Automatic Deployment of Virtual Labs on Cloud Infrastructure

Archita Jindal
Malaviya National
Institute of Technology
Jaipur, India
archita.jindal@gmail.com

Ashish Agarwal
The LNM Institute
of Information Technology
Jaipur, India
ashish.agarwal@lnmiit.ac.in

P. Amulya Sri
International Institute
of Information Technology
Hyderabad, India
pamulyasri@gmail.com

***Abstract*—Virtual labs are the collection of web-applications which interact with browser on one end and with set of virtual lab services at other end. At present, these virtual labs are deployed on physical servers. But, availability, scalability, and robustness becomes an issue in long run. Hence, the better option is to move these labs to a cloud infrastructure. This paper outlines an attempt by the institution, to deploy an Infrastructure as a Service cloud to enhance the student learning experience. As part of this, we introduce a python API that simplifies the deployment of Virtual Labs on cloud infrastructures. We describe the features and functionality of API, and how this API can be used to set up experiments on academic clouds such as OpenStack, Eucalyptus.**

***Keywords*—Eucalyptus, OpenStack, Virtual Labs.**

I. INTRODUCTION

Virtual labs: Virtual Lab is a complete Learning Management System. All the relevant information including the theory, lab-manual, additional web-resources, video-lectures, animated demonstrations and self-evaluations are available at a common place. Virtual labs are used to create interactive tutorials that provide students a visual demonstration of techniques. Virtual labs provide a self-paced learning environment, where individuals can learn from repeated practice, without using expensive resources or high maintenance equipments. Virtual Labs can be used in a complementary fashion to augment the efficacy of theory-based lectures. Small projects can also be carried out using some of the Virtual Labs. These labs can be effectively used to give lab-demonstrations to large classes.

These labs have been developed by lab developers from various educational institutes across India. Each lab has various pre-installation package dependencies and runtime requirements. These labs were initiated with a objective to share costly equipment and resources, which are otherwise available to limited number of users due to constraints on time and geographical distances. In order to make these labs remotly accessible, deployment of labs over cloud infrastructure was chosen.

Cloud Computing: Cloud computing is not just a service being offered from a remote data center. It is a set of approaches that can help organizations quickly, effectively add and subtract resources in almost real time. Cloud computing provides the means through which resources such as computing power, computing infrastructure and applications can be delivered to users as a service wherever and whenever they need over the Internet. The following are the variety of different service models in which required resources can be deployed,

- *Infrastructure as a Service (IaaS):* It is a provision model in which an organization outsources the equipment used to support operations, including storage, hardware, servers and networking components. Using this hardware the end user can run arbitrary software including operating systems or applications. The end user does not manage the underlying fixed infrastructure used to provide the cloud services instead manages how their applications are developed and designed.

- *Software as a Service (SaaS):* This service model allows the end user to use software provided by the cloud provider. The user accesses these services via a thin client interfaces such as web browsers.

- *Platform as a Service (PaaS):* This service model allows custom applications to be deployed into the cloud without the need for the end user to manage the hardware. In other words, it is a way allowing the user to rent virtualized servers and associated services for running existing applications. Examples of this would include Microsofts Azure or Googles App Engine.

To provide a sandbox environment, where each lab is independent from other, it would be preferable to use Infrastructure as a Service (IaaS) model. Because of this requirement, two open source cloud platform, namely Eucalyptus and OpenStack were chosen.

An overview of different cloud platforms is covered under section II. Section III describes the aspects of Virtual Labs deployment within the university environment. Our API

overview and implementation is covered under section IV. Section V describes Unit Tesing environment, followed by conclusion alongwith future work under section VI.

II. Background

A. *Eucalyptus*

Eucalytpus is a free and open source software architecture for implementing private and hybrid clouds on existing Private IT infrastructure. Eucalyptus is the acronym for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems. Eucalyptus implements an IaaS (Infrastructure as a Service) private cloud that is accessible via an API compatible with Amazon EC2 and Amazon S3.

The Eucalyptus cloud platform primarily comprises of five main components, Cloud Controller (CLC) that manages the virtualized resources; Cluster Controller (CC) controls the execution of virtual machines; Walrus is the storage system, Storage Controller (SC) provides block-level network storage including support for Amazon Elastic Block Storage (EBS) semantics; and Node Controller (NC) is installed in each compute node to control virtual machine activities, including the execution, inspection, and termination of virtual machine instances.

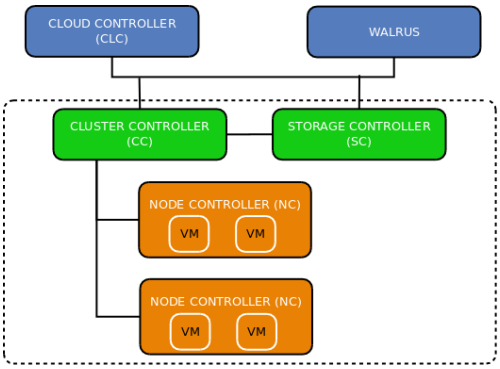

Fig. 1: Eucalyptus Architecture

Euca2ools are command-line interface to interact with web services. They are compatible with Amazon EC2, s3.

B. *OpenStack*

OpenStack is a collection of free and open source software cloud computing platform to deliver public and private clouds. The technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources through a data centre. OpenStack is deployed as an infrastructure as a service (IaaS) solution which can be managed or provisioned through a web-based dashboard, command line tools, or an API, which are also compatible with Amazon EC2 and Amazon S3, thus, can access the client applications written for Amazon Web Services with OpenStack. One of the key advantages to OpenStack is its greater flexibility in the management of users.

OpenStack uses various components currently include OpenStack Compute (called Nova), OpenStack Image Service (called Glance), OpenStack Networking (called Neutron), OpenStack Object Storage (called Swift), OpenStack Identity Service (called Keystone), OpenStack Volume Service (called Cinder) and OpenStack Dashboard (called Horizon).

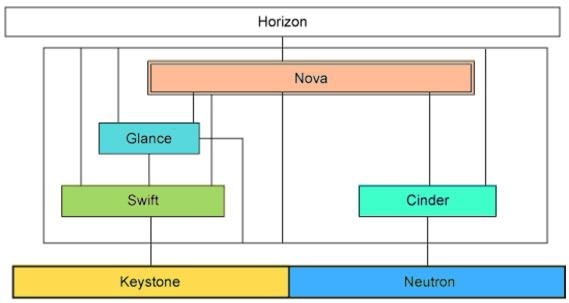

Fig. 2: Openstack Architecture

Nova is designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud computing platform. Glance provides discovery, registration, and delivery services for virtual disk images. Keystone acts as a common authentication system across the cloud operating system and supports multiple forms of authentication including standard username and password credentials.

C. *Supporting Libraries*

Boto provides an integrated API to access cloud services. It includes support for the Amazon EC2 and private cloud systems such as Eucalyptus, Openstack that implements EC2 compitable services. *Paramiko* provides secure (encrypted and authenticated) connetions to remote virtual machines throuch the SSH2 protocol.

The following section discribes the structure and layout of the automated deployment of the virtual lab over cloud infrastructure.

III. Deployment of Virtual Labs

Deployment of Virtual labs in a distributed environment through our API is a simple 4 step procedure:

1) Identifying Lab specifications (that includes OS, RAM and disk size),
2) Creating Virtual Machine with the help of identified specifications,
3) Automatic installation of lab dependencies (that includes, software such as Apache, PHP etc),
4) Automatic deployment of lab.

*1) **Identifying Lab specifications**:* As mentioned earlier each lab has different software package dependencies. So if we want to host these labs in a cloud framework, assuming one Virtual Machine per lab, each virtual machine should have customized environment where these labs would be running. Lab developer would provide these specifications in the form of a dictionary. Virtual machines configuration parameters are then determined from this dictionary.

*2) **Creating Virtual Machine with the help of identified specifications**:* In order to create a new virtual machine, user have to provide the image ID, flavor(ram size , disk size and no of cpu's), ssh key pair, security group.

a) **Images:** Images here refer to the operating system on which labs would be deployed. In our API, client can either use our customized cloud image or their own cloud images. Client can use any standard UNIX cloud images that do not need to contain predefined software. Any interaction with virtual machine is done over SSH connections and further dependencies can be installed. A customized cloud image with minimal lab dependencies can be used as an alternative. Considering this, our API provides user with a separate image class named *ImageAdapter* for both OpenStack and Eucalyptus . This class returns the image ID that is used as a parameter for virtual machine creation.

For Eucalyptus:
Image creation process in Eucalyptus involves 3 steps:
Bundling a root disk image and kernel/ramdisk pair,
Uploading the bundled image to Walrus bucket storage,
Registering the data with Eucalyptus.
As soon as image gets registered, Eucalyptus provides us with a unique image ID. It is often referred as EMI.

Eucalyptus Machine Image: A Eucalyptus Machine Image (EMI) is a template that can be used to create and run virtual machines. It is stored in Walrus. A EMI is a combination of: a kernel image (Eucalyptus Kernel Image - EKI), a ramdisk image (Eucalyptus Ramdisk Image - ERI) and one or more virtual hard disk image.

For Openstack:
For image creation in OpenStack, Identity (Keystone) and Image Service(glance) API is used. Firstly, the Keystone servers as a service catalog, which allows to look up a service by its type (for example image, volume, compute, network etc), and its endpoint type (publicURL, internalURL, adminURL), thus, letting the service accessed by a client application. After this, a client makes requests against the Identity(keystone) API by instantiating the appropriate keystone client Python object and calling its methods, associated with a specific version of the API. After making appropriate

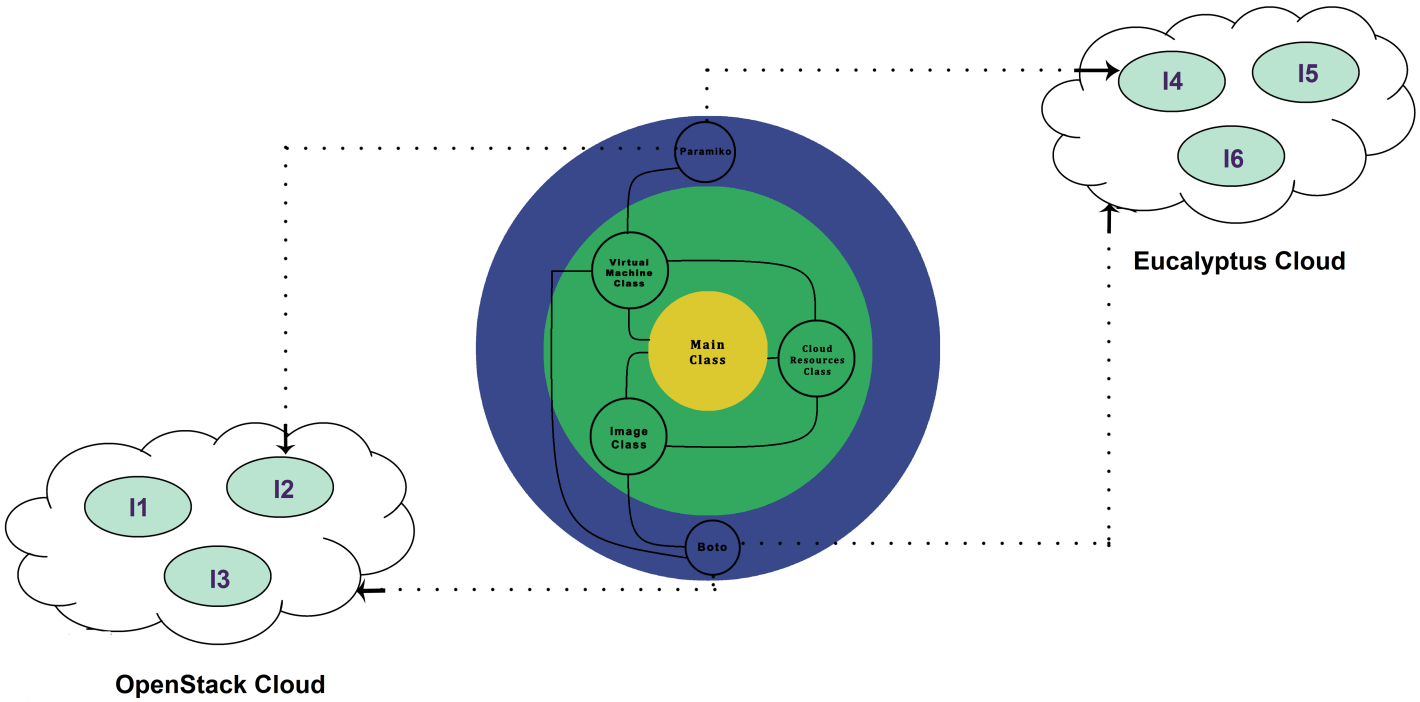

Fig. 3: Overview of APIs Architecture. The figure shows the functional layers in the system, starting with the Main class in the center, other API classes that deals with virtual machine management in the next layer, and supporting libraries (Boto and Paramiko) as the outer layer. The figure also highlights the fact that API can concurrently interact with multiple cloud infrastructures. Virtual Machine Instances are represented via I1, I2, I3 etc.

connections, Image gets created by uploading the unique cloud image with specific disk, container formats, etc required and storing it with a unique image id which can be accessed by the virtual machine for further creation.

b) **Flavor(ram size, disk size and no of cpu's):** These parameters are determined from the lab specification as mentioned earlier and can be used directly by passing as the parameter for virtual machine creation.

c) **SSH keypairs:** Keypairs are cryptographic keys used to verify access to a particular virtual machine. Creating a key pair generates two keys: a public key (saved within Cloud) and a corresponding private key (output to the user as a character string). When we create a virtual machine instance, the public key is then injected into the virtual machine. Later, when attempting to login to the virtual machine instance using SSH, the public key is checked against your private key to verify access.

d) **Security Groups:** A security group is defined as a named collection of access rules. For instance, if the user requires a specific set of opened ports, the rules that infer those ports can be added to the security group. Once the virtual machine gets created, both Eucalyptus and OpenStack provides virtual machine with unique ID and ip address.

*3) **Automatic Installation of lab dependencies**:* With the given ip address, we will be able to log in to the virtual machine. Boto and Paramiko along with virtual machine Manager are used to set up a lab on virtual machine. VMManager is a python API and a connection end-point for the *one virtual machine per lab* system. Once we ssh into the virtual machine, required lab dependencies can be installed using VMManager.

*4) **Automatic deployment of lab**:* The cloning of lab repository from git and deploying it on the virtual machine can be done automatically using VMManager.

The design and implementation details of the API are discussed briefly in the following section.

IV. API Implementation

We designed our API in a layered architecture, which is represented in concentric regions, as shown in the figure 3. The Main class refers to the central layer, while the internal application layer explains the various classes and functionalities associated with virtual machines. The external layer describes the libraries used to manage the resources.

API is built using Python, which was chosen because of its ease of use and ubiquity in academic computing. The Design of our API was motivated by the domain and technical requirements which are as follows :

A. Domain Requirements

- Each lab is deployed in a sandbox environment, **'One virtual machine per lab'**. Each lab was developed in a customized independent environment in such way that execution of one lab will not effect other virtual labs.

B. Technical Requirements

- Automatic logging. API calls are time stamped and logged.
- Basic handling of SSH keys and security groups in a fault tolerant manner.

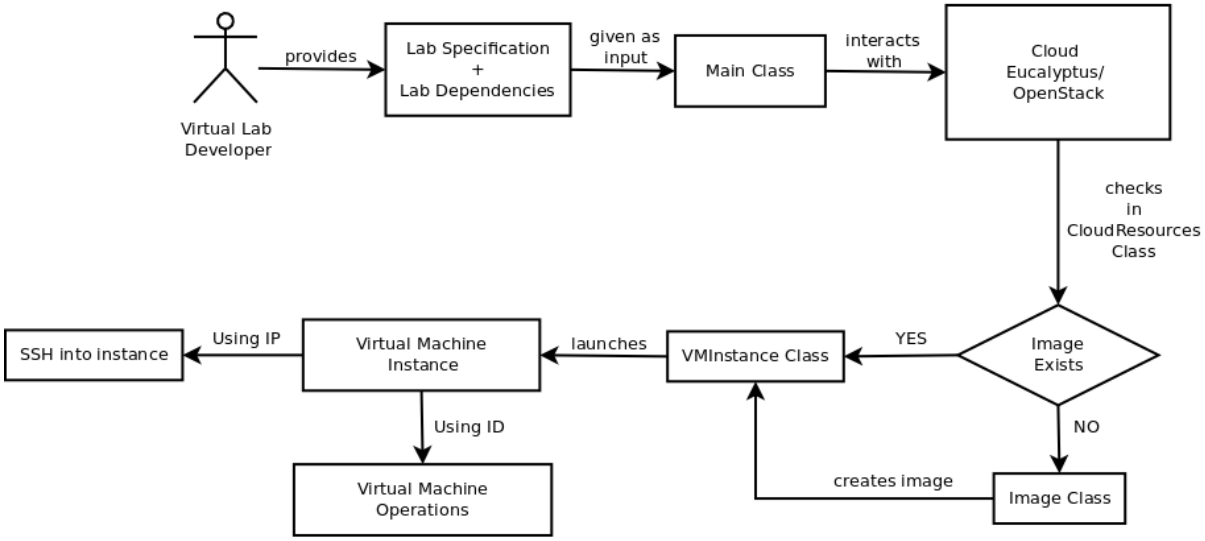

Fig. 4: Workflow API, Virtual Machine Instance

- Exception handling class, common for openstack and Eucalyptus.
- Unit test coverage for the API implemented.

C. Design Details

Our API can be divided into various modules as follows:

- **Virtual Machine Class** which creates, starts, stops and restarts the virtual machine,
- **Cloud Resources Class** containing methods to get various configuration parameters of virtual machine and cloud such as get_image_list, get_vm_list, etc,
- **Image Class** used to create Image, delete Image and
- **Main Class** which is used as controller for all these classes.

Lab developers will provide us with *Lab spec* (Lab specifications) and *Lab dependencies* in the form of dictionary. Lab specification includes, configuration parameters for the virtual machine such as operating system, ram size, disk size etc.

*1) **Virtual Machine Image**:* Our API takes the image from this dictionary and checks for its presence in the Cloud using CloudResources class. If it exists, Image object is returned. Else, API creates the desired image using *create_image* method in Image class. This image object provides us with image id which is passed to VMInstance class.

Create Image: Eucalyptus

```
def _create_(self, image_name, image_path, image_arch):
    """
    Register an EMI.

    :param image_name: The name of the EMI. Valid only for EBS-based images.
    :param image_path: Full path to your EMI manifest.
    :param image_arch: The architecture of the AMI. Valid choices are:
        * i386
        * x86_64
    """
    logging.info('Creating Image...')
    try:
        logging.info('Waiting for Image to CREATE...')
        bundle_path = self._bundle_(image_name, image_path, image_arch)
        upload_path = self._upload_(bundle_path)
        register_id = self._register_(image_name, upload_path)
        self.id = register_id
        self.name = image_name
        self.architecture = image_arch
        logging.info('Image is in PENDING state...')
        self._is_image_exists = True
    except:
        logging.error('Image not CREATED...')
        raise
```

*2) **Vitrual Machine Instance**:* Apart from image id, *create_vm* method in VMInstance class requires ssh key, security group and flavor(ram size, disk size and no of cpu's) to create the virtual machine as described in previous section. Once, virtual machine gets launched, it would be assigned with a virtual machine ID and ip address automatically. Virtual machine ID acts as a unique identifier through which API can perform all virtual machine related operations like start, stop, restart, destroy etc. Virtual machine ip is used to communicate with it using ssh. Once we ssh into the virtual machine, we can install all the required dependencies.

Create Image: OpenStack

```
def create(self, image_name, image_path):
    """
    Create an image instance.

    :param image_name: The name of the image to be created.
    :param image_path: The path of the image for creation.
    :rtype: :object: Image Instance object.
    """
    try:
        logging.info('Waiting for image instance to create.')
        endpoint = self.keystone.service_catalog.url_for(service_type='image',
                endpoint_type='publicURL')
        image_config = glanceclient.Client(version='1', endpoint=endpoint,
                token=self.keystone.auth_token)
        with open(image_path, 'rb') as fimage:
            image = image_config.images.create(name=image_name, is_public=True,
                disk_format="qcow2", container_format="bare", data=fimage)
        return image
    except:
        logging.error('Image not created.')
        raise
```

*3) **Virtual Lab Deployment**:* Setting up a virtual lab consists of two phases:

1) Establishing a connection to virtual machine on cloud and
2) Registering a pair of SSH keys.

API manages communication with virtual machine instances in a separate context from the users local system. This ensures that the virtual lab management does not interfere with an existing cloud setup of the system. Communication is performed through SSH key pairs and security groups. API uses the Boto and Paramiko libraries to communicate with the cloud providers and to manage SSH connections to the virtual machine instances. On API's first execution, a specific SSH key pair is created and stored on researchers local machine. On the remote cloud infrastructure, the key pair is automatically registered and a API security group is created.

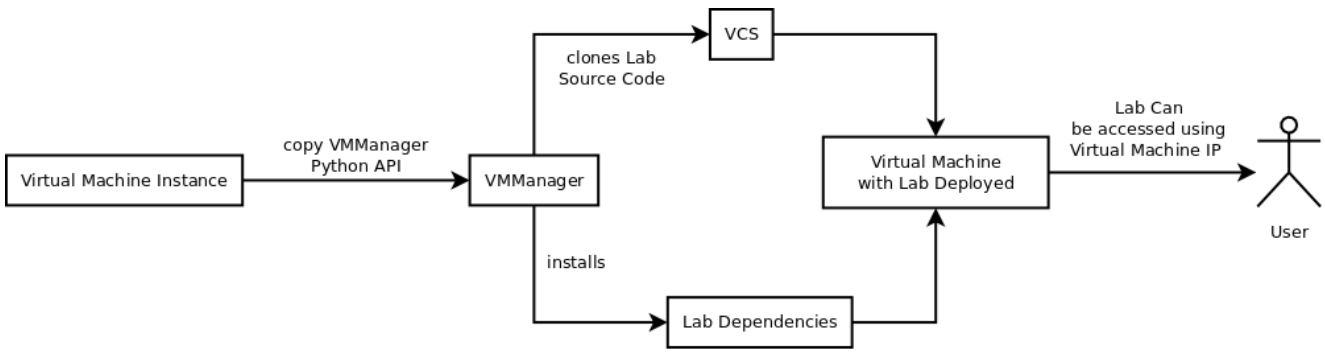

Fig. 5: Workflow API, Virtual Lab Deployment

After getting connected to the virtual machine remotely using ssh, we can install VMManager and all the specified *Lab Dependencies*. The lab dependencies are provided to API in the form of dictionary by lab developer. They include, various pre-installation packages such as *apache2, PHP, MySQL* which are required for labs functionality. These dependencies are then installed on to the virtual machine by VMManager. To deploy and host a lab, VMManager clones the source code from version controlled repository, into the virtual machine. This lab can be accessed using ip of virtual machine.

Create Virtual Machine: Eucalyptus

```
def _create_(self, image_id, security_key_name, security_group_name, instance_type):
    """
    Create a virtual machine instance.

    :param image_id: The ID of the EMI used to launch this virtual machine.
    :param security_key_name: The name of the SSH key associated with
                              the virtual machine.
    :param security_group_name: List of security Groups associated with
                                the virtual machine.
    :param instance_type: The type of instance (e.g. ml.small).
    """
    count = 0
    try:
        reservation = self.__conn__.run_instances(image_id=image_id,
                key_name=security_key_name, security_groups=security_group_name,
                instance_type=instance_type)
        instance = reservation.instances[0]
        logging.info('Waiting for instance to CREATE...')
        self.instance = instance
        self._is_instance_exists = True
        logging.info('Instance is in PENDING state...')
    except:
        logging.error('Instance not CREATED...')
        raise
```

4) ***Logging****:* To provide a complete record of the lab deployment, all major events in API get logged to the console using Pythons standard logging framework. The events are time stamped in the ISO 8601 format (YYYY-MM-DD hh:mm:ss). Logging includes virtual machine events such as creation of virtual machine and other operations. File transfers and remote command executions are logged as well. User can add their own events simply by also using the Python logging framework.

Create Virtual Machine: OpenStack

```
def create(self, vm_name, image_name, vm_ram, vm_disk, vm_vcpus):
    """
    Create an virtual machine.

    :param vm_name: The name of the virtual machine instance to create
    :param image_name: The name of the image through which we create.
    :param vm_ram: The ram size of the virtual machine instance.
    :param vm_disk: The disk size of the virtual machine instance.
    :param vm_vcpus: Number of CPU's required by the virtual machine instance.
    :rtype: :object: virtual machine Instance object.
    """
    try:
        self.key_name = 'key_pair1'
        logging.info('Waiting for virtual machine instance to create.')
        if not self.nova.keypairs.findall(name=self.key_name):
            with open(os.path.expanduser('~/.ssh/id_rsa.pub')) as fpubkey:
                self.nova.keypairs.create(name=self.key_name,
                        public_key=fpubkey.read())
        vm_flavor = self.set_flavor(vm_ram, vm_disk, vm_vcpus)
        image_object = self.nova.images.find(name=image_name)
        instance = self.nova.servers.create(name=vm_name, image=image_object,
                flavor=vm_flavor, meta=None, key_name=self.key_name)
        logging.info('Instance is in pending state.')
        self.instance = instance
        self.is_instance_present = True
        return self.instance
    except:
        logging.error('Instance not create.')
        raise
```

V. Testing Environment

Uncovering errors during unit testing only, lessens the probability of the propagation of errors to other phases in a large number. While this fact is applied to object oriented software, it is understood that the fundamental units with object oriented software are precisely the classes and hence the classes need to be thoroughly tested to accomplish unit testing. Testing of a class is analogous to testing the methods defined as part of the class.

Hence, as part of testing this API we use Python standard unit testing framework called *nose*. Nose comes with a number of built-in plugins to help us with output capture, error introspection, code coverage, doctests, and more. It also comes with plugin hooks for loading, running, watching and reporting on tests and test runs. Nose provides a library called nosetests that can use to automatically discover and run tests.

VI. Conclusion and New Research Directions

In this paper we have developed an API which provides a generic web application management tool for virtual labs deployment on cloud frameworks. We have provided an overview of the important methods of the API. We then demonstrated API features by performing a simple virtual machine creation and deployind a lab on it. Using this scheme, we can easily build the virtual machines and deploy lab on them. By deploying labs on the cloud, we can reduce the cost and makes these virtual labs easy to access from anywhere with great robustness. We would like to motivate research to a) emphasize on creation of customized cloud image which consists of all the dependent packages to deploy virtual labs. b) Cloud computing has got tremendous success with its, pay-as-much-as-you-use model. However, there is lot of scope required to develop an elastic model which helps to use infrastructure provided by cloud service providers in an optimal way.

Acknowledgment

This project is initiated and funded by Ministry of Human Resource Development, Government of India.

References

[1] Eucalyptus community official web site. [Online]. Available: http://open.eucalyptus.com/

[2] OpenStack official site. [Online]. Available: https://www.openstack.org/

[3] "Boto: A Python interface to Amazon Web Services,". [Online]. Available: https://github.com/boto/boto. [Online]. Available: https://github.com/boto/boto.

[4] Daniel ANurmi, Rich Wolski, Chris Grzegorczyk Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov , The Eucalyptus Open-source Cloud-computing System ,9th IEEE/ACM International Symposium on Cluster Computing and the Grid

[5] Huioon Kim; Hyounggyu Kim; Youngjoo Chung, "Building a Eucalyptus cloud automatically with Fully Automatic Installation," Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on , vol.2, no., pp.556,560, 25-27 May 2012

[6] Ruotz, C.; Schmaltz, J., "An Experience Report on an Industrial Case-Study about Timed Model-Based Testing with UPPAAL-TRON," Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on , vol., no., pp.39,46, 21-25 March 201

[7] Bonner, S.; Pulley, C.; Kureshi, I.; Holmes, V.; Brennan, J.; James, Y., "Using OpenStack to improve student experience in an H.E. environment," Science and Information Conference (SAI), 2013 , vol., no., pp.888,893, 7-9 Oct. 2013

[8] Azarnoosh, S.; Rynge, M.; Juve, G.; Deelman, E.; Niec, M.; Malawski, M.; da Silva, R.F., "Introducing PRECIP: An API for Managing Repeatable Experiments in the Cloud," Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on , vol.2, no., pp.19,26, 2-5 Dec. 2013

[9] von Laszewski, G.; Diaz, J.; Fugang Wang; Fox, G.C., "Comparison of Multiple Cloud Frameworks," Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on , vol., no., pp.734,741, 24-29 June 2012