|  |
| --- |
| **ASSIGNMENT NO.**  **Title:**  Design a GUI for VM Creation from user requirements using OpenNebula API.  **Aim**:  To design a GUI for creating and viewing VM instances running on OpenNebula cloud using OpenNebula Java API.  **Objective:**  To understand the OpenNebula API and the process of VM Creation and deployment.  **Theory:**  **OpenNebula**  OpenNebula is a cloud computing platform for managing heterogeneous distributed data center infrastructures. The OpenNebula platform manages a data center's virtual infrastructure to build private, public and hybrid implementations of infrastructure as a service. The two primary uses of the OpenNebula platform are data center virtualization solutions and cloud infrastructure solutions. The platform is also capable of offering the cloud infrastructure necessary to operate a cloud on top of existing infrastructure management solutions. OpenNebula is free and open-source software, subject to the requirements of the Apache License version 2.  OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services (e.g. compute clusters) as virtual machines on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies. The toolkit includes features for integration, management, scalability, security and accounting. It also claims standardization, interoperability and portability, providing cloud users and administrators with a choice of several cloud interfaces (Amazon EC2 Query, OGF Open Cloud Computing Interface and vCloud) and hypervisors (Xen, KVM and VMware), and can accommodate multiple hardware and software combinations in a data center.  The latest version of OpenNebula is **5.4.0** released in **July 2017**.  The OpenNebula Project's deployment model resembles classic cluster architecture which utilizes   * A front-end (master node) * Hypervisor enabled hosts (worker nodes) * Datastores * A physical network     OpenNebula supports KVM, QEMU, VMWare, etc. as it's virtualization backends. It also supports Amazon AWS for cloud bursting. It supports most popular operating systems for front end isntallation including RedHat, Ubuntu, CentOS, Debian, etc.  **OpenNebula Java API Reference:**   1. class **Client**   This class represents the connection with the core and handles the xml-rpc calls.  Created as:  Client client = new Client(String secret, String endpoint)  Where,  secret: A String of form <username>:<password>,  endpoint, A string of the form https://<host>:<port>/<endpoint>  2. class **VirtualMachinePool**  This class represents an OpenNebula VM pool. It also offers static XML-RPC call wrappers.    Created as:  VirtualMachinePool vmPool = new VirtualMachinePool(client)  Where,  Client: A OpenNebula Client object.    vmPool.infoAll():  This method loads the information about all the virtual machines available in the OpenNebula instance and allows us to iterate over them using vmPool.iterator().  3. class **Template**  This class represents an OpenNebula template. It also offers static XML-RPC call wrappers. It can also be used to instantiate a template using the Template.instantiate(String name, boolean onHold, String template) method.    4. class **OneResponse**  This class encapsulates OpenNebula's XML-RPC responses. Each response carries a boolean indicating if it is an error. It can also contain a success message, or an error message.  To get the message received as response, we can use the getMessage() method and to get the error, we can use the getErrorMesage() method.  **Mathematical Model:**  Let S be the system such that:  S={s,e,X,Y,F,Sc,Fc}  Where,  s= initial state  e= end state  X= set of inputs  Y= set of outputs  F= set of function  Sc= Success cases  Fc= Failure cases  Let S’ be system in observation  Where S’ C S  S’ = {s,e,X,Y,F,Sc,Fc}   * S= start state   { Logged out of OpenNebula Instance. Waiting for Username, password from user.}   * e= end state   exit(0) ….success   * X= {Username, Password, VM Name, VM RAM Size} * Y= {Y1, Y2}   Where ,  {Y1,}Є success  { Y2} Є failure   * F= {F1, F2,F3,F4,F5,F6,F7}   F1 = doLogin()  F2 = refreshUi()  F3 = loadVMs()  F4 = instantiateVM()   * Sc= {Y1,}   where Y1 = VMs viewed and created successfully after login.   * Fc = {Y2}   where Y1 = Incorrect login credentials or insufficient memory.  **Input:** Username, Password, VM Name, VM RAM Size  **Output:**  List of VMs, Creation of VMs.  **Platform:** Ubuntu 16.04  **Conclusion:**  Thus, we have successfully created a GUI application to list VMs available in OpenNebula and create a VM according to users requirements. |