Amazon Web Services Cloudmesh Extension

GREGOR VON LASZEWSKI[1], MILIND SURYAWANSHI[1], AND PIYUSH RAI[1]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*

project-006, May 12, 2017

Cloudmesh client provides and easy abstraction to gain access to hybrid clouds from the commandline. In contrast to other systems it provides a mechanism of adding plugins to the command. The command can not only be used as a commandline tool, but as a shell. Due to the swift developments in the area of cyber infrastructure such a convenient plugin allows us to integrate easily new services easily. This work uses our next generation cloudmesh shell and demonstrates the ease of integrating Amazon Web Services services into it. Hence Cloudmesh client can via other plugins achieve interoperability for virtual machine management by providing plugins for OpenStack, Azure, Comet Cloud Services and other frameworks including containers. Here we focus on extensions to bring its capability to work with Amazon EC2 services. As our integration demonstrates how to leverage libcloud as a backend, more than 30 other providers could easily be targeted.

Keywords: Cloud, I524

Report: https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P006/report/report.pdf
Code: https://github.com/cloudmesh/cloudmesh.aws

1. INTRODUCTION

Amazon provides a web service in a cloud (AWS) to deploy virtual machines (VM) with numerous operating systems environment available in different flavors. The images are called Amazon Machine Image (AMI) and are available as pre-configured for different applications. Once can also create his own custom environment [1].

The Amazon EC2 drivers provides numerous functionalities to authenticate into the AWS, list available configurations and create and boot VMs. Many interfaces exist providing access to many different frameworks and platforms. Being able to utilize access through Web services allows developers to integrate AWS into their service offerings. Within Indiana University we have provided for the last seven years cloud services based on a variety of cloud services. While observing the evolution and practical use of these services it has become clear that in order to switch easily from one cloud to another we need a uniform interfaces that provides the most elementary service offerings of managing virtual machines. Our design offers tha ability to switch easily between different cloud providers. Figure 1 demonstrates this convenient feature. Here we define in our cloudmesh client a cloud and set it to AWS, we than start a vm, we can do the same for OpenStack as demonstrated. Important to note is that all details of name assignment, flavor and image management, are abstracted out, but can be controlled and are provided as part of an easy to manage configuration framework. This has advantages for university settings as we can distribute such configurations for particular classes and customize the use for many hundrets of non cloud experts. In fact using our abstractions allows non cloud experts easily to use clouds.

```
set default cloud=aws
vm boot
set default cloud=openstack
vm boot
```

Fig. 1. cms aws image refresh

2. GETTING STARTED

One needs to create an aws account first to be able to access its cloud. The instructions for it are available at [11]. A pair of access key and secret keys are required to be generated to authenticate into the cloud [12]. These keys are required to be kept confidential. They are to be specified in a yaml configuration file. Default vm image and flavor can also be specified in the configuration file. Following are some of the configuration entries:

```
cloudmesh:
clouds:
aws:
credentials:
```

```
EC2_ACCESS_KEY: 'ACCESS KEY'
EC2_SECRET_KEY: 'SECRET KEY'
.
.

default:
image: 'IMAGE ID'
size: 'SIZE ID'
location: 'LOCATION'
```

The user need to ensure that the combination of image, size and location is valid and that it's account has the required privileges for it. The instructions for installation of AWS client can be found at [13]. Once, the client has been installed and configurations settings enabled, the mongodb and the rest services to access the database can be started by following command from the root directory of the code:

```
make rest
```

The above services will be required by AWS client to store cloud related information locally. The user can now execute the client commands e.g.:

```
cms aws flavor refresh
```

This will fetch the list of image sizes available on Amazon EC2 cloud.

3. CLOUDMESH DESIGN

The cloudmesh client toolkit [2] is a lightweight client interface of accessing heterogeneous clouds, clusters, and workstations right from the users computer. The user can manage her own set of resources she would like to utilize. Thus the user has the freedom to customize the integration of a variety of cyber infrastructure resources into Cloudmesh. Cloudmesh client includes an API, a command-line client, and a command-line shell. It strives to abstract backends to databases that are used to manage the workflow utilizing different infrastructures and services. As explained earlier, switching to activate virtual machines from OpenStack clouds to Amazon is as simple as specifying the name of the cloud. Moreover, cloudmesh client can be installed on Linux, MacOSX, and in future Windows.

3.1. General Requirements and Goals

Cloudmesh has from its inception followed the following general design goals and requirements.

Technology agnostic. An important aspect of cloudmesh is to offer access to useful services APIs and interfaces in a technology agnostic fashion. Thus it should be possible for example to switch easily between different IaaS providers. CM has excellently protected us during the changes of the OpenStack interfaces and libraries from the very first version of OpenStack and other IaaS frameworks [3].

Easy to use. Cloudmesh simplifies access of complex workflow to integrate with IaaS and deploy on them new platforms and software stacks. This advanced feature is not only to be performed by expert IT personal or programmers, but in fact by data scientists which we found in practice have less experience in such areas. As we deal often with many compute nodes it is often insufficient to just provide a graphical user interface or portal, but we need to provide APIs, REST interfaces and especially command-line and shell access in an easy comprehensible fashion.

Expandable. While working over the last years in the area it is obvious that the technology is rapidly evolving and new features need to be integrated, Thus it is important that cloudmesh is easy to expand and new features can be added while leveraging a core set of functionality and services.

Documented. It is important that we provide from the start documentation to existing and new features and make it easy for the developers to contribute documented add ons, but also allow users to have access to documentation including examples. This will include easy to follow documented installation and configuration steps to guarantee successful deployment and use

Repeatable and automated deployment. When we install software stacks an a variety of platforms it is expected that that can easily be replicated and repeated automatically.

Portable. It is important to provide services in cross platforms compatible fashion. It ensures working, executable software deployment on multiple platforms. This includes not only the installation of the software, but the integration of external services and tools such as DevvOps or workflow frameworks that could support the general mission of a data scientist.

Abstractions. To address some of the design issues our requirements implicitly asks for the existence of a number of abstractions and interfaces that can be used to enable portable and crossplatform services and tools.

3.2. Architecture Requirements and Goals

In addition to the general requirements we set some specific architectural requirements and goals that have been growing from previous versions of cloudmesh and our earlier work in this area.

Client based. Cloudmesh client as the name indicates is a client based toolkit that is installed and run on the users computers. An add on component to use the client within a portal is available. Thus we distinguish the client that contains most of the functionality and the portal that can access the functionality through a locally maintaine Web portal. Important to note is that the user manages its own credentials and thus security and credential management is done directly on the users machine instead through a hosted Web portal. This increases the security as access to any credential is managed by the user and is not part of a credential management system.

REST. Although Cloudmesh provides a client interface, it will provide a REST interface to many of its services in order to support service based deployments. The basic APIs developed for the client can easily be reused to implement such interfaces.[1]

Layered Architecture. Cloudmesh client has a layered architecture that allows easy development of new features. This

[1]we have demonstrated that cloudmesh APIs can be used to implement REST interfaces in a variety of frameworkks such as Flask, Django, and Cherypy

also allows contribution by the community while developing integrated and smaller sub components. Figure **??** depicts the various layers. A resource abstraction layer allows the integration of a multitude of resources spanning HPC, Containers, and Cloud resources. (At this time we focus on OpenStack and Slurm resources. We are working on reintegrating resources such as Azure, AWS, Maui, Moab, and others which we previously supported, as well as new resources such as docker).

Management Framework. Cloudmesh client contains a management framework, and its components are depicted in Figure B. cloudmesh allows easy management of virtual machines, containers, and the data associated with them. We are currently developing a choreography framework that leverages Ansible, chef, and heat. All of the functionality is easily usable through a comand-shell that also can be used from the command-line, and a Python API. IN future we will be providing a REST API.

Database Agnostic. Cloudmesh contains some state about the resource and environment that a user may want to use. The information is managed in an database abstraction that would allow storing the data in a variety of databases such as SQL and MongoDB. At this time we have chosen SQLite to be the default database as it does not require any additional setup and is universally available on all operating systems without change.

Command-shell and Command-line. Cloudmesh contains a comand-shell allowing scripts to be developed and run. However we designed the comand-shell in such a way that each command can also be called from the command-line. Through the cloudmesh state machine the state between comand-shell, command-client, and the portal is shared.

Cloudmesh Client Portal. Previously, we distributed cloudmesh with client, server, and a portal components in one package. This however turned out to be to complex to be installed for some of our less technically skilled user community. Thus we split up the install into two independent packages. The cloudmesh client and the cloudmesh portal. The portal provides some elementary features to manage virtual machines and HPC jobs. At this time the portal is considered to be alpha technology. Just as the client the portal is to be run on the local user machine in oredr to allow increased security by managing the credentials locally rather than on a server.

Cloudmesh Two Factor Authentication. We have an exploratory project in place that looks at the use of Yubikeys for cloudmesh, client and cloudmesh portal.

Cloudmesh Comet. We are actively developing the client interface for SDSC's comet supercomputer allowing bare metal provisioning. The interface reuses cloudmesh components and technologies while interfacing with the comet cloud REST interface. The goal here is to manage virtual clusters.

3.3. Architecture

4. IMPLEMENTATION CONSIDERATIONS

The AWS client is based on cloudmesh.cmd5 [4] and cloudmesh.common [5]. It uses mongodb in the back-end to store the cloud information such as list of available images or

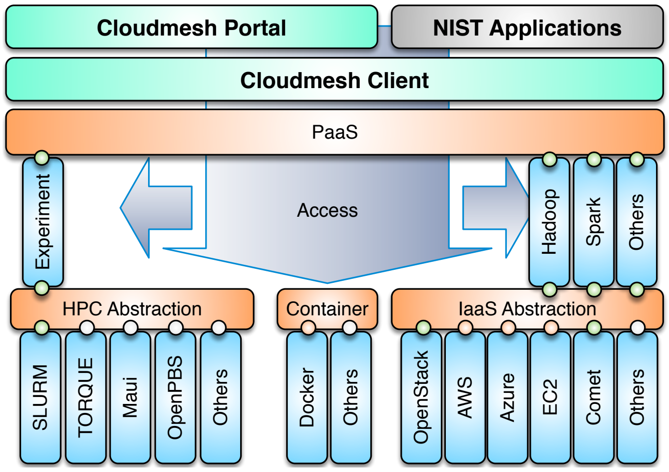


Fig. 2. Cloudmesh layered architecture.

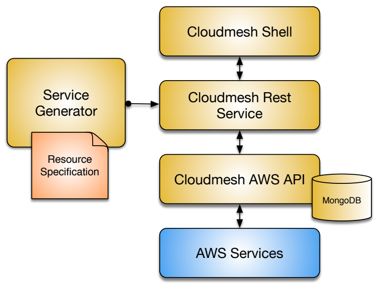


Fig. 3. Cloudmesh AWS REST Service.

instances running on the cloud. Requests library [6] is used to connect to the back-end through rest services. The rest service is deployed using cloudmesh REST framework [7].

Design Diagram is missing

```
client -> rest (eve) -> database (nosql)
          ^ object definitions
```

5. IMPLEMENTATION

Some details about REST service and auto creation from examples. Gregor may have text.

6. ARCHITECTURE

The commands are implemented as methods of a class AwsCommand which is based on PluginCommand from cloudmesh shell. Depending on the arguments passed, corresponding routine is called from Aws client API which acts as a wrapper around the libcloud EC2 drivers [8] and is also responsible to connect with back-end database apart from reading the configuration from yaml file. The entire code is written in Python.

6.1. Technologies Used

To adhere to the principal of code reuse we are using a number of core technologies that allow us to keep the overall developed new code small. This includes leveraging our previous *cloudmesh* efforts, the reuse of *mongodb* and the automatic creation of a REST service based on example objects with *evengine*. Particularly the following software components have been leveraged:

Libcloud EC2 driver [8]: The driver provides a number of functions for various functionalities such as listing the available nodes, generating a key pair and deploying a VM.

MongoDb [9]: MongoDB is an open source document store database. It's used by AWS client to store information about various VM configuration options available on the Amazon EC2 cloud. It's also used to store information regarding the VMs that are running on the cloud. The information in the database is refreshed whenever it's fetched from the cloud.

Cloudmesh.common [5]: A library of many commonly useful functions and classes shared among all cloudmesh projects.

Cloudmesh.cmd5: The cmd5 is a dynamically extensible CMD based command shell [4].

Cloudmesh.evegenie [10]: The schema for the collections are specified in json files which are then converted to Eve syntax using evegenie. It creates the configuration file for starting up the rest services using cloudmesh rest framework.

Cloudmesh.rest [7]: The cloudmesh rest framework is used to deploy and start the mongodb and rest services. The schema for the objects to be stored in the database is collectively specified in json file called *all.json*. The schema defined in the file is closely associated with the code in awsclient.py which is responsible for fetching the information from cloud and passing it to mongodb through rest services. From our experience during the development of this project, we observed that rest services required the schema to be precise and didn't handle null values for collection fields. There's another file all.settings.py which contains the configuration information for rest services such as port mongodb is running on along with the database name. It contains the schema for the collections and the list of methods to be provided by the rest services. The database connectivity was initially developed using pymongo library. However, during the review it was suggested that the code based on pymongo is quite low level and does not have adequate security features. This led to the use of Pyhthon Requests library.

7. USAGE

One needs to create an AWS account first to be able to access its cloud. The instructions for it are available at [11]. A pair of access key and secret keys are required to be generated to authenticate into the cloud [12]. These keys are required to be kept confidential. They are to be specified in a yaml configuration file. Default vm image and flavor can also be specified in the configuration file (see FIgure 4).

The user need to ensure that the combination of image, size and location is valid and that it's account has the required privileges for it. The instructions for installation of AWS client can be found at [13]. Once, the client has been installed and configurations settings enabled, the mongodb and the rest services to access the database can be started by following command from the root directory of the code:

```
make rest
```

```
cloudmesh:
  clouds:
    aws:
      credentials:
        EC2_ACCESS_KEY: 'ACCESS KEY'
        EC2_SECRET_KEY: 'SECRET KEY'
        . . .
    default:
      image: 'IMAGE ID'
      size: 'SIZE ID'
      location: 'LOCATION'
```

Fig. 4. Configuration

The above services will be required by AWS client to store cloud related information locally. The user can now execute the client commands e.g.:

```
cms aws flavor refresh
```

This will fetch the list of image sizes available on Amazon EC2 cloud.

8. CLOUDMESH REST SERVICE

```
{
    "aws_flavor": {
        "id": "t1.micro",
        "name": "Micro Instance",
        "ram": 627,
        "disk": 15,
        "price": 0.025
    }
}
```

Fig. 5. Specification for flavor.

```
{
    "aws_image": {
        "id": "ami-0180d261",
        "name": "ubuntu/images-testing/ubuntu-....-server-20170123",
        "driver": "string"
    }
}
```

Fig. 6. Specification for image.

9. AWS COMMANDS

10. BENCHMARKING CLOUDMESH AWS

We benchmarked the aws commands that fetch information either from the cloud over the network or from the local database. The information is then displayed to the user. The informations that is fetched over the network is also updated in the database. The performance statistics observed give us the overview of the difference in execution time of the two kind of commands. Each command was run 50 times with its execution time reocrded. The mean and standard deviation of the timings were calculated using statistical methods in R. Table 3 shows the observed values in seconds.

The commands that fetch the information from local database are on an average 85% faster than the commands that fetch information from the cloud and also have lesser variation in their timings. The difference in performance tends to decrease when the information to be fetched is more. This may be due to the limitation of the machine on which it was tested.

Table 1. Cloudmesh AWS Commands

Function	Description	Command
Refresh	Whether to always fetch the information from the cloud over the network or display it from the local database when asked can be configured by setting the configuration variable *refresh* to either *on* or *off*. When the value is set to on, the information will always be fetched from the cloud.This functionality is implemented for only some of the commands as of now.	`aws refresh on`
Image refresh	The images available on the cloud could be listed using this command. The database will also be updated with the newly fetched list. The command may take up to minutes to run as the list contains more than 24,000 images.	`aws image refresh`
Image list	Depending on whether the value of *refresh* is set to *on* or *off*, the list is either fetched from the cloud or from the local database.	`aws image list`
Flavor refresh	This will list the different sizes of VMs that are available on the cloud. The information is fetched from the cloud and stored locally.	`aws flavor refresh`
Flavor list	The flavor list is either fetched from the cloud or from the local database depending on whether the value of *refresh* is set to *on* or *off*. `aws flavor list`	
Start/Boot vm	It creates a new node instance and start that node automatically. The required parameter is *IMAGE_ID*. This command assumes user has created keypair with name *AWS1*. The default values; flavor and location are taken from cloudmesh.yaml. Those values are required to be set before executing this command. Once it has been created, it can be seen using vm list.	`aws vm boot IMAGE_ID`
Reboot vm	This command is used to reboot a running vm instance. The state of the node gets changed from RUNNING to REBOOTING while its restart. The rebooting time of node depends on the configuration the configuration that was selected while its creation. *vm reboot* requires *NODE_UUID* of the node to be rebooted which can be retrieved using either *vm list* or *vm refresh*.	`aws vm reboot NODE_UUID`
VM delete	It destroys the instance of a node and all the data associated with it including backup. It takes few seconds to terminated the instance. At first, the state of the node changes to *TERMINATED* which can be observed using *vm refresh* . The node gets removed from the list subsequently.	`aws vm delete UUID`
VM list	It lists out all the nodes along with their status as stored in the database.	`aws vm list`
Keypair create	To create the node, one of the essential component is *key pair*. Amazon EC2 uses public key of the user to encrypt the password and then recipient uses private key to decrypt it. These public and private keys are know as a *key pair*. They are required to be given a name.	`aws keypair create NAME`
Keypair delete	It allows user to delete a created *key pair* which is no longer in use.	`aws keypair delete NAME`
Keypair refresh	It refreshes the list of created *key pairs* in the database and also displays it on the screen.	`aws keypair refresh`
Keypairs list	It lists out all the *key pairs* information stored locally in the database.	`aws keypair list`
Keypair get	It returns the key pair object, which has the name of *key pairs*, driver and hash key.	`aws keypair get NAME`

Table 2. Cloudmesh AWS Commands

Locations refresh	It will show all the available locations associated with Amazon EC2 account. For free tier, user will get two locations. More locations will be available in paid service. The database records are also updated with this command.	`aws location refresh`
Locations list	It lists the locations information stored in the database.	`aws location list`
Volume create	It creates the volume for vm. The size of volume is specified in GB, the default value is set to 1 GB. The maximum number of volumes that can be attached to a vm will depend on its operating system.	`aws volume create VOLUME_NAME`
Volume re-fresh	This command shows the created volumes with the id, size and the driver name. It also updates the database.	`aws volume refresh`
Volume list	This command displays the volume information stored in the database.	`aws volume list`
Volume delete	User can delete the unwanted volumes using the *VOLUME_ID*.	`aws volume delete VOLUME_ID`

```
{
    "aws_keypair": {
        "name": "AWSTEST",
        "fingerprint": "e7:db:fe:a6:5e:...:d2:d3:f8:79"
    }
}
```

Fig. 7. Specification for keypair.

```
{
    "aws_location": {
        "region_name": "us-west-1",
        "availability_zone": "us-west-1a",
        "country": "USA",
        "zone_state": "available",
        "id": 1,
        "name": "us-west-1a"
    }
}
```

Fig. 8. Specification for location.

11. USE CASE: CREATE AN AWS NODE

The following are the pre-requisites to be able to create an EC2 node.

11.1. Prerequisite

URLS IN REFRENCES

1. User should have a valid AWS account. ‘

2. Create a IAM [14] user for which the access and secret key will have to be generated [15]. The keys will ahve to added to the yaml cloudmesh configuration file.

3. The required permissions are needed to be granted to the user.

   (a) visit IAM home [16]
   (b) select **policie**s in left hand menu
   (c) create **administrator policy** from amazons existing policies
   (d) select **administrator checkbox** and **attach** to your user

4. Open the ../cloudmesh.yaml configuration file and update with EC2_ACCESS_KEY, EC2_SECRET_KEY, you just copied. Now set default flavor, image and location, as t2.micro, ami-0183d861 and us-west-1 respectively in same file.

5. Mongodb server should be up and running (please refer section 2. Getting started)

```
{
    "aws_node": {
        "state": "RUNNING",
        "provider": "provider",
        "uuid": "e257efd0e6763e9fdc04b79a00f5147fcc21ee7a",
        "name": "test1"
    }
}
```

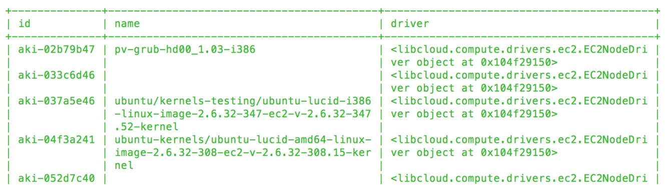
Fig. 9. Specification for node.

```
{
    "aws_volume": {
        "id": "vol-0c30e77f6ca2a263a",
        "size": 1,
        "driver": "Amazon EC2"
       }
}
```

Fig. 10. Specification for volume.

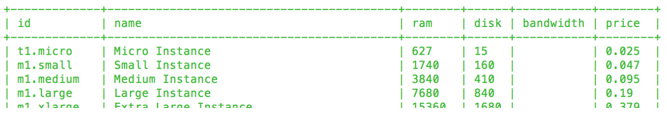
Table 3. Banchmarking of AWS commands

Commands	Mean	Standard Deviation
flavor refresh	0.612	0.059
flavor list	0.207	0.024
keypair refresh	1.967	0.964
keypair list	0.240	0.062
location refresh	1.933	1.246
location list	0.172	0.039
vm refresh	2.342	1.876
vm list	0.160	0.023
volume refresh	2.184	1.878
volume list	0.170	0.020



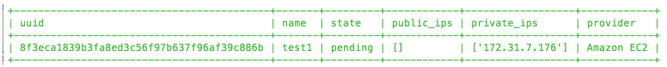
```
+-------------+-----------------------------------------+-----------------------------------------+
| id          | name                                    | driver                                  |
+-------------+-----------------------------------------+-----------------------------------------+
| aki-02b79b47 | pv-grub-hd00_1.03-i386                 | <libcloud.compute.drivers.ec2.EC2NodeDri |
|             |                                         | ver object at 0x104f29150>              |
| aki-033c6d46 |                                        | <libcloud.compute.drivers.ec2.EC2NodeDri |
|             |                                         | ver object at 0x104f29150>              |
| aki-037a5e46 | ubuntu/kernels-testing/ubuntu-lucid-i386 | <libcloud.compute.drivers.ec2.EC2NodeDri |
|             | -linux-image-2.6.32-347-ec2-v-2.6.32-347 | ver object at 0x104f29150>             |
|             | .52-kernel                              |                                         |
| aki-04f3a241 | ubuntu-kernels/ubuntu-lucid-amd64-linux- | <libcloud.compute.drivers.ec2.EC2NodeDri |
|             | image-2.6.32-308-ec2-v-2.6.32-308.15-ker | ver object at 0x104f29150>             |
|             | nel                                     |                                         |
| aki-052d7c40 |                                        | <libcloud.compute.drivers.ec2.EC2NodeDri |
```

Fig. 11. aws image refresh



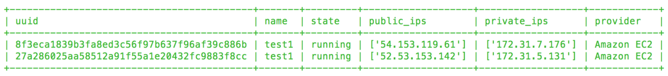
```
+-------------+-----------------------------------------+--------+------+-----------+--------+
| id          | name                                    | ram    | disk | bandwidth | price  |
+-------------+-----------------------------------------+--------+------+-----------+--------+
| t1.micro    | Micro Instance                          | 627    | 15   |           | 0.025  |
| m1.small    | Small Instance                          | 1740   | 160  |           | 0.047  |
| m1.medium   | Medium Instance                         | 3840   | 410  |           | 0.095  |
| m1.large    | Large Instance                          | 7680   | 840  |           | 0.19   |
| m1.xlarge   | Extra Large Instance                    | 15360  | 1680 |           | 0.379  |
```

Fig. 12. aws flavor refresh



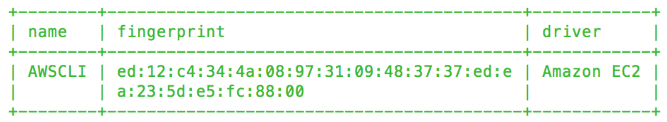
```
+----------------------------------+-------+---------+------------+-----------------+------------+
| uuid                             | name  | state   | public_ips | private_ips     | provider   |
+----------------------------------+-------+---------+------------+-----------------+------------+
| 8f3eca1839b3fa8ed3c56f97b637f96af39c886b | test1 | pending | []  | ['172.31.7.176'] | Amazon EC2 |
+----------------------------------+-------+---------+------------+-----------------+------------+
```

Fig. 13. aws vm boot ami-0183d861



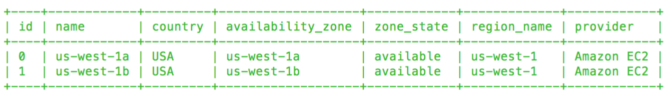
```
+------------------------------------------+-------+---------+-------------------+------------------+------------+
| uuid                                     | name  | state   | public_ips        | private_ips      | provider   |
+------------------------------------------+-------+---------+-------------------+------------------+------------+
| 8f3eca1839b3fa8ed3c56f97b637f96af39c886b | test1 | running | ['54.153.119.61'] | ['172.31.7.176'] | Amazon EC2 |
| 27a286025aa58512a91f55a1e20432fc9883f8cc | test1 | running | ['52.53.153.142'] | ['172.31.5.131'] | Amazon EC2 |
+------------------------------------------+-------+---------+-------------------+------------------+------------+
```

Fig. 14. aws vm list



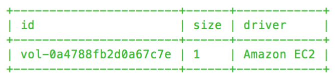
```
+--------+---------------------------------------------+------------+
| name   | fingerprint                                 | driver     |
+--------+---------------------------------------------+------------+
| AWSCLI | ed:12:c4:34:4a:08:97:31:09:48:37:37:ed:e    | Amazon EC2 |
|        | a:23:5d:e5:fc:88:00                         |            |
+--------+---------------------------------------------+------------+
```

Fig. 15. aws keypair create AWSCLI



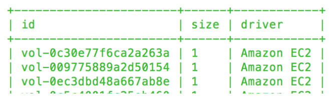
```
+----+------------+---------+-------------------+------------+-------------+------------+
| id | name       | country | availability_zone | zone_state | region_name | provider   |
+----+------------+---------+-------------------+------------+-------------+------------+
| 0  | us-west-1a | USA     | us-west-1a        | available  | us-west-1   | Amazon EC2 |
| 1  | us-west-1b | USA     | us-west-1b        | available  | us-west-1   | Amazon EC2 |
+----+------------+---------+-------------------+------------+-------------+------------+
```

Fig. 16. aws location refresh



```
+-----------------------+------+------------+
| id                    | size | driver     |
+-----------------------+------+------------+
| vol-0a4788fb2d0a67c7e | 1    | Amazon EC2 |
+-----------------------+------+------------+
```

Fig. 17. aws volume create VOL_TEST_1



```
+-----------------------+------+------------+
| id                    | size | driver     |
+-----------------------+------+------------+
| vol-0c30e77f6ca2a263a | 1    | Amazon EC2 |
| vol-009775889a2d50154 | 1    | Amazon EC2 |
| vol-0ec3dbd48a667ab8e | 1    | Amazon EC2 |
```

Fig. 18. aws volume refresh

11.2. Create Node

1. Create keypair name using command

   ```
   cms aws keypair create  AWS1
   ```

   It will create the *keypair name*, it is essential to create the *node*. To verify it, *cms aws keypair list* command will list down all created *keypairs* so far.

2. Now create the node

   ```
   cms aws vm boot ami-0183d861
   ```

   Above command will create a node instance with the image of ami-0183d8661.

REFERENCES

[1] Web Page. [Online]. Available: https://aws.amazon.com/ec2/details/
[2] Github. [Online]. Available: https://github.com/cloudmesh/client
[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 124–131.
[4] G. von Laszewski, "cloudmesh.cmd5," Github. [Online]. Available: https://github.com/cloudmesh/cloudmesh.cmd5
[5] G. von Laszewski, "cloudmesh.common," Github. [Online]. Available: https://github.com/cloudmesh/cloudmesh.common
[6] Web Page. [Online]. Available: https://pypi.python.org/pypi/requests
[7] github. [Online]. Available: https://github.com/cloudmesh/cloudmesh.rest
[8] Web Page. [Online]. Available: http://libcloud.readthedocs.io/en/latest/compute/drivers/ec2.html
[9] "Mongodb," Web Page. [Online]. Available: https://www.mongodb.com/
[10] G. von Laszewski, "cloudmesh.evengine," Github. [Online]. Available: https://github.com/cloudmesh/cloudmesh.eveengine
[11] Web Page. [Online]. Available: http://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/AboutAWSAccounts.html
[12] Web Page. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.htmll
[13] Github. [Online]. Available: https://github.com/cloudmesh/cloudmesh.aws
[14] Amazon, "Create an iam." [Online]. Available: https://console.aws.amazon.com/iam/home?#/users
[15] Web Page. [Online]. Available: http://stackoverflow.com/questions/28222445/aws-cli-client-unauthorizedoperation-even-when-keys-are-set
[16] Amazon, "Iam home." [Online]. Available: https://console.aws.amazon.com/iam/home

AUTHOR BIOGRAPHIES

Milind Suryawanshi received his BE (Electronics and Telecommunication) in 2010 from The University of Pune. His research interests include Big Data analytics for intelligence and research.
Piyush Rai received his BE (Computer) in 2011 from The University of Pune. His research interests also include Big Data analytics for military intelligence and frinancial markets.
Gregor von Laszewski has written more than 100 publications in the are of Grid and Cloud computing. He obtained a Ph.D. from Syracuse University and has worked at Argonne National Laboratory. His team is providing with cloudmesh a client interface to XSEDE SDSC comet virtual clusters.