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

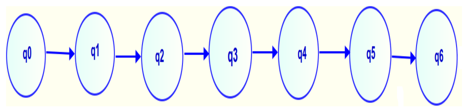
- CloudSim libraries imported in eclipse IDE
- Docker ubuntu image downloaded which is uncorrupt

Fu = Failure case

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

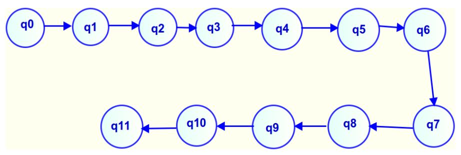
State Diagram

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

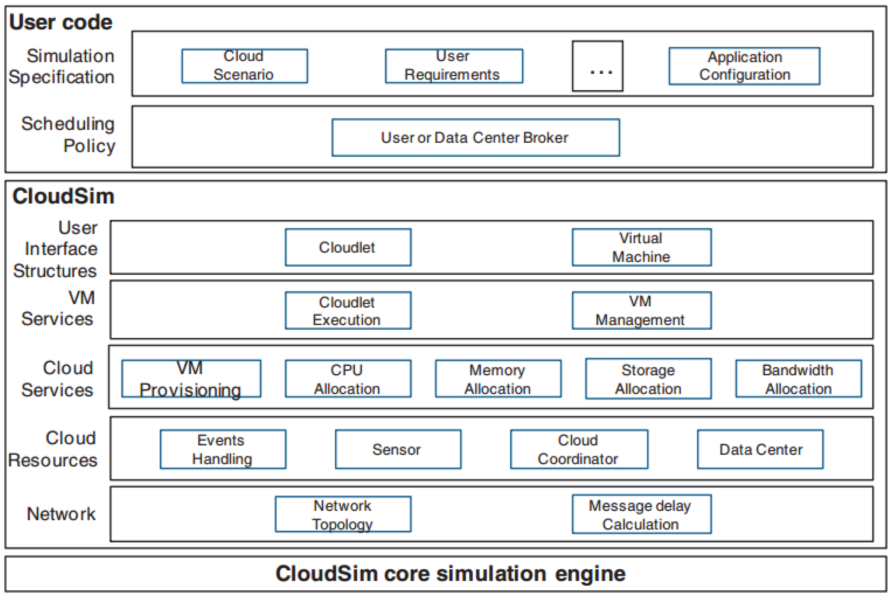


Fig : CloudSim Layered Architecture

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.

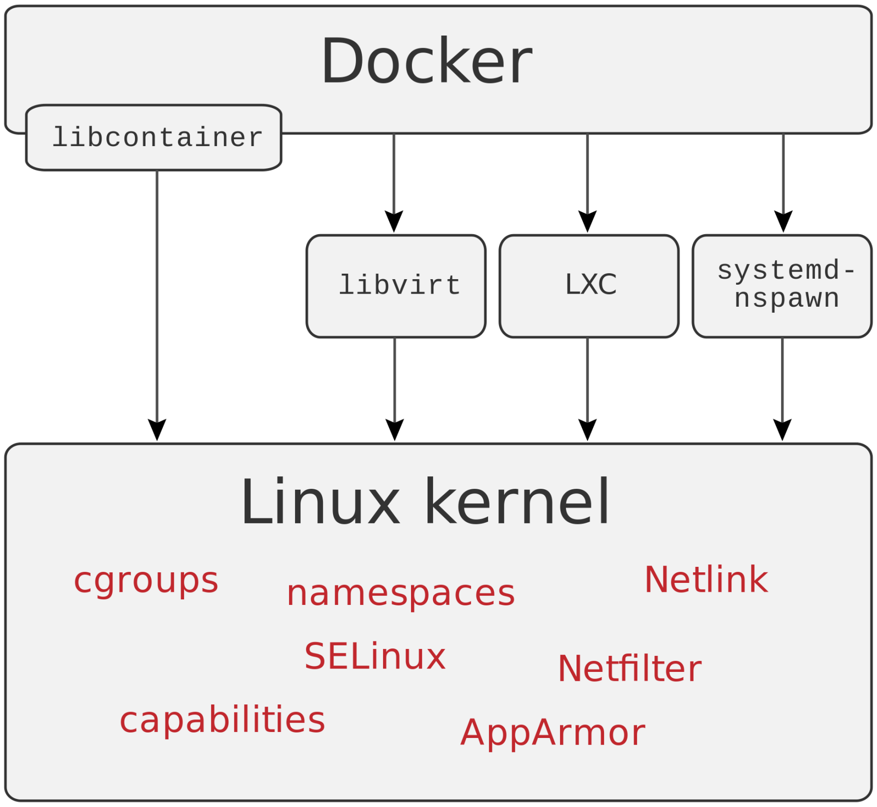


Fig : Docker

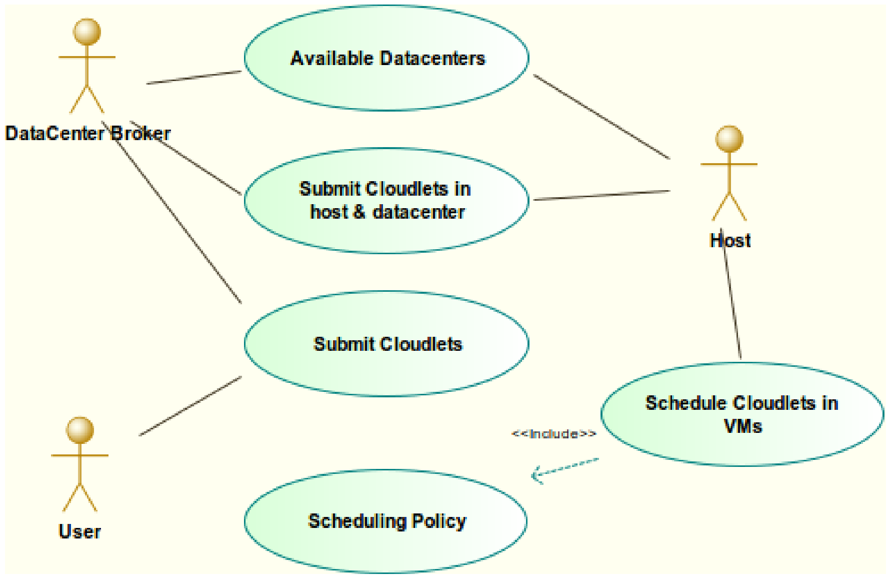
Docker implements a high-level API to provide lightweight containers that run processes in isolation.

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.

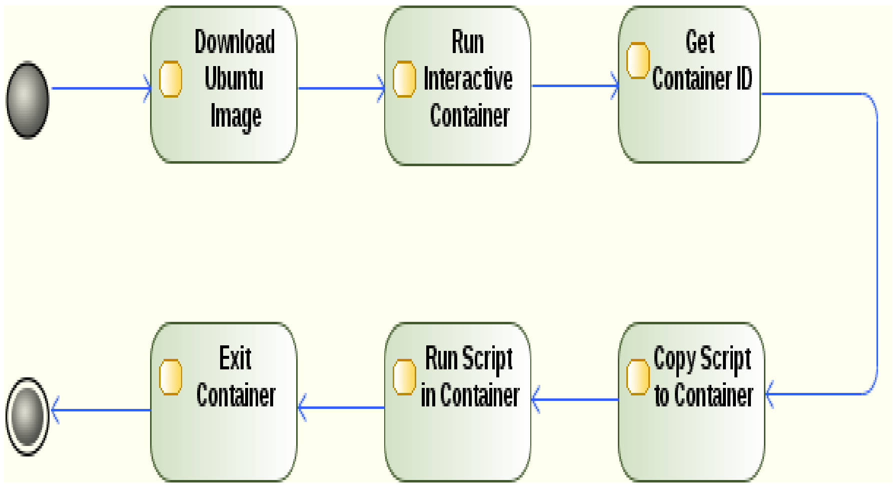
By using containers, resources can be isolated, services restricted, and processes provisioned to have an almost completely private view of the operating system with their own process ID space, file system structure, and network interfaces. Multiple containers share the same kernel, but each container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

UML Diagrams

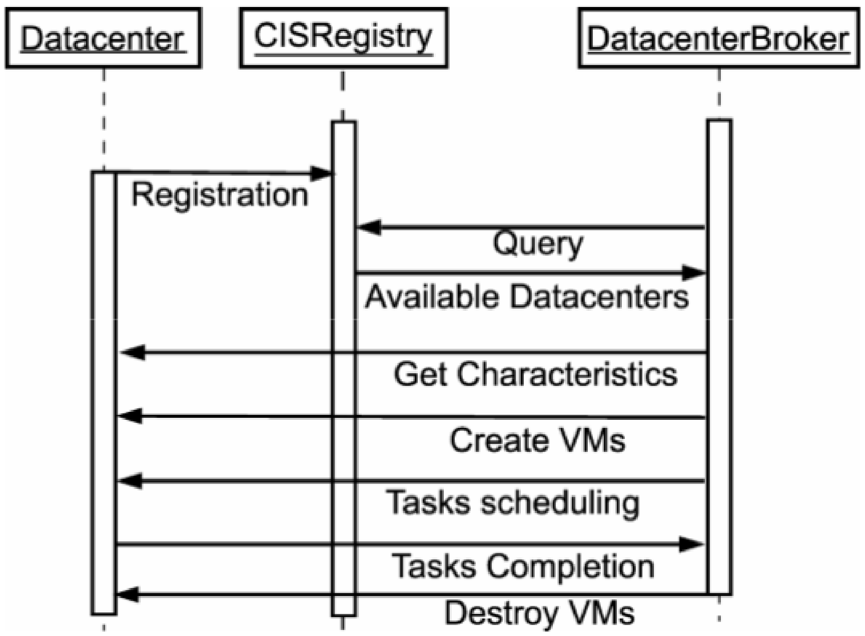
USE-CASE Diagram



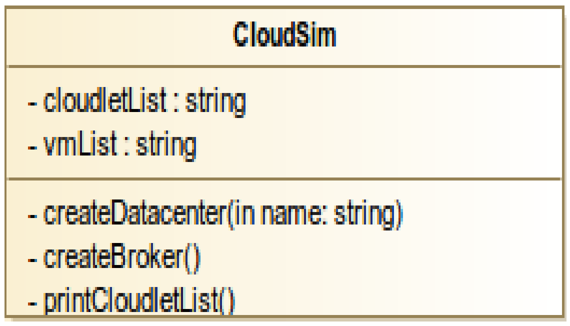
Activity Diagram



Sequence Diagram



Class Diagram



Conclusion

Thus, we understood concepts of Cloudsim, created two virtual machines and transferred a cloudlet from one VM to another successfully. We also installed Docker and hosted an image on it. A script was successfully executed on this .iso image.