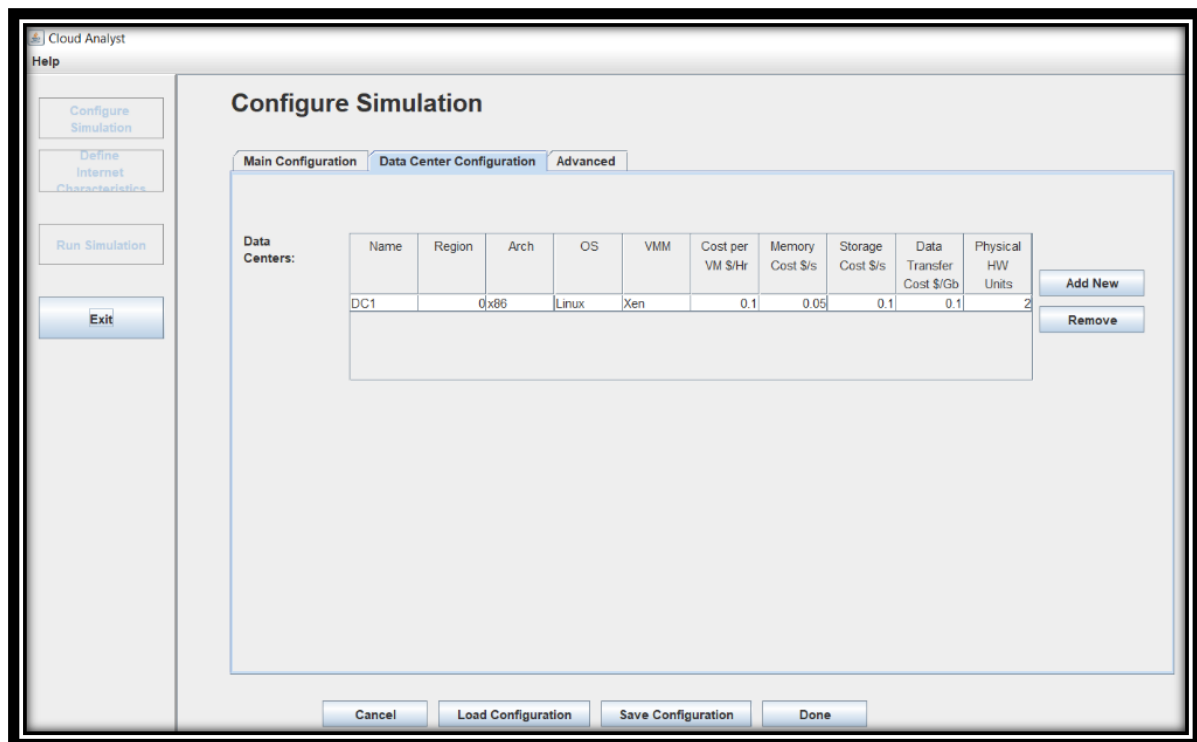

```
java -cp jars\simjava2.jar;jars\gridsim.jar;jars\iText-2.1.5.jar;classes;.
cloudsim.ext.gui.GuiMain
```



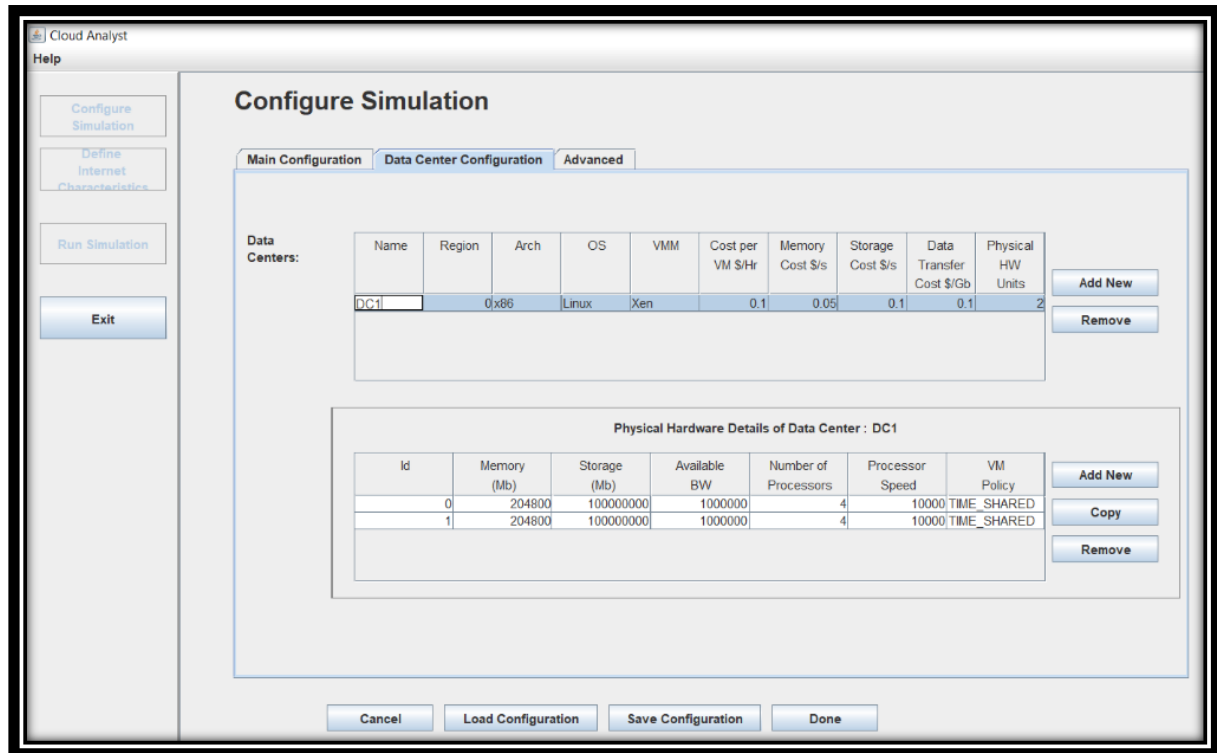
Click on CONFIGURE Simulation.

Here you can configure:

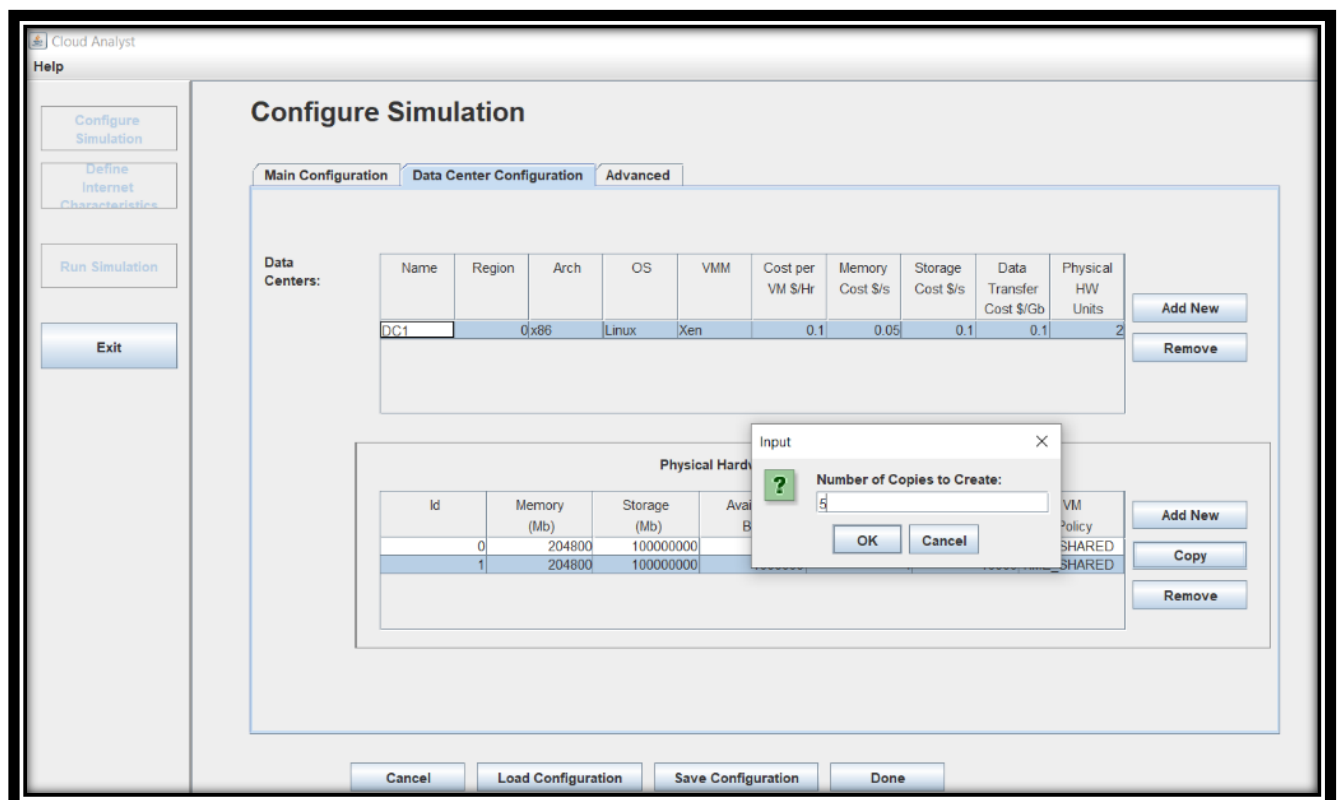
Data Center Configuration where you can manage Physical Hardware Details of Data center.



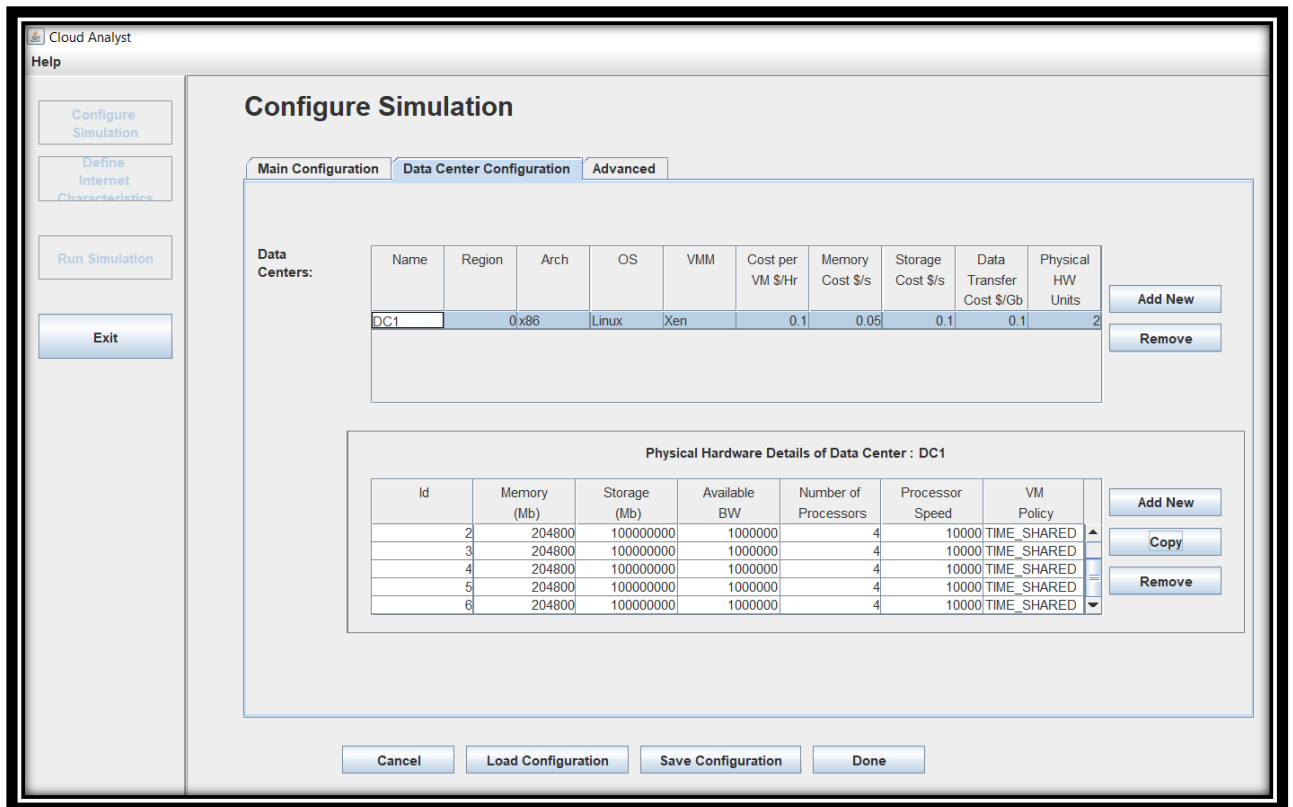
We can double click on Datacenter to open Host information.



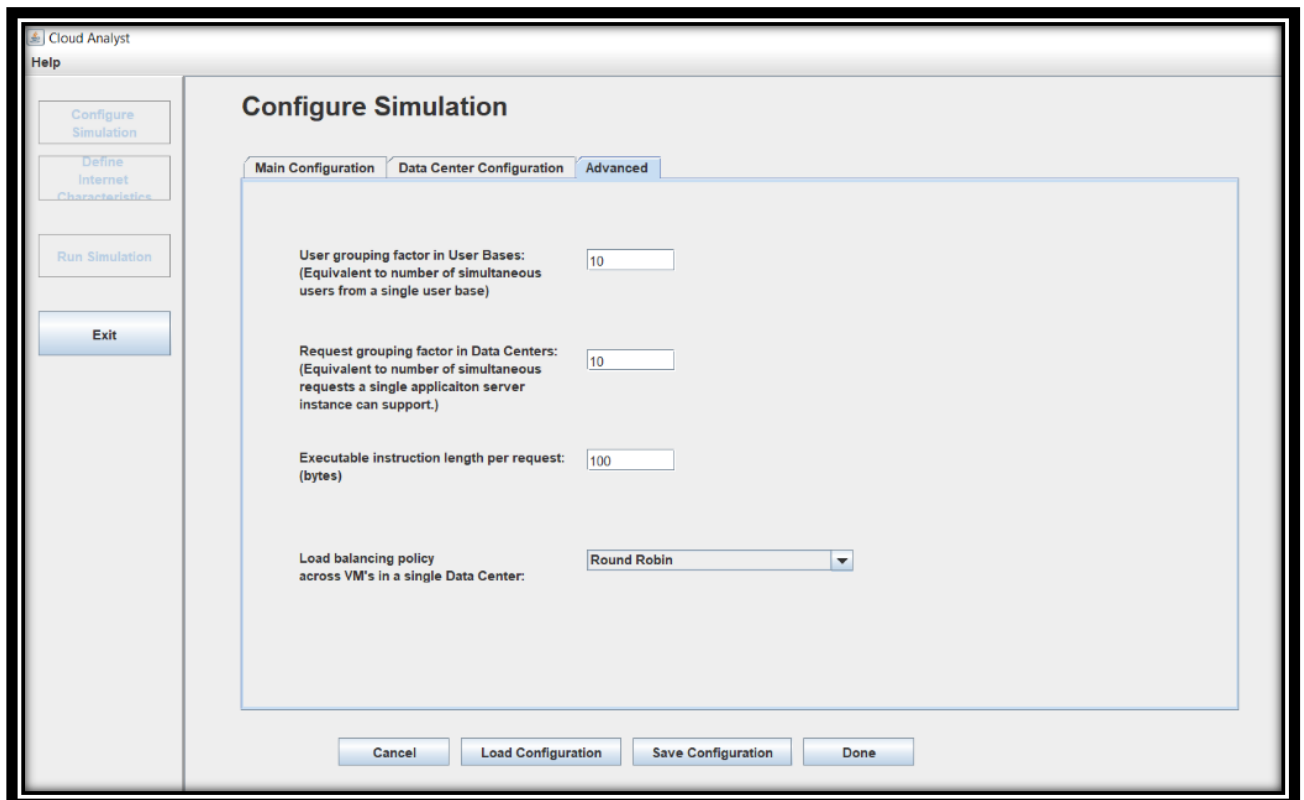
We can click on any host and copy it as many time as we want.



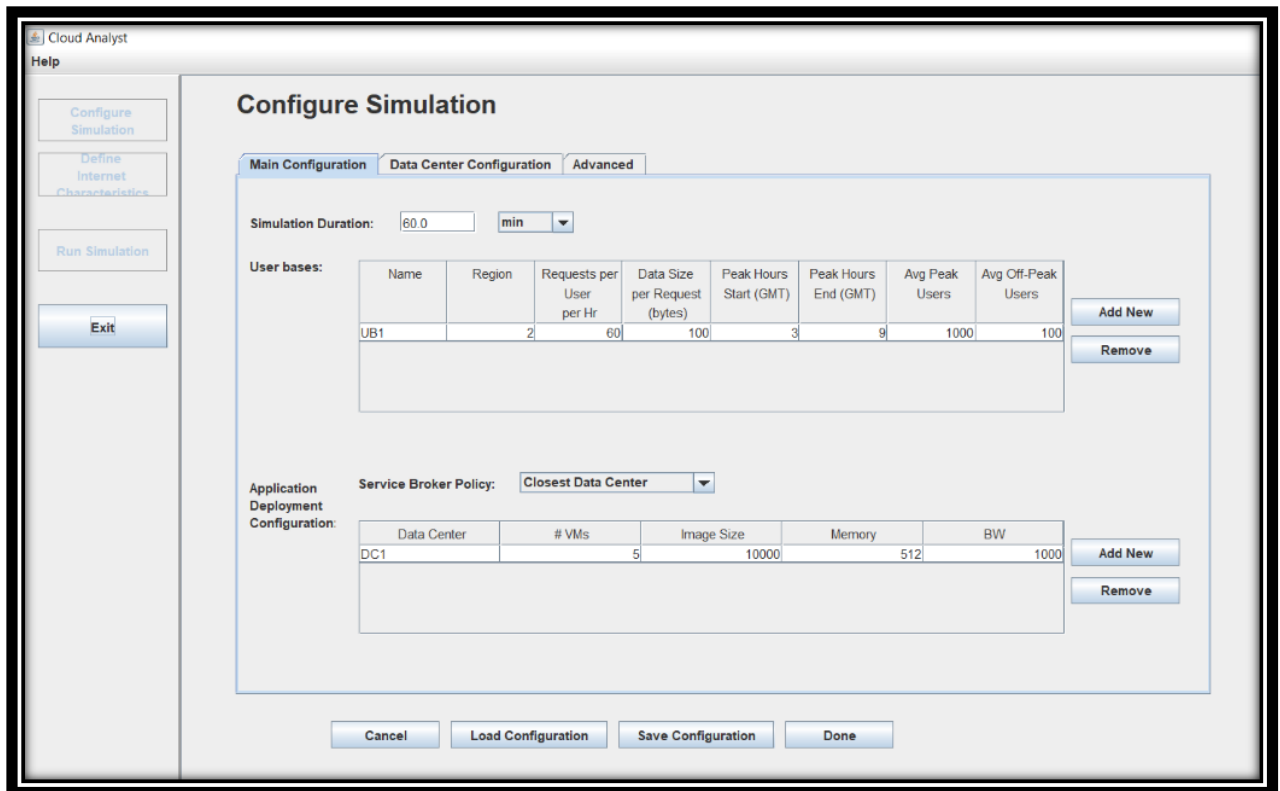
Here we are creating 5 copies of one of them so it will give us total of 6 hosts.



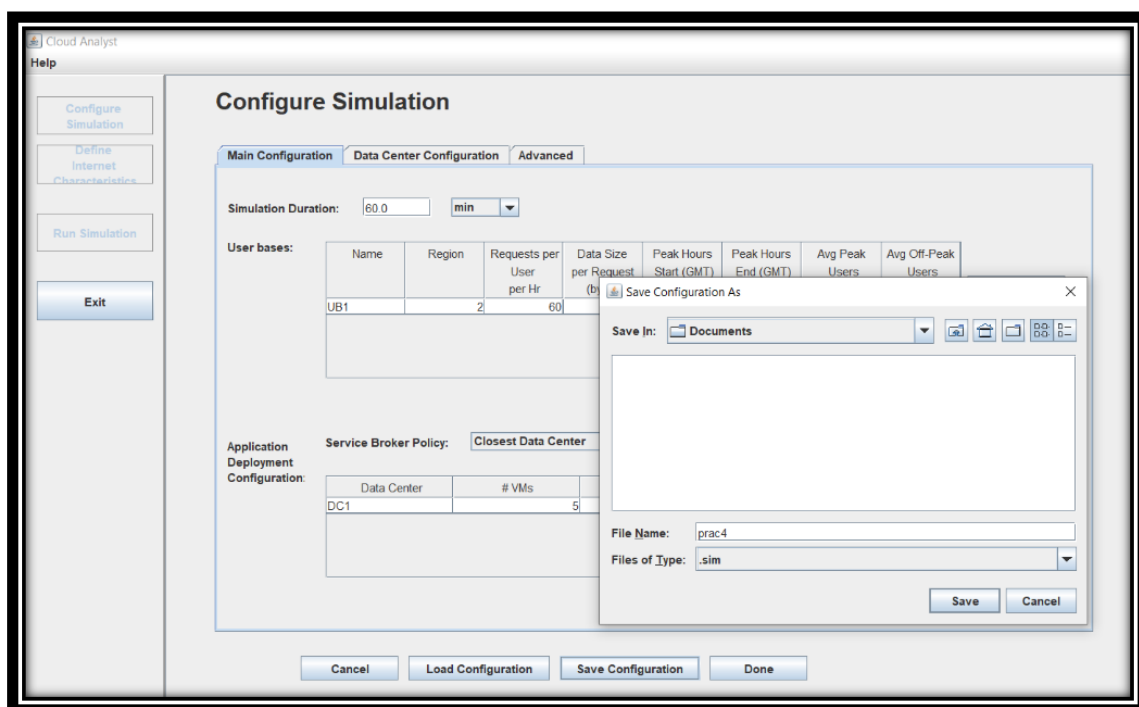
We can also configure advanced settings.



We can also customize user base that is models a group of users and generates traffic representing the users.



You can Save this Configuration as well in case you want to use it later. It is stored as .sim file. XML data is generated and saved as Sim file.

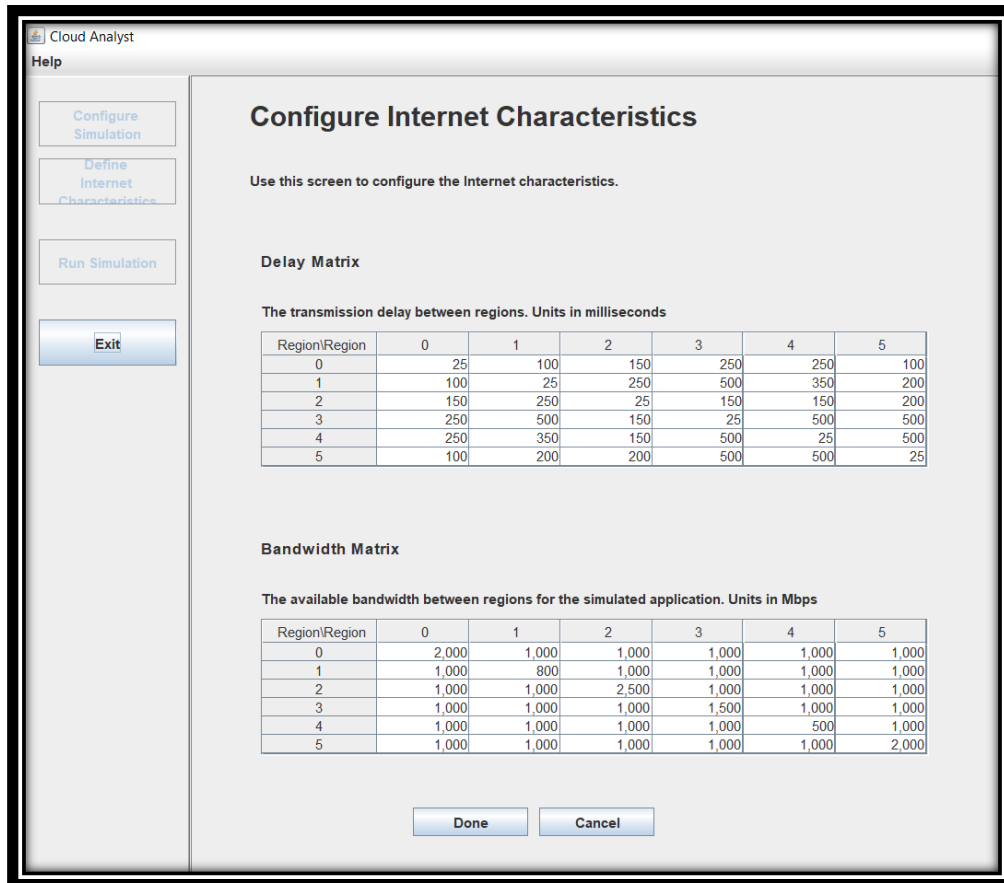


Saved configuration can be loaded anytime easily into CloudAnalyst.

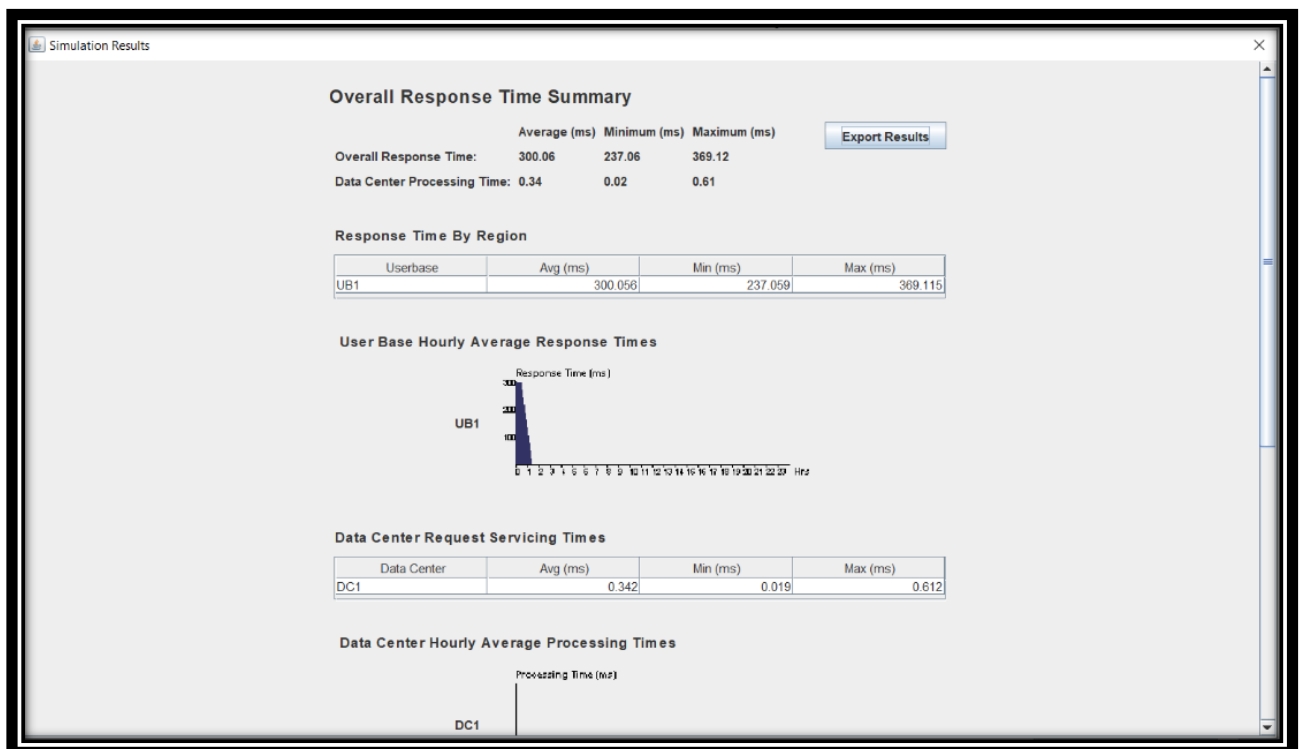
So you need to enter data each time you want to run simulation.

Once your are done with Configuration; click on Done!!!

We can check bandwidth and delay between regions in “Define Internet Characteristic”

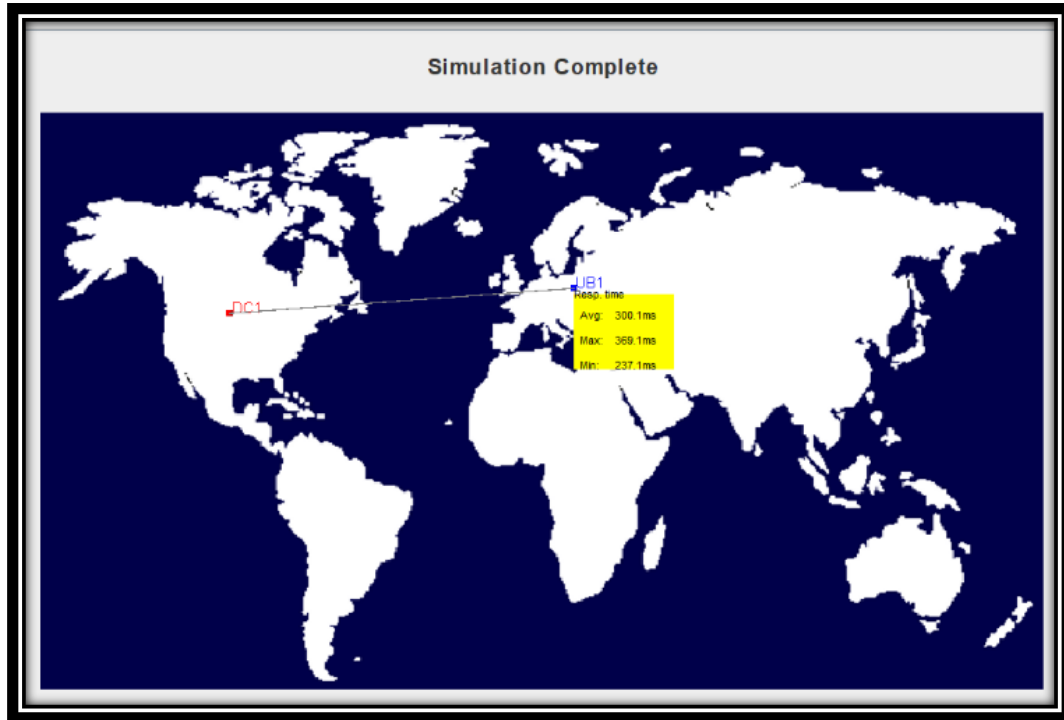


Then we can run simulation that would give us overall report of simulation.



We can close it.

Main Window will give all statistics.



CONCLUSION:

In this practical, we learnt about cloudAnalyst and simulated a simple case with single datacenter with 6 hosts and single user base.

PRACTICAL 5

AIM:

Choose any real time Cluster application and make analysis on that.

THEORY:

Cluster Computing:

Everyone might have faced the situation of low-speed services and content criticality. Cluster computing resolves the need for content criticality and process services in a quicker approach. As Internet Service Providers look for enhanced availability in a scalable approach, cluster computing will provide this. And even, this technology is the heavy need for the film industry as they require this for rendering extended quality of graphics and cartoons. Implementation of the cluster through the Beowulf method also resolves the requirement of statistics, fluid dynamics, genetic analysis, astrophysics, economics, neural networks, engineering, and finance. Many of the organizations and IT giants are implementing this technology to augment their scalability, processing speed, availability and resource management at the economic prices.

Cluster computing provides a relatively inexpensive, unconventional to the large server or mainframe computer solutions. Modernized advancements in both the hardware and software technologies are probable to allow cluster computing to be completely progressive. Many predict that the future of cluster computing also gains more prominence.

Advantages of Cluster Computing:

There are numerous advantages of implementing cluster computing in the applications. Few of them to be discussed are as follows:

Cost efficacy – Even mainframe computers seems to be extremely stable, cluster computing is more in implementation because of their cost-effectiveness and economical. Also, these systems provide enhanced performance than that of mainframe computer networks.

Processing speed – The cluster computing systems offer the same processing speed as that of mainframe computers and the speed is also equal to supercomputers.

Extended resource availability – Computers come across frequent breakdowns, so to eliminate this failure, cluster computers are available with high availability. So, when one node gets failed, the other nodes will be active and will function as a proxy for the failed node. This makes sure for enhanced availability.

Expandability – The next crucial advantage of this cluster computing is its enhanced scalability and expandability. As they instantiate the prospect to combine multiple additional resources or the networks to the prevailing computer system.

Flexibility – Cluster computing can be upgraded to the superior specification or extended through the addition of additional nodes (computer systems).

Some of the popular implementations of cluster computing are Google search engine, Earthquake Simulation, Petroleum Reservoir Simulation, and Weather Forecasting system.

We're going to focus on Clustering search Engine in this report.

Clustering Search Engines:

Clusters are widely used with respect to the criticality of the data or content handled and the processing speed expected. Sites and applications which expect extended availability without downtime and expecting heavy load balancing ability use these cluster concepts to a large extent.

Clustering Search Engines unlike traditional Search Engines provide clusters that are related to the original search phrase. Clusters help users see search results by topic so they can zero in on exactly what they are looking for or discover unexpected relationships between items. So instead of having to look through pages of search results, the clusters help users find results they might have missed or that were buried deep in the ranked list.

These suggested clusters provide an interesting way of finding relevant topics to extract keywords from.

Although this method finds relevant topics, it has some fairly serious shortcomings, some of which are:

Most search engines do not allow the automated querying of their services.

Automated querying where allowed takes too long.

The clusters generated by clustering search engines are often not Closely Related Themes.

For these reasons the use of clustering search engines is not considered a viable method of keyword discovery.

Several commercial clustering engines have been launched recently; the most popular one among them is the Vivisimo engine. Vivisimo won the "best meta-search engine award" assigned by Search Engine <http://watch.com/> from 2001 to 2003

Vivisimo is possibly the most popular commercial clustering search engine. Vivisimo calls search engines such as Yahoo and Google to extract relevant information (titles, URLs, and short descriptions) from the result retrieved. It groups documents in the retrieved result based on summarized information. The Vivisimo search clustering engine was sold to Yippy, Inc. in 2010. Grouper uses snippets obtained by the search engines. It is an interface for the results of the Husky Search meta-search engine. Grouper uses the Suffix Tree Clustering (STC) algorithm to cluster together documents that have great common subphrases. Carrot2 is a clustering search engine solution that uses search results from various search engines including Yahoo, Google, and MSN. It uses five different clustering algorithms (STC,

FussyAnts, Lingo, HAOG-STC, and Rough k-means) where Lingo Algorithm is the default clustering algorithm used. The output is a flat folder structure; overlapping folders are revealed when the user places the mouse over a document title. The system presented in is a meta-search clustering engine, called the Search Clustering System (SCS), which organizes the results returned by conventional Web search engines into a cluster hierarchy. The hierarchy is produced by the Cluster Hierarchy Construction Algorithm (CHCA). Unlike most other clustering algorithms, CHCA operates on nominal data: its input is a set of binary vectors representing Web documents. Document representations are based either on snippets or on the full contents of the retrieved pages.

All of the above clustering engines except use snippets that probably contain terms that are part of the query keywords. Snippets are not necessarily good representative of the whole document contents, which affects the quality of the clusters. Proposed solution uses whole documents rather than titles and short snippets, to ensure proper extraction of the semantics of the retrieved documents. While all of the above clustering engines have been mostly performed without explicit use of lexical semantics, proposed work takes into consideration both lexical and semantics similarities.

CONCLUSION:

In this practical, we learnt about cluster computing and its advantages. We analysed the cluster search engine and different examples of it.

PRACTICAL 6

AIM:

Choose any one real time Grid Application and analyze its applications.

THEORY:**Grid Computing:**

Grid Computing can be defined as a network of computers working together to perform a task that would rather be difficult for a single machine. All machines on that network work under the same protocol to act like a virtual supercomputer. The task that they work on may include analysing huge datasets or simulating situations which require high computing power. Computers on the network contribute resources like processing power and storage capacity to the network.

Grid Computing is a subset of distributed computing, where a virtual super computer comprises of machines on a network connected by some bus, mostly Ethernet or sometimes the Internet. It can also be seen as a form of Parallel Computing where instead of many CPU cores on a single machine, it contains multiple cores spread across various locations. The concept of grid computing isn't new, but it is not yet perfected as there are no standard rules and protocols established and accepted by people.

Working:

A Grid computing network mainly consists of these three types of machines

Control Node:

A computer, usually a server or a group of servers which administrates the whole network and keeps the account of the resources in the network pool.

Provider:

The computer which contributes it's resources in the network resource pool.

User:

The computer that uses the resources on the network.

When a computer makes a request for resources to the control node, control node gives the user access to the resources available on the network. When it is not in use it should ideally contribute it's resources to the network. Hence a normal computer on the node can swing in between being a user or a provider based on it's needs. The nodes may consist of machines with similar platforms using same OS called homogeneous networks, else machines with

different platforms running on various different OS called heterogeneous networks. This is the distinguishing part of grid computing from other distributed computing architectures.

For controlling the network and its resources a software/networking protocol is used generally known as Middleware. This is responsible for administrating the network and the control nodes are merely its executors. As a grid computing system should use only unused resources of a computer, it is the job of the control node that any provider is not overloaded with tasks.

Another job of the middleware is to authorize any process that is being executed on the network. In a grid computing system, a provider gives permission to the user to run anything on its computer, hence it is a huge security threat for the network. Hence a middleware should ensure that there is no unwanted task being executed on the network.

The meaning of the term Grid Computing has changed over the years, according to “The Grid: Blueprint for a new computing infrastructure” by Ian Foster and Carl Kesselman published in 1999, the idea was to consume computing power like electricity is consumed from a power grid. This idea is similar to current concept of cloud computing, whereas now grid computing is viewed as a distributed collaborative network. Currently grid computing is being used in various institutions to solve a lot of mathematical, analytical and physics problems.

Advantages of Grid Computing:

- It is not centralized, as there are no servers required, except the control node which is just used for controlling and not for processing.
- Multiple heterogeneous machines i.e. machines with different Operating Systems can use a single grid computing network.
- Tasks can be performed parallelly across various physical locations and the users don't have to pay for it(with money).

General Applications of Grid Computing

- Distributed Supercomputing
- High-throughput Supercomputing
- On-demand Supercomputing
- Data-intensive Supercomputing
- Collaborative Supercomputing

Applications of Grid Computing Across Different Sectors

- Movie Industry
- Gaming Industry
- Life Sciences
- Engineering and Design

- Government

Distributed Supercomputing:

A distributed supercomputer is a supercomputer with processors spread around the world, connected via the internet. It's conceptually the exact same as a classic supercomputer, but distributed. These processors are located in the desktops and laptops used by average people every day. In a distributed supercomputer, device owners allow other people to use their processors when they aren't using them, and the devices are wirelessly connected to the distributed supercomputer. That's pretty neat, but there's a downside. Currently, distributed supercomputers can only solve certain, very specific, problems.

This is why the future is Hypercomputing by Hypernetwork.

Hypercomputing is a term we adapted to describe distributed supercomputing that is done much faster, and in a truly parallel manner. Because it is parallel, Hypernet is not limited in its computational abilities in the way that other distributed supercomputers are. This means it can be used for complex parallel processing tasks, like the identification of cancer cells, the prediction of traffic patterns, or even predicting natural disasters!

the advantage of distributed systems is that relative to supercomputers they are much less expensive. Many distributed systems make use of cheap, off-the-shelf computers for processors and memory, which only require minimal cooling costs. In addition, they are simpler to scale, as adding an additional processor to the system often consists of little more than connecting it to the network. However, unlike supercomputers, which send data short distances via sophisticated and highly optimized connections, distributed systems must move data from processor to processor over slower networks making them unsuitable for many real-time applications.

Distributed systems are most useful for problems that are not as sensitive to latency. For example, when NASA's Jet Propulsion Laboratory (JPL) needed to process high volumes of image data collected by its Mars rovers, a computer cluster hosted on the Amazon Cloud was a natural fit. Such tasks are not substantially hindered by small delays in individual computations, so distributed systems offered the most pragmatic solution. Other distributed computing applications include large-scale records management and text mining.

CONCLUSION:

In this practical, we learnt about grid computing and listed different applications of it. We chose distributed supercomputing to analyse in detail.

PRACTICAL 7

AIM:

Case study on GoogleApp Engine API for PaaS in cloud computing.

THEORY:

PaaS:

Platform as a service (PaaS) is a cloud computing model where a third-party provider delivers hardware and software tools to users over the internet. Usually, these tools are needed for application development. A PaaS provider hosts the hardware and software on its own infrastructure. As a result, PaaS frees developers from having to install in-house hardware and software to develop or run a new application.

PaaS tools tend to be touted as simple to use and convenient. Users will normally have to pay on a per-use basis. An organization may find the move to a PaaS compelling considering potential cost savings over using on-premises alternatives.

Google App Engine:

Google AppEngine is a scalable runtime environment mostly devoted to executing Web applications. These take advantage of the large computing infrastructure of Google to dynamically scale as the demand varies over time. AppEngine provides both a secure execution environment and a collection of services that simplify the development of scalable and high-performance Web applications. These services include in-memory caching, scalable data store, job queues, messaging, and cron tasks. Developers can build and test applications on their own machines using the AppEngine software development kit (SDK), which replicates the production runtime environment and helps test and profile applications. Once development is complete, developers can easily migrate their application to AppEngine, set quotas to contain the costs generated, and make the application available to the world. The languages currently supported are Python, Java, and Go.

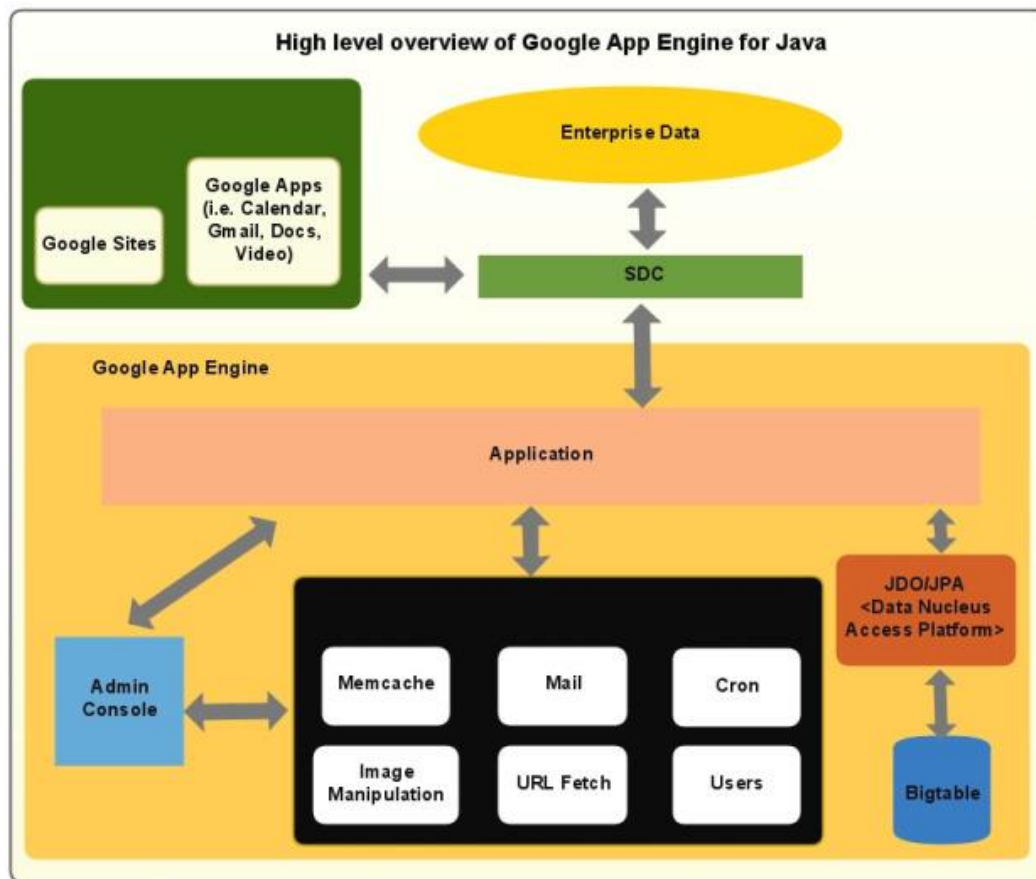
Advantages of Google App Engine:

- Easy to build.
- Easy to maintain.
- Easy to scale as the traffic and storage needs grow.

Is it free?

Yes, free for upto 1 GB of storage and enough CPU and bandwidth to support 5 million page views a month. 10 Applications per Google account.

Architecture:



Key Features of Google App Engine:

- Dynamic web serving, with full support for common web technologies
- Persistent storage with queries, sorting, and transactions
- Automatic scaling and load balancing
- APIs for authenticating users and sending email using Google Accounts
- A fully featured local development environment that simulates Google App Engine on your computer
- Task queues for performing work outside of the scope of a web request
- Scheduled tasks for triggering events at specified times and regular intervals

Programming Language Support of Google App Engine:

Java:

- App Engine runs JAVA apps on a JAVA 7 virtual machine (currently supports JAVA 6 as well).
- Uses JAVA Servlet standard for web applications:
 - WAR (Web Applications ARchive) directory structure.
 - Servlet classes
 - Java Server Pages (JSP)

- Static and data files
- Deployment descriptor (web.xml)
- Other configuration files

Python:

- Uses WSGI (Web Server Gateway Interface) standard.
- Python applications can be written using:
 - Webapp2 framework
 - Django framework
 - Any python code that uses the CGI (Common Gateway Interface) standard.

PHP (Experimental support):

- Local development servers are available to anyone for developing and testing local applications.
- Only whitelisted applications can be deployed on Google App Engine.

Google's Go:

- Go is a Google's open source programming environment.
- Tightly coupled with Google App Engine.
- Applications can be written using App Engine's Go SDK.

Data storage in Google App Engine:**App Engine Datastore:**

- NoSQL schema-less object based data storage, with a query engine and atomic transactions.
- Data object is called a "Entity" that has a kind (~ table name) and a set of properties (~ column names).
- JAVA JDO/ JPA interfaces and Python datastore interfaces.

Google cloud SQL:

- Provides a relational SQL database service.
- Similar to MySQL RDBMS.

Google cloud store:

- RESTful service for storing and querying data.
- Fast, scalable and highly available solution.
- Provides Multiple layers of redundancy. All data is replicated to multiple data centers.
- Provides different levels of access control.
- HTTP based APIs.

Services of Google App Engine:

URL Fetch:

Facilitates the application's access to resources on the internet, such as web services or data.

Mail:

Facilitates the application to send e-mail messages using Google infrastructure.

Memcache:

High performance in-memory key-value storage.

Can be used to store temporary data which doesn't need to be persisted.

Security in Google App Engine:**The sandbox:**

All hosted applications run in a secure environment that provides limited access to the underlying operating system.

Sandbox isolates the application in its own secure, reliable environment that is independent of hardware, operating system and physical location of a web server.

Limitations imposed by sandbox (for security):

An application can only access other computers over internet using the provided URL fetch and email services. Other computers can only connect to the application through HTTP/HTTPS requests on the standard ports (80/ 443).

Applications cannot write to local file system in any of the runtime environments.

Application code runs only in response to a web request, a queued task or a scheduled task and must return the response data within 60 seconds. A request handler cannot spawn a sub-process or execute code after the response has been sent.

When to use Google App Engine:

- You don't want to get troubled for setting up a server.
- You want instant for-free nearly infinite scalability support.
- Your application's traffic is spiky and rather unpredictable.
- You don't feel like taking care of your own server monitoring tools.
- You need pricing that fits your actual usage and isn't time-slot based (App engine provides pay-per-use cost model).
- You are able to chunk long tasks into 60 second pieces.
- You are able to work without direct access to local file system.

Pricing of Google App Engine:

Resource	Unit	Unit cost (in US \$)
Instances*	Instance hours	\$0.05
Outgoing Network Traffic	Gigabytes	\$0.12
Incoming Network Traffic	Gigabytes	Free
Datastore Storage	Gigabytes per month	\$0.18
Blobstore, Logs, and Task Queue Stored Data	Gigabytes per month	\$0.026
Dedicated Memcache	Gigabytes per hour	\$0.06
Logs API	Gigabytes	\$0.12
SSL Virtual IPs** (VIPs)	Virtual IP per month	\$39.00
Sending Email, Shared Memcache, Pagespeed, Cron, APIs (URLFetch, Task Queues, Image, Sockets, Files, and Users)		No Additional Charge

CONCLUSION:

In this practical, we learnt about PaaS and Google App Engine. We analysed different aspects of Google App engine and learnt deeply about how it works and how we should use it.

PRACTICAL 8

Aim:

An organization is in its initial planning to start the cloud infrastructure at its workplace. The organization is now focused to check the Service Level Agreement (SLA) which gives service availability of 98% of time. The organization application runs on public cloud for 14 hours/day. The task is to search out that SLA is violated or not for this organization with outage after one month is 10.25 hours.

Theory:

A cloud SLA (cloud service-level agreement) is an agreement between a cloud service provider and a customer that ensures a minimum level of service is maintained. It guarantees levels of reliability, availability and responsiveness to systems and applications; specifies who governs when there is a service interruption; and describes penalties if service levels are not met.

For the given Problem, it is given that:

- The organization claims to give service availability **98%** of time
- The organization application runs on public cloud for **14 hours/day**.
- Outage after one month is **10.25 hours**

We want to find out whether the provider has violated the initial availability guarantee.

$$\begin{aligned}\text{Total Time for which application runs in a month} &= 14 * 30 \\ &= 420\end{aligned}$$

$$\text{Outage time} = 10.25 \text{ Hrs}$$

$$\begin{aligned}\text{Therefore, Service Duration} &= (420 - 10.25) \\ &= 409.75 \text{ Hrs}\end{aligned}$$

$$\begin{aligned}\text{Percentage Availability} &= (1 - 10.25/409.75) * 100 \\ &= (1 - 0.02501) * 100 \\ &= 97.49 \%\end{aligned}$$

Since Initial Service Guarantee = 98%

So, Final Service Availability < Initial Service Availability

So, the SLA is Violated.

CONCLUSION:

In this practical, we learnt about SLA and how to check if SLA is satisfied or not.

PRACTICAL 9

AIM:

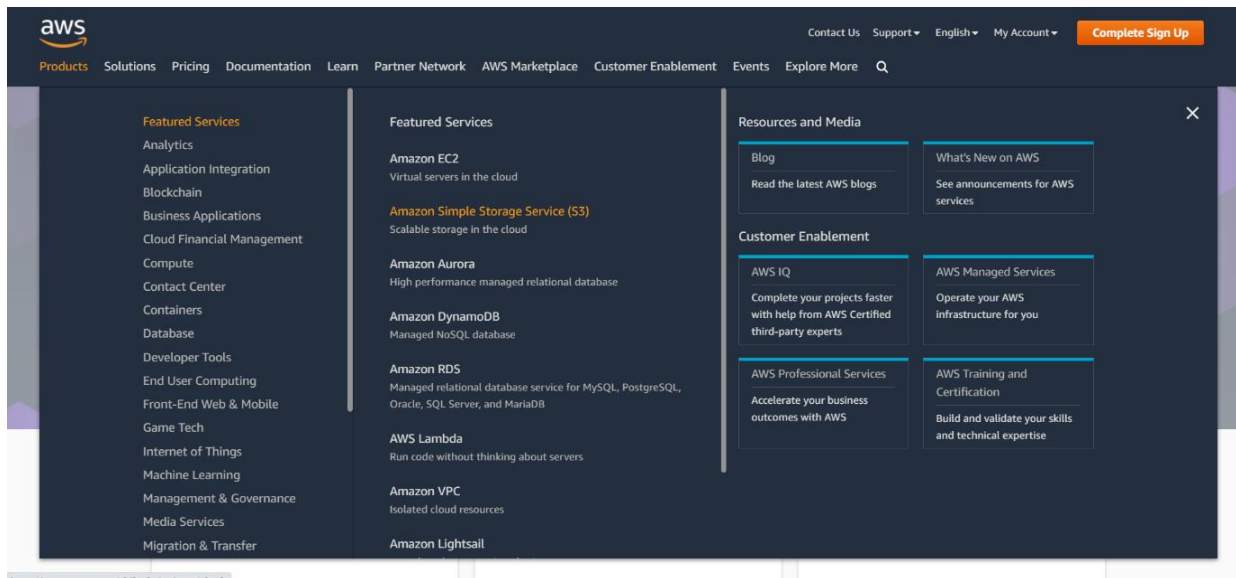
Amazon Simple Storage Service (Amazon S3) is an object storage service and it can have features such as industry-leading scalability, data availability and performance. It is used to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, and enterprise applications. It provides easy-to-use management to organize your data and to meet your organizational requirements. Deploy AWS S3 bucket for serverless computing and host static website using AWS S3 bucket.

THEORY:

Amazon Web Services, Inc. (AWS) is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. These cloud computing web services provide a variety of basic abstract technical infrastructure and distributed computing building blocks and tools. One of these services is Amazon Elastic Compute Cloud (EC2), which allows users to have at their disposal a virtual cluster of computers, available all the time, through the Internet. AWS's virtual computers emulate most of the attributes of a real computer, including hardware central processing units (CPUs) and graphics processing units (GPUs) for processing; local/RAM memory; hard-disk/SSD storage; a choice of operating systems; networking; and pre-loaded application software such as web servers, databases, and customer relationship management (CRM).

The AWS technology is implemented at server farms throughout the world, and maintained by the Amazon subsidiary. Fees are based on a combination of usage (known as a "Pay-as-you-go" model), hardware, operating system, software, or networking features chosen by the subscriber required availability, redundancy, security, and service options. Subscribers can pay for a single virtual AWS computer, a dedicated physical computer, or clusters of either. As part of the subscription agreement, Amazon provides security for subscribers' systems. AWS operates from many global geographical regions including 6 in North America.

Amazon markets AWS to subscribers as a way of obtaining large scale computing capacity more quickly and cheaply than building an actual physical server farm. All services are billed based on usage, but each service measures usage in varying ways. As of 2017, AWS owns 33% of all cloud (IaaS, PaaS) while the next two competitors Microsoft Azure and Google Cloud have 18%, and 9% respectively, according to Synergy Group.



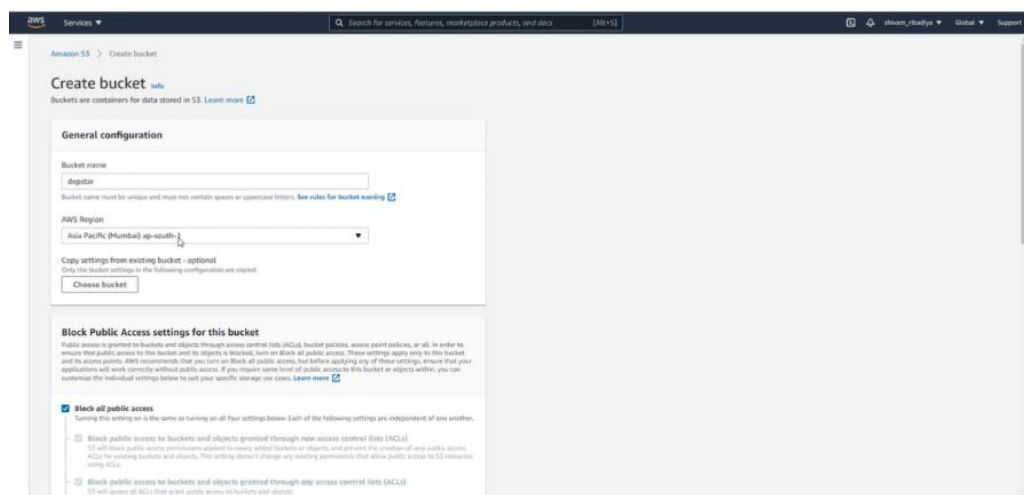
PRACTICAL:

Login to your AWS account and go to products and select Amazon Simple Storage Service (S3).

Before you begin hosting your awesome static website out of S3, you need a bucket first. For this blog post, it is critical that your bucket has the same name as your domain name.

If your website domain is `www.my-awesome-site.com`, then your bucket name must be `www.my-awesome-site.com`.

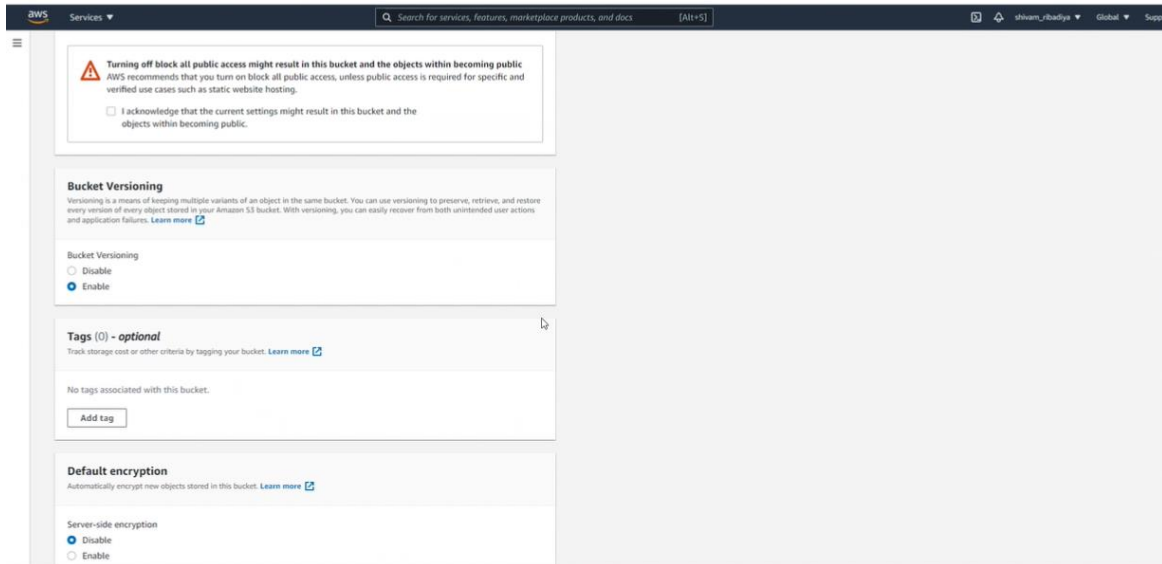
The reasoning for this has to do with how requests are routed to S3. The request comes into the bucket, and then S3 uses the Host header in the request to route to the appropriate bucket.



Alright, you have your bucket. It has the same name as your domain name, yes? Time to configure the bucket for static website hosting.

- Navigate to S3 in the AWS Console.

- Click into your bucket.
- Click the “Properties” section.
- Click the “Static website hosting” option.
- Select “Use this bucket to host a website”.
- Enter “index.html” as the Index document.



Your bucket is configured for static website hosting, and you now have an S3 website url like this <http://www.my-awesome-site.com.s3-website-us-east-1.amazonaws.com/>.

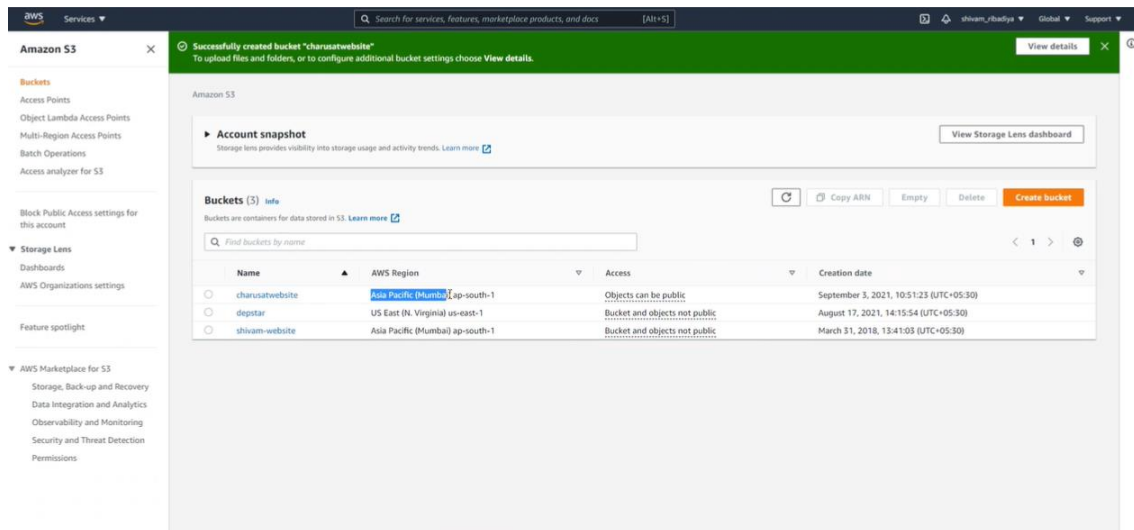
By default, any new buckets created in an AWS account deny you the ability to add a public access bucket policy. This is in response to the recent leaky buckets where private information has been exposed to bad actors. However, for our use case, we need a public access bucket policy. To allow this you must complete the following steps before adding your bucket policy.

- Click into your bucket.
- Select the “Permissions” tab at the top.
- Under “Public Access Settings” we want to click “Edit”.
- Change “Block new public bucket policies”, “Block public and cross-account access if bucket has public policies”, and “Block new public ACLs and uploading public objects” to be false and Save.

Now you must update the Bucket Policy of your bucket to have public read access to anyone in the world. The steps to update the policy of your bucket in the AWS Console are as follows:

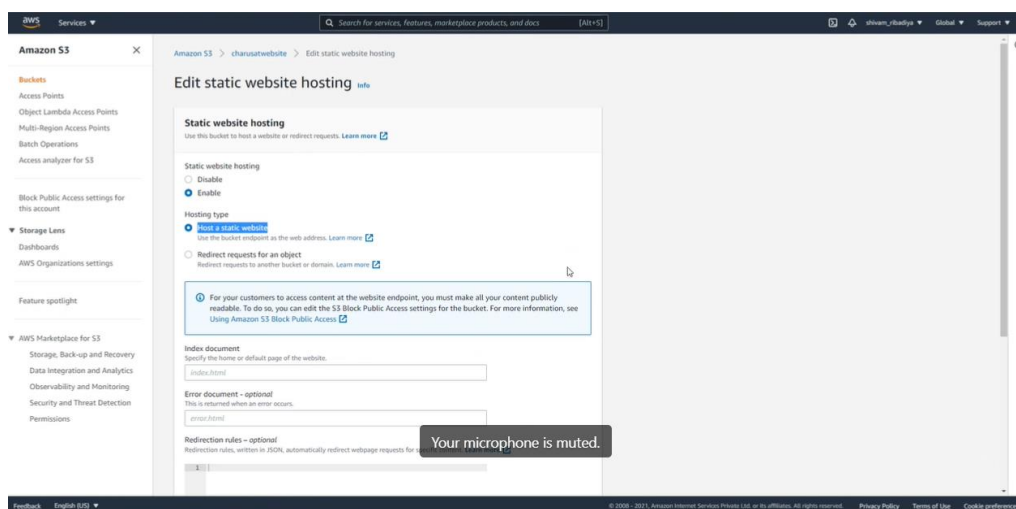
- Navigate to S3 in the AWS Console.
- Click into your bucket.
- Click the “Permissions” section.
- Select “Bucket Policy”.
- Add the following Bucket Policy and then Save

Remember S3 is a flat object store, which means each object in the bucket represents a key without any hierarchy. While the AWS S3 Console makes you believe there is a directory structure, there isn't. Everything stored in S3 is keys with prefixes.

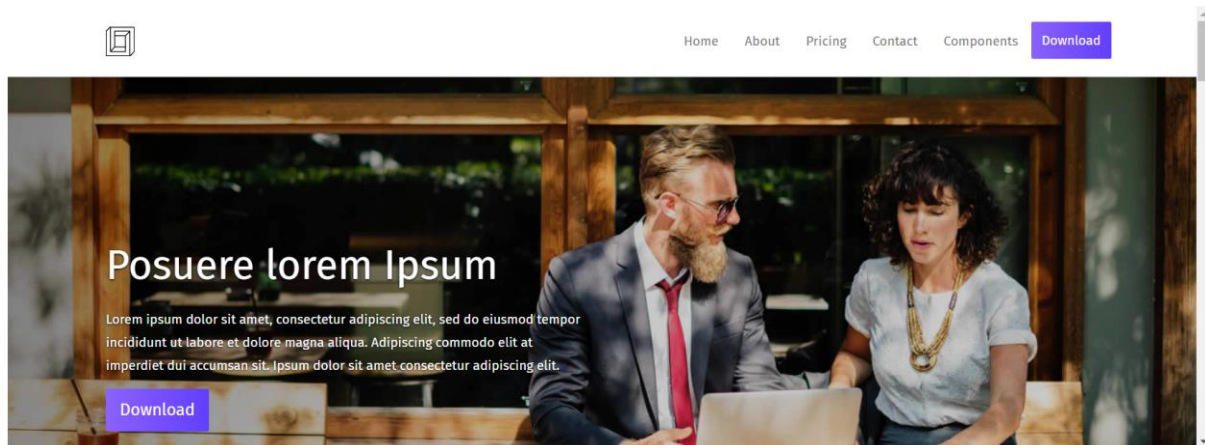


If you are a GUI person, then you upload your static website to S3 via the AWS Console by completing these steps:

- Navigate to S3 in the AWS Console.
- Click into your bucket.
- Click the “Upload” button.
- Drag and drop or select “Add files”, and add the entire static website directory.
- Click “Next”.
- Leave the default permissions S3 offers.
- Click “Next”.
- Leave the default permissions for “Set properties”.
- Click “Next”.
- Click “Upload”.



Your static website has been uploaded to your S3 website bucket. You can go to www.my-awesome-site.com and your static website loads from your S3 bucket.



CONCLUSION:

In this practical, we learnt about AWS and hosted our static website on AWS using S3 service.

PRACTICAL 10

AIM:

Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry and it is used to store, manage, and deploy Docker container images. It hosts images in scalable architecture and users can deploy containers for their applications. Deploy AWS Elastic Container Registry (ECR) for containers file storage facility in cloud computing.

THEORY:

Docker is a free software developed by Docker Inc. It was presented to the general public on March 13, 2013 and has become since that day a must in the world of IT development.

It allows users to create independent and isolated environments to launch and deploy its applications. These environments are then called containers.

This will let the developer run a container on any machine.

As you can see, with Docker, there are no more dependency or compilation problems. All you have to do is launch your container and your application will launch immediately.

Docker container technology was launched in 2013 as an open source Docker Engine.

It leveraged existing computing concepts around containers and specifically in the Linux world, primitives known as cgroups and namespaces. Docker's technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

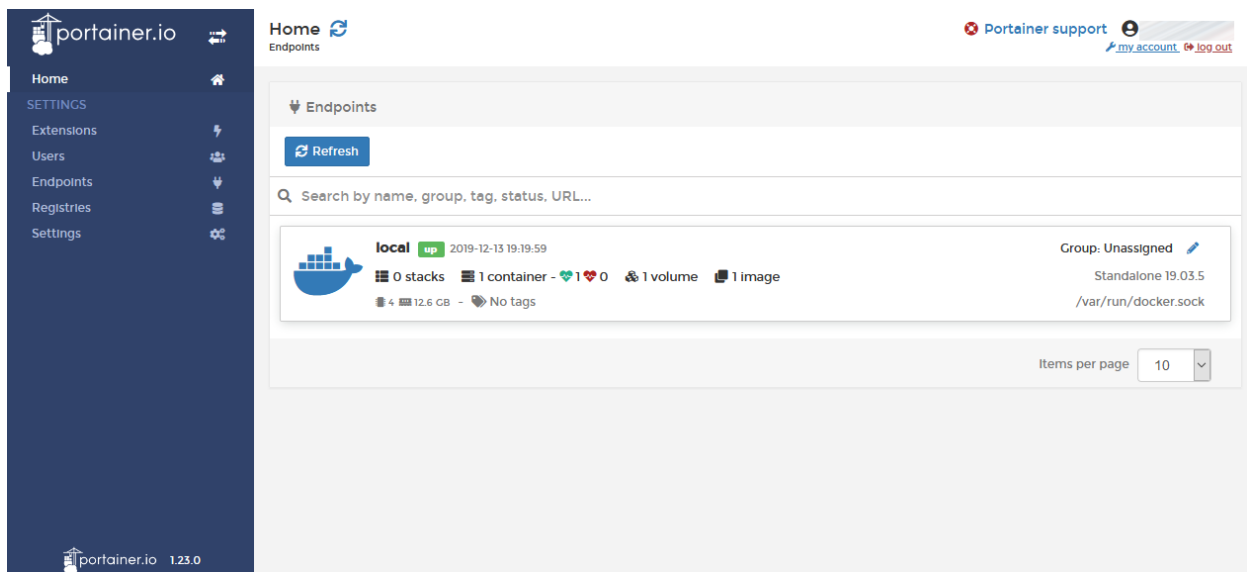
Success in the Linux world drove a partnership with Microsoft that brought Docker containers and its functionality to Windows Server (sometimes referred to as Docker Windows containers).

Technology available from Docker and its open source project, Moby has been leveraged by all major data centre vendors and cloud providers. Many of these providers are leveraging Docker for their container-native IaaS offerings. Additionally, the leading open source serverless frameworks utilize Docker container technology.

The launch of Docker in 2013 jump started a revolution in application development - by democratizing software containers. Docker developed a Linux container technology - one that is portable, flexible and easy to deploy. Docker open sourced libcontainer and partnered with a worldwide community of contributors to further its development. In June 2015, Docker donated the container image specification and runtime code now known as runc, to the Open Container Initiative (OCI) to help establish standardization as the container ecosystem grows and matures.

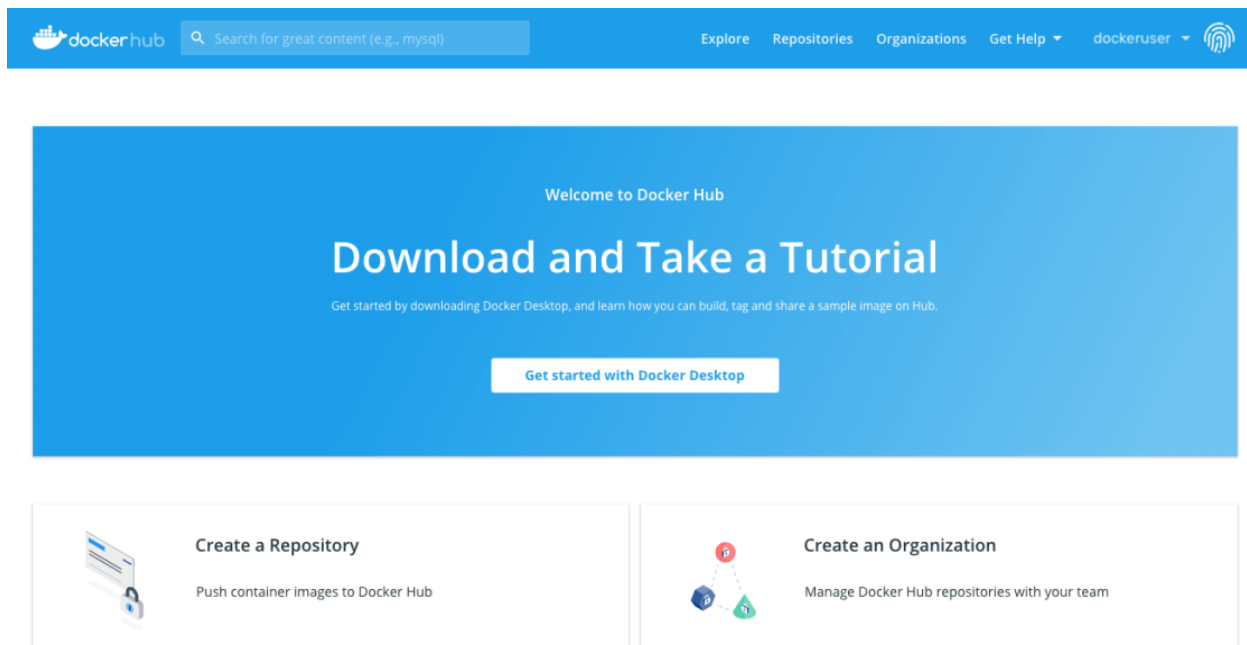
Following this evolution, Docker continues to give back with the containerd project, which Docker donated to the Cloud Native Computing Foundation (CNCF) in 2017. containerd is an industry-standard container runtime that leverages runc and was created with an emphasis

on simplicity, robustness and portability. containerd is the core container runtime of the Docker Engine.



PRACTICAL:

Install Docker on your machine



Create your project.

In order to create your first Docker application, I invite you to create a folder on your computer. It must contain the following two files:

A 'main.py' file (python file that will contain the code to be executed).

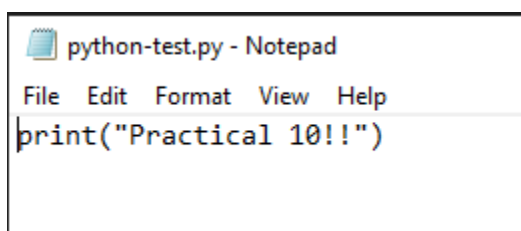
A 'Dockerfile' file (Docker file that will contain the necessary instructions to create the environment).

Edit the Python file

You can add the following code to the 'main.py' file:

```
#!/usr/bin/env python3  
print("Practical 10!!")
```

Nothing exceptional, but once you see "Docker is magic!" displayed in your terminal you will know that your Docker is working.



Create the Docker image

Once your code is ready and the Dockerfile is written, all you have to do is create your image to contain your application.

```
$ docker build -t python-test .
```

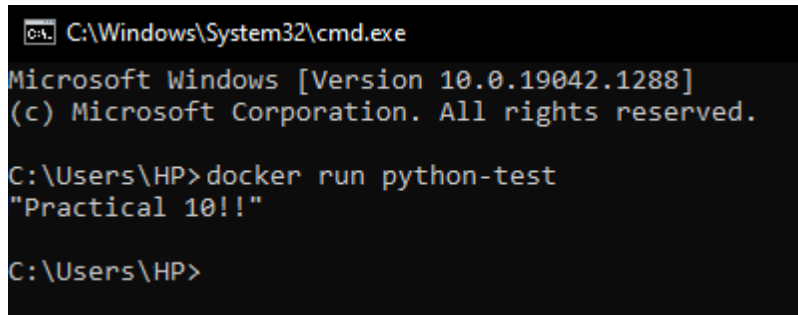
The '-t' option allows you to define the name of your image. In our case we have chosen 'python-test' but you can put what you want.

Run the Docker image

Once the image is created, your code is ready to be launched.

```
$ docker run python-test
```

You need to put the name of your image after 'docker run'.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>docker run python-test
"Practical 10!!"

C:\Users\HP>
```

CONCLUSION:

In this practical, we learnt about docker and container. We created a small python script and run it using docker image.