Assignment No. 6

Aim

Installation & demonstration of CloudSim

Problem Definition

Install following Cloud Simulators/Tools and frame suitable assignments to demonstarte its use: CloudSim
Assignment : Simulate the following

1. Create a datacenter with one host and run one cloudlet on it using CloudSim
2. Create and Configure the data center and user base to show response time, request servicing time and data center loading
3. Install the client to launch the application running in the container using docker images.

Learning Objectives

- To learn the concept of CloudSIm and Docker
- TO implement ClouSim and Docker images

Learning Outcome

- Understood the concept of CloudSim and Docker
- Successfully implemented Docker image of ubuntu and CloudSim Datacentre

Software And Hardware Requirements

- Latest 64-BIT Version of Linux Operating System
- CloudSim Library downloaded
- Docker installed
- Eclipse IDE

Mathematical Model

Let S be the system of solution set for given problem statement such that,
S = { s, e, X, Y, F, DD, NDD, Su, Fu }
where,

s = start state
such that, y = { }
e = end state
such that, y = { CS }
where, CS = CloudSim created
X = set of inputs
such that, X = { X1, X2 }
where, X1 = Input for CloudSim that X1 = { x1, x2, x3, x4, x5, x6. x7 }
x1 = Allocated ram
x2 = Virtual HDD size
x3 = CPU threshold
x4 = Video memory
x5 = OS installation ISO
x6 = MAC address
x7 = Architecture type

X2 = { x_a, x_b }
where, x_a = Image name
x_b = Source program /script file

Y = set of output
such that Y = { y1, y2, y3 }
where, y1 = Cloudlet run & Datacenter created
y2 = Source code / script run in given container
y3 = Container ID
F = set of function
such that F = { f1, f2 }
where,
f1 = functions in Cloudsim implementation
such that, f1 = { f_{a1}, f_{a2}, f_{a3}, f_{a4} }
f_{a1} = function to initialize cloudsim library and packages
f_{a2} = function to create datacenter
f_{a3} = function to create broker
f_{a4} = function to create virtual machine

f2 = function in Docker implementation
such that, f2 = { f_{b1}, f_{b2}, f_{b3} }
f_{b1} = function to download ubuntu image
f_{b2} = function to create container
f_{b3} = function to run host code/script

DD = Deterministic data
such that, DD = { X1, X2 }
NDD = Nondeterministic data
such that, NDD = { y1, y3 }
Su = Success case

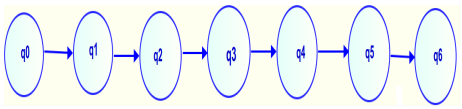
- CloudSim libraries imported in eclipse IDE
- Docker ubuntu image downloaded which is uncorrupt

Fu = Failure case

- CloudSim libraries are not imported
- Docker image of ubuntu is not downloaded or corrupt

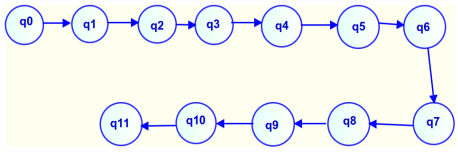
State Diagram

CloudSim



where,
q0 = Start state
q1 = Download Ubuntu Docker image
q2 = Create interactive docker container
q3 = Get container id
q4 = Copy code/script executable from localhost to container
q5 = Run code/script in container
q6 = End state

Docker



where,
q0 = Start state
q1 = Initialize the CloudSim package
q2 = Initialize the CloudSim library

q3 = Create Datacenter
q4 = Create Broker
q5 = Create one virtual machine
q6 = Submit vm list to the broker
q7 = Create one Cloudlet
q8 = Submit cloudlet list to the broker
q9 = Start the simulation
q10 = Stop the simulation
q11 = Print result (End state)

Theory

Introduction

Recently, cloud computing emerged as the leading technology for delivering reliable, secure, fault-tolerant, sustainable, and scalable computational services, which are presented as Software, Infrastructure, or Platform as services (SaaS, IaaS, PaaS). Moreover, these services may be offered in private data centers (private clouds), may be commercially offered for clients (public clouds), or yet it is possible that both public and private clouds are combined in hybrid clouds.

These already wide ecosystem of cloud architectures, along with the increasing demand for energy-efficient IT technologies, demand timely, repeatable, and controllable methodologies for evaluation of algorithms, applications, and policies before actual development of cloud products. Because utilization of real testbeds limits the experiments to the scale of the testbed and makes the reproduction of results an extremely difficult undertaking, alternative approaches for testing and experimentation leverage development of new Cloud technologies.

A suitable alternative is the utilization of simulations tools, which open the possibility of evaluating the hypothesis prior to software development in an environment where one can reproduce tests. Specifically in the case of Cloud computing, where access to the infrastructure incurs payments in real currency, simulation-based approaches offer significant benefits, as it allows Cloud customers to test their services in repeatable and controllable environment free of cost, and to tune the performance bottlenecks before deploying on real Clouds. At the provider side, simulation environments allow evaluation of different kinds of resource leasing scenarios under varying load and pricing distributions. Such studies could aid the providers in optimizing the resource access cost with focus on improving profits. In the absence of such simulation platforms, Cloud customers and providers have to rely either on theoretical and imprecise evaluations, or on try-and-error approaches that lead to inefficient service performance and revenue generation.

CloudSim

CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. Originally built primarily at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, The University of Melbourne, Australia. CloudSim has become one of the most popular open source cloud simulators in the research and academia. CloudSim is completely written in Java.

This tool allows to:

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

Though CloudSim itself does not have a graphical user interface, extensions such as CloudReports offer a GUI for CloudSim simulations.

CloudSim Layered Architecture

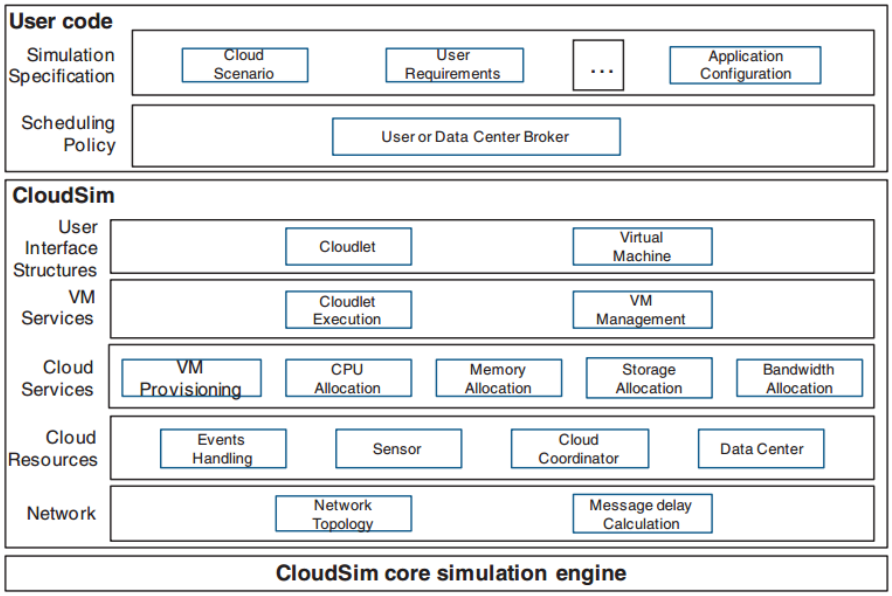


Fig : CloudSim Layered Architecture

Overview of CloudSim functionalities:

- Support for modeling and simulation of large scale Cloud computing data centers

- Support for modeling and simulation of virtualized server hosts, with customizable policies for provisioning host resources to virtual machines
- Support for modeling and simulation of energy-aware computational resources
- Support for modeling and simulation of data center network topologies and message-passing applications
- Support for modeling and simulation of federated clouds
- Support for dynamic insertion of simulation elements, stop and resume of simulation
- Support for user-defined policies for allocation of hosts to virtual machines and policies for allocation of host resources to virtual machines

Docker:

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable filesystem such as aufs and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.

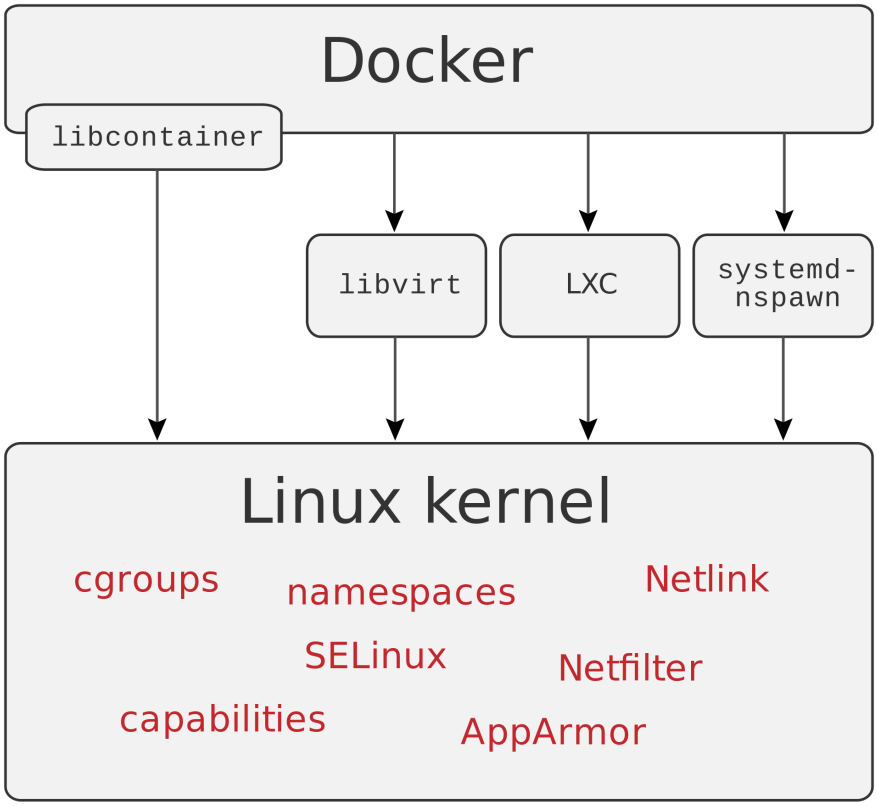


Fig : Docker

Docker implements a high-level API to provide lightweight containers that run processes in isolation.

Building on top of facilities provided by the Linux kernel (primarily cgroups and namespaces), a Docker container, unlike a virtual machine, does not require or include a separate operating system. Instead, it relies on the kernel's functionality and uses resource isolation (CPU, memory, block I/O, network, etc.) and separate namespaces to isolate the application's view of the operating system. Docker accesses the Linux kernel's virtualization features either directly using the libcontainer library, which is available as of Docker 0.9, or indirectly via libvirt, LXC (Linux Containers) or systemd-nspawn.

By using containers, resources can be isolated, services restricted, and processes provisioned to have an almost completely private view of the operating system with their own process ID space, file system structure, and network interfaces. Multiple containers share the same kernel, but each container can be constrained to only use a defined amount of resources such as CPU, memory and I/O.

Program Code

```
package org.cloudbus.cloudsim.examples;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

public class CloudExample {
/** 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) {
Scanner reader = new Scanner(System.in);
Log.printLine("Starting CloudSimExample1...");

try {
```

```
// First step: Initialize the CloudSim package. It should be called
// before creating any entities.
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
// Initialize the CloudSim library
CloudSim.init(num_user, calendar, trace_flag);
// Second step: Create Datacenters
// Datacenters are the resource providers in CloudSim. We need at
// list one of them to run a CloudSim simulation
Datacenter datacenter0 = createDatacenter("Datacenter_0");
// Third step: Create Broker
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();
// Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();
// VM description
int vmid = 0;
int mips = 1000;
long size = 10000; // image size (MB)
int ram = 1000; // 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.
 */
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!
```

Docker Output

####Terminal 1:

```
pmd@pmD-PC:~$ sudo docker run ubuntu /bin/echo 'Hello world'
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
203137e8afd5: Pull complete
2ff1bbbe9310: Pull complete
933ae2486129: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:1bea66e185d3464fec1abda32ffaf2a11de69833cfcf81bd2b9a5be147776814
Status: Downloaded newer image for ubuntu:latest
Hello world

pmd@pmD-PC:~$ echo "Hello PICT!"
Hello PICT!
pmd@pmD-PC:~$ sudo docker run ubuntu /bin/echo 'Hello world'
Hello world

pmd@pmD-PC:/home/pmd# docker run -i -t ubuntu /bin/bash
root@56026341dff3:/# cd /
root@56026341dff3:/# ls
bin   dev   lib    media  opt   root  sbin  sys  usr   boot
etc   home      lib64  mnt    proc  run   srv   tmp  var
```

####Terminal 2:

```
pmd@pmD-PC:~$ sudo docker ps
[sudo] password for pmd:
CONTAINER ID        IMAGE               COMMAND
CREATED
STATUS              PORTS               NAMES
56026341dff3        ubuntu              "/bin/bash"
About a minute ago
 Up About a minute                      pensive_yonath
pmd@pmD-PC:~$ sudo docker cp hello.sh 56026341dff3:/hello.sh
pmd@pmD-PC:~$
```

####Terminal 1:

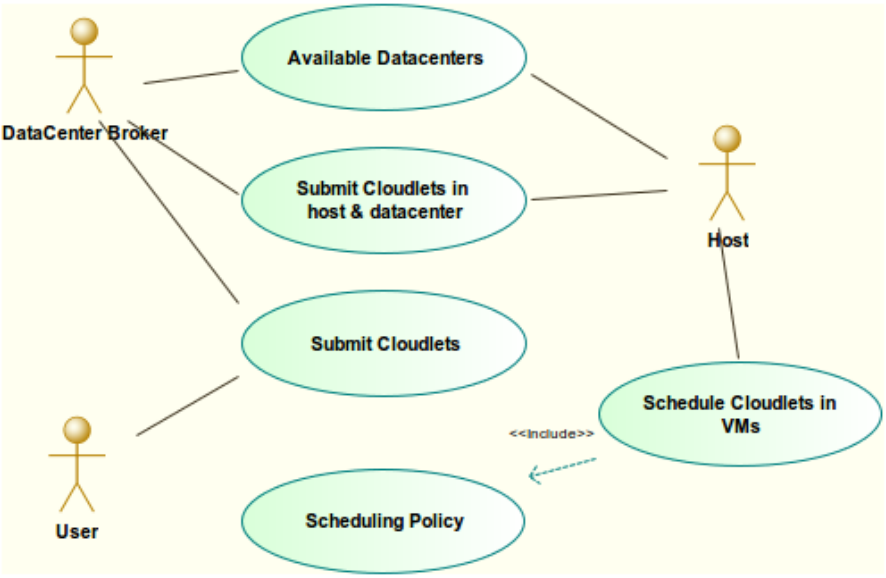
```
root@56026341dff3:/# ls
bin   dev  hello.sh  lib    media  opt   root  sbin  sys
usr
boot  etc  home      lib64  mnt    proc  run   srv   tmp
var
root@56026341dff3:/# ./hello.sh
Hello PICT!
root@56026341dff3:/# exit
exit
pmd@pmD-PC:/home/pmd#
```

####Terminal 2:

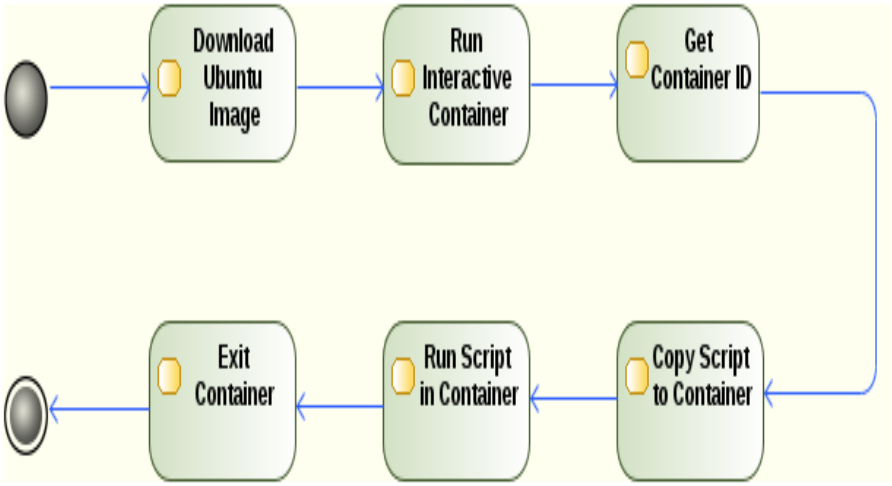
```
pmd@pmD-PC:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED
STATUS              PORTS               NAMES
pmd@pmD-PC:~$
```

UML Diagrams

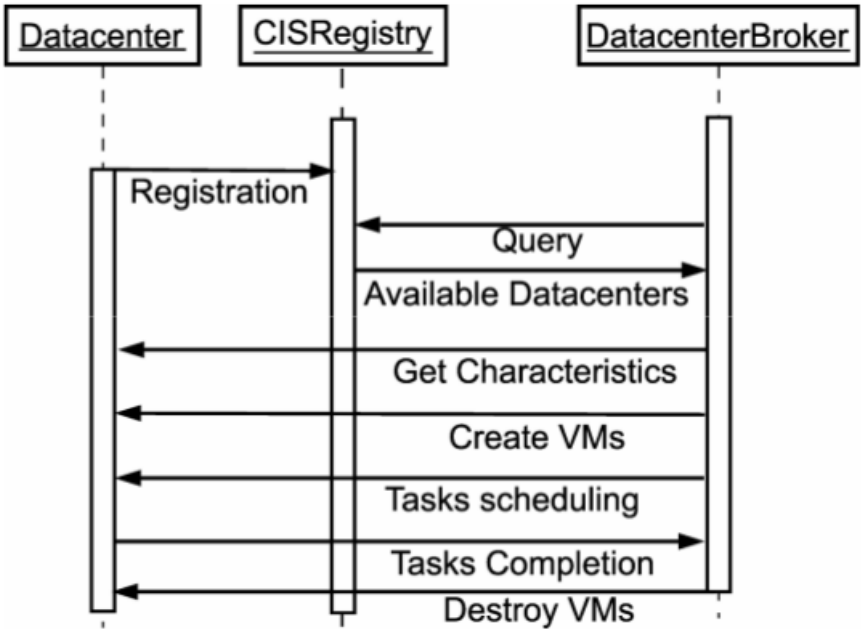
USE-CASE Diagram



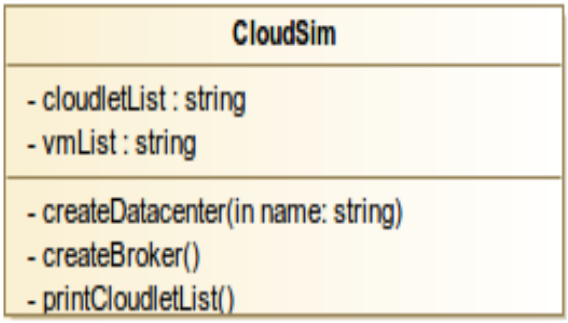
Activity Diagram



Sequence Diagram



Class Diagram



Conclusion

Thus, we understood concepts of Cloudsim and docker and successfully implemented both successfully.