Cloudmesh Docker Extension

KARTHICK VENKATESAN[1] AND ASHOK VUPPADA[2]

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

S17-IR-P009, April 26, 2017

Cloudmesh client is a simple client to enable access to multiple cloud environments form a command shell and command line. The user's can manage their set of resources right from their workstation.Currently cloudmesh client supports managing Virtual Machines across multiple clouds. In this project we have added the capability to manage/provision docker and swarm containers to cloudmesh client through a simple and extensible command line interface.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P009/report/report.pdf

1. INTRODUCTION

In the time of Big data and Micro-Services,it is common to have a set of services running on multiple clouds.Docker[1] with the build and ship model made the micro-services architecture work better.Since there would multiple containers running across multiple remote clouds managing them can become tedious.In project we have added the capability to provision and manage docker[1] and swarm[2] containers to cloudmesh client[3].

Clouldmesh Client[3] capability is detailed in Figure 1,it aims at managing vm instances in multiple heterogeneous clouds remotely via a command line interface.

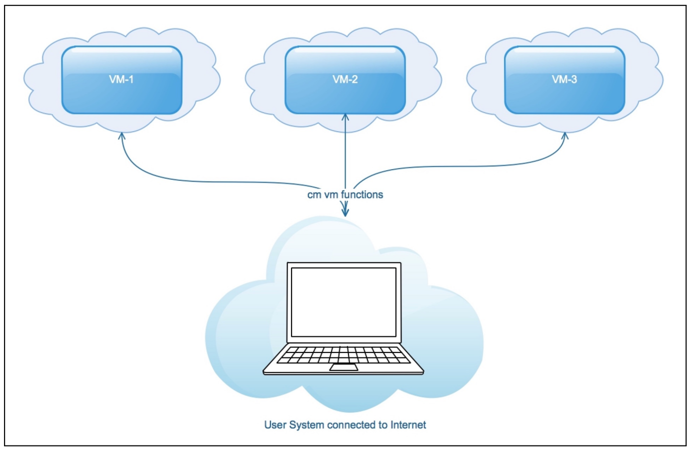

Fig. 1. Cloudmesh client

The cloudmesh docker application build can be broadly divided into Docker and Swarm modules.

The Docker Module capabilities are detailed in Figure 2. This module would help user perform various docker functions mentioned in Section 3

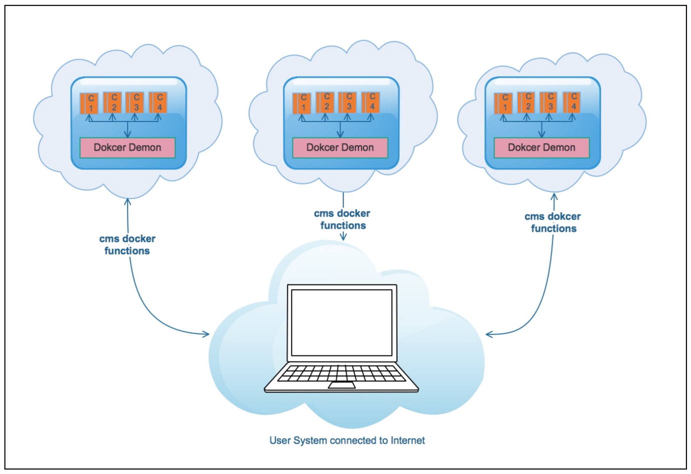

Fig. 2. Docker Mode

The Swarm module facilitates performing various swarm operations Figure 3.

Users can choose to use cloudmesh docker application from a remote terminal outside the network of the data center as in Figure2 or locally from a provisioning or configuration server inside the data center as in Figure 4

2. APPLICATION ARCHITECTURE

The architecture of the application is depicted in Figure 5.The commands developed can be broadly classified as 'action com-

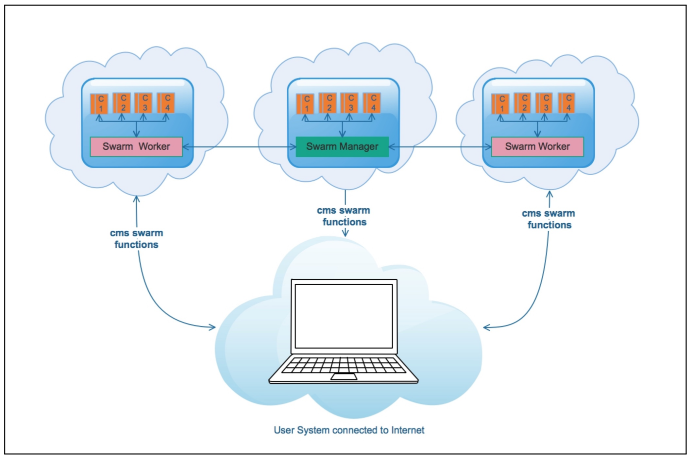

Fig. 3. Swarm Mode

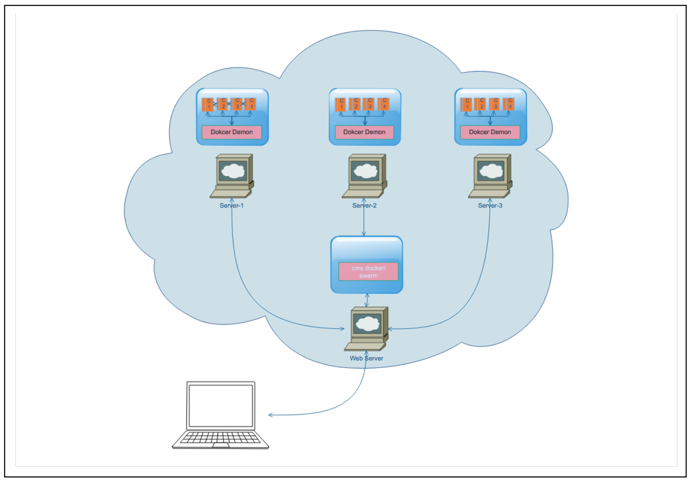

Fig. 4. Docker/Swarm Remote

mands','Inquiry commands'.
The action commands are which would create or alter an entity.The entity can be a host/container/node. we use the corresponding API call to get the latest values for the changed entity.Docker API module is developed for addressing the docker commands and Swarm API module is used for swarm commands.
The Inquiry commands have two flavours a list and refresh mode.The list commands fetch the data locally from the Database and the refresh command will refresh the current state of the corresponding entity from the hosts.

2.1. Technologies Used

Name	Purpose
docker [1]	Docker Server and Api for managing containers
mongodb [4]	Nosql DBMS
Python-eve [5]	Restful webservices interface to mongoDB
python [6]	development
ansible [7]	automated deployment

Table 1. Technology Name and Purpose

2.2. MongoDB and Python-eve

The cloudmesh docker application uses MongoDB[4] for data storage.The access to the database is all through restful services supported through Python-eve[5].The following are the entities for which collections are defined in eve and mongoDB.

1. Host
2. Image
3. Container
4. Network
5. Service
6. Node

A key benefit of using a NoSQL data base like mongoDB is that it allowed us to store the data in the the DB in the native form as returned by the Docker Api without the need for much marshaling of the data .The application uses the cloudmesh.rest repository for managing mongo and eve services.

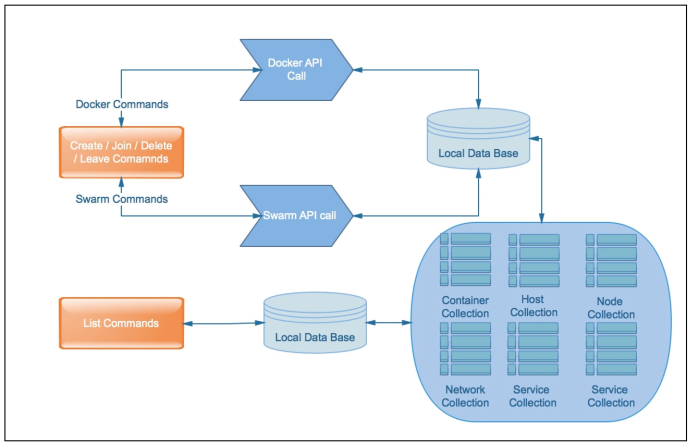

Fig. 5. cms docker extension application architecture

2.3. Cloudmesh Common

The application makes extensive use of common functions and tightly integrated into common functions available in the cloudmesh common repository for display formatting , YAML config management and timers

2.4. Ansible

As part of the project we have built Ansible[7] scripts to automate installation of docker in remote hosts and also deployment of docker images in these remote docker hosts.Below is the list of Ansible scripts that are built and used in the project

1. Install Docker in remote hosts and enable them for remote API access
2. Install Docker images in docker hosts .As part of the script the local docker files are synced with remote hosts and the images are built .
3. Setup /etc/hosts for remote hosts.This script allows to setup host names and IP in remote hosts which allows the applications to be configured to access by the standard host names instead of the IP address of the machine.

3. DOCKER COMMANDS

1. **Host Set/Add** This command is to be used to setup the docker host on which the user wants to operate .The docker commands following would be executed on the host setup in this step. The host details will be also captured in the database with this command.

   ```
   cms docker host docker1 docker1:4243
   ```

2. **Host List** This command would list the hosts available.This command would display the Ip,Name,Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarmmanager Ip would be blank if the host is manager or not part of swarm.

   ```
   cms docker host list
   ```

Table 2. cms docker host list

Ip	Name	port	Swarmmode	SwarmManagerIp
docker1	docker1	4243	Manager	
docker2	docker2	4243	Worker	docker1
docker3	docker1	4243	Host	

3. **Host Delete** This command would be used to delete a host from the setup.This would also delete the host details from the database.User can do a host list to see the updated host details.

   ```
   cms docker host delete docker1:4243
   ```

4. **Image List** This command would be used to display the images available on a host. It would display the Ip of the host,the Image id , repository and the size of the image.Please note that this command would display the results from the local dB.

   ```
   cms docker image list
   ```

5. **Image Refresh** This command would refresh the images across the hosts available the results would update the local data base , and display the updated reults to the user.

Table 3. cms docker image list

Ip	Id	Repository	Size(GB)
docker1	5545f4e3b27e	cloudmesh:docker	5.59
docker2	45f4e3b2799e	elasticsearch:swarm	0.45

   ```
   cms docker image refresh
   ```

6. **Container Create** This command would used to create a container on a given host.The arguments for this command would be the name of the container and the image from which the container needs to be created.The image in the argument must be available through image list command above on the given host.

   ```
   cms docker container create test1 \
   elasticsearch:docker
   ```

7. **Container Start** This command would be used to start a container.The container should be already created using the above command.

   ```
   cms docker container start test1
   ```

8. **Container Stop** This command would be used to stop and container which is running.The container can be started using the above command

   ```
   cms docker container stop test1
   ```

9. **Container List** This command would display the list of containers running across the hosts.This would return the Ip,Container Id , Name , Image , status and start time of the container.The details would be shown from the local database maintained.

   ```
   cms docker container list
   ```

Table 4. cms docker container list

Ip	Id	Name	Image	Status	StartedAt
docker1	5545f4e3b27e	test1	image1	exited	12.00PM

10. **Container Refresh** This command would refresh the current state of the containers across the hosts, This command would connect to the host and run the native docker container list to get the latest information and update the local database for refreshing the data.

   ```
   cms docker container refresh
   ```

11. **Container Delete** This command would be used to delete a required container on the host. The arguments required are the container name.The updated container list can be viewed by running cms docker container list command.

```
cms docker container delete test1
```

12. **Container Run** This command would be used to run a container instead of creating and starting in two steps.This arguments for the function are the name of the container and the image from which it needs to be run.

```
cms docker container run test1 /
elasticsearch:docker
```

13. **Container Pause** This command would pause the container which is currently running.Use can run a cms docker container list to observe the status change.

```
cms docker container pause test1
```

14. **Container Unpause** This command would unpause the container which is currently paused.Use can run a cms docker container list to observe the status change.

```
cms docker container unpause test1
```

15. **Network Refresh** This command would refresh the network across the docker containers and hosts the updated results would be stored in the local database.

```
cms docker network refresh
```

16. **Network List** This command would display the results of the refreshed network.This would display the host ip where the network established , the network id , name and containers in the network

```
cms docker network list
```

Table 5. **cms docker network list**

Ip	Id	Name	Containers
docker1	5545f4e3b27e	network1	test1

4. SWARM COMMANDS

1. **Host Set/Add** The command would be used to setup the current host.The docker commands following would be executed on the host setup in this step. the host details will be also captured in the database with this command.

```
cms swarm host docker1 docker1:4243
```

2. **Host List** This command would list the hosts available,This command would display the Ip,Name,Port,and if the host is swarm manager and the swarm manager Ip. Please note the Swarmmanager Ip would be blank if the host is manager or not part of swarm.

```
cms swarm host list
```

Table 6. **cms docker host list**

Ip	Name	port	Swarmmode	SwarmManagerIp
docker1	docker1	4243	Manager	
docker2	docker2	4243	Worker	docker1
docker3	docker1	4243	Host	

3. **Host Delete** This command would be used to delete a host from the setup.This would also delete the host details from the database.User can do a host list to see the updated host details.

```
cms swarm host delete docker1:4243
```

4. **Image List** This command would be used to display the images available on a host. It would display the Ip of the host,the Image id , repository and the size of the image.Please note that this command would display the results from the local dB.

```
cms swarm Image list
```

5. **Swarm Create** This would create swarm on the host in use.There is no arguments required for this command, After this command is run the current host status would be treated as 'manager'.user can run a node list or host list to see the updated result.To setup the current host user needs to use cms swarm host ADDR command shown above.

```
cms swarm create
```

6. **Swarm Join** This command would be applicable for the host which is not manager,user needs to setup a new current host with cms swarm host command and run cms swarm join so that current host would be joined with the swarm created in the last step.User needs to pass the swarm host Ip and address the host being joined

```
cms swarm join docker3 docker4:4243 worker
```

(assuming docker3 is already a swarm manager)

7. **Swarm Leave** This command is applicable for the swarm manager or worker , this would let the host leave swarm.If manager has multiple workers ,workers needs to be removed(leave) before manager can leave.This command would treat current host is to be removed(leave) the swarm.User may need to set up the current host before processing the command.

```
cms swarm leave
```

8. **Network Create** This command would be used to create the network which can be used by the swarm containers later.The arguments it would need is the name of the containers.

```
cms swarm network create network1
```

9. **Network List** This command would display the results of the refreshed network.This would display the host ip where the network established , the network id , name and containers in the network

```
cms swarm network list
```

Table 7. cms swarm network list

Ip	Id	Name	Containers
docker1	5545f4e3b27e	network1	test1

10. **Network Refresh** This command would refresh the network across the docker containers and hosts the updated results would be stored in the local database.

```
cms swarm network refresh
```

11. **Network Delete** This command would be used to delete an existing network.The inputs required for this command is just the network name.

```
cms swarm network delete network1
```

12. **Service Create** This command would be used to create a service,the arguments required are the image name and the name of service.This command would record the service details into the local database.

```
cms swarm service create elasticsearch \
elasticsearch:swarm
```

13. **Service List** This command would list the current services running ,the data being displayed would be from the local data base, if the most current details are required user can run service refresh command below.

```
cms swarm service list
```

The number of replicas below indicates the number of containers which are running the services.

Table 8. cms swarm service list

Ip	Id	Name	Image	Replicas
docker1	5545f4e3b27e	elasticsearch	elastic:swarm	3

14. **Service Delete** This command would be used to delete a running service,the arguments required are the service name.This command would delete the service details into the local database.

```
cms swarm service delete elasticsearch
```

15. **Service Refresh** This command would be used to refresh the services status based on the current condition.This command would refresh the local database so that service list would show the updated results.

```
cms swarm service refresh
```

16. **Node List** This command would display the list of the nodes across the hosts available.The results would come from the local database. The command would display the node id,,Id,Role,status and Manager Ip.

```
cms swarm node list
```

Table 9. cms swarm node list

Id	Ip	Role	Status	Manager Ip
5545f4e3b27e	docker3	Manager	Ready	
7645f4f4b27e	docker2	Worker	Ready	docker4

17. **Image Refresh** This command would refresh the images across the hosts available the results would update the local data base, user can run docker image list to view the updated results.

```
cms swarm image refresh
```

18. **Image List** This command would be used to display the images available on a host. It would display the Ip of the host,the Image id , repository and the size of the image.Please note that this command would display the results from the local dB.

```
cms swarm image list
```

19. **Container Refresh** This command would refresh the current state of the containers across the hosts, This command would connect to the host and run the native docker container list to get the latest information and update the local database for refreshing the data.

Table 10. cms swarm image list

Ip	Id	Repository	Size(GB)
docker1	5545f4e3b27e	cloudmesh:docker	5.59
docker2	45f4e3b2799e	elasticsearch:swarm	0.45

```
cms swarm container refresh
```

20. **Container List** This command would display the list of containers running across the hosts.This would return the Ip,Container Id , Name , Image , status and start time of the container.The details would be shown from the local database maintained.

```
cms swarm container list
```

Table 11. container list

Ip	Id	Name	Image	Status	StartedAt
docker1	5545f4e3b27e	test1	image1	exited	12.00PM

5. USE CASE - ELASTICSEARCH CLUSTER

Elasticsearch[8] is an open-source, broadly-distributable, readily-scalable, enterprise-grade search engine. Accessible through an extensive and elaborate API, Elasticsearch can power extremely fast searches that support your data discovery applications [?] [2]

Using Cloudmesh client , Ansible and Cloudmesh Docker application we deployed and provisioned a Elasticsearch cluster on remote hosts in Chameleon cloud in docker and swarm mode .We benchmarked the cluster using esrally[9] have compared the results between the elastic search clusters in docker and swarm mode.

5.1. Elasticsearch Cluster Docker Mode

For provisioning the Elasticsearch cluster in docker hosts below are the steps done

1. Created 3 Virtual Machines using Cloud Mesh Client .2 of the Virtual Machines are to be used for the docker Elasticsearch cluster and 1 Virtual machine is the Benchmark server for the Kibana and esrally docker images

2. Using Ansible scripts Install docker in 3 Virtual Machines and enable the docker daemon for remote access.

3. Using Ansible scripts Install Images of Elasticsearch on hosts for docker cluster and the Image of Esrally in the Benchmark server .

4. Using the Cloudmesh Docker application we start 4 containers 2 in each of the virtual machines .To enable clustering of Elasticsearch applications running in the docker containers we need set the below parameters in container creation

```
network_mode=host
environment=
["http.host=0.0.0.0",
"transport.host=0.0.0.0",
"discovery.zen.ping.unicast.hosts=docker1,docker2"]
```

The network mode set to host allows the Elasticsearch containers use the underlying Virtual Machines network for networking and leveraging the Elasticsearch unicast discovery find and form a cluster along with the other Elasticsearch instances running in other containers either on the same host or different hosts.

5.2. Elasticsearch Cluster Swarm Mode

For provisioning the Elasticsearch cluster in docker hosts in swarm mode below are the steps done

1. Created 3 Virtual Machines using Cloud Mesh Client .2 of the Virtual Machines are to be used for the docker Elasticsearch cluster and 1 Virtual machine is the Benchmark server for the Kibana and esrally docker images

2. Using Ansible scripts Install docker in 3 Virtual Machines and enable the docker daemon for remote access.

3. Using Ansible scripts Install Images of Elasticsearch on hosts for docker cluster and the Images of Kibana and Esrally in the Benchmark server .

4. Using the Cloudmesh Docker application we first create a swarm cluster with the two docker hosts.Then we create a service in the Swarm Manager Node.Along with the creation of the service we pass parameters to specify the number of replicas ,the network to be used , the mode of replication and the service name.

```
ServiceMode.mode="replicated"
ServiceMode.replicas=4
EndpointSpec.ports=["9200:9200"]
networks=["elastic_cluster"]
env=["SERVICE_NAME=elasticsearch"]
```

Swarm mode containers cannot use the the underlying host network as in the docker mode to enable the communication between the swarm containers we created a "overlay" network in the swarm manager.This network is passed in the service creation.So every container that is created by the swarm mode Manager will run on this network .In the swarm mode to enable elastic search unicast discovery on start of the elastic search cluster using the Servicename environmental variable we identify other containers available in the cluster and dynamical set the

```
discovery.zen.ping.unicast.hosts
```

parameter to enable elastic search to find and form a cluster with other Elasticsearch applications in the swarm.

5.3. Elasticsearch cluster Docker and Swarm mode benchmark results

6. BENCHMARKING CLOUDMESH DOCKER

We performed benchmarking of the cloudmesh docker application for docker and swarm commands .The benchmark was performed both in remote mode (Cloudmesh docker client is

Table 12. Elastic search Benchmark Results Docker Vs Swarm

Operation	Unit	Docker	Swarm
Flush time	min	0.9709	1.34333
Indexing time	min	117.888	136.951
Merge throttle time	min	75.5648	87.8035
Merge time	min	146.693	179.403
Refresh time	min	27.4014	32.6458
articles_monthly_agg_cached	ops/s	20.0178	20.0175
articles_monthly_agg_uncached	ops/s	20.0085	20.0093
default	ops/s	20.0133	20.007
force-merge	ops/s	1.75528	0.943048
index-append	docs/s	535.527	461.233
index-stats	ops/s	49.8993	50.2674
node-stats	ops/s	49.6913	50.2767
phrase	ops/s	20.0127	20.0129
scroll	ops/s	1.31822	0.457152
term	ops/s	20.0126	20.011

run on a network outside the cloud data center) and local mode (Cloudmesh docker client is run from a VM inside the cloud data center) . We performed the benchmarking for both the options on both the Amazon Webservices[10] and Chameleon cloud[11]. The results are plotted and tabulated as below

Each of the benchmark runs was performed 100 times for a defined set of operations similar to the steps performed for setting up a elastic search cluster in docker and swarm.The results were gathered as a csv file and plotted using a Ipython[12].

The hardware specifications used on both the clouds is listed below

Table 13. Hardware Specification

Parameter	Chameleon	Aws
VM	3	3
OS	Ubuntu 16.04	Ubuntu 16.04
Flavor	m1.large	m1.large
Secgroup	default	default
Assign floating IP	True	True

6.1. Docker Mode - Results

Below are the categories of the bench mark results

1. Chameleon Docker Mode Local Client Figure 6
2. Chameleon Docker Mode Remote Client Figure 7
3. Aws Docker Mode Local Client Figure 8
4. Aws Docker Mode Remote Client Figure 9

Based on the benchmark reults we can infer the below details

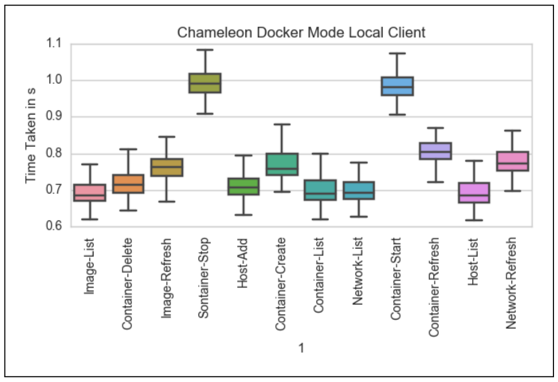

Fig. 6. Chameleon Docker Mode Local Client

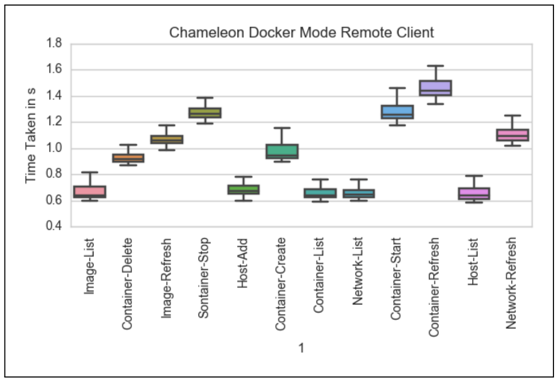

Fig. 7. Chameleon Docker Mode Remote Client

1. In the battle of the clouds Aws is around 20 percent faster than Chameleon cloud in docker mode
2. Cloudmesh docker operations for the docker command performed in a local network are between 25 and 30 percent faster.We also noticed some network issues when performing the test from a remote network however we chose to ignore those outliers in the plot.
3. The standard deviation of the response times is significantly lower for Aws than Chameleon indicating that Aws is much more stable and reliabe in performance than the chameleon cloud
4. The mean container create times range between 0.5 to 1 s which is significantly faster than normal VM boot times on the cloud .

6.2. Swarm Mode - Results

Below are the categories of the Benchmark results

1. Chameleon Swarm Mode Local Client Figure 11
2. Chameleon Swarm Mode Remote Client Figure 10
3. Aws Swarm Mode Local Client Figure 12
4. Aws Swarm Mode Remote Client Figure 13

Based on the benchmark reults we can infer the below details

1. In the battle of the clouds Aws is around 20 percent faster than Chameleon cloud in swarm mode

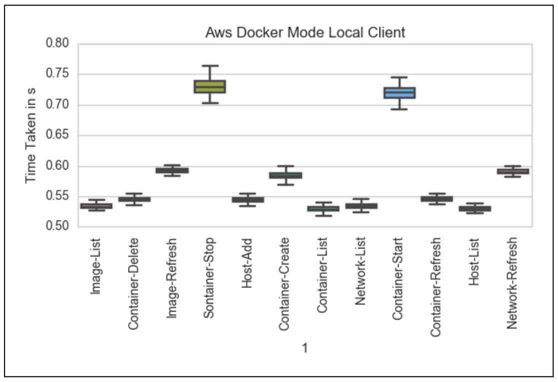

Fig. 8. Aws Docker Mode Local Client

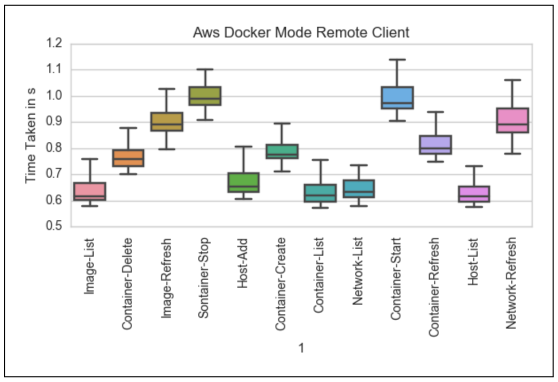

Fig. 9. Aws Docker Mode Remote Client

2. Cloudmesh swarm operations for the swarm command performed in a local network are between 25 and 30 percent faster.

3. The standard deviation of the response times is significantly lower for Aws than Chameleon indicating that Aws is much more stable and reliabe in performance than the chameleon cloud

4. The mean service create times range between 2 to 1.6 s for 4 replicated containers which is significantly faster than normal boot times for a similar number of VM on the same cloud .

Table 14. Docker Mode AWS VS Chameleon Local Vs Remote

		Aws		Chameleon	
		Local	**Remote**	**Local**	**Remote**
Image-List	mean	0.534	0.661	0.695	0.704
Image-List	std	0.004	0.128	0.039	0.197
Container-Delete	mean	0.545	0.785	0.721	0.951
Container-Delete	std	0.004	0.090	0.040	0.115
Image-Refresh	mean	0.592	0.925	0.763	1.139
Image-Refresh	std	0.005	0.109	0.040	0.298
Sontainer-Stop	mean	0.730	1.017	0.992	1.299
Sontainer-Stop	std	0.014	0.122	0.041	0.113
Host-Add	mean	0.544	0.691	0.710	0.727
Host-Add	std	0.004	0.114	0.038	0.194
Container-Create	mean	0.584	0.798	0.767	1.007
Container-Create	std	0.006	0.075	0.042	0.164
Container-List	mean	0.529	0.655	0.697	0.689
Container-List	std	0.004	0.110	0.038	0.141
Network-List	mean	0.534	0.668	0.700	0.679
Network-List	std	0.005	0.098	0.035	0.115
Container-Start	mean	0.720	1.018	0.985	1.310
Container-Start	std	0.011	0.145	0.044	0.169
Container-Refresh	mean	0.546	0.824	0.805	1.509
Container-Refresh	std	0.004	0.070	0.033	0.208
Host-List	mean	0.530	0.659	0.693	0.708
Host-List	std	0.004	0.125	0.042	0.273
Network-Refresh	mean	0.591	0.946	0.780	1.137
Network-Refresh	std	0.005	0.171	0.043	0.151

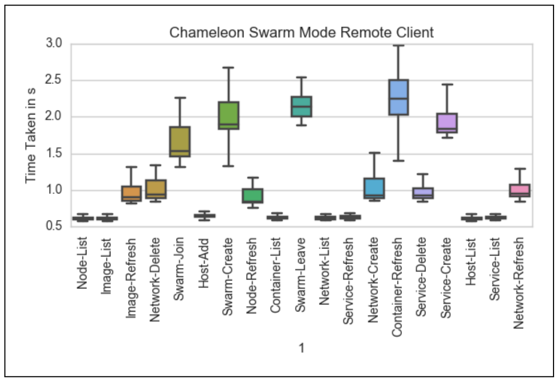

Fig. 10. Chameleon Swarm Mode Remote Client

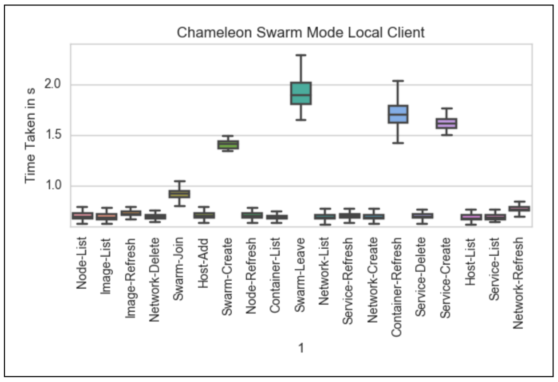

Fig. 11. Chameleon Swarm Mode Local Client

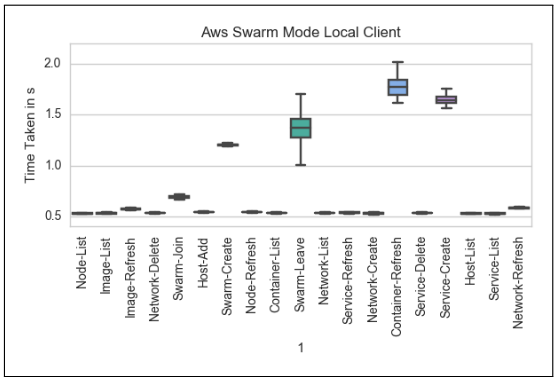

Fig. 12. Aws Swarm Mode Local Client

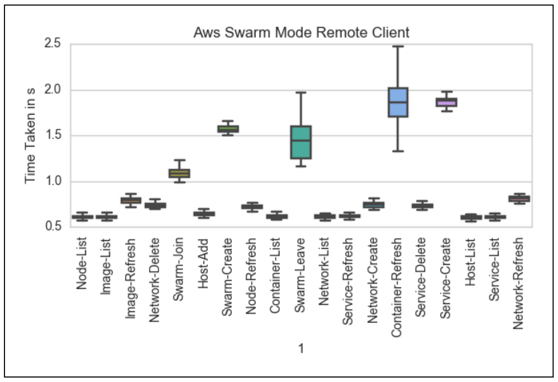

Fig. 13. Aws Swarm Mode Remote Client

Table 15. Swarm Mode AWS VS Chameleon Local Vs Remote

		Aws		Chameleon	
		Local	**Remote**	**Local**	**Remote**
Node-List	mean	0.529	0.612	0.704	0.631
Node-List	std	0.004	0.022	0.037	0.056
Image-List	mean	0.532	0.614	0.696	0.626
Image-List	std	0.004	0.041	0.039	0.053
Image-Refresh	mean	0.571	0.818	0.732	0.981
Image-Refresh	std	0.006	0.139	0.035	0.220
Network-Delete	mean	0.536	0.822	0.701	1.024
Network-Delete	std	0.005	0.280	0.035	0.247
Swarm-Join	mean	0.690	1.178	0.925	1.666
Swarm-Join	std	0.015	0.345	0.053	0.270
Host-Add	mean	0.542	0.653	0.714	0.665
Host-Add	std	0.005	0.062	0.035	0.076
Swarm-Create	mean	1.201	1.640	1.320	1.963
Swarm-Create	std	0.046	0.342	0.213	0.340
Node-Refresh	mean	0.542	0.738	0.714	0.911
Node-Refresh	std	0.004	0.107	0.039	0.127
Container-List	mean	0.533	0.622	0.696	0.638
Container-List	std	0.004	0.039	0.033	0.050
Swarm-Leave	mean	1.419	1.460	1.932	2.143
Swarm-Leave	std	0.275	0.288	0.241	0.186
Network-List	mean	0.532	0.625	0.698	0.625
Network-List	std	0.004	0.050	0.035	0.027
Service-Refresh	mean	0.536	0.631	0.710	0.645
Service-Refresh	std	0.004	0.089	0.039	0.099
Network-Create	mean	0.531	0.781	0.698	1.009
Network-Create	std	0.005	0.156	0.035	0.162
Container-Refresh	mean	1.666	1.846	1.693	2.273
Container-Refresh	std	0.374	0.348	0.181	0.386
Service-Delete	mean	0.535	0.781	0.704	0.991
Service-Delete	std	0.004	0.205	0.039	0.140
Service-Create	mean	1.661	1.905	1.636	1.938
Service-Create	std	0.071	0.164	0.115	0.273
Host-List	mean	0.529	0.608	0.693	0.629
Host-List	std	0.004	0.030	0.033	0.083
Service-List	mean	0.528	0.617	0.698	0.639
Service-List	std	0.005	0.052	0.035	0.075
Network-Refresh	mean	0.583	0.834	0.776	1.009
Network-Refresh	std	0.005	0.145	0.039	0.126

7. CONCLUSION

In this project we have succefully integrated docker and swarm capabilities into cloudmesh client.We have also demonstrated its use for a practical use case of setting up a Elastic search cluster in docker and swarm modes.We have also benchmarked the commands for multiple clouds(AWS and Chameleon) in both local and remote modes and detailed the results and insights.The ansible scripts as part of the project along with the capabilities built in the cloudmesh docker application provide for a seamless capability in deploying and provisioning applications in docker and swarm containers.

8. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

9. APPENDICES

Appendix A: Work Distribution The co-authors of this report worked together on the design of the technical solutions, implementation, testing and documentation. Below given is the work distribution

- Karthick Venkatesan
  - Design and Implementation of Docker and Swarm Commands .
  - Integration of Docker and Swarm Commands to Docker API.
  - Integration to cloudmesh.common,cloudmesh.rest ,cloudmesh.cmd5 repositories.
  - Framework definition and wrapper class built for Python-Eve
  - Ansible scripts for docker image installation and setup of etc hosts
  - Test scripts for Docker and Swarm command
  - Dockerfile for installation of cloudmesh.docker
  - Create Benchmark scripts for Local and Remote Benchmarking on Chameleon and AWS
  - Execute Benchmark scripts for Chameleon and Aws and plot the results in Ipython
  - Scripts for setup of Elasticsearch docker cluster
  - Benchmark Elastic search swarm cluster using ESRally and dcoument reults
  - Writing related sections in this report.
- Ashok Vuppuda
  - Design of Docker and Swarm Commands .
  - Integration into cloudmesh.rest
  - Python EVE integration and implementation for docker and Swarm Modes
  - Ansible scripts for docker installation
  - Test application on Aws and Chameleon clouds
  - Execute Benchmark scripts for Chameleon and Aws and plot the results in Ipython
  - Benchmark Elastic search docker cluster using ESRally and dcoument reults
  - Writing related sections in this report.

REFERENCES

[1] Docker Inc., “Docker,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.docker.com/

[2] Docker Inc., “Swarm,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://docs.docker.com/engine/swarm/

[3] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, “Accessing Multiple Clouds with Cloudmesh,” in *Proceedings of the 2014 ACM International Workshop on Software-defined Ecosystems*, ser. BigSystem ’14. New York, NY, USA: ACM, 2014, pp. 21–28. [Online]. Available: http://doi.acm.org/10.1145/2609441.2609638

[4] MongoDB, Inc., “MongoDB,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.mongodb.com/

[5] N. Iarocci, “Python Eve,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: http://python-eve.org/

[6] Python Software Foundation, “Python,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.python.org/

[7] Red Hat, Inc., “Ansible,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.ansible.com/it-automation

[8] Elasticsearch BV, “ElasticSearch,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.elastic.co/

[9] D. Mitterdorfer, “ESRally,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: http://esrally.readthedocs.io/en/latest/quickstart.html

[10] Amazon Web Services, Inc., “Amazon Webservices,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://aws.amazon.com/?nc2=h_lg

[11] National Science Foundation, “Chameleon,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://www.chameleoncloud.org/

[12] IPython development team, “Ipython,” Web Page, 2017, accessed 2017-04-23. [Online]. Available: https://ipython.org/