Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm

Nitin Naik
School of Informatics and Digital Engineering, Aston University, United Kingdom
Email: n.naik1@aston.ac.uk

Abstract—**The design of distributed systems in multiple clouds have been gaining popularity due to various benefits of the multi-cloud infrastructure such as minimizing vendor lock-in, data loss and downtime. Nonetheless, this multi-cloud infrastructure also poses several challenges such as compatibility, interoperability, complex provisioning and configuration due to the variation in technologies and services of each cloud provider. Consequently, it is a tedious task to design distributed systems in multiple clouds. Virtualization is regarded as the base technology of the cloud and therefore, most cloud-based distributed systems are based on it. Nevertheless, virtual machines require substantial resources and cause several issues across multiple clouds such as provisioning, configuration management, load balancing and migration. Docker Swarm is a container-based clustering tool that resolves some of these issues and supports the design of distributed systems in multiple clouds. It has also incorporated several inbuilt attributes of the distributed system, however, it is still evolving. This paper initially presents the simulated development of a Docker Swarm-based distributed system which can be easily replicated in multiple clouds. Subsequently, based on the simulated Docker Swarm-based distributed system, it performs an evaluation of several attributes of this distributed system such as high availability and fault tolerance; automatic scalability, load balancing and maintainability of services; and scalability of large clusters.**

Keywords—Distributed System; Docker Swarm; Multiple Clouds; Container; Containerization; Virtual Machine; Virtualization; High Availability; Fault Tolerance; Scalability; Maintainability; Load Balancing.

I. Introduction

A distributed system is a collection of interconnected independent systems that enables systems to coordinate their activities and share their resources for acting as one large virtual system [1], [2]. The designing of distributed systems has many challenges such as successful handling of failure of machines, disks, networks, and software. Distributed systems can be made more effective if they are designed in multiple clouds by leveraging several benefits of the multi-cloud infrastructure such as minimizing vendor lock-in, data loss and downtime [3], [4]. Nonetheless, this multi-cloud infrastructure also poses several challenges such as compatibility, interoperability, complex provisioning and configuration due to the variation in technologies and services of each cloud provider [5]. Inevitably, it increases the complexity of design process of distributed systems and operations across multiple clouds [4]. Virtualization is regarded as the base technology of the cloud and therefore, most cloud-based distributed systems are based on it. Nevertheless, virtual machines demand substantial resources and cause several issues across multiple clouds such as provisioning, configuration management, load balancing and migration [6].

Docker is an open-source platform which makes the development of software applications easy using containers [7]. It bundles a software application with all its required dependencies into a container for making a complete development pipeline more efficient. This makes the Docker container a preferred choice for software developers. Docker container is an OS-level virtualization and it requires fewer resources than a virtual machine, thus, it resolves the speed and performance issues of virtualization for software developers [3]. Docker Swarm is a container-based clustering tool that supports the design of multi-cloud distributed systems in those clouds which are supported by Docker [4]. It has also incorporated several inbuilt attributes of the distributed system, however, it is still evolving.

This paper illustrates the design of a distributed system in multiple clouds using a Docker Swarm cluster. Subsequently, it presents the simulated development of a Docker Swarm-based distributed system in VirtualBox, which can be easily replicated in multiple clouds. Finally, based on the simulated Docker Swarm-based distributed system, it performs an evaluation of several attributes of the Docker Swarm-based distributed system such as high availability and fault tolerance; automatic scalability, load balancing and maintainability of services; and scalability of large clusters. This evaluation demonstrates that the Docker Swarm-based distributed system is relatively easy to design and act as a natural distributed systems due to several inbuilt attributes of distributed systems. Additionally, Docker supports many virtual and cloud environments, therefore, this Docker Swarm-based distributed system can also be replicated into multiple clouds.

The remainder of this paper is organised as follows: Section II explains about containerization, Docker container and Docker Swarm; Section III illustrates the design of a distributed system in multiple clouds using Docker Swarm; Section IV presents the simulated development of a Docker Swarm-based distributed system which can be easily replicated in multiple clouds. Section V performs an evaluation of several attributes of the Docker Swarm-based distributed system; Section VI concludes the paper and suggests some future areas of extension.

II. Containerization, Docker Container and Docker Swarm

A. *Containerization*

Containerization or container-based virtualization is moderately different technique from virtualization, where an isolated environment (container) is created similar to a virtual machine

2021 IEEE International Systems Conference (SysCon) | 978-1-6654-4439-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/SysCon48628.2021.9447123

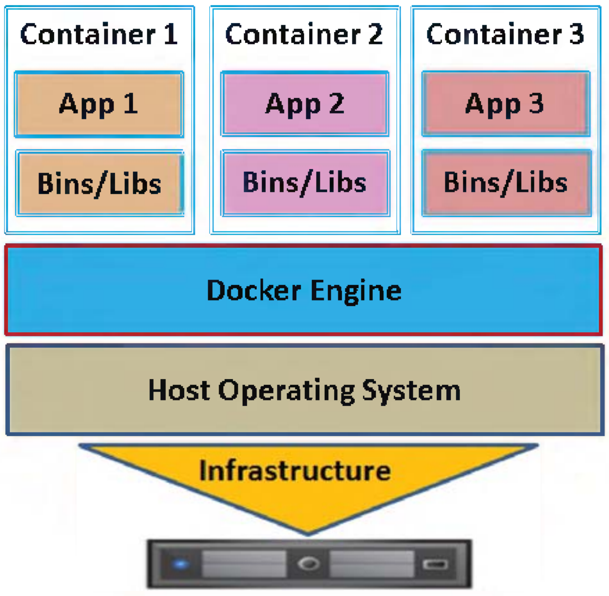


Fig. 1. Docker container architecture on a Linux machine

but without virtual hardware emulation [3]. Container is a very old technique in Unix and Linux but now it is reintroduced at commercial level due to its benefits as compared to a VM. Containerization can be considered as an OS-level virtualization because containers are created in the user space above the OS kernel [8]. Multiple containers can be created in multiple user spaces on a single host but with very fewer resources than VMs [8].

B. Docker Container

Docker container is an instance of containerization. Docker is a container-based technology for an easy and automated creation, deployment and execution of applications by employing containers [9]. It facilitates an isolated environment (container) similar to a VM but without having its own OS, therefore all containers share the same OS kernel via Docker Engine as shown in Figs. 1 and 2. However, a container consists of all the binary and library files required to run an application. If Docker container is used on Linux then Linux OS acts as a default Docker Host (see Fig. 1), but when it is used on non-Linux machine then this Docker Host needs to be installed separately (see Fig. 2). This Docker Host is a lightweight VM and needs minimum resources as compared to the actual VM in virtualization. Docker has given the name *default* to this Docker Host because it comes with the default installation and requires to run the Docker Engine.

C. Docker Swarm

Docker Swarm is a cluster management and orchestration tool that connects and controls several Docker nodes to form a single virtual system [9], [10]. It is an enhancement of Docker container technology for creating a cluster of nodes across several machines or clouds and designing distributed systems in multiple clouds. Docker Swarm offers several essential attributes of a distributed system to the Swarm cluster such as availability, reliability, fault tolerance, maintainability and scalability, which is an added advantage to the basic container and transforms this container-based system into a distributed system.

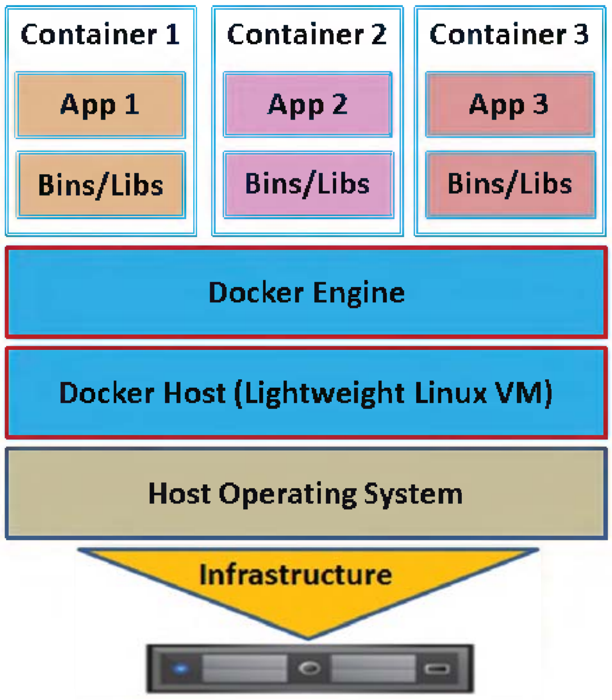


Fig. 2. Docker Host (a lightweight VM) is required to run Docker Engine on a non-Linux machine

III. Design of a Distributed System in Multiple Clouds Using Docker Swarm

Fig. 3 illustrates the design of a distributed system using Docker Swarm cluster of three manager and two worker nodes on multiple clouds, which will be simulated later. All five clouds in the illustrated design are supported by Docker [11]. The number of manager and worker nodes can be chosen depending on the specific requirement of the design. A Docker Swarm cluster requires one or more managers to manage resources and services within the cluster, where one manager is the leader of the cluster and other managers are followers [12]. Workers perform only basic operations such as execution of tasks and routing data traffic related to containers [13]. Managers manage these workers and perform scheduling of tasks for them and their regular health check to ensure that they are up and running [13]. The workflow of manager and worker is elicited in Fig. 4, which is based the *Raft Consensus Algorithm*.

A leader is the main manager and controller in the cluster, while other managers/followers coordinate with it and ensure that leader is immediately replaced with any of the manager in case of its sudden failure [12]. During the normal working of a cluster, it is possible to communicate with any manager other than the leader, but all these communications are automatically copied to the leader. The leader sends periodic heartbeats to all other nodes in order to inform them that it is alive. If other managers/followers receive no communication over a period of time called the *election timeout*, then they assume that the leader has crashed or there is no viable leader and begin an election to choose a new leader. The leader election process

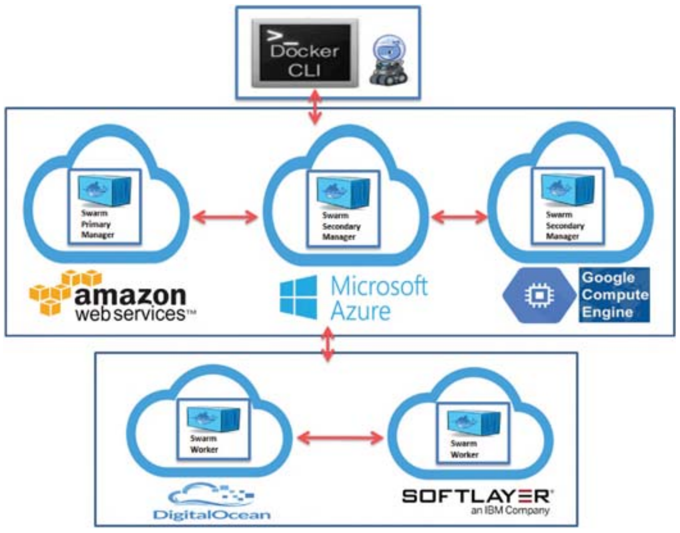


Fig. 3. Docker Swarm-based Distributed System in Multiple Clouds

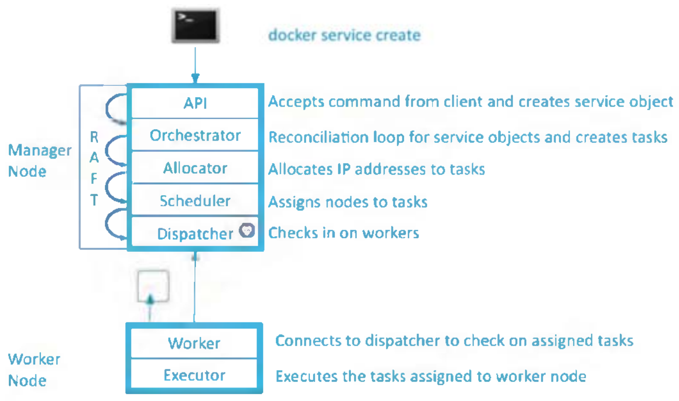
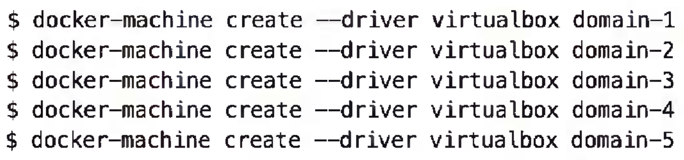

docker service create
API
Accepts command from client and creates service object
Orchestrator
Reconciliation loop for service objects and creates tasks
Allocator
Allocates IP addresses to tasks
Scheduler
Assigns nodes to tasks
Dispatcher
Checks in on workers
Manager Node
R A F T
Worker
Connects to dispatcher to check on assigned tasks
Executor
Executes the tasks assigned to worker node
Worker Node

Fig. 4. Docker Swarm Node Breakdown and Workflow [13]

requires the quorum of managers, i.e., a majority of managers must be available to promote a new manger as a leader. Availability of several managers and their implicit coordination make this Docker Swarm cluster a highly available and reliable distributed system [12].

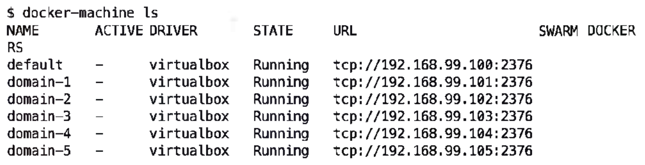
IV. Simulated Development of a Distributed System Using Docker Swarm

This section presents the simulated development of the previously described distributed system using Docker Swarm. This simulation is carried out in VirtualBox, where five docker machines are created; however, this VirtualBox design can be replicated with any preferred cloud provider to develop this system in the cloud. Fig. 5 shows the creation of five Docker machines (i.e., lightweight VMs) in VirtualBox to form a distributed system. These Docker machines are named domain-1 to domain-5 and assigned private IP addresses and standard Docker port as shown in Fig. 6. Subsequently, for creating the Docker Swarm cluster, three swarm managers are created on domain-1 to domain-3, and two Swarm workers are created on domain-4 and domain-5 (see Fig. 7). Here, the domain-1 is elected as a leader of the Docker Swarm cluster (see Fig. 7). This complete development process of a Docker Swarm-based distributed system is relatively easy process and can



```
$ docker-machine create --driver virtualbox domain-1
$ docker-machine create --driver virtualbox domain-2
$ docker-machine create --driver virtualbox domain-3
$ docker-machine create --driver virtualbox domain-4
$ docker-machine create --driver virtualbox domain-5
```

Fig. 5. Creating Docker lightweight virtual machines (VMs)



```
$ docker-machine ls
NAME       ACTIVE  DRIVER       STATE     URL                          SWARM  DOCKER
RS
default    -       virtualbox   Running   tcp://192.168.99.100:2376
domain-1   -       virtualbox   Running   tcp://192.168.99.101:2376
domain-2   -       virtualbox   Running   tcp://192.168.99.102:2376
domain-3   -       virtualbox   Running   tcp://192.168.99.103:2376
domain-4   -       virtualbox   Running   tcp://192.168.99.104:2376
domain-5   -       virtualbox   Running   tcp://192.168.99.105:2376
```

Fig. 6. Cluster of five running domains (lightweight VMs) with different private IP addresses and standard Docker Port 2376

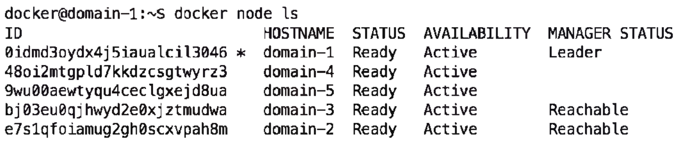
be replicated on multiple clouds (shown in Fig. 3), however, it requires an account/subscription on these clouds [14], and then replacing the driver name with the corresponding cloud in Fig 5, for example: *–driver virtualbox* to *–driver amazonec2/azure/google/digitalocean/softlayer* [11].

V. Performance Evaluation of a Distributed System Using Docker Swarm

This section presents the performance evaluation of the Docker Swarm-based distributed system based on some common attributes of distributed systems and it also compares with other container-based distributed systems.

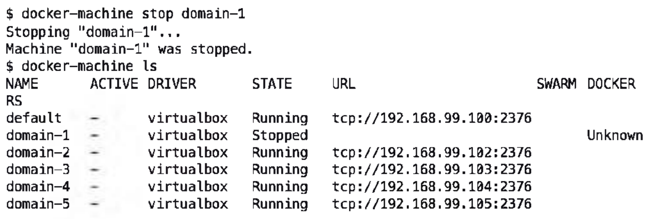
A. *High Availability and Fault Tolerance*

Both high availability and fault tolerance are the inbuilt attributes of Docker Swarm that allows a distributed system to benignly manage the failover of the Swarm leader and continues the normal working of the cluster. For evaluating the high availability and fault tolerance of the Docker Swarm-based distributed system, the leader on the Docker machine (domain-1) is suddenly stopped (see Fig. 8) because this could affect the normal working of the distributed system, as the leader controls the complete cluster.



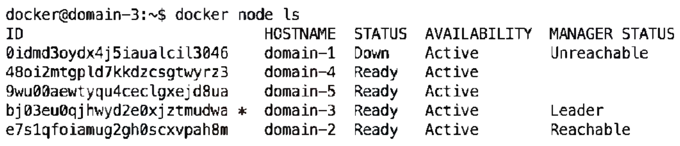
```
docker@domain-1:~S docker node ls
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046 *   domain-1  Ready   Active        Leader
48oi2mtgpld7kkdzcsgtwyrz3     domain-4  Ready   Active
9wu00aewtyqu4ceclgxejd8ua     domain-5  Ready   Active
bj03eu0qjhwyd2e0xjztmudwa     domain-3  Ready   Active        Reachable
e7s1qfoiamug2gh0scxvpah8m     domain-2  Ready   Active        Reachable
```

Fig. 7. Docker Swarm cluster with 3 managers (with domain-1 as a leader) and 2 workers



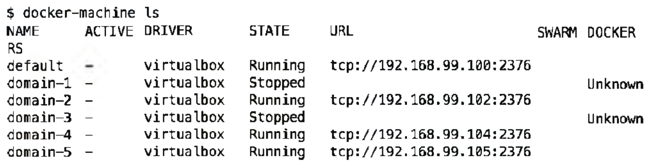
```
$ docker-machine stop domain-1
Stopping "domain-1"...
Machine "domain-1" was stopped.
$ docker-machine ls
NAME       ACTIVE  DRIVER       STATE     URL                          SWARM  DOCKER
RS
default    -       virtualbox   Running   tcp://192.168.99.100:2376
domain-1   -       virtualbox   Stopped                                       Unknown
domain-2   -       virtualbox   Running   tcp://192.168.99.102:2376
domain-3   -       virtualbox   Running   tcp://192.168.99.103:2376
domain-4   -       virtualbox   Running   tcp://192.168.99.104:2376
domain-5   -       virtualbox   Running   tcp://192.168.99.105:2376
```

Fig. 8. Failover procedure of a manager (leader) on the domain-1 when the domain-1 has stopped/failed



```
docker@domain-3:~$ docker node ls
ID                            HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046     domain-1   Down    Active        Unreachable
48oi2mtgpld7kkdzcsgtwyrz3     domain-4   Ready   Active
9wu00aewtyqu4ceclgxejd8ua     domain-5   Ready   Active
bj03eu0qjhwyd2e0xjztmudwa *   domain-3   Ready   Active        Leader
e7s1qfoiamug2gh0scxvpah8m     domain-2   Ready   Active        Reachable
```

Fig. 9. Successful failover procedure of a manager (leader) on the domain-1 when it is automatically taken over by the domain-3 as a new manager (leader)



```
$ docker-machine ls
NAME      ACTIVE DRIVER     STATE     URL                         SWARM DOCKER
RS
default   -      virtualbox Running   tcp://192.168.99.100:2376
domain-1  -      virtualbox Stopped                                     Unknown
domain-2  -      virtualbox Running   tcp://192.168.99.102:2376
domain-3  -      virtualbox Stopped                                     Unknown
domain-4  -      virtualbox Running   tcp://192.168.99.104:2376
domain-5  -      virtualbox Running   tcp://192.168.99.105:2376
```

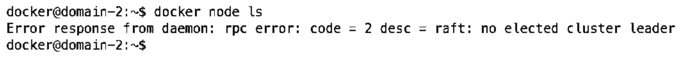
Fig. 10. Failover procedure of a manager (leader) on the domain-3 when the domain-3 has stopped/failed

Despite the failure of the leader on the Docker machine (domain-1), the Docker Swarm cluster was working normally and promoted the manager of the Docker machine (domain-3) as the leader of the cluster (see Fig. 9). This Swarm-based distributed system tolerates the first failure and provides high availability due to the readiness of extra managers and the quorum of managers to promote a new manger as a leader (see Table I).

This Docker Swarm-based distributed system has left with two active managers and it is again tested for its high availability and fault tolerance. For the second evaluation, the leader on the Docker machine (domain-3) is suddenly stopped (see Fig. 10) to affect the normal working of the cluster.

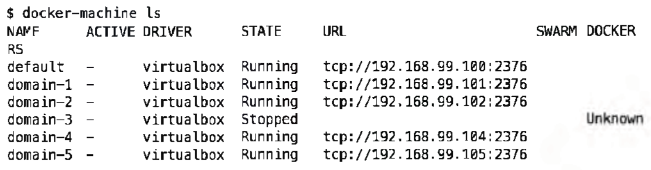
Now two managers have failed on the Docker machines (domain-1 and domain-3) and the only manager is active on the Docker machine (domain-2). Now, the Docker Swarm cluster could not continue its normal operation (see Fig. 11) because it loses the quorum of managers, meaning it requires minimum two managers to promote a new manger as a leader and for successful failover procedure.

To verify the above requirement of two managers (i.e., quorum), the manager of Docker machine (domain-1) is reinstated to its running position as shown in Fig. 12. As soon as this manager has started, the Swarm cluster has promoted the manager of Docker machine (domain-2) as a leader and started normal working again (see Fig. 13).



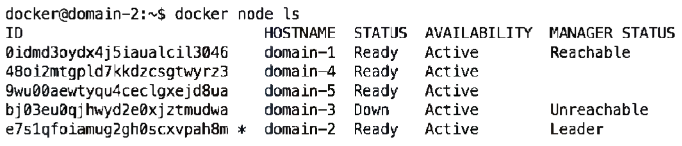
```
docker@domain-2:~$ docker node ls
Error response from daemon: rpc error: code = 2 desc = raft: no elected cluster leader
docker@domain-2:~$
```

Fig. 11. Unsuccessful failover procedure of a manager (leader) on the domain-3 by the domain-2 manager when the domain-3 is failed



```
$ docker-machine ls
NAME      ACTIVE DRIVER     STATE     URL                         SWARM DOCKER
RS
default   -      virtualbox Running   tcp://192.168.99.100:2376
domain-1  -      virtualbox Running   tcp://192.168.99.101:2376
domain-2  -      virtualbox Running   tcp://192.168.99.102:2376
domain-3  -      virtualbox Stopped                                     Unknown
domain-4  -      virtualbox Running   tcp://192.168.99.104:2376
domain-5  -      virtualbox Running   tcp://192.168.99.105:2376
```

Fig. 12. Failover procedure of a manager (leader) on the domain-3 when the domain-1 is again up and running



```
docker@domain-2:~$ docker node ls
ID                            HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046     domain-1   Ready   Active        Reachable
48oi2mtgpld7kkdzcsgtwyrz3     domain-4   Ready   Active
9wu00aewtyqu4ceclgxejd8ua     domain-5   Ready   Active
bj03eu0qjhwyd2e0xjztmudwa     domain-3   Down    Active        Unreachable
e7s1qfoiamug2gh0scxvpah8m *   domain-2   Ready   Active        Leader
```

Fig. 13. Successful failover procedure of a manager (leader) on the domain-3 by the domain-2 manager when the domain-3 is failed but the domain-1 is again up and running

TABLE I. FAILURE TOLERATION CAPABILITY OF DOCKER SWARM DEPENDABLE SYSTEM

Controller and Replicas of Manager/Leader	**Number of Failures Tolerated**
3	1
5	2
7	3
9	4

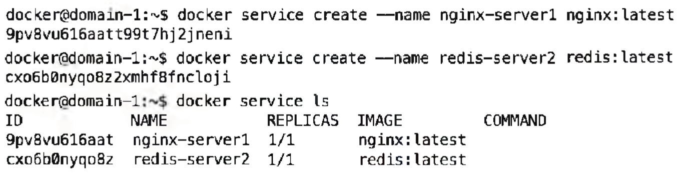
This experimental demonstration shows that the Docker Swarm-based distributed system can offer high availability and fault tolerance for one failure when it has three active managers. A Docker Swarm cluster employs *Raft Consensus Algorithm* for the coordination among their managers, therefore, the Docker Swarm-based distributed system can offer fault tolerance for maximum *f* manager failures when it has incorporated *2f+1* managers in the cluster (see Table I) [15].

Though a swarm loses the quorum of managers, swarm tasks on existing worker nodes continue to run. Nonetheless, swarm nodes cannot be added, updated, or removed, and new or existing tasks cannot be started, stopped, moved, or updated. Additionally, Docker Swarm also ensures the availability of worker nodes in the cluster. Nonetheless, this failure of a node can only affect the individual worker node but not the entire cluster. The manager regularly checks the health of a worker node by listening their heartbeat to ensure the progress of their assigned tasks. The time period to report the health of a worker is decided by the leader/manager.

B. *Automatic Scalability, Load Balancing and Maintainability of Services*

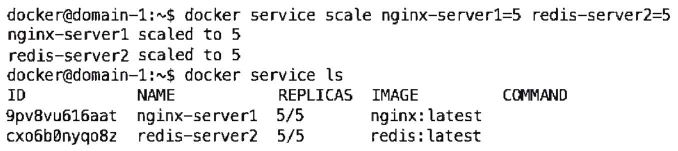
A Docker Swarm-based distributed system also offers automatic scalability, load balancing and maintainability of services. For evaluating these attributes, two services *nginx* and *redis* are created on the nodes of the Docker Swarm cluster, which are called nginx-server1 and redis-server2 respectively and shown in Fig. 14.

Firstly, these services can be easily scaled depending on the



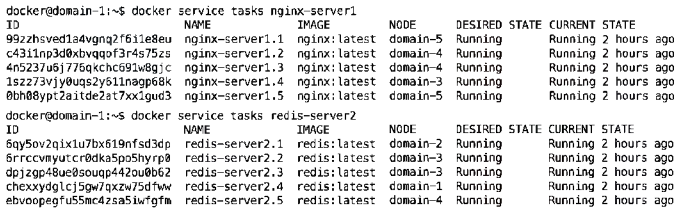
```
docker@domain-1:~$ docker service create --name nginx-server1 nginx:latest
9pv8vu616aatt99t7hj2jneni
docker@domain-1:~$ docker service create --name redis-server2 redis:latest
cxo6b0nyqo8z2xmhf8fncloji
docker@domain-1:~$ docker service ls
ID            NAME           REPLICAS  IMAGE         COMMAND
9pv8vu616aat  nginx-server1  1/1       nginx:latest
cxo6b0nyqo8z  redis-server2  1/1       redis:latest
```

Fig. 14. Creating two services *nginx* and *redis* on the Docker Swarm nodes



```
docker@domain-1:~$ docker service scale nginx-server1=5 redis-server2=5
nginx-server1 scaled to 5
redis-server2 scaled to 5
docker@domain-1:~$ docker service ls
ID            NAME           REPLICAS  IMAGE         COMMAND
9pv8vu616aat  nginx-server1  5/5       nginx:latest
cxo6b0nyqo8z  redis-server2  5/5       redis:latest
```

Fig. 15. Scaling two services *nginx* and *redis* on the Docker Swarm nodes



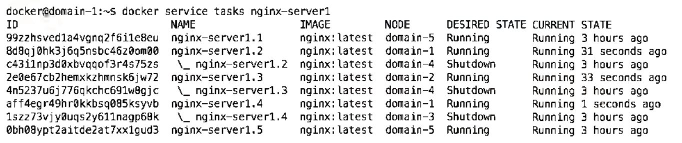
```
docker@domain-1:~$ docker service tasks nginx-server1
ID                         NAME             IMAGE         NODE      DESIRED STATE CURRENT STATE
99zzhsved1a4vgnq2f6i1e8eu  nginx-server1.1  nginx:latest  domain-5  Running       Running 2 hours ago
c43i1np3d0xbvqqof3r4s75zs  nginx-server1.2  nginx:latest  domain-4  Running       Running 2 hours ago
4n5237u6j776qkchc691w8gjc  nginx-server1.3  nginx:latest  domain-4  Running       Running 2 hours ago
1szz73vjy0uqs2y611nagp68k  nginx-server1.4  nginx:latest  domain-3  Running       Running 2 hours ago
0bh08ypt2aitde2at7xx1gud3  nginx-server1.5  nginx:latest  domain-5  Running       Running 2 hours ago
docker@domain-1:~$ docker service tasks redis-server2
ID                         NAME             IMAGE         NODE      DESIRED STATE CURRENT STATE
6qy5ov2qix1u7bx619nfsd3dp  redis-server2.1  redis:latest  domain-2  Running       Running 2 hours ago
6rrccvmyutcr0dka5po5hyrp0  redis-server2.2  redis:latest  domain-3  Running       Running 2 hours ago
dpjzgp48ue0souqp442ou0b62  redis-server2.3  redis:latest  domain-3  Running       Running 2 hours ago
chexxydglcj5gw7qxzw75dfww  redis-server2.4  redis:latest  domain-1  Running       Running 2 hours ago
ebvoopegfu55mc4zsa5iwfgfm  redis-server2.5  redis:latest  domain-4  Running       Running 2 hours ago
```

Fig. 16. Automatic load balancing of all the replicas of the two services *nginx* and *redis* on the Docker Swarm nodes

system's requirements; here, they are scaled to five instances as shown in Fig. 15.

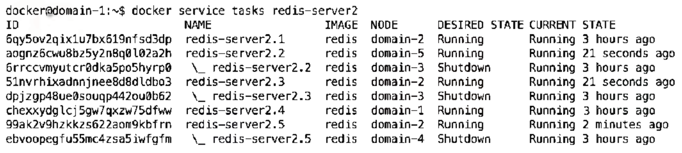
Once they are scaled, Docker Swarm automatically balances the load of the cluster and allocates all these replicas to the nodes, which have the best available resources at that moment. Fig. 16 shows that ten instances of these two services are appropriately assigned to the different Swarm nodes (across the domain-1 to domain-5).

Maintainability is a very important feature of a distributed system by which the system can be restored to the same operational status of services after a failure occurs on any of the node within the cluster. For testing the maintainability of two services *nginx-server1* and *redis-server2*, two nodes domain-3 and domain-4 are abruptly stopped. Later, the status of all the instances of the first service *nginx-server1* are checked. Fig. 17 shows that three instances nginx-server1.2, nginx-server1.3 and nginx-server1.4; which were running on domain-3 and domain-4 are automatically switched on to the other running nodes after the failure of these nodes. Similarly, Fig. 18 shows the maintainability of *redis-server2*, where three instances redis-server2.2, redis-server2.3 and redis-server2.5; which were running on domain-3 and domain-4 are automatically switched on to the other running nodes after the failure of these nodes.



```
docker@domain-1:~$ docker service tasks nginx-server1
ID                         NAME                 IMAGE         NODE      DESIRED STATE CURRENT STATE
99zzhsved1a4vgnq2f6i1e8eu  nginx-server1.1      nginx:latest  domain-5  Running       Running 3 hours ago
8d8qj0hk3j6q5nsbc46z0om00  nginx-server1.2      nginx:latest  domain-1  Running       Running 31 seconds ago
c43i1np3d0xbvqqof3r4s75zs   \_ nginx-server1.2  nginx:latest  domain-4  Shutdown      Running 3 hours ago
2e0e67cb2hemxkzhmnsk6jw72  nginx-server1.3      nginx:latest  domain-2  Running       Running 33 seconds ago
4n5237u6j776qkchc691w8gjc   \_ nginx-server1.3  nginx:latest  domain-4  Shutdown      Running 3 hours ago
aff4egr49hr0kkbsq085ksyvb  nginx-server1.4      nginx:latest  domain-1  Running       Running 1 seconds ago
1szz73vjy0uqs2y611nagp68k   \_ nginx-server1.4  nginx:latest  domain-3  Shutdown      Running 3 hours ago
0bh08ypt2aitde2at7xx1gud3  nginx-server1.5      nginx:latest  domain-5  Running       Running 3 hours ago
```

Fig. 17. Maintainability of all the replicas of the service *nginx* from the failure nodes on to the running nodes



```
docker@domain-1:~$ docker service tasks redis-server2
ID                         NAME                 IMAGE  NODE      DESIRED STATE CURRENT STATE
6qy5ov2qix1u7bx619nfsd3dp  redis-server2.1      redis  domain-2  Running       Running 3 hours ago
aognz6cwu8bz5y2n8q0l02a2h  redis-server2.2      redis  domain-5  Running       Running 21 seconds ago
6rrccvmyutcr0dka5po5hyrp0   \_ redis-server2.2  redis  domain-3  Shutdown      Running 3 hours ago
51nvrhixadnnjnee8d8dldbo3  redis-server2.3      redis  domain-2  Running       Running 21 seconds ago
dpjzgp48ue0souqp442ou0b62   \_ redis-server2.3  redis  domain-3  Shutdown      Running 3 hours ago
chexxydglcj5gw7qxzw75dfww  redis-server2.4      redis  domain-1  Running       Running 3 hours ago
99ak2v9hzkkzs622aom9kbfrn  redis-server2.5      redis  domain-2  Running       Running 2 minutes ago
ebvoopegfu55mc4zsa5iwfgfm   \_ redis-server2.5  redis  domain-4  Shutdown      Running 3 hours ago
```

Fig. 18. Maintainability of all the replicas of the service *redis* from the failure nodes on to the running nodes

This experiment demonstrates that the Docker Swarm-based distributed system provides automatic scalability, load balancing and maintainability of services.

C. Scalability of Large Clusters

The several recent research studies [16], [17], [18], [19] have been conducted to test and compare the scalability performance of Docker Swarm with other *Containers*, where Docker Swarm and Google Kubernetes were tested and compared based on the large cluster size. Initially, the scalability performance test was carried out by both organizations Docker [16] and Google Kubernetes [17] for the large cluster size. Subsequently, a Docker-sponsored study for the comparison of the scalability performance of Docker Swarm and Google Kubernetes was performed by an independent technology consultant Jeff Nickoloff [18], [19]. He designed a *Cloud Container Cluster Common Benchmark* framework (available on GitHub [20]) to test the performance of both container platforms while running 30,000 containers across 1,000 node in a cluster.

This automated test framework mainly compared the scalability performance of Docker Swarm and Kubernetes based on two main criteria: (1) container startup time and (2) system responsiveness under the load as the cluster is built. A fully loaded cluster is 1,000 nodes running 30,000 containers (30 containers per node). As nodes were added to the cluster, the automated test was stopped and measured the container startup time, and system responsiveness under the load. There were five breakpoints during the complete test when the cluster was 10%, 50%, 90%, 99%, and 100% full. At each of these load levels 1,000 test iterations are executed. The 10% full cluster represents 100 nodes, and 3,000 containers and test harness was stopped at this stage for adding new nodes. Subsequently, it measured the time it takes to startup a new container (in this case the 3,001st container), and how long it takes to list all the running containers (3,001). This test was repeated 1,000 times to obtain precise results. The final result shows that Docker Swarm is on average five times faster in terms of container startup time and seven times faster in terms of system responsiveness under the load (in delivering operational insights necessary to run a cluster at scale in production).

In summary, both Docker Swarm and Google Kubernetes frameworks are different in attributes, therefore, each will perform better under certain circumstances. The comparative study assumed that Docker Swarm is a good choice for small-scale workloads, however, large-scale workloads are best handled by Kubernetes or some other orchestration framework such as Mesos [21].

VI. Conclusion

This paper presented the performance evaluation of a distributed system using Docker Swarm. It illustrated the design and simulated development of the distributed system in multiple clouds using Docker Swarm. Subsequently, the paper demonstrated an evaluation of several attributes of the distributed system such as high availability and fault tolerance; automatic scalability, load balancing and maintainability of services; and scalability of large clusters. This evaluation demonstrated that the Docker Swarm-based distributed system is relatively easy to design and act as a natural distributed systems due to several inbuilt attributes of distributed systems. Additionally, Docker supports many virtual and cloud environments, therefore, this Docker Swarm-based distributed system can also be replicated into multiple clouds. However, this simulated distributed system was a small Docker Swarm cluster, therefore, in the future, it will be useful to perform an evaluation on a large Docker Swarm cluster. Moreover, it may be worthwhile to develop and evaluate this distributed system in the multiple clouds to analyse their communication and interoperability.

REFERENCES

[1] W. Emmerich, "Distributed component technologies and their software engineering implications," in *Proceedings of the 24th international conference on Software engineering*. ACM, 2002, pp. 537–546.

[2] A. S. Tanenbaum and M. Van Steen, *Distributed Systems*. Prentice-Hall, 2007.

[3] N. Naik, "Building a virtual system of systems using Docker Swarm in multiple clouds," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016.

[4] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 887–894.

[5] N. Naik, "Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker Swarm," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016.

[6] ——, "Migrating from virtualization to dockerization in the cloud: Simulation and evaluation of distributed systems," in *IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA)*. IEEE, 2016, pp. 1–8.

[7] Docker.com. (2016) Docker use cases. [Online]. Available: https://www.docker.com/use-cases

[8] J. Turnbull, *The Docker Book: Containerization is the new Virtualization*. James Turnbull, 2014.

[9] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2017.

[10] Docker.com. (2016) Docker swarm. [Online]. Available: https://www.docker.com/products/docker-swarm

[11] ——. (2016) Supported drivers. [Online]. Available: https://docs.docker.com/machine/drivers/

[12] ——. (2016) High availability in docker swarm. [Online]. Available: https://docs.docker.com/swarm/multi-manager-setup/

[13] ——. (2016, July 28) Docker built-in orchestration ready for production: Docker 1.12 goes ga. [Online]. Available: https://blog.docker.com/2016/07/docker-built-in-orchestration-ready-for-production-docker-1-12-goes-ga/

[14] N. Naik, "Connecting Google cloud system with organizational systems for effortless data analysis by anyone, anytime, anywhere," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016.

[15] Docker.com. (2016) Set up high availability. [Online]. Available: https://docs.docker.com/ucp/high-availability/set-up-high-availability/

[16] A. Luzzardi. (2015, November 16) Scale testing docker swarm to 30,000 containers. [Online]. Available: https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/

[17] W. Tyczynski. (2015, September 10) Kubernetes performance measurements and roadmap. [Online]. Available: http://blog.kubernetes.io/2015/09/kubernetes-performance-measurements-and.html

[18] J. Nickoloff. (2016, March 9) Evaluating container platforms at scale. [Online]. Available: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c#.uj0blrkt9

[19] M. Coleman. (2016, March 9) Docker swarm exceeds kubernetes performance at scale. [Online]. Available: https://blog.docker.com/2016/03/swarmweek-docker-swarm-exceeds-kubernetes-scale/

[20] J. Nickoloff. (2016, March 9) Cloud container cluster common benchmark. [Online]. Available: https://github.com/allingeek/c4-bench

[21] J. Jackson. (2016, March 10) Docker swarm wins scaling benchmark but don't take that as gospel. [Online]. Available: https://thenewstack.io/docker-swarm-wins-scaling-benchmark-dont-take-gospel/