PRACTICAL 01: Execute & Check the Performance of existing Algorithms using CloudSim.

1. Download *cloudsim-3.0.3.zip* from: <https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3>
2. Unzip the Package to an appropriate Location. [Here: S:\]
3. If the System Variable CLASSPATH does not exist, create it and add the following two Values to it: *S:\cloudsim-3.0.3\jars\\*;*

*S:\cloudsim-3.0.3\examples\org\cloudbus\cloudsim\examples*

1. **Using Command Prompt**, compile and run the Example .java Files in *CloudSim* Folder:

javac *CloudSimExample1*.java

java *org.cloudbus.cloudsim.examples.CloudSimExample1*

1. Now **to run the Program in an IDE**, create a new Application in IDE *Eclipse*.
2. To add cloudsim-3.0.3.jar file, go to: **Application (Right Click) > Properties > Java Build Path > Add External Jars > Browse S:\cloudsim-3.0.3\jars > Choose: cloudsim-3.0.3.jar > OK**
3. Copy the following Code:

package practical.pkg1;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.DatacenterCharacteristics;

import org.cloudbus.cloudsim.Host;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.Pe;

import org.cloudbus.cloudsim.Storage;

import org.cloudbus.cloudsim.UtilizationModel;

import org.cloudbus.cloudsim.UtilizationModelFull;

import org.cloudbus.cloudsim.Vm;

import org.cloudbus.cloudsim.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerTimeShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\* A simple example showing how to create a datacenter with one host and run one

\* cloudlet on it.

\*/

/\*M.Sc (2019-2020). Part-II, Sem-III: GHADIGAONKAR AMOL ASHOK VEENA \*/

public class CloudSimDemo {

/\*\* 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 {

//1st step: Initialize the CloudSim package. It should be called before creating any entities.

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

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events Initialize the CloudSim library

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

// Second step: Create Datacenters

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

// list one of them to run a CloudSim simulation

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

// Third step: Create Broker

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

// Fourth step: Create one virtual machine

vmlist = new ArrayList<Vm>();

// VM description

int vmid = 0;

int mips = 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

//we are not adding SAN devices by now

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

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()));

}

}

}

}

1. Run as Application.
2. Output:

PRACTICAL 02: Install a Cloud Analyst and integrate with IDE Eclipse/Netbeans. Monitor the performance of an Existing Algorithm.

**Steps**

1. Download Cloud Analyst from: [www.cloudbus.org/cloudsim/CloudAnalyst.zip](http://www.cloudbus.org/cloudsim/CloudAnalyst.zip)
2. Unzip the Package to an appropriate Location. [Here: S:\]
3. Start IDE Eclipse. Go to: **File > Open Projects from File-System > Directory (Browse to the unzipped Folder) > Check *Cloud Analyst >* Finish***.*
4. Go to Project: *CloudAnalyst* > source > *cloudsim*.ext.gui > *GuiMain*.java
5. Run the File to open the GUI as follows:
6. Click the *Configure Simulation > Main Configuration.*

Click **Add New** *of User Bases* to add new Record.

1. Go to **Data Center Configuration** Tab. **Add** **new** Data Center.
2. Select any Data Center by clicking on it. In the Physical Hardware Details that appear, **remove** one Detail.
3. Select the **Advanced** Tab and set Values as follows: 6, 3, 100
4. Save the Configuration as a **.sim** file at an appropriate Location [Here: S:\]
5. Click the **Show Boundaries Region** Button **(**Bottom Right Corner**).**
6. Click **Run Simulation**.
7. Click **Export Results**. Store the resultant .pdf file at an appropriate Location [S:\]
8. Check the Output, in the **output Console Window** of IDE Eclipse.

Output:

simulation time =3600000.0ms

Starting Simulation...

Initialising...

Creating new broker DC1-Broker

0.0 Creating new user base UB1

0.0 Creating new user base UB2

Starting GridSim version 4.2

Entities started.

Starting user base 7 UB1

Starting broker 6 name=DC1-Broker

Starting internet 11

Starting user base 9 UB2

5.0: DC1-Broker: Cloud Resource List received with 1 resource(s)

5.0: DC1-Broker: Trying to Create VM #0

5.0: DC1-Broker: Trying to Create VM #1

5.0: DC1-Broker: Trying to Create VM #2

5.0: DC1-Broker: Trying to Create VM #3

5.0: DC1-Broker: Trying to Create VM #4

Gathering simulation data.

UB1 finalizing. Messages sent:1035, Received:1035

UB1 requests sent=6058 , received=6058

UB2 finalizing. Messages sent:1054, Received:1054

UB2 requests sent=6187 , received=6187

Got response for 701023 but it seems to be completed.

DC1-Broker finalizing, submitted cloudlets=4121 processing cloudlets=0 ,allRequestsProcessed=12245

Simulation completed.

\*\*\*\*\*\*\*\*\*\*\*\* Vm allocations in DC1

0->1650

1->1650

2->1650

3->1649

4->1649

\*\*\*\*\*Datacenter: DC1\*\*\*\*\*

User id Debt

6 5128

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Simulation finished at 3647500.0

PRACTICAL 03: Build an Application on Private Cloud.

1. Download **PuTTY** from: [*https://www.puttygen.com/download-putty*](https://www.puttygen.com/download-putty)

***PuTTY***, a popular terminal emulator, is an open-source, light-weight, and free SSH client. It was developed by Simon Tatham in C language. Its primary function is to connect to a computer remotely while offering the facilities of transferring files, manipulating data, etc.

It offers support to a variety of network protocols like SSH, Telnet, Serial, SCP, SFTP, etc. *PuTTY* also comes with a command-line tool called “psftp” which can securely transfer files between computers over an SSH connection.

Use of *PuTTY*:

In a multi-user operating system like Unix, the interface is generally of command-line type, just like the command prompt or MS-DOS. As such the user needs to type in the command in the command line program to get anything processed by the system.

Generally, these commands can quickly be run over a network from a different computer on a different location (client) and the response is transferred over the network to the client.

The arrangement mentioned above is made possible with the help of network protocols like SSH, Telnet, Rlogin, etc. Interestingly, users can give commands to multiple computers simultaneously.

SSH (Secure Shell) protocol is a cryptographic network protocol that allows you to access an internet server while encrypting any information sent to that server. Some of the other protocols include Telnet, Rlogin only if either you are connected to a Unix system or you have a login account on a web server (a shell account). *PuTTY* is one such application that enables this kind of transfer.

*PuTTY* supports the following protocols

SCP (Secure Copy)

SSH (Secure Shell)

Telnet

Rlogin

Raw socket connection.

***PuTTYgen*** is a key generator tool for creating pairs of public and private SSH keys. It is one of the components of the open-source networking client PuTTY. Although originally written for Microsoft Windows operating system, it is now officially available for multiple operating systems including macOS, Linux. PuTTYgen.exe is the graphical tool on Windows OS. While on the other side, Linux OS has the only command-line version could be accessible using SSH commands.

**To run PuTTYgen on Windows:**

**Go to Windows -> Start Menu -> All Programs -> PuTTY -> PuTTYgen**

The key generation utility – PuTTYgen can create various public-key cryptosystems including Rivest–Shamir–Adleman (RSA), Digital Signature Algorithm (DSA), Elliptic Curve Digital Signature Algorithm (ECDSA), and Edwards-curve Digital Signature Algorithm (EdDSA) keys.

The aforementioned public-key cryptosystems principally focus on secure data transmission and digital signatures.

Although PuTTYgen collects keys in its native file format i.e. .ppk files, the keys can easily be converted to any file format. For Windows, the software interface is PuTTYgen.exe, whereas, for Linux OS the command-line adaptation is available using SSH commands.

How to use PuTTYgen?

PuTTYgen is used to generate public or private key pair for creating SSH keys. Below is the complete guidance about how to generate RSA key in the Windows operating system:

Once you install the PuTTY on your machine, you can easily run PuTTYgen. For the same, go to Windows -> Start Menu -> All Programs -> PuTTY -> PuTTYgen.

You will see the PuTTY key generator dialog box on your screen

You will find a “Generate” button in that dialog. Clicking on it will lead to generating the keys for you.

Now you will need to add a unique key passphrase in the Key passphrase and Confirm passphrase field.

Click on the “Save Public Key” and “Save Private Key” buttons to save your public and private keys.

You will see the text starting with ssh-RSA in the Public key for pasting into OpenSSH authorized\_keys file field which is located at the top of the window. Copy that entire text to your clipboard by pressing ctrl+c as you will require the key to paste on your clipboard in the public key tool of control panel or directly on the cloud server.

Various Ways to Use RSA Key Pair

RSA key pair generated through PuTTYgen is used in two various ways defined as below:

* To assign while creating a new cloud server
* You can choose the public key from the given list of keys at the time of creating a cloud server. If you don’t find your key in that list, then first add and then assign it.
* Assign to an existing cloud server
* At the time of connecting to the cloud server, first of all, you need to tell PuTTY to use it for utilizing your newly created RSA key pair.

1. Run **aws.amazon.com** on the Browser.
2. Go to > MyAccount > AWS Management Console > Sign In > Account ID: *ccdemo*

IAM User Name: *user3*

Password: *patkar123*

1. After login, go to: **All Services > EC2 > Launch Instance** to get the following window:

Choose the very first: 1. **Amazon Linux 2 AMI (HVM), SSD Volume Type**

1. Choose: Instance Type: *T2 Micro*

Click Button: ***Configure Instance Details***

1. Number of Instances: 1

Click Button: ***Add Storage***

1. Click: **Next: Add Tags**

Click: Add Tag. Key: KyAmol Value: DEFAULT-VPC

Click: **Configure Security Group**

1. Add Role: **SSH, HTTP, HTTPS**

Click: **Review and Launch**

1. Select: **Launch** and then **Create a new Key Pair**.

Click: **Download Key Pair** to download **Private Key File.**

1. Click: **Launch Instances** and **View Instances.**
2. Copy the Public DNS: **ec2-18-217-209-203.us-east-2.compute.amazonaws.com** to Clipboard.
3. Launch the PuTTY (**not** PuTTYGen)

Host Name: [EC2\_User@ec2-18-217-209-203.us-east-2.compute.amazonaws.com](mailto:EC2_User@ec2-18-217-209-203.us-east-2.compute.amazonaws.com)

Session Name: SSN\_AMOL > Save

1. In the **Category** Section, Go to: **Connection > SSH > Auth** > **Browse to the Private Key File (.PEM) and open.**
2. In the PuTTY Security Alert Dialog Box, click **NO.**
3. Launch the **PuTTY Generator.**

Click: **Load** to browse to and load the Private Key File: (.PEM) File.

Click **OK** and then **Save Private Key** at a desired Location**.**

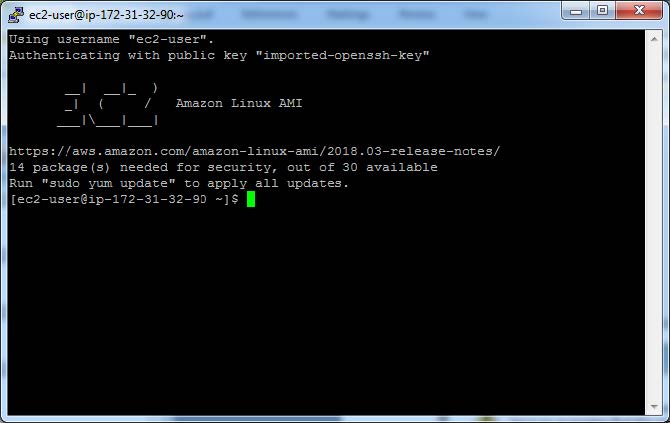
1. Click **Yes** in the next Dialog Box to save the Private Key (.ppk) File to a desired Location.
2. Launch PuTTY again. Enter:

Host Name: [EC2\_User@ec2-18-217-209-203.us-east-2.compute.amazonaws.com](mailto:EC2_User@ec2-18-217-209-203.us-east-2.compute.amazonaws.com)

1. In the **Category** Section, Go to: **Connection > SSH > Auth** > **Browse to the Private Key File (.ppk) and open.**

Hit **NO** to carry on connecting just Once.

1. The Ec2 Amazon Linux will be launched.



1. In the Command Prompt:

Execute: **sudo yum update –y**

1. Run the Public DNS in the Browser for the Final Output of a Test Page.

PRACTICAL 04: Demonstrate any Cloud Monitoring Tool.

Login into [www.amazon,com](http://www.amazon,com) and Click on the **Monitoring** Tab.

PRACTICAL 05: Implement FOSS-Cloud Functionality – Virtual Desktop Infrastructure (VDI).

**Foss Cloud**

1. Download and Install ***WinSCP*** from *https://winscp.net/eng/download.php*

User Name*:* ***root*** Password*:* ***admin***

Click **Login.**

2 In the following Dialog Box**,** Click **No.** This lets you in the ***root*** Folder of the Server.

3. On the Server, go to **<root> var > Virtualization > iso-Choosable.**

4. Drag-Drop the **Ubuntu.iso** File (from the local Machine on the right) into the above Destination Folder on the Server.

5. Now **to implement the FOSS-Cloud Functionality**, **in the FOSS-Cloud,** in the **existing** Templates, in the **Run-Action** Column, click the **Green Arrow**, to start the Template.

6. In the **Links** Section (on the left-bottom), click **Download Spice Client,** to visit and download from its Site: *http://www.foss-cloud.org/en/wiki/Spice-Client* and install its latest stable Version.

7. Go back to the **Action** Column and click the **Blue Icon** to start the following:

8. Click **Try Ubuntu**

**PRACTICAL 06**: Implement FOSS-Cloud Functionality – Virtual Server Infrastructure (VSI) – Infrastructure as a Service (IaaS).

1. Go to: *http://192.168.2.86/vm-manager/*

Enter:

User Name: ***admin***

Password: ***admin***

3. Go to: **Virtual Machine > VM Templates > Create > Profile > Linux > Ubuntu > i686 > multi**

4. Make Choices as in the following Picture:

5. Click **Create**.

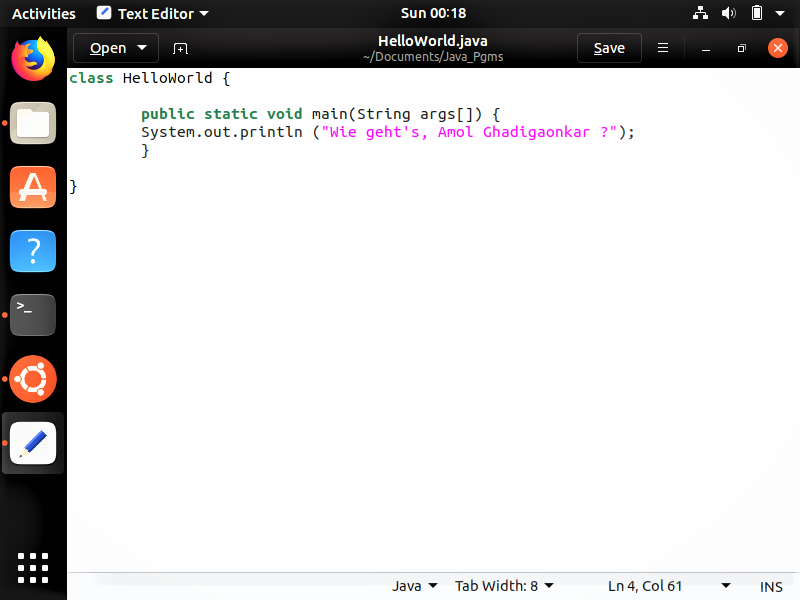
PRACTICAL 07: Implement FOSS-Cloud Functionality – Virtual Server Infrastructure (VSI) – Platform as a Service (PaaS).

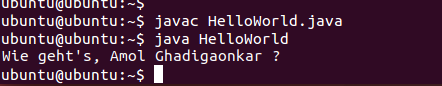












PRACTICAL 08: Explore FOSS-Cloud Functionality - Storage Cloud

