INDUS UNIVERSITY OF TECHNOLOGY & ENGINEERING

CLOUD COMPUTING (CE0723)

LAB PRACTICALS

Name: Your Name

Enrollment No: Enrol. Num.

Class : CE- A/B/C

Semester : 7

Submitted by: Your Name

Submitted to: Prof. Devarsh Shah

INDEX

SR. NO.	EXPERIMENT	SUBMISSION DATE	REMARKS
1	Sketch out and analyze the architecture of Cloudsim and identify different entities to understand the structure of cloudsim.		
2.	Create a scenario in cloudsim to create a datacenter along with one host. Also create one virtual machine with static configuration to run one cloudlet on it.		
3.	Illustrate a scenario in cloudsim to create one datacenter and one host. Also implement required virtual machines to run two cloudlets on it. Assume that cloudlets run in VMs with the same MIPS requirements. The cloudlets will take the same time to complete the execution.		
4.	Implement a datacenter with two hosts and run two cloudlets on it in cloudsim. Consider the cloudlets run in VMs with different MIPS requirements. The cloudlets will take different time to complete the execution depending on the requested VM performance.		
5.	Design a program in cloudsim to create two data centers with one host and run two cloudlets on it.		
6.	Construct a case in cloudsim to create two datacenters with one host each & run cloudlets of two users on them.		
7.	Make and perform scenarios to pause and resume the simulation in cloudsim, and create simulation entities (a Datacenter Broker) dynamically.		
8.	Organize a case in cloudsim for simulation entities (a Datacenter Broker) in run-time using a global manager entity (Global Broker).		
9.	Sketch out and analyze the architecture of Microsoft Azure.		
10.	OEP (Open Ended Problem)		

PRACTICAL - 1

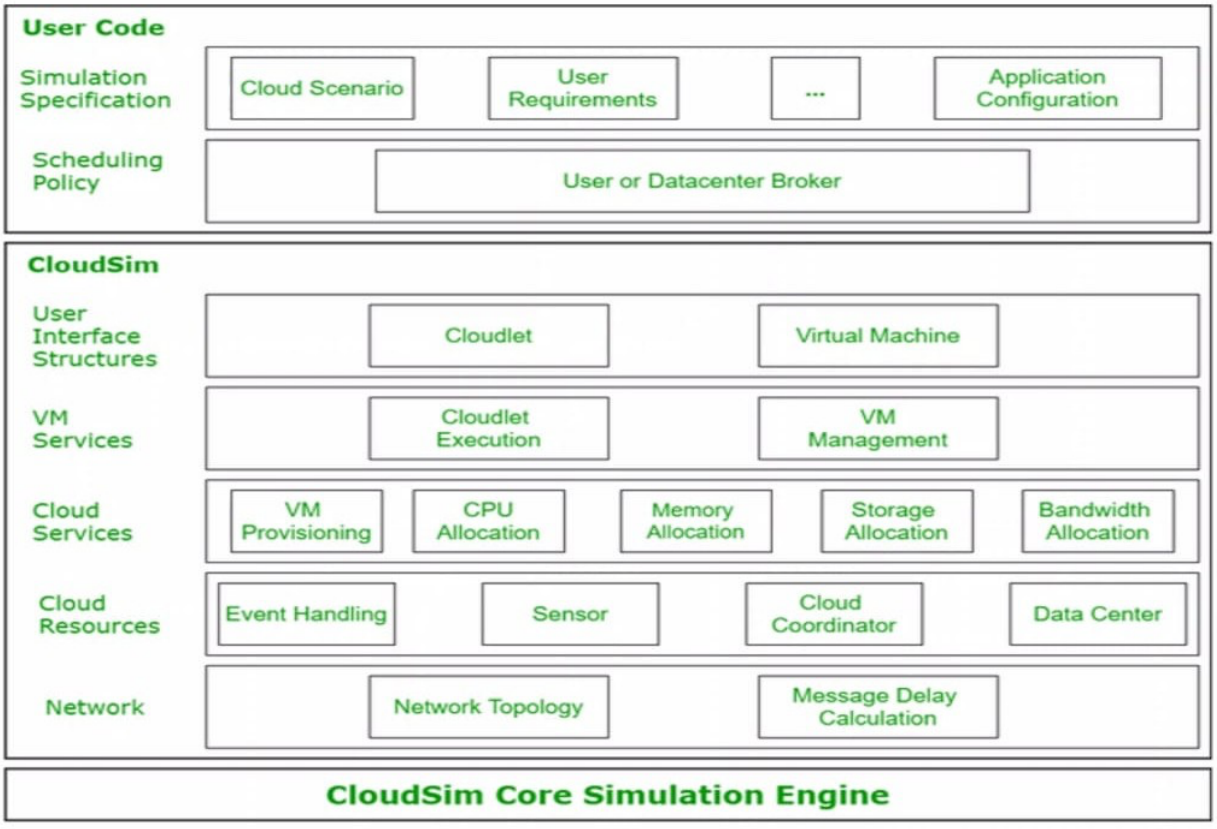
AIM: Sketch out and analyze the architecture of Cloudsim and identify different entities to understand the structure of cloudsim.

THEORY:

1A. What is Cloudsim?

CloudSim is an open-source framework, which is used to simulate cloud computing infrastructure and services. It is developed by the CLOUDS Lab organization and is written entirely in Java. It is used for modeling and simulating a cloud computing environment as a means for evaluating a hypothesis prior to software development in order to reproduce tests and results. By using Cloudsim, developers can focus on specific system design issues that they want to investigate, without getting concerned about details related to cloud-based infrastructures and services.

1B. Cloudsim Architecture



The above diagram demonstrates the layered architecture of the CloudSim Simulation Toolkit.
The **CloudSim Core simulation engine** provides support for modeling and simulation of virtualized Cloud-based data center environments including queuing and processing of events, creation of cloud system entities (like data center, host, virtual machines, brokers, services, etc.) communication between components and management of the simulation clock.

The **CloudSim layer** provides dedicated management interfaces for Virtual Machines, memory, storage, and bandwidth. Also, it manages the other fundamental issues, such as provisioning of hosts to Virtual Machines, managing application execution, and monitoring dynamic system state(e.g. Network topology, sensors, storage characteristics, etc), etc.

The **User Code layer** is a custom layer where the user writes their own code to redefine the characteristics of the stimulating environment as per their new research findings.

1C. Entities of CloudSim

1. **Cloudlet:** A cloudlet represents a task or workload that needs to be executed in the cloud. It encapsulates the computational requirements, data size, and other parameters of a specific job or application.

2. **Cloudlet Scheduler:** This entity is responsible for scheduling and allocating cloudlets to virtual machines (VMs) in the cloud. It determines the order and placement of cloudlets on VMS based on various scheduling policies.

3. **Datacenter:** A datacenter represents a physical infrastructure that hosts multiple virtual machines. It consists of physical hosts or servers, which are capable of running V MS, and networking components that enable communication among the hosts.

4. **Datacenter Broker:** The Datacenter Broker acts as an intermediary between cloud service users (clients) and the cloud data centers. It receives cloudlet requests from clients and submits them to appropriate data centers for execution.

5. **Virtual Machine (VM):** A VM is a software emulation of a physical computer within which cloudlets are executed. CloudSim allows the creation and management of multiple VMS in a data center.

6. **Host:** A host is a physical machine or server within a data center that can accommodate multiple VMs. It provides computing resources such as CPU, memory, and storage to the VMs.

7. **CloudSimEntity:** This is the base class for all entities in CloudSim. It provides common attributes and methods that are inherited by other entities.

8. **Data center characteristics:** This entity represents the characteristics or properties of a data center, such as its geographical location, power consumption, cost models, and network parameters.

9. **CloudletSchedulerSpaceShared:** It is a specific implementation of a cloudlet scheduler that allocates resources to cloudlets using a space-sharing policy, where each cloudlet is assigned a portion of resources based on its demands.

10. **CloudSimTags:** CloudSim uses tags to identify and categorize different types of events and messages exchanged among entities. CloudSimTags define the constants representing these tags.

PRACTICAL - 2

AIM: Create a scenario in cloudsim to create a datacenter along with one host. Also create one virtual machine with static configuration to run one cloudlet on it.

LO: Working of the data centers in a cloud environment.

THEORY:

2A. CloudSim Setup using Eclipse

Cloudsim is a standalone toolkit that allows us to simulate the cloud functionality on any commodity machine. This toolkit is developed using JAVA programming language and is an extensible framework.

Requirements for CloudSim Projects

- **Integrated Development Environment (IDE)**
  - Java Development Kit (JDK)
  - Eclipse/Netbeans IDE
- **Operating System (OS)**
  - Windows Vista (32 or 64 bit)
  - Windows 7 (32 or 64 bit)
  - Windows 8 and 10 (32 or 64 bit)
  - Windows XP (32 bit)
  - Mac OS X 10.5.8 or later (32 bit only)
  - Linux (Lucid Lynx and Ubuntu Linux)

Step 1

Java Development Kit (JDK): As the CloudSim simulation toolkit is a class library written in the Java programming language, therefore, the latest version of Java(JDK) should be installed on your machine, which can be downloaded from Oracle's Java portal.

Step 2

Download the **CloudSim 3.0.3 zip** file from **GitHub** and extract the folders within a folder **CloudSimSetup** in your computer storage.

Step 3

Download Apache Commons Math 3.6.1 bin-zip file and extract the folders within a folder **CloudSimSetup**. Cut & paste **commons-math3-3.6.1.jar** file in the folder **CloudSimSetup**. It is required as it provides a faster, more accurate, portable alternative to the regular Math and StrictMath classes for large-scale computation in java files.

Step 4

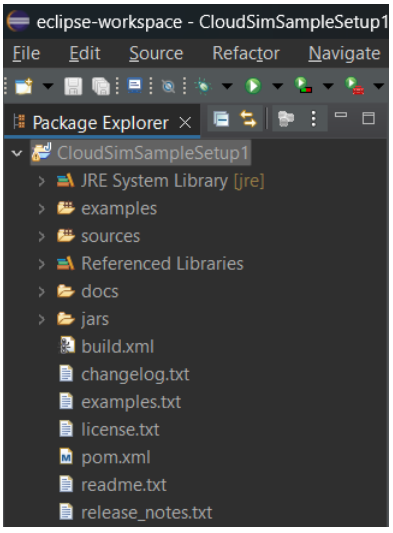
Open Eclipse IDE and create a new Java Project named CloudSimSampleSetup. Untick UseDefaultLocation. Select the folder path where CloudSim-3.0.3 is extracted till cloudsim-3.0.3. click on Next.

Step 5

In the JAVA Settings window, select tab Libraries. Click on Add External JARs from the right buttons. The JAR file commons-math3-3.6.1.jar from the extracted Apache Commons Math 3.6.1 folder needs to be added to the JAR files of CloudSim.

Step 6

Click on Finish. Now the CloudSim Environment has been set up in the Eclipse IDE.



From the Eclipse menu, add Window-->show view-->console. When the script is run, output is displayed in the console.

To implement AIM: A simple example showing how to create a datacenter with one host and run one cloudlet on it.

STEPS:

I. Initialize the CloudSim package.
II. Create Data Centers
III. Create Broker
IV. Create one virtual machine
V. Create one Cloudlet
VI. Starts the simulation
VII. Print results when simulation is over

<u>Here are the steps needed to create a Power-Datacenter:</u>

1. We need to create a list to store
2. A Machine contains one or more PEs or CPUs/Cores.
3. Create PEs and add these into a list.

4. Create Host with its id and list of PEs and add them to the list of machines
5. Create a Datacenter Characteristics 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).
6. Finally, we need to create a Power-Datacenter object.

2B. CONCLUSION:

We have successfully created a datacenter along with one host. Also created one virtual machine with static configuration to run one cloudlet on it.

2C. Cloud Computing Simulation using CloudSim

cloudsim lets the user design and imitate the cloud system entities. So, one should have more knowledge on modeling cloud simulation. For that, to know the classes with its supportive functionalities, list of some main classes are as follows:

To simulate the workloads

- org.cloudbus.cloudsim ➔ "Cloudlet.java"

To simulate the regions and datacenters

- org.cloudbus.cloudsim ➔ "Datacenter.java"

To simulate the task offloading and policy-related services

- org.cloudbus.cloudsim ➔ "VmAllocationPolicy.java", "CloudletScheduler.java", "DatacenterBroker.java", etc

Similarly, discrete event simulation engine is used to collect the workloads from heterogeneous hardware

CODE:

```
package org.cloudbus.cloudsim.examples;
import java.text.DecimalFormat;
import java.util.*;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample1 {

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

    /** The vmlist. */
    private static List < Vm > vmlist;

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

```
     * @param args the args
     */
    @SuppressWarnings("unused")
    public static void main(String[] args) {

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

        try {
       // First step: Initialize the CloudSim package. It should be called before creating any entities.
            int num_user = 1; // number of cloud users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false; // mean trace events

            // Initialize the CloudSim library
            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
           // Datacenters are the resource providers in CloudSim. We need at least 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 = 1000;
            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 vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

            // add the VM to the vmList
            vmlist.add(vm);
```

```
            // submit vm list to the broker
            broker.submitVmList(vmlist);

            // Fifth step: Create one Cloudlet
            cloudletList = new ArrayList < Cloudlet > ();

            // Cloudlet properties
            int id = 0;
            long length = 400000;
            long fileSize = 300;
            long outputSize = 300;
            UtilizationModel utilizationModel = new UtilizationModelFull();

             Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
            cloudlet.setUserId(brokerId);
            cloudlet.setVmId(vmid);

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

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

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

            CloudSim.stopSimulation();

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

            Log.printLine("CloudSimExample1 finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Unwanted errors happen");
        }
    }

    /**
     * Creates the datacenter.
     *
     * @param name the name
```

```
     *
     * @return the datacenter
     */
    private static Datacenter createDatacenter(String name) {

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store our machine
        List < Host > hostList = new ArrayList < Host > ();

        // 2. A Machine contains one or more PEs or CPUs/Cores.
        // In this example, it will have only one core.
        List < Pe > peList = new ArrayList < Pe > ();
        int mips = 1000;

        // 3. Create PEs and add these into a list.
          peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS
Rating
        // 4. Create Host with its id and list of PEs and add them to the list of machines
        int hostId = 0;
        int ram = 2048; // host memory (MB)
        long storage = 1000000; // host storage
        int bw = 10000;

        hostList.add(
           new Host(
              hostId,
              new RamProvisionerSimple(ram),
              new BwProvisionerSimple(bw),
              storage,
              peList,
              new VmSchedulerTimeShared(peList)
           )
        ); // This is our machine

          // 5. Create a DatacenterCharacteristics object that stores the properties of a data center:
architecture, OS, list of Machines, allocation policy: time- or space-shared, time zone and its
price (G$/Pe time unit).
        String arch = "x86"; // system architecture
        String os = "Linux"; // operating system
        String vmm = "Xen";
        double time_zone = 10.0; // time zone this resource located
        double cost = 3.0; // the cost of using processing in this resource
        double costPerMem = 0.05; // the cost of using memory in this resource
        double costPerStorage = 0.001; // the cost of using storage in this resource
```

```
        double costPerBw = 0.0; // the cost of using bw in this resource
        LinkedList < Storage > storageList = new LinkedList < Storage > (); // we are not adding SAN devices by now

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

        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return datacenter;
    }

    // We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according to the specific rules of the simulated scenario
    /**
     * Creates the broker.
     *
     * @return the datacenter broker
     */
    private static DatacenterBroker createBroker() {
        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return broker;
    }

    /**
     * Prints the Cloudlet objects.
     *
     * @param list list of Cloudlets
     */
    private static void printCloudletList(List < Cloudlet > list) {
```

```
        int size = list.size();
        Cloudlet cloudlet;

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

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

           if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
              Log.print("SUCCESS");
               Log.printLine(indent + indent + cloudlet.getResourceId() + indent + indent + indent +
cloudlet.getVmId() + indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent +
indent + dft.format(cloudlet.getExecStartTime()) + indent + indent +
dft.format(cloudlet.getFinishTime()));
           }
        }
     }
}
```

OUTPUT:

```
Starting CloudSimExample1...
Initialising...
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
400.1: Broker: Cloudlet 0 received
400.1: Broker: All Cloudlets executed. Finishing...
400.1: 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        400        0.1        400.1
CloudSimExample1 finished!
```

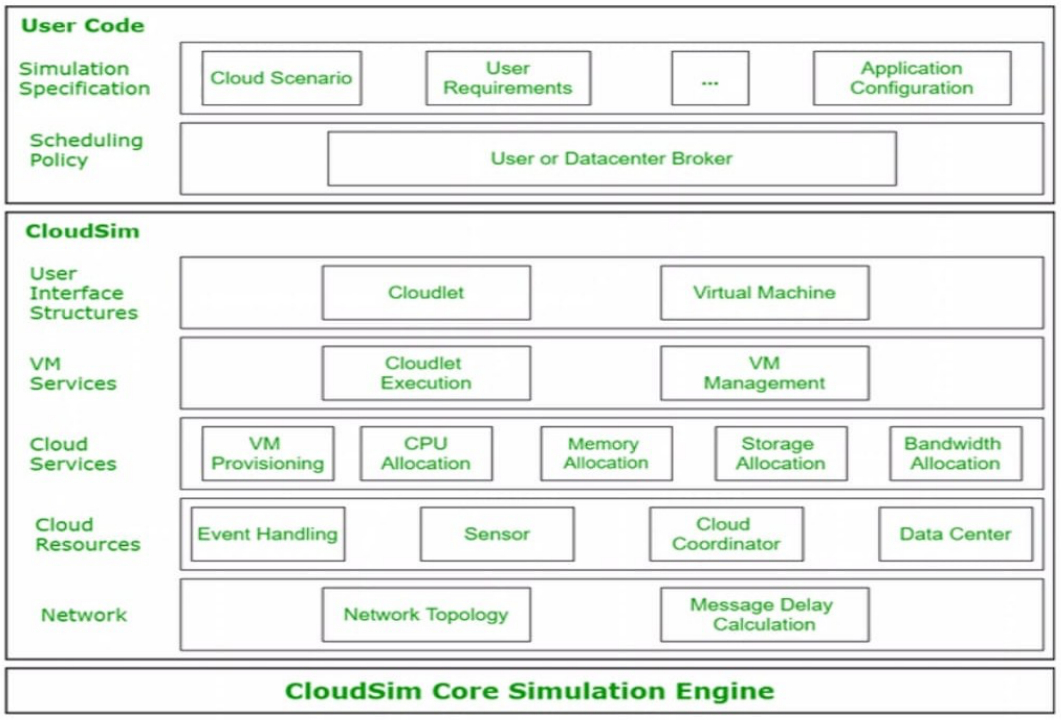
PRACTICAL - 3

AIM: Illustrate a scenario in cloudsim to create one datacenter and one host. Also implement required virtual machines to run two cloudlets on it. Assume that cloudlets run in VMs with the same MIPS requirements. The cloudlets will take the same time to complete the execution.

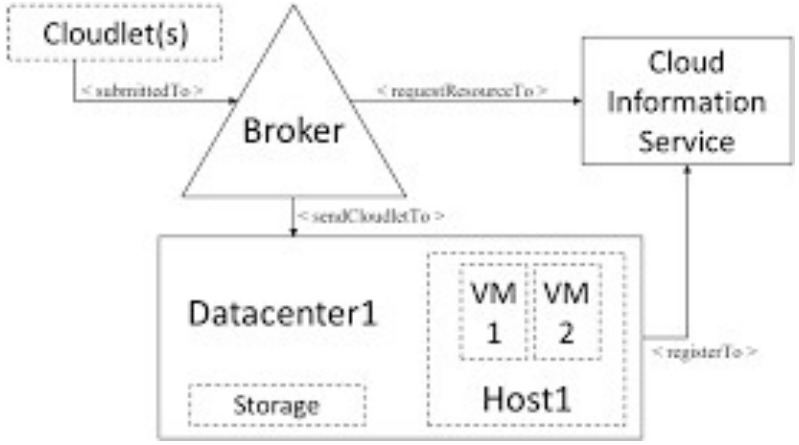
LO: Understanding the cloudlets requests in a cloud scenario

THEORY:

3A. CloudSim Architecture –



3B. Simulation flow –



CloudSim - General steps to follow

- Initiate the CloudSim simulation.
- Create a datacenter.
- Create a datacenter broker.
- Create VMs/cloudlet and add it to respective lists.
- Submit vm and cloudlet list to broker.
- Start simulation.
- Stop simulation.
- Print the end results.

1. Datacenter:

This class models the core infrastructure-level services (i.e. hardware) that are offered by Cloud providers (Amazon, Azure, and App Engine). ***It encapsulates a set of hosts(resembling server machine model) instances that can either be homogeneous or heterogeneous concerning their hardware configurations (memory, cores, capacity, and storage).*** Also, every Datacenter component takes care of generalized application provisioning that enforces a set of policies for the allocation of bandwidth, memory, and storage devices to hosts and their related VMs.

```
// Create Datacenters
// Datacenters are the resource providers in CloudSim.
// At Least one of them is needed to run a CloudSim simulation.
Datacenter datacenter0 = createDatacenter("Datacenter_0");
/**
 * Creates the datacenter.
 *
 * @param name the name
 *
 * @return the datacenter
 */
private static Datacenter createDatacenter(String name) {

    // Here are the steps needed to create a PowerDatacenter:
    // 1. We need to create a list to store our machine
    List < Host > hostList = new ArrayList < Host > ();

    // 2. A Machine contains one or more PEs or CPUs/Cores.
    // In this example, it will have only one core.
    List < Pe > peList = new ArrayList < Pe > ();
    int mips = 1000;

    // 3. Create PEs and add these into a list.
    peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS
Rating
```

```
    // 4. Create Host with its id and list of PEs and add them to the list of machines
    int hostId = 0;
    int ram = 2048; //host memory (MB)
    long storage = 1000000; //host storage
    int bw = 10000;
    hostList.add(
        new Host(
            hostId,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList,
            new VmSchedulerTimeShared(peList)
        )
    ); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the properties of a data center:
architecture, OS, list of Machines, allocation policy: time- or space-shared, time zone and its
price (G$/Pe time unit).
    String arch = "x86"; // system architecture
    String os = "Linux"; // operating system
    String vmm = "Xen";
    double time_zone = 10.0; // time zone this resource located
    double cost = 3.0; // the cost of using processing in this resource
    double costPerMem = 0.05; // the cost of using memory in this resource
    double costPerStorage = 0.001; // the cost of using storage in this resource
    double costPerBw = 0.0; // the cost of using bw in this resource
    LinkedList < Storage > storageList = new LinkedList < Storage > (); //we are not adding SAN
devices by now
    DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
        arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
    // 6. Finally, we need to create a PowerDatacenter object.
    Datacenter datacenter = null;
    try {
        datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return datacenter;
}
```

2. Datacenter Broker or Cloud Broker:

This class model is a broker, which is responsible for mediating negotiations between SaaS and Cloud providers, and such negotiations are driven by QoS requirements. The broker class acts on behalf of applications. Its prime role is to query the CIS to discover suitable resources/services and undertake negotiations for the allocation of resources/services that can fulfill the application's QoS needs. This class must be extended for evaluating and testing custom brokering policies.

```
// Create Broker
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();
// We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according to the specific rules of the simulated scenario
/**
 * Creates the broker.
 *
 * @return the datacenter broker
 */
private static DatacenterBroker createBroker() {
    DatacenterBroker broker = null;
    try {
        broker = new DatacenterBroker("Broker");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return broker;
}
```

3. Cloudlet:

This class model(define specific attributes such as length of instruction, input/output file size, no of processor required, etc) the Cloud-based application services (program-based tasks) such as content delivery, social networking, etc. CloudSim implements the complexity of an application in terms of its computational requirements. As a developer, we know that every individual executable application/service workload has a pre-defined instruction length and requires certain network data flow (both pre and post-fetches) overhead that it needs to undertake during its life cycle. This class allows modeling all the above-said requirements.

```
// Create CloudletList
cloudletList = new ArrayList<Cloudlet>();
// Cloudlet properties
// Create one Cloudlet
int id = 0;
long length = 400000;
long fileSize = 300;
long outputSize = 300;
```

```
UtilizationModel utilizationModel = new UtilizationModelFull();
Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
cloudlet.setUserId(brokerId);
cloudlet.setVmId(vmid);
// add the cloudlet to the list
cloudletList.add(cloudlet);
// submit cloudlet list to the broker
broker.submitCloudletList(cloudletList);
```

4. VM

This class model is a Virtual Machine (VM), which is managed and hosted by a Cloud host component. Every VM component has access to a component that stores the following characteristics related to a VM (i.e.) accessible memory, processor, storage size, and the VM's internal provisioning policy that is extended from an abstract class called the CloudletScheduler.

```
// VM description
int vmid = 0; // virtual machine ID
int mips = 1000; // million instructions per second
long size = 10000; // image size (MB)
int ram = 512; // vm memory (MB)
long bw = 1000; //bandwidth megabits/sec
int pesNumber = 1; // number of cpus
String vmm = "Xen"; // VMM name
// create VM
Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
// add the VM to the vmList
vmlist.add(vm);
// submit vm list to the broker
broker.submitVmList(vmlist);
```

3C. CONCLUSION

We have successfully created a datacenter along with one host. Also created two virtual machines with static configuration to run two cloudlets on it.

CODE:

```
package org.cloudbus.cloudsim.examples;
import java.text.DecimalFormat;
import java.util.*;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample2 {

  /** 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.printLine("Starting CloudSimExample2...");
    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
      @SuppressWarnings("unused")
      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 two VMs
    Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
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;
pesNumber = 1;
long length = 250000;
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: Starts the simulation
        CloudSim.startSimulation();

        // Final step: Print results when simulation is over
        List < Cloudlet > newList = broker.getCloudletReceivedList();
        CloudSim.stopSimulation();
        printCloudletList(newList);
        Log.printLine("CloudSimExample2 finished!");
      } catch (Exception e) {
        e.printStackTrace();
        Log.printLine("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.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
       "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent +
"Finish Time");

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

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

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

OUTPUT:

```
Starting CloudSimExample2...
Initialising...
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
0.0: 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
1000.1: Broker: Cloudlet 0 received
1000.1: Broker: Cloudlet 1 received
1000.1: Broker: All Cloudlets executed. Finishing...
1000.1: Broker: Destroying VM #0
1000.1: 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...
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        1000        0.1        1000.1
    1        SUCCESS        2            1        1000        0.1        1000.1
CloudSimExample2 finished!
```

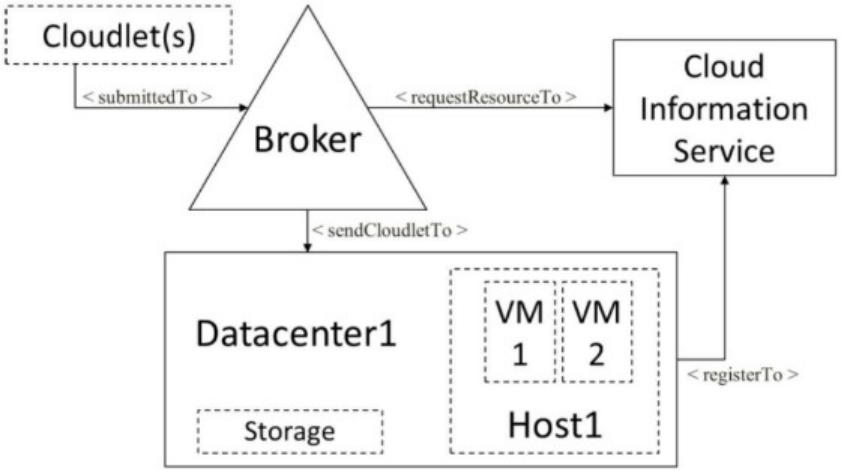
PRACTICAL - 4

AIM: Implement a datacenter with two hosts and run two cloudlets on it in cloudsim. Consider the cloudlets run in VMs with different MIPS requirements. The cloudlets will take different time to complete the execution depending on the requested VM performance.

LO: Analyzing the cloud performance in the distinct scenario.

THEORY:

- **Simulation flow for basic scenario**



1. Datacenter:
This class models the core infrastructure-level services (i.e. hardware) that are offered by Cloud providers (Amazon, Azure, and App Engine). It encapsulates a set of hosts(resembling server machine model) instances that can either be homogeneous or heterogeneous concerning their hardware configurations (memory, cores, capacity, and storage). Also, every Datacenter component takes care of generalized application provisioning that enforces a set of policies for the allocation of bandwidth, memory, and storage devices to hosts and their related VMs.

2. Datacenter Broker or Cloud Broker:
This class model is a broker, which is responsible for mediating negotiations between SaaS and Cloud providers, and such negotiations are driven by QoS requirements. The broker class acts on behalf of applications. Its prime role is to query the CIS to discover suitable resources/services and undertake negotiations for the allocation of resources/services that can fulfill the application's QoS needs. This class must be extended for evaluating and testing custom brokering policies.

3. Cloudlet:
This class model(define specific attributes such as length of instruction, input/output file size, no of processor required, etc) the Cloud-based application services (program-based tasks) such as

content delivery, social networking, etc. CloudSim implements the complexity of an application in terms of its computational requirements. As a developer, we know that every individual executable application/service workload has a pre-defined instruction length and requires certain network data flow (both pre and post-fetches) overhead that it needs to undertake during its life cycle. this class allows modeling all the above-said requirements

4. VM:

This class models a Virtual Machine (VM), which is managed and hosted by a Cloud host component. Every VM component has access to a component that stores the following characteristics related to a VM (i.e.) accessible memory, processor, storage size, and the VM's internal provisioning policy that is extended from an abstract class called the CloudletScheduler.

CONCLUSION:

We have successfully created a datacenter along with two hosts. Also created two virtual machines with static configuration to run two cloudlets on it. The cloudlets run in VMs with different MIPS requirements. The cloudlets will take different time to complete the execution depending on the requested VM performance.

CODE:

```
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.*;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample3 {

 /** 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.printLine("Starting CloudSimExample3...");
```

```
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
    @SuppressWarnings("unused")
    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 = 2048; //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 * 2, 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: Starts the simulation
CloudSim.startSimulation();

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

CloudSim.stopSimulation();

printCloudletList(newList);
Log.printLine("CloudSimExample3 finished!");
```

```
    } catch (Exception e) {
      e.printStackTrace();
      Log.printLine("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 Hosts 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 first machine

    //create another machine in the Data center
    List < Pe > peList2 = new ArrayList < Pe > ();

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

```
    hostId++;

    hostList.add(
      new Host(
        hostId,
        new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList2,
        new VmSchedulerTimeShared(peList2)
      )
    ); // This is our second 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.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
        "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent +
"Finish Time");

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

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

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

OUTPUT:

```
Starting CloudSimExample3...
Initialising...
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
0.0: 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 #1
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
80.1: Broker: Cloudlet 1 received
160.1: Broker: Cloudlet 0 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...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    1        SUCCESS        2            1        80        0.1        80.1
    0        SUCCESS        2            0        160        0.1        160.1
CloudSimExample3 finished!
```

PRACTICAL - 5

AIM: Design a program in cloudsim to create two data centers with one host and run two cloudlets on it.

LO: Multiple cloudlet execution in the cloud.

THEORY:

1. Datacenter:
This class represents a datacenter in the cloud computing environment. It encapsulates the characteristics of a datacenter, such as its hosts, resource provisioning policies, and other attributes.

2. DatacenterBroker:
This class Represents a broker entity responsible for managing the interaction between cloud users and data centers. It submits VMs and cloudlets to datacenters and handles their scheduling.

3. VM:
Represents a virtual machine in the cloud environment. It encapsulates the attributes of a VM, including MIPS (Million Instructions Per Second), RAM, storage, and scheduling policy.

4. Cloudlet:
This class represents a cloudlet, which is a task or workload that is executed on a VM. It encapsulates information about the cloudlet's attributes, such as length, file size, and utilization models.

5. Various Provisioner Classes (PeProvisionerSimple, RamProvisionerSimple, BwProvisionerSimple):
These classes are used to provision resources (CPU, RAM, bandwidth) to PEs, hosts, and VMs, respectively.

CONCLUSION:
This code provides a simple demonstration of creating data centers, VMs, and cloudlets within a CloudSim simulation environment and observing their execution. It's a basic example that showcases the fundamental components of CloudSim and how they interact in a cloud computing simulation scenario.

CODE:

```
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.*;

import org.cloudbus.cloudsim.*
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*

 public class CloudSimExample4 {

   /** 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.printLine("Starting CloudSimExample4...");

     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 GridSim 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");

       // 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());

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: Starts the simulation
        CloudSim.startSimulation();

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

        CloudSim.stopSimulation();

        printCloudletList(newList);

        Log.printLine("CloudSimExample4 finished!");
      } catch (Exception e) {
        e.printStackTrace();
        Log.printLine("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 VMAllocatonPolicy 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,
    peList,
    new VmSchedulerSpaceShared(peList)
  )
); // This is our first 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.printLine();
      Log.printLine("========== OUTPUT ==========");
      Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
          "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent +
"Finish Time");

      DecimalFormat dft = new DecimalFormat("###.##");
      for (int i = 0; i < size; i++) {
        cloudlet = list.get(i);
        Log.print(indent + cloudlet.getCloudletId() + indent + indent);
```

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

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

OUTPUT:

```
Starting CloudSimExample4...
Initialising...
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
[VmScheduler.vmCreate] Allocation of VM #1 to Host #0 failed by MIPS
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Creation of VM #1 failed in Datacenter #2
0.1: Broker: Trying to Create VM #1 in Datacenter_1
0.2: Broker: VM #1 has been created in Datacenter #3, Host #0
0.2: Broker: Sending cloudlet 0 to VM #0
0.2: Broker: Sending cloudlet 1 to VM #1
160.2: Broker: Cloudlet 0 received
160.2: Broker: Cloudlet 1 received
160.2: Broker: All Cloudlets executed. Finishing...
160.2: Broker: Destroying VM #0
160.2: 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        0.2        160.2
    1        SUCCESS        3            1        160        0.2        160.2
CloudSimExample4 finished!
```

PRACTICAL - 6

AIM: Construct a case in cloudsim to create two datacenters with one host each and run cloudlets of two users on them.

LO: Multiple data center approach and its performance on the cloud.

CODE:

```
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.*;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample5 {

 /** The cloudlet lists. */
 private static List < Cloudlet > cloudletList1;
 private static List < Cloudlet > cloudletList2;

 /** The vmlists. */
 private static List < Vm > vmlist1;
 private static List < Vm > vmlist2;

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

  Log.printLine("Starting CloudSimExample5...");

  try {
   // First step: Initialize the CloudSim package. It should be called before creating any entities.
   int num_user = 2; // 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");

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

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

//Fourth step: Create one virtual machine for each broker/user
vmlist1 = new ArrayList < Vm > ();
vmlist2 = 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: 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());

//add the VMs to the vmlists
vmlist1.add(vm1);
vmlist2.add(vm2);

//submit vm list to the broker
broker1.submitVmList(vmlist1);
broker2.submitVmList(vmlist2);

//Fifth step: Create two Cloudlets
```

```
    cloudletList1 = new ArrayList < Cloudlet > ();
    cloudletList2 = 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);

    //add the cloudlets to the lists: each cloudlet belongs to one user
    cloudletList1.add(cloudlet1);
    cloudletList2.add(cloudlet2);

    //submit cloudlet list to the brokers
    broker1.submitCloudletList(cloudletList1);
    broker2.submitCloudletList(cloudletList2);

    // 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();

    CloudSim.stopSimulation();

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

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

    Log.printLine("CloudSimExample5 finished!");
  } catch (Exception e) {
    e.printStackTrace();
```

```
    Log.printLine("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 VMAllocatonPolicy 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,
     peList,
     new VmSchedulerSpaceShared(peList)
    )
  ); // This is our first 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.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
```

```
      "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent +
"Finish Time");
   DecimalFormat dft = new DecimalFormat("###.##");
   for (int i = 0; i < size; i++) {
    cloudlet = list.get(i);
    Log.print(indent + cloudlet.getCloudletId() + indent + indent);
    if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
    Log.print("SUCCESS");
         Log.printLine(indent + indent + cloudlet.getResourceId() + indent + indent + indent +
cloudlet.getVmId() +
               indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime()) +
        indent + indent + dft.format(cloudlet.getFinishTime()));
    }
   }
  }
}
```

OUTPUT:

```
Starting CloudSimExample5...
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Datacenter_1 is starting...
Broker1 is starting...
Broker2 is starting...
Entities started.
0.0: Broker1: Cloud Resource List received with 2 resource(s)
0.0: Broker2: Cloud Resource List received with 2 resource(s)
0.0: Broker1: Trying to Create VM #0 in Datacenter_0
0.0: Broker2: Trying to Create VM #0 in Datacenter_0
[VmScheduler.vmCreate] Allocation of VM #0 to Host #0 failed by MIPS
0.1: Broker1: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker1: Sending cloudlet 0 to VM #0
0.1: Broker2: Creation of VM #0 failed in Datacenter #2
0.1: Broker2: Trying to Create VM #0 in Datacenter_1
0.2: Broker2: VM #0 has been created in Datacenter #3, Host #0
0.2: Broker2: Sending cloudlet 0 to VM #0
160.1: Broker1: Cloudlet 0 received
160.1: Broker1: All Cloudlets executed. Finishing...
160.1: Broker1: Destroying VM #0
Broker1 is shutting down...
160.2: Broker2: Cloudlet 0 received
160.2: Broker2: All Cloudlets executed. Finishing...
160.2: Broker2: Destroying VM #0
Broker2 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...
Broker1 is shutting down...
Broker2 is shutting down...
Simulation completed.
Simulation completed.
=============> User 4
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        160        0.1        160.1
=============> User 5
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        3            0        160        0.2        160.2
CloudSimExample5 finished!
```

PRACTICAL - 7

AIM: Make and perform scenario to pause and resume the simulation in cloudsim, and create simulation entities (a Datacenter Broker) dynamically

LO: Understanding the Broker concept with example

CODE:

```
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.*;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample7 {

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

 /** The vmlist. */
 private static List < Vm > vmlist;

 private static List < Vm > createVM(int userId, int vms, int idShift) {
  //Creates a container to store VMs. This list is passed to the broker later
  LinkedList < Vm > list = new LinkedList < Vm > ();

  //VM Parameters
  long size = 10000; //image size (MB)
  int ram = 512; //vm memory (MB)
  int mips = 250;
  long bw = 1000;
  int pesNumber = 1; //number of cpus
  String vmm = "Xen"; //VMM name

  //create VMs
  Vm[] vm = new Vm[vms];
  for (int i = 0; i < vms; i++) {
        vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
    list.add(vm[i]);
  }
```

```
    return list;
  }

  private static List < Cloudlet > createCloudlet(int userId, int cloudlets, int idShift) {
    // Creates a container to store Cloudlets
    LinkedList < Cloudlet > list = new LinkedList < Cloudlet > ();

    //cloudlet parameters
    long length = 40000;
    long fileSize = 300;
    long outputSize = 300;
    int pesNumber = 1;
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for (int i = 0; i < cloudlets; i++) {
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
      // setting the owner of these Cloudlets
      cloudlet[i].setUserId(userId);
      list.add(cloudlet[i]);
    }
    return list;
  }

  ////////////////////////// STATIC METHODS ///////////////////////
  /**
   * Creates main() to run this example
   */
  public static void main(String[] args) {
    Log.printLine("Starting CloudSimExample7...");

    try {
      // First step: Initialize the CloudSim package. It should be called before creating any entities.
      int num_user = 2; // number of grid 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");

    //Third step: Create Broker
    DatacenterBroker broker = createBroker("Broker_0");
    int brokerId = broker.getId();

    //Fourth step: Create VMs and Cloudlets and send them to broker
    vmlist = createVM(brokerId, 5, 0); //creating 5 vms
    cloudletList = createCloudlet(brokerId, 10, 0); // creating 10 cloudlets
    broker.submitVmList(vmlist);
    broker.submitCloudletList(cloudletList);

    // A thread that will create a new broker at 200 clock time
    Runnable monitor = new Runnable() {
     @Override
     public void run() {
      CloudSim.pauseSimulation(200);
      while (true) {
       if (CloudSim.isPaused()) {
        break;
       }
       try {
        Thread.sleep(100);
       } catch (InterruptedException e) {
        e.printStackTrace();
       }
      }

      Log.printLine("\n\n\n" + CloudSim.clock() + ": The simulation is paused for 5 sec \n\n");

      try {
       Thread.sleep(5000);
      } catch (InterruptedException e) {
       e.printStackTrace();
      }

      DatacenterBroker broker = createBroker("Broker_1");
      int brokerId = broker.getId();
```

```
        //Create VMs and Cloudlets and send them to broker
        vmlist = createVM(brokerId, 5, 100); //creating 5 vms
        cloudletList = createCloudlet(brokerId, 10, 100); // creating 10 cloudlets
        broker.submitVmList(vmlist);
        broker.submitCloudletList(cloudletList);
        CloudSim.resumeSimulation();
      }
    };

    new Thread(monitor).start();
    Thread.sleep(1000);

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

    // Final step: Print results when simulation is over
    List < Cloudlet > newList = broker.getCloudletReceivedList();
    CloudSim.stopSimulation();
    printCloudletList(newList);
    Log.printLine("CloudSimExample7 finished!");
   } catch (Exception e) {
    e.printStackTrace();
    Log.printLine("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 one or more Machines
   List < Host > hostList = new ArrayList < Host > ();

    // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, you should create a list
to store these PEs before creating a Machine.
   List < Pe > peList1 = new ArrayList < Pe > ();
   int mips = 1000;

   // 3. Create PEs and add these into the list. for a quad-core machine, a list of 4 PEs is required:
    peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS
Rating
   peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
   peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
   peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

   //Another list, for a dual-core machine
```

```
List < Pe > peList2 = new ArrayList < Pe > ();
peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int hostId = 0;
int ram = 16384; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

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

hostId++;
hostList.add(
new Host(
    hostId,
    new RamProvisionerSimple(ram),
    new BwProvisionerSimple(bw),
    storage,
    peList2,
    new VmSchedulerTimeShared(peList2)
  )
); // Second 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 = 0.1; // the cost of using storage in this resource
double costPerBw = 0.1; // 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(String name) {
    DatacenterBroker broker = null;
    try {
      broker = new DatacenterBroker(name);
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
    return broker;
  }

  /**
   * Prints the Cloudlet objects
   * @param list list of Cloudlets
   */
  private static void printCloudletList(List < Cloudlet > list) {
    int size = list.size();
    Cloudlet cloudlet;
    String indent = "    ";
    Log.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
        "Data center ID" + indent + "VM ID" + indent + indent + "Time" + indent + "Start Time" + indent + "Finish Time");
    DecimalFormat dft = new DecimalFormat("###.##");
    for (int i = 0; i < size; i++) {
```

```
    cloudlet = list.get(i);
    Log.print(indent + cloudlet.getCloudletId() + indent + indent);

    if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
      Log.print("SUCCESS");
        Log.printLine(indent + indent + cloudlet.getResourceId() + indent + indent + indent +
cloudlet.getVmId() +
        indent + indent + indent + dft.format(cloudlet.getActualCPUTime()) +
          indent + indent + dft.format(cloudlet.getExecStartTime()) + indent + indent + indent +
dft.format(cloudlet.getFinishTime()));
    }
   }
  }
}
```

OUTPUT:

```
Starting CloudSimExample7...
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Datacenter_1 is starting...
Broker_0 is starting...
Entities started.
0.0: Broker_0: Cloud Resource List received with 2 resource(s)
0.0: Broker_0: Trying to Create VM #0 in Datacenter_0
0.0: Broker_0: Trying to Create VM #1 in Datacenter_0
0.0: Broker_0: Trying to Create VM #2 in Datacenter_0
0.0: Broker_0: Trying to Create VM #3 in Datacenter_0
0.0: Broker_0: Trying to Create VM #4 in Datacenter_0
0.1: Broker_0: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker_0: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker_0: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker_0: VM #3 has been created in Datacenter #2, Host #1
0.1: Broker_0: VM #4 has been created in Datacenter #2, Host #0
0.1: Broker_0: Sending cloudlet 0 to VM #0
0.1: Broker_0: Sending cloudlet 1 to VM #1
0.1: Broker_0: Sending cloudlet 2 to VM #2
0.1: Broker_0: Sending cloudlet 3 to VM #3
0.1: Broker_0: Sending cloudlet 4 to VM #4
0.1: Broker_0: Sending cloudlet 5 to VM #0
0.1: Broker_0: Sending cloudlet 6 to VM #1
0.1: Broker_0: Sending cloudlet 7 to VM #2
0.1: Broker_0: Sending cloudlet 8 to VM #3
0.1: Broker_0: Sending cloudlet 9 to VM #4



200.0: The simulation is paused for 5 sec


Adding: Broker_1
Broker_1 is starting...
200.0: Broker_1: Cloud Resource List received with 2 resource(s)
200.0: Broker_1: Trying to Create VM #100 in Datacenter_0
200.0: Broker_1: Trying to Create VM #101 in Datacenter_0
200.0: Broker_1: Trying to Create VM #102 in Datacenter_0
200.0: Broker_1: Trying to Create VM #103 in Datacenter_0
200.0: Broker_1: Trying to Create VM #104 in Datacenter_0
200.1: Broker_1: VM #100 has been created in Datacenter #2, Host #1
200.1: Broker_1: VM #101 has been created in Datacenter #2, Host #0
200.1: Broker_1: VM #102 has been created in Datacenter #2, Host #1
```

```
200.1: Broker_1: Sending cloudlet 100 to VM #100
200.1: Broker_1: Sending cloudlet 101 to VM #101
200.1: Broker_1: Sending cloudlet 102 to VM #102
200.1: Broker_1: Sending cloudlet 103 to VM #103
200.1: Broker_1: Sending cloudlet 104 to VM #104
200.1: Broker_1: Sending cloudlet 105 to VM #100
200.1: Broker_1: Sending cloudlet 106 to VM #101
200.1: Broker_1: Sending cloudlet 107 to VM #102
200.1: Broker_1: Sending cloudlet 108 to VM #103
200.1: Broker_1: Sending cloudlet 109 to VM #104
320.096: Broker_0: Cloudlet 0 received
320.096: Broker_0: Cloudlet 5 received
320.096: Broker_0: Cloudlet 1 received
320.096: Broker_0: Cloudlet 6 received
320.096: Broker_0: Cloudlet 2 received
320.096: Broker_0: Cloudlet 7 received
320.096: Broker_0: Cloudlet 4 received
320.096: Broker_0: Cloudlet 9 received
320.096: Broker_0: Cloudlet 3 received
320.096: Broker_0: Cloudlet 8 received
320.096: Broker_0: All Cloudlets executed. Finishing...
320.096: Broker_0: Destroying VM #0
320.096: Broker_0: Destroying VM #1
320.096: Broker_0: Destroying VM #2
320.096: Broker_0: Destroying VM #3
320.096: Broker_0: Destroying VM #4
Broker_0 is shutting down...
519.996: Broker_1: Cloudlet 101 received
519.996: Broker_1: Cloudlet 106 received
519.996: Broker_1: Cloudlet 103 received
519.996: Broker_1: Cloudlet 108 received
519.996: Broker_1: Cloudlet 100 received
519.996: Broker_1: Cloudlet 105 received
519.996: Broker_1: Cloudlet 102 received
519.996: Broker_1: Cloudlet 107 received
519.996: Broker_1: Cloudlet 104 received
519.996: Broker_1: Cloudlet 109 received
519.996: Broker_1: All Cloudlets executed. Finishing...
519.996: Broker_1: Destroying VM #100
519.996: Broker_1: Destroying VM #101
519.996: Broker_1: Destroying VM #102
519.996: Broker_1: Destroying VM #103
519.996: Broker_1: Destroying VM #104
Broker_1 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_0 is shutting down...
Broker_1 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            320        0.1            320.1
    5        SUCCESS        2            0            320        0.1            320.1
    1        SUCCESS        2            1            320        0.1            320.1
    6        SUCCESS        2            1            320        0.1            320.1
    2        SUCCESS        2            2            320        0.1            320.1
    7        SUCCESS        2            2            320        0.1            320.1
    4        SUCCESS        2            4            320        0.1            320.1
    9        SUCCESS        2            4            320        0.1            320.1
    3        SUCCESS        2            3            320        0.1            320.1
    8        SUCCESS        2            3            320        0.1            320.1
CloudSimExample7 finished!
```

PRACTICAL - 8

AIM: Organize a case in cloudsim for simulation entities (a Datacenter Broker) in run-time using a global manager entity (Global Broker)

LO: Understanding the global manager in the cloud.

CODE:

```
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.*;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.*;
import org.cloudbus.cloudsim.provisioners.*;

public class CloudSimExample8 {

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

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

 private static List < Vm > createVM(int userId, int vms, int idShift) {
  //Creates a container to store VMs. This list is passed to the broker later
  LinkedList < Vm > list = new LinkedList < Vm > ();

  //VM Parameters
  long size = 10000; //image size (MB)
  int ram = 512; //vm memory (MB)
  int mips = 250;
  long bw = 1000;
  int pesNumber = 1; //number of cpus
  String vmm = "Xen"; //VMM name

  //create VMs
  Vm[] vm = new Vm[vms];

  for (int i = 0; i < vms; i++) {
        vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
    list.add(vm[i]);
```

```
    }
    return list;
  }

  private static List < Cloudlet > createCloudlet(int userId, int cloudlets, int idShift) {
    // Creates a container to store Cloudlets
    LinkedList < Cloudlet > list = new LinkedList < Cloudlet > ();

    //cloudlet parameters
    long length = 40000;
    long fileSize = 300;
    long outputSize = 300;
    int pesNumber = 1;
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for (int i = 0; i < cloudlets; i++) {
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
      // setting the owner of these Cloudlets
      cloudlet[i].setUserId(userId);
      list.add(cloudlet[i]);
    }

    return list;
  }

  ////////////////////////// STATIC METHODS ///////////////////////
  /**
   * Creates main() to run this example
   */
  public static void main(String[] args) {
    Log.printLine("Starting CloudSimExample8...");

    try {
      // First step: Initialize the CloudSim package. It should be called before creating any entities.
      int num_user = 2; // number of grid users
      Calendar calendar = Calendar.getInstance();
      boolean trace_flag = false; // mean trace events

      // Initialize the CloudSim library
      CloudSim.init(num_user, calendar, trace_flag);
```

```
    GlobalBroker globalBroker = new GlobalBroker("GlobalBroker");

    // 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");

    //Third step: Create Broker
    DatacenterBroker broker = createBroker("Broker_0");
    int brokerId = broker.getId();

    //Fourth step: Create VMs and Cloudlets and send them to broker
    vmList = createVM(brokerId, 5, 0); //creating 5 vms
    cloudletList = createCloudlet(brokerId, 10, 0); // creating 10 cloudlets

    broker.submitVmList(vmList);
    broker.submitCloudletList(cloudletList);

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

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

    CloudSim.stopSimulation();

    printCloudletList(newList);

    Log.printLine("CloudSimExample8 finished!");
   } catch (Exception e) {
    e.printStackTrace();
    Log.printLine("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 one or more Machines
   List < Host > hostList = new ArrayList < Host > ();
```

```
// 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should create a list to store these PEs before creating a Machine.
   List < Pe > peList1 = new ArrayList < Pe > ();
   int mips = 1000;

   // 3. Create PEs and add these into the list. for a quad-core machine, a list of 4 PEs is required:
     peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
   peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
   peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
   peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

   //Another list, for a dual-core machine
   List < Pe > peList2 = new ArrayList < Pe > ();

   peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
   peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

   //4. Create Hosts with its id and list of PEs and add them to the list of machines
   int hostId = 0;
   int ram = 16384; //host memory (MB)
   long storage = 1000000; //host storage
   int bw = 10000;

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

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

```
        new VmSchedulerTimeShared(peList2)
      )
    ); // Second 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 = 0.1; // the cost of using storage in this resource
    double costPerBw = 0.1; // 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(String name) {
    DatacenterBroker broker = null;
    try {
      broker = new DatacenterBroker(name);
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
    return broker;
  }
```

```
  /**
   * Prints the Cloudlet objects
   * @param list list of Cloudlets
   */
  private static void printCloudletList(List < Cloudlet > list) {
    int size = list.size();
    Cloudlet cloudlet;
    String indent = "    ";
    Log.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
       "Data center ID" + indent + "VM ID" + indent + indent + "Time" + indent + "Start Time" +
indent + "Finish Time");
    DecimalFormat dft = new DecimalFormat("###.##");
    for (int i = 0; i < size; i++) {
      cloudlet = list.get(i);
      Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

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

  public static class GlobalBroker extends SimEntity {
    private static final int CREATE_BROKER = 0;
    private List < Vm > vmList;
    private List < Cloudlet > cloudletList;
    private DatacenterBroker broker;

    public GlobalBroker(String name) {
      super(name);
    }

    @Override
    public void processEvent(SimEvent ev) {
      switch (ev.getTag()) {
      case CREATE_BROKER:
```

```
    setBroker(createBroker(super.getName() + "_"));
    //Create VMs and Cloudlets and send them to broker
    setVmList(createVM(getBroker().getId(), 5, 100)); //creating 5 vms
    setCloudletList(createCloudlet(getBroker().getId(), 10, 100)); // creating 10 cloudlets
    broker.submitVmList(getVmList());
    broker.submitCloudletList(getCloudletList());
    CloudSim.resumeSimulation();
    break;

  default:
    Log.printLine(getName() + ": unknown event type");
    break;
  }
}

@Override
public void startEntity() {
  Log.printLine(super.getName() + " is starting...");
  schedule(getId(), 200, CREATE_BROKER);
}

@Override
public void shutdownEntity() {}

public List < Vm > getVmList() {
  return vmList;
}

protected void setVmList(List < Vm > vmList) {
  this.vmList = vmList;
}

public List < Cloudlet > getCloudletList() {
  return cloudletList;
}

protected void setCloudletList(List < Cloudlet > cloudletList) {
  this.cloudletList = cloudletList;
}

public DatacenterBroker getBroker() {
  return broker;
}
```

```
    protected void setBroker(DatacenterBroker broker) {
      this.broker = broker;
    }
  }
}
```

OUTPUT:

```
Starting CloudSimExample8...
Initialising...
Starting CloudSim version 3.0
GlobalBroker is starting...
Datacenter_0 is starting...
Datacenter_1 is starting...
Broker_0 is starting...
Entities started.
0.0: Broker_0: Cloud Resource List received with 2 resource(s)
0.0: Broker_0: Trying to Create VM #0 in Datacenter_0
0.0: Broker_0: Trying to Create VM #1 in Datacenter_0
0.0: Broker_0: Trying to Create VM #2 in Datacenter_0
0.0: Broker_0: Trying to Create VM #3 in Datacenter_0
0.0: Broker_0: Trying to Create VM #4 in Datacenter_0
0.1: Broker_0: VM #0 has been created in Datacenter #3, Host #0
0.1: Broker_0: VM #1 has been created in Datacenter #3, Host #0
0.1: Broker_0: VM #2 has been created in Datacenter #3, Host #0
0.1: Broker_0: VM #3 has been created in Datacenter #3, Host #1
0.1: Broker_0: VM #4 has been created in Datacenter #3, Host #0
0.1: Broker_0: Sending cloudlet 0 to VM #0
0.1: Broker_0: Sending cloudlet 1 to VM #1
0.1: Broker_0: Sending cloudlet 2 to VM #2
0.1: Broker_0: Sending cloudlet 3 to VM #3
0.1: Broker_0: Sending cloudlet 4 to VM #4
0.1: Broker_0: Sending cloudlet 5 to VM #0
0.1: Broker_0: Sending cloudlet 6 to VM #1
0.1: Broker_0: Sending cloudlet 7 to VM #2
0.1: Broker_0: Sending cloudlet 8 to VM #3
0.1: Broker_0: Sending cloudlet 9 to VM #4
Adding: GlobalBroker_
GlobalBroker_ is starting...
200.0: GlobalBroker_: Cloud Resource List received with 2 resource(s)
200.0: GlobalBroker_: Trying to Create VM #100 in Datacenter_0
200.0: GlobalBroker_: Trying to Create VM #101 in Datacenter_0
200.0: GlobalBroker_: Trying to Create VM #102 in Datacenter_0
200.0: GlobalBroker_: Trying to Create VM #103 in Datacenter_0
200.0: GlobalBroker_: Trying to Create VM #104 in Datacenter_0
200.1: GlobalBroker_: VM #100 has been created in Datacenter #3, Host #1
200.1: GlobalBroker_: VM #101 has been created in Datacenter #3, Host #0
200.1: GlobalBroker_: VM #102 has been created in Datacenter #3, Host #1
200.1: GlobalBroker_: VM #103 has been created in Datacenter #3, Host #0
200.1: GlobalBroker_: VM #104 has been created in Datacenter #3, Host #1
200.1: GlobalBroker_: Sending cloudlet 100 to VM #100
200.1: GlobalBroker_: Sending cloudlet 101 to VM #101
200.1: GlobalBroker_: Sending cloudlet 102 to VM #102
```

```
200.1: GlobalBroker_: Sending cloudlet 103 to VM #103
200.1: GlobalBroker_: Sending cloudlet 104 to VM #104
200.1: GlobalBroker_: Sending cloudlet 105 to VM #100
200.1: GlobalBroker_: Sending cloudlet 106 to VM #101
200.1: GlobalBroker_: Sending cloudlet 107 to VM #102
200.1: GlobalBroker_: Sending cloudlet 108 to VM #103
200.1: GlobalBroker_: Sending cloudlet 109 to VM #104
320.1: Broker_0: Cloudlet 0 received
320.1: Broker_0: Cloudlet 5 received
320.1: Broker_0: Cloudlet 1 received
320.1: Broker_0: Cloudlet 6 received
320.1: Broker_0: Cloudlet 2 received
320.1: Broker_0: Cloudlet 7 received
320.1: Broker_0: Cloudlet 4 received
320.1: Broker_0: Cloudlet 9 received
320.1: Broker_0: Cloudlet 3 received
320.1: Broker_0: Cloudlet 8 received
320.1: Broker_0: All Cloudlets executed. Finishing...
320.1: Broker_0: Destroying VM #0
320.1: Broker_0: Destroying VM #1
320.1: Broker_0: Destroying VM #2
320.1: Broker_0: Destroying VM #3
320.1: Broker_0: Destroying VM #4
Broker_0 is shutting down...
520.1: GlobalBroker_: Cloudlet 101 received
520.1: GlobalBroker_: Cloudlet 106 received
520.1: GlobalBroker_: Cloudlet 103 received
520.1: GlobalBroker_: Cloudlet 108 received
520.1: GlobalBroker_: Cloudlet 100 received
520.1: GlobalBroker_: Cloudlet 105 received
520.1: GlobalBroker_: Cloudlet 102 received
520.1: GlobalBroker_: Cloudlet 107 received
520.1: GlobalBroker_: Cloudlet 104 received
520.1: GlobalBroker_: Cloudlet 109 received
520.1: GlobalBroker_: All Cloudlets executed. Finishing...
520.1: GlobalBroker_: Destroying VM #100
520.1: GlobalBroker_: Destroying VM #101
520.1: GlobalBroker_: Destroying VM #102
520.1: GlobalBroker_: Destroying VM #103
520.1: GlobalBroker_: Destroying VM #104
GlobalBroker_ 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_0 is shutting down...
GlobalBroker_ is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    0        SUCCESS        3            0            320        0.1            320.1
    5        SUCCESS        3            0            320        0.1            320.1
    1        SUCCESS        3            1            320        0.1            320.1
    6        SUCCESS        3            1            320        0.1            320.1
    2        SUCCESS        3            2            320        0.1            320.1
    7        SUCCESS        3            2            320        0.1            320.1
    4        SUCCESS        3            4            320        0.1            320.1
    9        SUCCESS        3            4            320        0.1            320.1
    3        SUCCESS        3            3            320        0.1            320.1
    8        SUCCESS        3            3            320        0.1            320.1
    101        SUCCESS        3            101            320        200.1            520.1
    106        SUCCESS        3            101            320        200.1            520.1
    103        SUCCESS        3            103            320        200.1            520.1
    108        SUCCESS        3            103            320        200.1            520.1
    100        SUCCESS        3            100            320        200.1            520.1
    105        SUCCESS        3            100            320        200.1            520.1
    102        SUCCESS        3            102            320        200.1            520.1
    107        SUCCESS        3            102            320        200.1            520.1
    104        SUCCESS        3            104            320        200.1            520.1
    109        SUCCESS        3            104            320        200.1            520.1
CloudSimExample8 finished!
```

PRACTICAL - 9

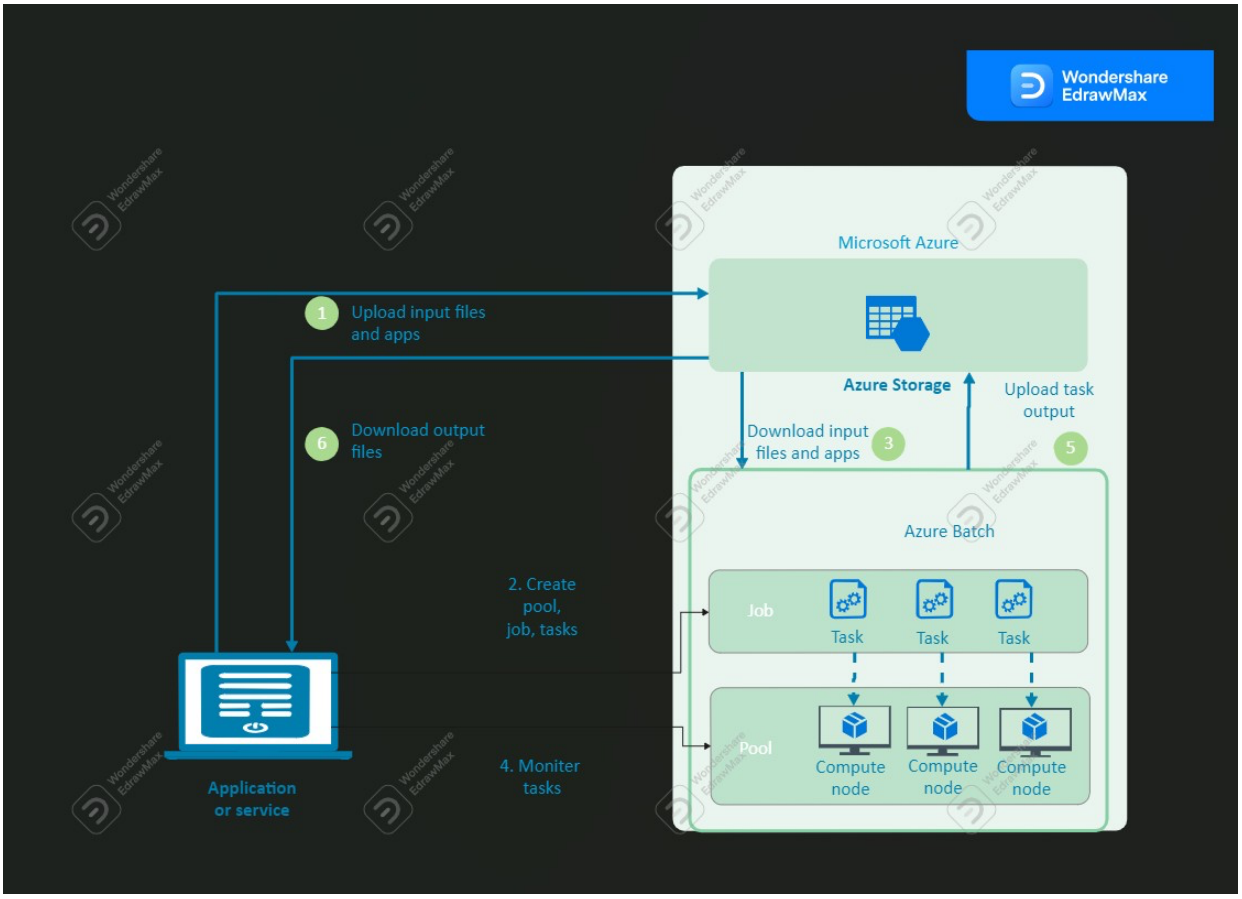
AIM: Sketch out and analyze the architecture of Microsoft Azure.

THEORY:

9A. What is Microsoft azure?

Microsoft Azure, often referred to as Azure. It is a cloud computing platform run by Microsoft, which offers access, management, and development of applications and services through global data centers. It provides a range of capabilities, including software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Microsoft Azure supports many programming languages, tools, and frameworks, including Microsoft-specific and third-party software and systems.

9B. Azure Architecture



- An azure architecture diagram visualizes the deployment and hosting of any application on azure cloud services.
- Azure architecture is the most popular cloud service used by most of the 500 fortune companies. Organizations use azure architectures because it is cost-effective, quick, and flexible.

- An azure diagram provides a better understanding of your cloud infrastructure by depicting your organization's network topologies and databases.
- Azure diagrams depict the cloud computing services of an enterprise. Creating a cloud architecture is hard because the infrastructure of an organization's cloud services is complex.
- Making these diagrams is essential if you plan to use azure architecture services.

9C. Working of Microsoft Azure.

- Microsoft offers through the Azure portal, a number of third-party vendors also make software directly available through Azure.
- The cost billed for third-party applications varies widely but may involve paying a subscription fee for the application, plus a usage fee for the infrastructure used to host the application.
- Microsoft provides the following five different customer support options for Azure:

1. Basic
2. Developer
3. Standard
4. Professional Direct
5. Enterprise (Premier)

- These customer support plans vary in terms of scope and price. Basic support is available to all Azure accounts, but Microsoft charges a fee for the other support offerings.
- Developer support costs $29 per month, while Standard support costs $100 per month and Professional Direct support is $1,000 per month.

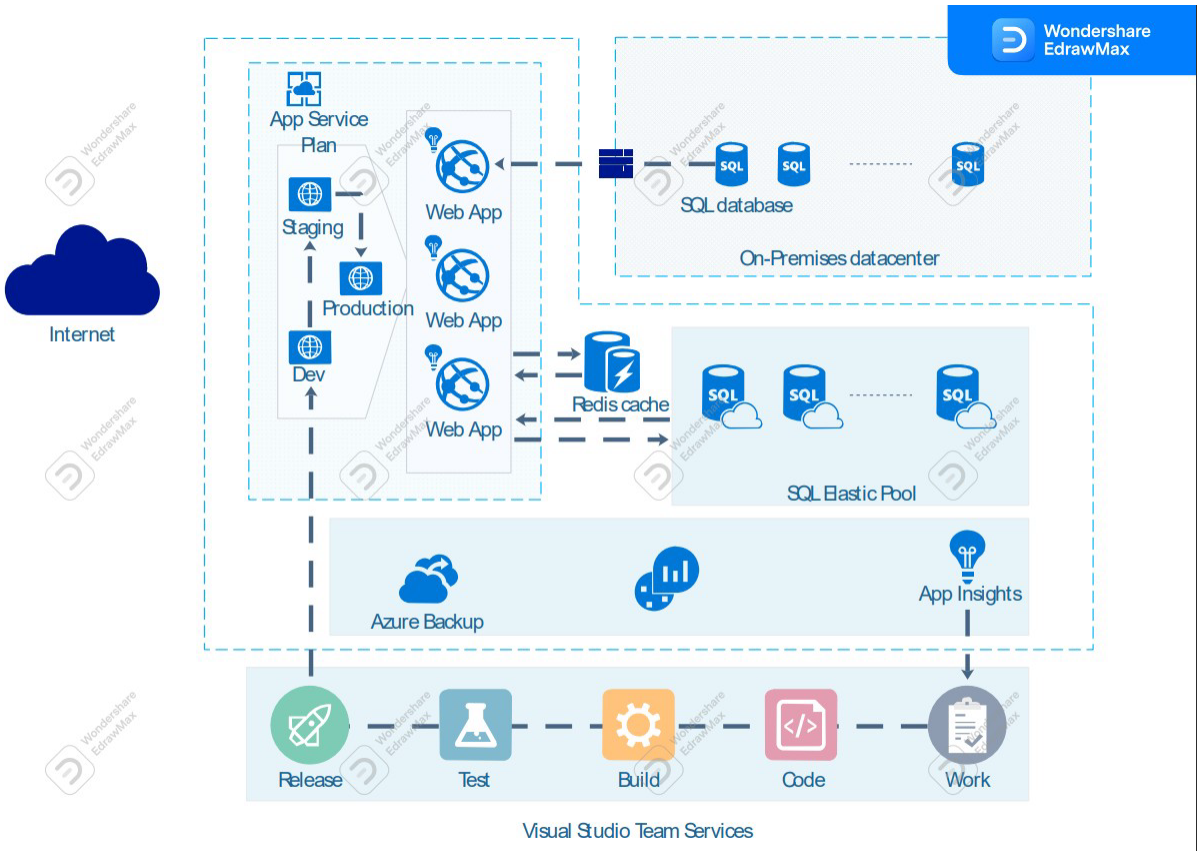
9D. Creation of Azure Architecture diagram.

STEP 1: Log in to EdrawMax Online or create a new account.

STEP 2: You can make your diagram using a template or open a blank canvas to draw it from scratch. Go to new, and click on the cloud service category. Select azure and click on the "+" button to get a new canvas, or select a template to edit it right away.

STEP 3: The next step is to customize your azure architecture diagram. Go to the symbols library and insert icons and symbols with a simple drag and drop. Add connectors to visualize the relationship between components. Align your diagram, adjust the layout, and style it by changing fonts. Customize it using drawing tools, auto themes, and color fill.

STEP 4: After your diagram is complete, the next step is to download it. EdrawMax Online supports various document formats, so you can export in any format you want. You can also share and print it.



[Azure Architecture Diagram Complete Guide | EdrawMax Online]

PRACTICAL - 10

AIM: OEP (Open Ended Problem)