Cloudmesh client extension for AWS

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 9, 2017

ASW client is an extension to the Cloudmesh client [1] for ineteracting with Amazon Web Services. Cloudmesh client is command line tool to access numerous cloud envirnments and manage the deployment of virtual machines on them. It has it's own command shell [2]. AWS client extends its capability to work with Amazoz EC2 and is based on Libcloud EC2 driver [3].

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

AWS provides a cloud service to deploy virtual machines (VM) with numerous operating systems environment availble in different flavors. The images are called Amazon Machine Image (AMI) and are available as preconfigured for different applications. Once can also create his own custom environment [4].

The Amazon EC2 drivers provides numerous functionalities to authenticate into the aws, list available configurations and create and boot VMs. The AWS Client is based on cloudmesh.cmd5 and cloudmesh.common. It uses mongodb in the backend to store the cloud information such as list of availble images or instances running on the cloud. Requests library [5] is used to connect to the backend through rest services. The rest service is deployed using cloudmesh REST framework [6].

2. GETTING STARTED

One needs to create an aws account first to be able to acces its cloud. The instructions for it are available at [7]. A pair of access key and secret keys are required to be generated to authenticate into the cloud [8]. These keys are required to be kept confidential. These keys are required 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 priviliges for it. The instructions for installation of AWS client can be found at [9]. Once, the client has been installed and configurations settings enabled, the mongodb and the rest servies to access the database can be started by following command:

```
cms admin mongodb start
cms admin rest start
```

The above services will be requried 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. 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 wrppaer around the libcloud EC2 drivers and is also responsible to connect with backend database apart from reading the configuratio from yaml file. The entire code is written in Python.

3.1. Technologies Used

1. Cloudmesh.common and Cloudmesh.cmd5: The common library contains number of modules such as for printing and measuring execution time. The cmd5 is an "dynamically extensible CMD based command shell" [2].

2. Libcloud EC2 driver: The driver provides a number of functions for various functionalities such as from listing the available nodes to generating a key pair and deplying a VM.

3. MongoDb: 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.

4. Cloudmesh.rest: The cloudmesh rest framework is used to deploy the 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 responsible for fetching the information from cloud and passing it to mongodb through rest services. From our expericence 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 alongwith 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 intially 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.

4. AWS COMMANDS

1. 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 onformation will always be fetched from the cloud.

```
cms aws refresh on
```

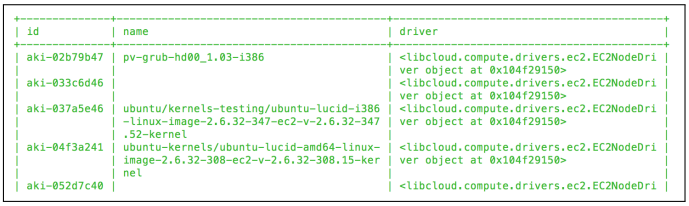
2. Image refresh: The images available on the cloud could be listed using this command. The databse will also be updated with the newly fetched list.

```
cms aws image refresh
```

3. 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.

```
cms aws image list
```

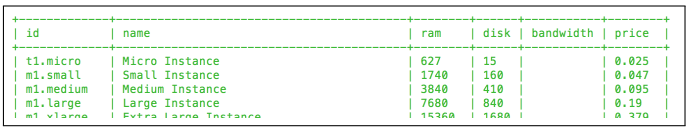
4. Flavor refresh: This will list the different sizes of VMs that are available on the cloud. The information is fetched from the cloud and stored locally.



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 -linux-image-2.6.32-347-ec2-v-2.6.32-347 .52-kernel	<libcloud.compute.drivers.ec2.EC2NodeDri ver object at 0x104f29150>
aki-04f3a241	ubuntu-kernels/ubuntu-lucid-amd64-linux- image-2.6.32-308-ec2-v-2.6.32-308.15-ker nel	<libcloud.compute.drivers.ec2.EC2NodeDri ver object at 0x104f29150>
aki-052d7c40		<libcloud.compute.drivers.ec2.EC2NodeDri

Fig. 1. cms aws image refresh

```
cms aws flavor 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

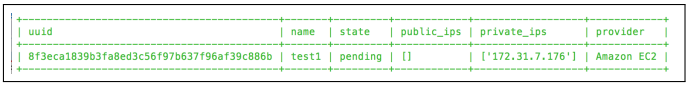
Fig. 2. aws flavor refresh

5. 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'.

```
cms aws flavor list
```

6. Start/Boot vm: 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 being take from cloudmesh.yaml (user need to set those value prior firing the command). You can see the create node instance on console, once it get created.

```
cms aws vm boot IMAGE_ID
```



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

Fig. 3. cms aws vm boot ami-0183d861

7. Reboot vm: If vm is running and reboot command fired, running vm gets reboot. The state of the node gets changed from RUNNING to REBOOTING. The rebooting time of node depends on the configuration we selected, while creating it. *vm reboot* required *NODE_UUID*. Once user fired vm list, all the running node *NODE_UUID* will be shown.

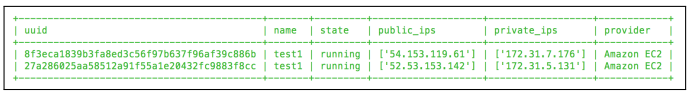
```
cms aws vm reboot NODE_UUID
```

8. Delete vm: Destoyes the instance of a node. Also destory all the associated data with that node, including backup. Node will take time to terminate and remove from the list. Once it fired, very first the state of the node changes to *TERMINATED* and after certain time it will get remove from node list as well.

```
cms aws vm delete UUID
```

9. List created vm : List out all the created nodes present in different states for aws.

```
cms aws vm list
```

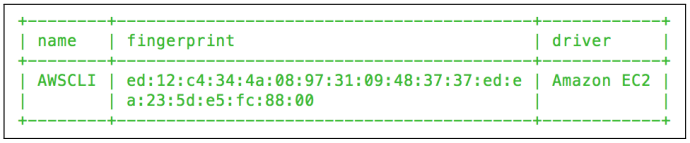


uuid	name	state	public_ips	private_ips	provider
8f3eca1839b3fa8ed3c56f97b637f96af39c886b	test1	running	['54.153.119.61']	['172.31.7.176']	Amazon EC2
27a286025aa58512a91f55a1e20432fc9883f8cc	test1	running	['52.53.153.142']	['172.31.5.131']	Amazon EC2

Fig. 4. cms aws vm list

10. Create keypair: To create the node, one of the essential component is *key pair*. Amazon EC2 uses public key to encrypt and decrypt the password and then recipient uses private key to decrypt it. This public and private keys are know as a *key pair*. User needs to give any (alpha numeric) name to that key.

```
cms aws keypair create NAME
```



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

Fig. 5. cms aws keypair create AWSCLI

11. Delete keypair: Allows user to delete created *key pair*, which is no longer in use.

```
cms aws keypair delete NAME
```

12. List created keypairs: List out all the created *key pairs*.

```
cms aws keypair list
```

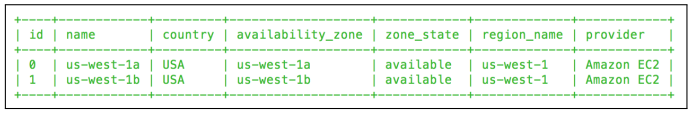
13. Get keypair: Returns the key pair object, which has the name of *key pairs*, driver and hash key.

```
cms aws keypair get NAME
```

14. Get locations: 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.

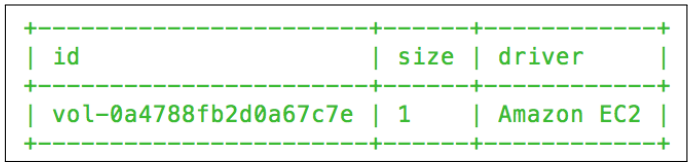
```
cms aws location list
```

15. Create volume: Creates the volume for vm, the size of volume is in GB, the default value is set to 1 GB. The maximum number of volumes that we can attach to vm will be depends on its operating system.



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. 6. cms aws location list



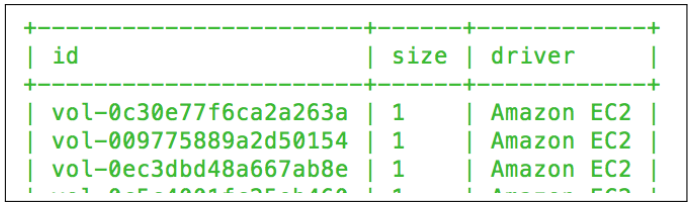
id	size	driver
vol-0a4788fb2d0a67c7e	1	Amazon EC2

Fig. 7. cms aws volume create VOL_TEST_1

```
cms aws volume create VOLUME_NAME
```

16. List created volumes: Command shows the created volumes with the id, size and the driver name.

```
cms aws volume list
```



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

Fig. 8. cms aws volume list

17. Delete volume: User can delete the unwanted volumes using the *VOLUME_ID*.

```
cms aws volume delete VOLUME_ID
```

5. CREATE AWS NODE

To create own node instance, following steps need to be followed.

5.1. Prerequisite

(a) User should have a valid AWS account.

(b) Create a IAM user for which the access and secret key being generated. After creating user, access key and secret key gets generated, copy that keys [**?**].

(c) Once the user has created, now add permission to created user

i. visit IAM home

ii. select **policie**s in left hand menu

iii. create **administrator policy** from amazons existing policies

iv. select **administrator checkbox** and **attach** to your user

(d) 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.

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

5.2. Create Node

i. 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.

ii. Now create the node

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

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

5.3. Create Node

ACKNOWLEDGEMENTS

Prof. Gregor von Laszewski originally suggested this project and provided the objectives in simplistic form. He reviewd the code as it was developed during the course of this project. His inputs helped us to make the code more secure and efficient. He provided us with different resources to look for help and overcome the challenges faced during the course.

REFERENCES

[1] Web Page. [Online]. Available: https://github.com/cloudmesh/client
[2] Web Page. [Online]. Available: https://github.com/cloudmesh/cloudmesh.cmd5
[3] Web Page. [Online]. Available: http://libcloud.readthedocs.io/en/latest/compute/drivers/ec2.html
[4] Web Page. [Online]. Available: https://aws.amazon.com/ec2/details/
[5] Web Page. [Online]. Available: https://pypi.python.org/pypi/requests
[6] Web Page. [Online]. Available: https://github.com/cloudmesh/cloudmesh.rest
[7] Web Page. [Online]. Available: http://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/AboutAWSAccounts.html
[8] Web Page. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.htmll
[9] Web Page. [Online]. Available: https://github.com/cloudmesh/cloudmesh.aws

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.

A. WORK BREAKDOWN

The work on this project was distributed as follows between the authors:

Milind Suryawanshi. Explored the libcloud EC2 drivers, investigated the various arguments requried by their different routines and implemented their use accordingly.

Piyush Rai. Worked on configurable parameters and database connectivity using pymongo and rest services.