Docker Swarm Notes

By Andhika A. Sentosa <andhika.as@gmail.com>

for 1rstwap.com (at 9 March 2019)

Last Updated: 13. June 2019

Table of Contents

Tutorial Objectives

1. Understand how to install, setup and configure/administrate Docker container and Docker Swarm.
2. Install and configure following container inside Docker Swarm :
   - Nginx
   - MariaDB Galera Cluster
   - Minio

A. Introductions

Docker Docs:

- https://docs.docker.com/machine/overview/
- https://docs.docker.com/engine/swarm/
- https://docs.docker.com/get-started/part4/

A swarm is a group of machines that are running Docker and joined into a cluster. After that has happened, you continue to run the Docker commands you're used to, but now they are executed on a cluster by a **swarm manager**. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as **nodes**.

You can use Docker Machine to:

- Install and run Docker on Mac or Windows
- Provision and manage multiple remote Docker hosts
- Provision Swarm clusters

Swarms are a cluster of nodes that consist of the following components:

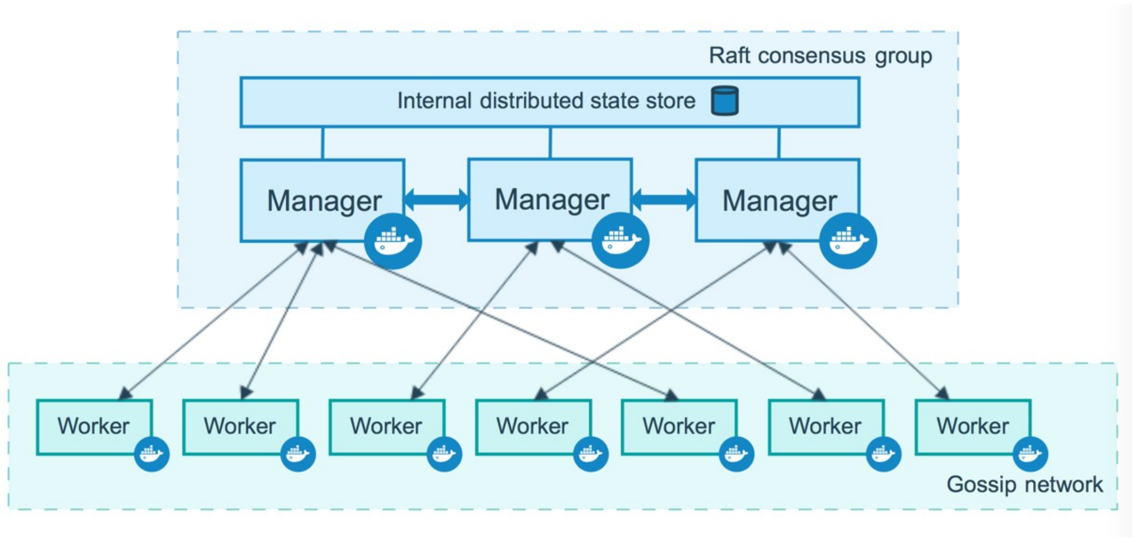
● **Manager Nodes:** Tasks involved here include Control Orchestration, Cluster Management, and Task Distribution.

● **Worker Nodes:** Functions here include running containers and services that have been assigned by the manager node.

● **Services:** This gives a description of the blueprint via which an individual container can distribute itself across the nodes.

● **Tasks:** These are slots in which single containers place their work.

There are two types of nodes: **managers** and **workers**. A swarm consists of one or more nodes: physical or virtual machines running Docker Engine.



A. 1. Manager nodes

Manager nodes handle cluster management tasks:

- maintaining cluster state
- scheduling services
- serving swarm mode HTTP API endpoints

Using a Raft implementation, the managers maintain a consistent internal state of the entire swarm and all the services running on it. An `N` manager cluster tolerates the loss of at most `(N-1)/2` managers.

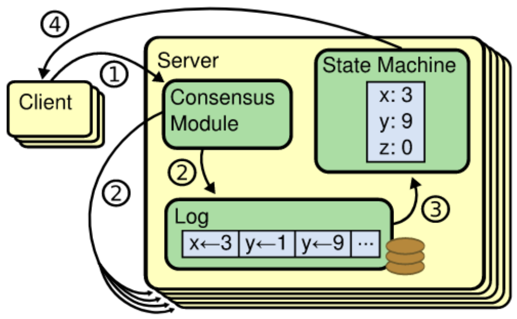


Figure 1: Raft machine arch.

More: https://raft.github.io/

A. 2. Worker nodes

Worker nodes are also instances of Docker Engine whose sole purpose is to execute containers. Worker nodes don't participate in the Raft distributed state, make scheduling decisions, or serve the swarm mode HTTP API.

You can create a swarm of one manager node, but you cannot have a worker node without at least one manager node. By default, all managers are also workers. In a single manager node cluster, you can run commands like $ `docker service create` and the scheduler places all tasks on the local Engine.

You can promote a worker node to be a manager by running $ `docker node promote`

A. 3. Swarm Feature highlights

- **Cluster management integrated with Docker Engine**

- **Decentralized design:** Docker Engine handles any specialization at runtime from a single disk image to create multiple nodes.
- **Declarative service model:** Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack.
- **Scaling:** For each service, you can declare the number of tasks you want to run.

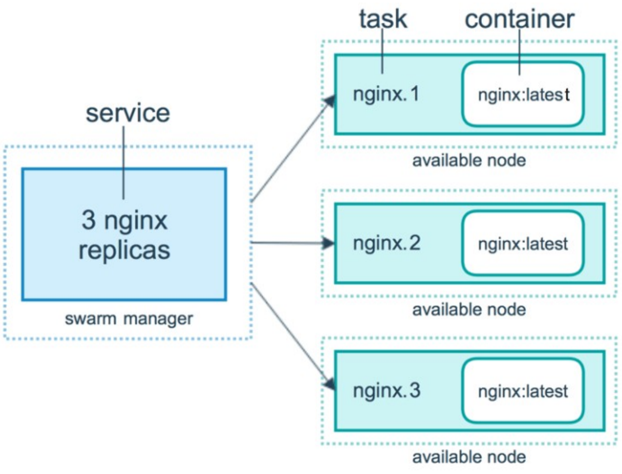


Figure 2: Example of Scaling task of nginx

- **Desired state reconciliation:** The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state.
- **Multi-host networking:** You can specify an overlay network for your services.
- **Service discovery:** Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers.
- **Load balancing:** You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.

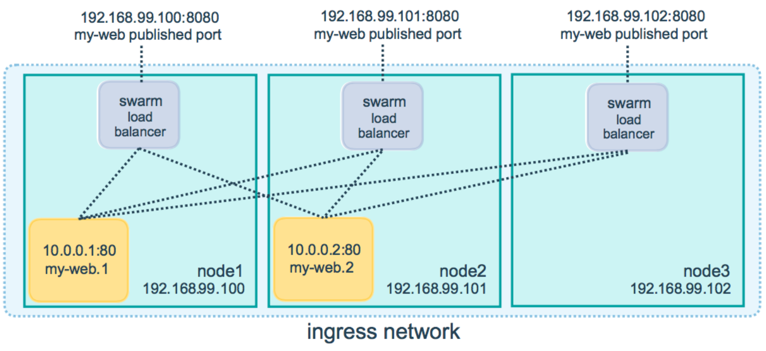


Figure 3: Swarm load balancer

- **Secure by default:** Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes.
- **Rolling updates:** The swarm manager lets you control the delay between service deployment to different sets of nodes.

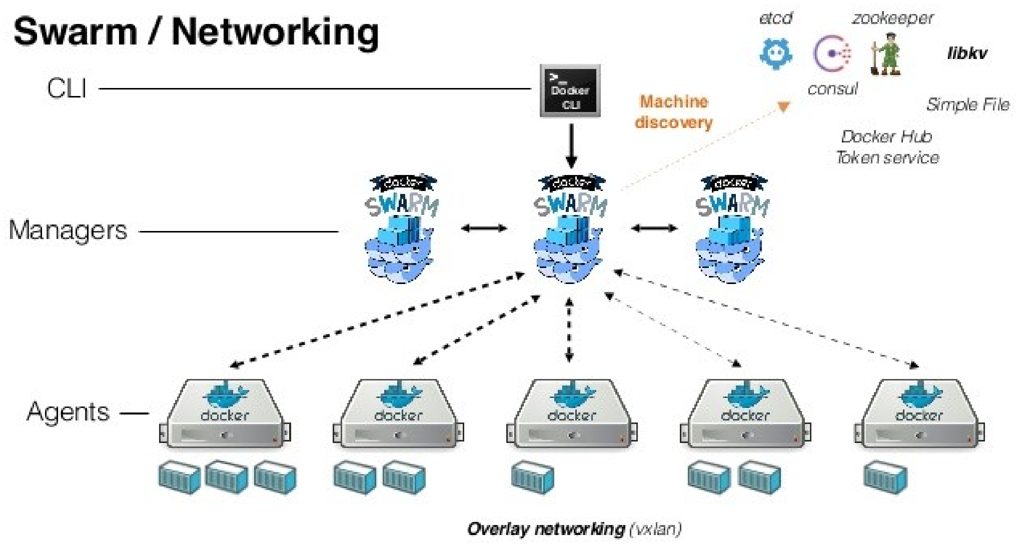
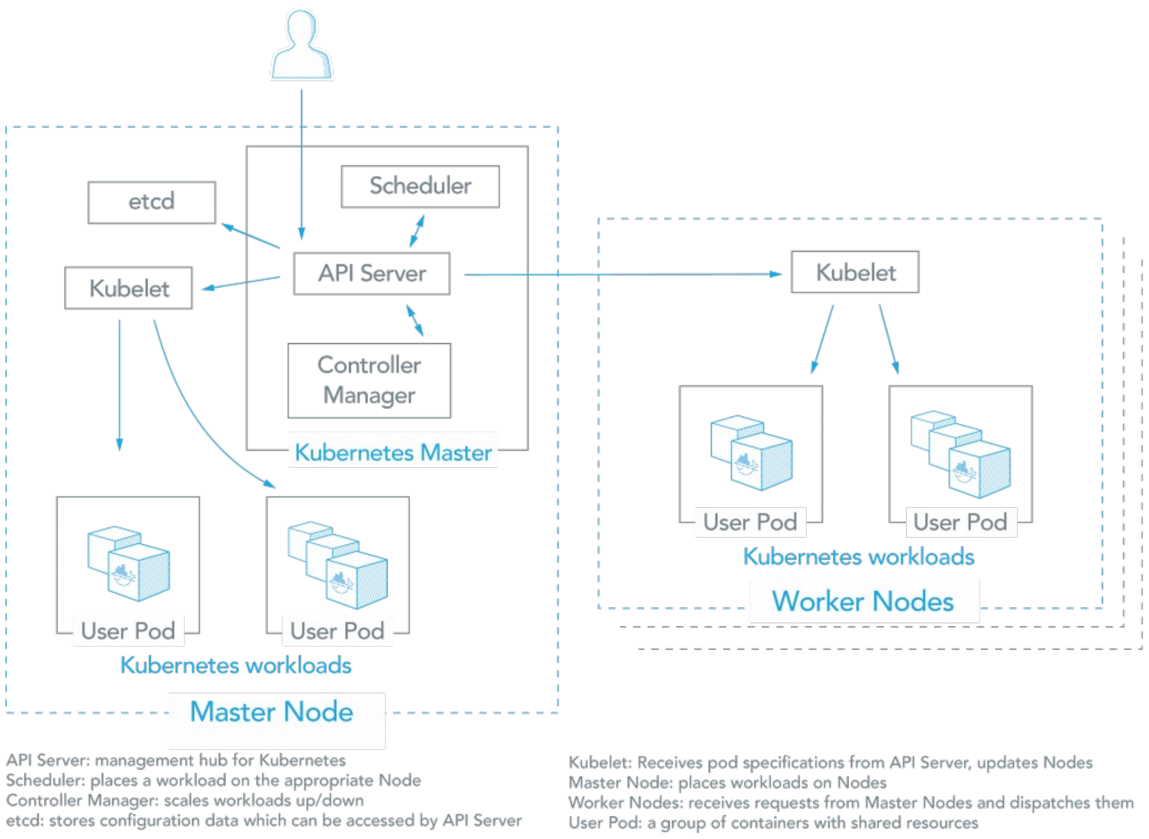


Figure 4: https://technologyadvice.com/blog/information-technology/kubernetes-vs-docker/

A. 4. Docker Swarm vs Kubernetes

Reff: https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/



API Server: management hub for Kubernetes
Scheduler: places a workload on the appropriate Node
Controller Manager: scales workloads up/down
etcd: stores configuration data which can be accessed by API Server

Kubelet: Receives pod specifications from API Server, updates Nodes
Master Node: places workloads on Nodes
Worker Nodes: receives requests from Master Nodes and dispatches them
User Pod: a group of containers with shared resources

Figure 5: Kubernetes Nodes

Kubernetes

Kubernetes is an open-source platform created by Google for container deployment operations, scaling up and down, and automation across the clusters of hosts. This production-ready, enterprise-grade, self-healing (auto-scaling, auto-replication, auto-restart, auto-placement) platform is modular, and so it can be utilized for any architecture deployment.

Kubernetes also distributes the load amongst containers. It aims to relieve the tools and components from the problem faced due to running applications in private and public clouds by placing the containers into groups and naming them as logical units. Their power lies in easy scaling, environment agnostic portability, and flexible growth.

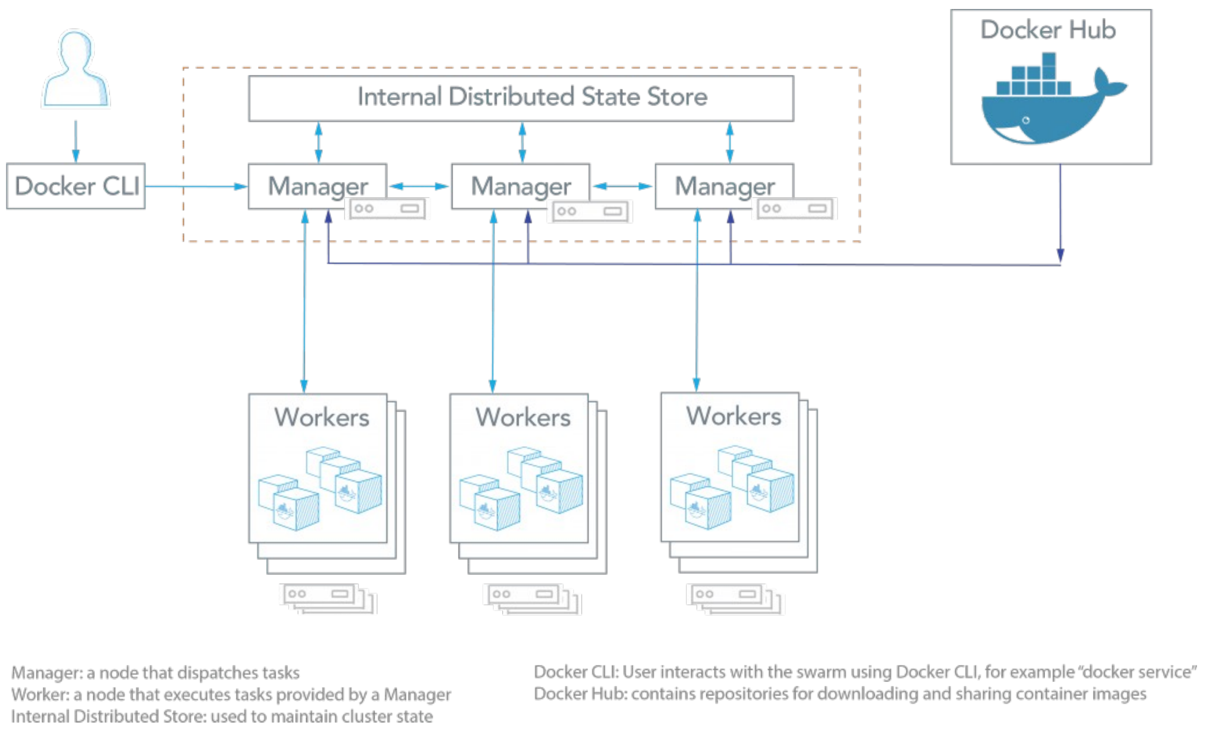

Docker Hub
Internal Distributed State Store
Docker CLI
Manager
Manager
Manager
Workers
Workers
Workers
Manager: a node that dispatches tasks
Worker: a node that executes tasks provided by a Manager
Internal Distributed Store: used to maintain cluster state
Docker CLI: User interacts with the swarm using Docker CLI, for example "docker service"
Docker Hub: contains repositories for downloading and sharing container images

Figure 6: Docker Swarm

Docker Swarm

As a platform, Docker has revolutionized the manner software was packaged. Docker Swarm or simply Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker. Any software, services, or tools that run with Docker containers run equally well in Swarm. Also, Swarm utilizes the same command line from Docker.

Swarm turns a pool of Docker hosts into a virtual, single host. Swarm is especially useful for people who are trying to get comfortable with an orchestrated environment or who need to adhere to a simple deployment technique but also have more just one cloud environment or one particular platform to run this on.

Reff: https://blog.docker.com/2016/03/swarmweek-docker-swarm-exceeds-kubernetes-scale/

Docker Swarm allows organizations to leverage the full power of the native Docker CLI and APIs. It allows developers to work in a consistent way, regardless of where their applications are developed or where they will run. Also, Swarm smooths your transition to different providers.

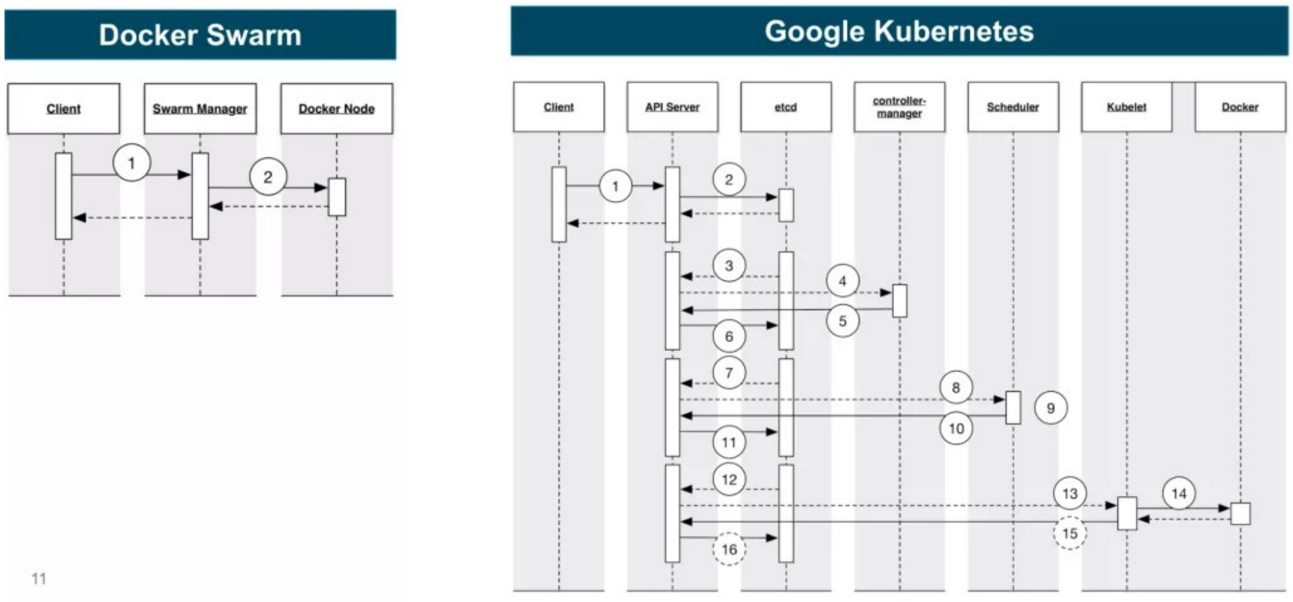
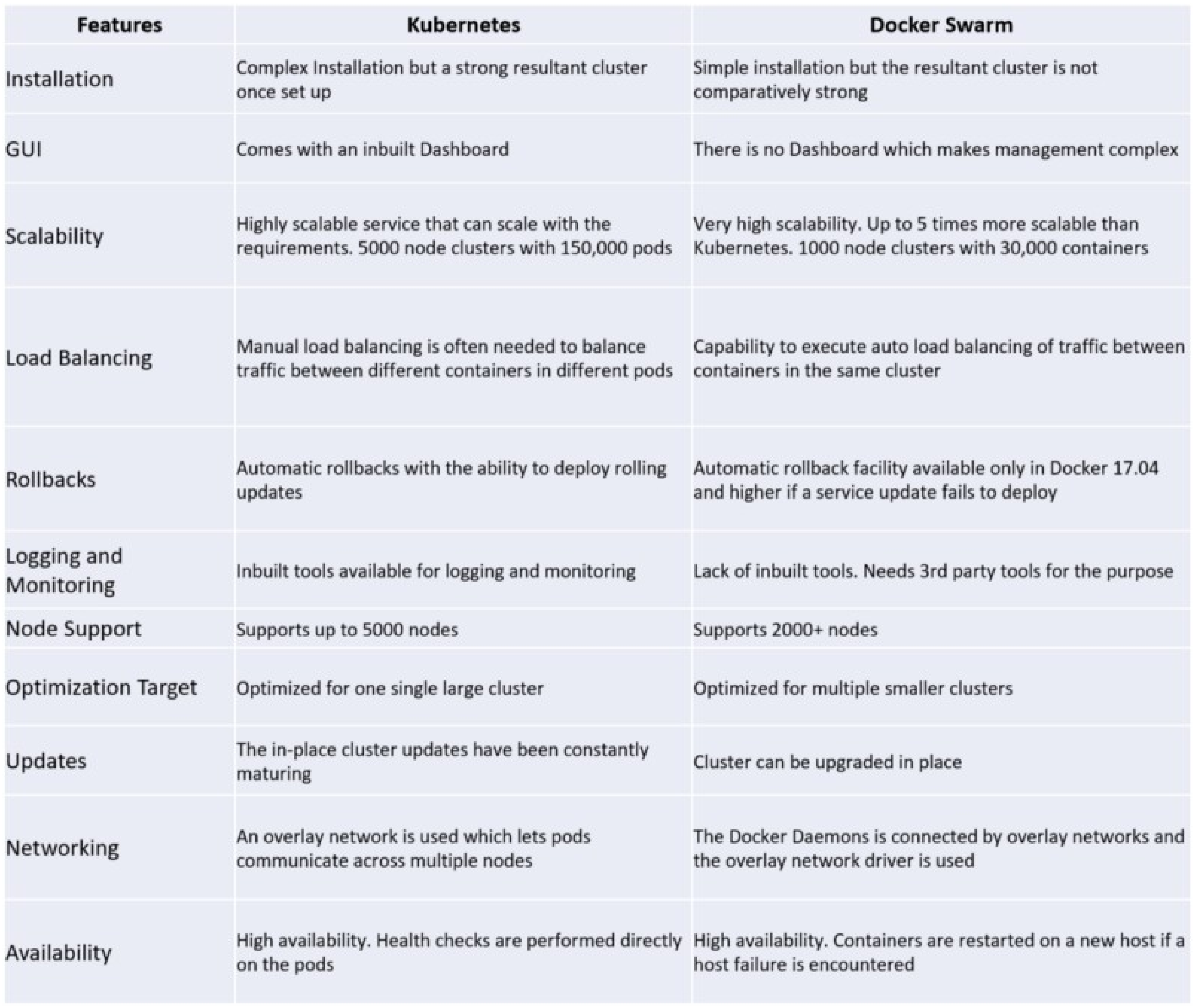


Figure 7: Architecture for making new node

Ref: https://hackernoon.com/kubernetes-vs-docker-swarm-a-comprehensive-comparison-73058543771e



Features	Kubernetes	Docker Swarm
Installation	Complex Installation but a strong resultant cluster once set up	Simple installation but the resultant cluster is not comparatively strong
GUI	Comes with an inbuilt Dashboard	There is no Dashboard which makes management complex
Scalability	Highly scalable service that can scale with the requirements. 5000 node clusters with 150,000 pods	Very high scalability. Up to 5 times more scalable than Kubernetes. 1000 node clusters with 30,000 containers
Load Balancing	Manual load balancing is often needed to balance traffic between different containers in different pods	Capability to execute auto load balancing of traffic between containers in the same cluster
Rollbacks	Automatic rollbacks with the ability to deploy rolling updates	Automatic rollback facility available only in Docker 17.04 and higher if a service update fails to deploy
Logging and Monitoring	Inbuilt tools available for logging and monitoring	Lack of inbuilt tools. Needs 3rd party tools for the purpose
Node Support	Supports up to 5000 nodes	Supports 2000+ nodes
Optimization Target	Optimized for one single large cluster	Optimized for multiple smaller clusters
Updates	The in-place cluster updates have been constantly maturing	Cluster can be upgraded in place
Networking	An overlay network is used which lets pods communicate across multiple nodes	The Docker Daemons is connected by overlay networks and the overlay network driver is used
Availability	High availability. Health checks are performed directly on the pods	High availability. Containers are restarted on a new host if a host failure is encountered

By using Kubernetes, you will be able to define how your applications should run and the manner in which they interact with other applications. Empowering the user with interfaces and composable platform primitives, Kubernetes allows high degrees of flexibility and reliability.

Docker Swarm has been designed around four pillars of principles:

● Simple and powerful user experience through the Docker Universal Control Plane single interface

using Docker-machine for all interaction to swarm.

● Resilient architecture with a single point of failure

Swarm is resilient to failures and the swarm can recover from any number of temporary node failures (machine reboots or crash with restart) or other transient errors. However, a swarm cannot automatically recover if it loses a quorum (Majority). Tasks on existing worker nodes continue to run, but administrative tasks are not possible, including scaling or updating services and joining or removing nodes from the swarm. The best way to recover is to bring the missing manager nodes back online. (Source: Recover from loosing a quorum)

(https://docs.docker.com/engine/swarm/raft/) The reason why *Docker swarm mode* is using a consensus algorithm is to make sure that all the manager nodes that are in charge of managing and scheduling tasks in the cluster, are storing the same consistent state.

Having the same consistent state across the cluster means that in case of a failure, any Manager node can pick up the tasks and restore the services to a stable state. For example, if the *Leader Manager* which is responsible for scheduling tasks in the cluster dies unexpectedly, any other Manager can pick up the task of scheduling and re-balance tasks to match the desired state.

● Backward compatibility with existing components (highly relevant for existing users)

Every Engine release strives to be backward compatible with its predecessors, and interface stability is always a priority at Docker. (https://docs.docker.com/engine/breaking_changes/)

Since every node in swarm are full docker machine, Compatibility is highly relevant.

● Automatically generated certificates that make it secure

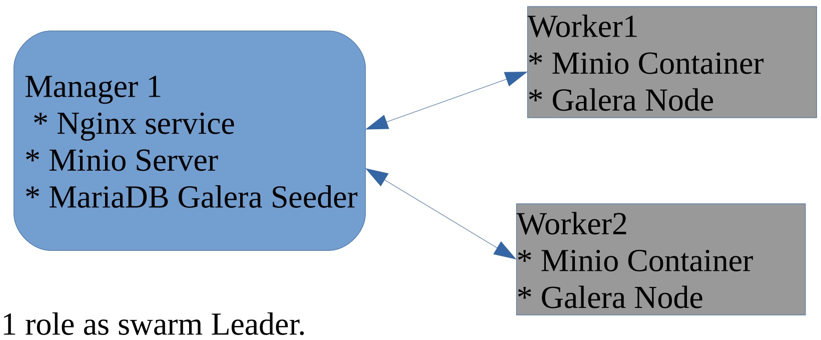
B. Install Docker + Docker-machine (swarm) :

1. Install Docker
    https://docs.docker.com/install/linux/docker-ce/ubuntu/ (Ubuntu)
    And Post script install:
    https://docs.docker.com/install/linux/linux-postinstall/

2. Install Virtualbox sebagai VM driver
    Ubuntu: sudo apt install virtualbox

3. Install docker-machine
    1. https://docs.docker.com/machine/install-machine/

C. Swarm Design for Objectives:



Manager 1 role as swarm Leader.

D. Initialize swarm:

Make docker swarm (manager & workers)

1. **Create Manager and Worker Nodes:**

```
$ docker-machine create --driver virtualbox [nama VM]
 swarm as design:
$ docker-machine create --driver virtualbox manager1
$ docker-machine create --driver virtualbox worker1
$ docker-machine create --driver virtualbox worker2
```

Note: each docker-machine create will use at least 1 GB of RAM. Use

```
--virtualbox-memory “[memory size in MiB]”
```

2. **docker swarm init, Initialize the swarm Leader by assign a machine:**

```
$ docker swarm init --advertise-addr [IP manager] --listen-addr
    [network_name local]:2377
```

Note: Any initialized swarm will be assigned as manager.

You can find the IP using: **$ docker node ls**

Fast-script:

```
docker-machine ssh manager1 "docker swarm init --listen-addr $(docker-
    machine ip manager1) --advertise-addr $(docker-machine ip
    manager1)"
```

3. **Get Leader token :**

```
export manager_token=`docker-machine ssh manager1 "docker swarm join-
    token manager -q"`
export worker_token=`docker-machine ssh manager1 "docker swarm join-
    token worker -q"`
```

Note: 1 Swarm must have at least 1 managers & >= 0 workers, and 1 swarm must have 1 leader.
Join manager to swarm using leader $manager_token.
Join worker to swarm using leader $worker_token.

4. **Assign Workers to manager using worker_token:**

Worker1:

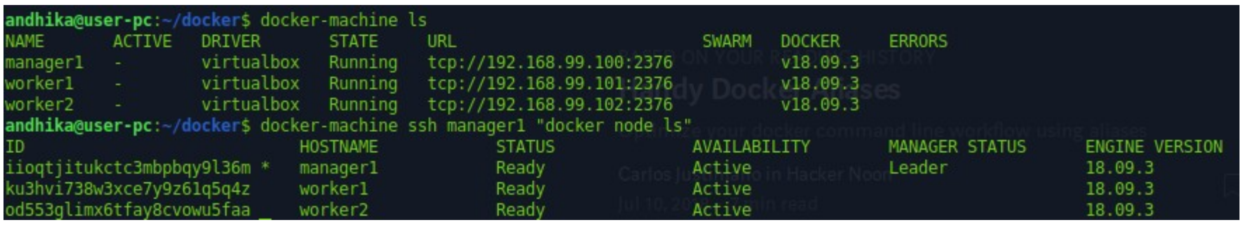
```
docker-machine ssh worker1 \
    "docker swarm join \
    --token $worker_token \
    --listen-addr $(docker-machine ip worker1) \
    --advertise-addr $(docker-machine ip worker1) \
    $(docker-machine ip manager1):2377"
```

Worker2:

```
docker-machine ssh worker2 \
     "docker swarm join \
     --token $worker_token \
     --listen-addr $(docker-machine ip worker2) \
     --advertise-addr $(docker-machine ip worker2) \
     $(docker-machine ip manager1):2377"
```

Cek swarm at manager node:

```
    docker-machine ssh manager1 "docker node ls"
```



```
andhika@user-pc:~/docker$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
manager1   -        virtualbox   Running   tcp://192.168.99.100:2376           v18.09.3
worker1    -        virtualbox   Running   tcp://192.168.99.101:2376           v18.09.3
worker2    -        virtualbox   Running   tcp://192.168.99.102:2376           v18.09.3
andhika@user-pc:~/docker$ docker-machine ssh manager1 "docker node ls"
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
iioqtjitukctc3mbpbqy9l36m *   manager1            Ready               Active              Leader              18.09.3
ku3hvi738w3xce7y9z61q5q4z     worker1             Ready               Active                                  18.09.3
od553glimx6tfay8cvowu5faa     worker2             Ready               Active                                  18.09.3
```

5. Link host machine with manager1

```
[host machine]$ eval $(docker-machine env manager1)
```

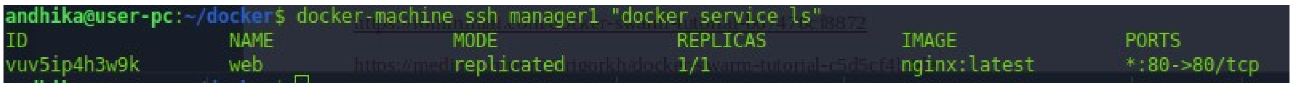
E. Create Service to swarm:

1. Create Nginx service at manager1

```
$ docker-machine ssh manager1 "docker service create -p 80:80 --name web
    nginx:latest"
```

Then check manager1 service:

```
    $ docker-machine ssh manager1 "docker service ls"
```



```
andhika@user-pc:~/docker$ docker-machine ssh manager1 "docker service ls"
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
vuv5ip4h3w9k        web                 replicated          1/1                 nginx:latest        *:80->80/tcp
```

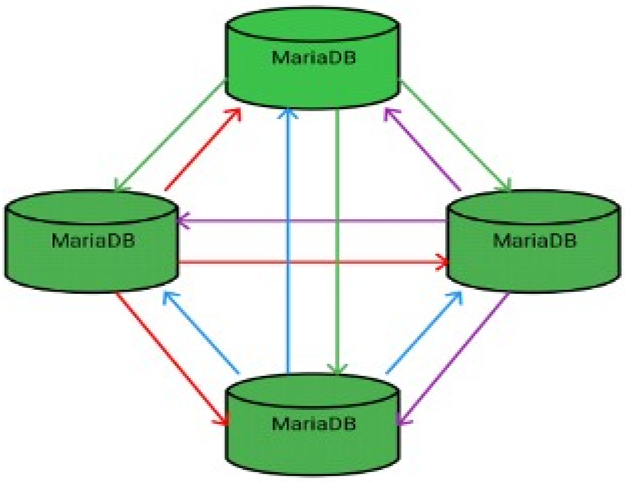
Check by open your IP manager1 at browser. (for default: http://192.168.99.100)

Note: You can remote internally using normal ssh by using, for modifying the Nginx

```
    $ docker-machine ssh [Node Name]
```

Expanding to another nodes is optional.

2. Create MariaDB galera at docker Swarm :



MariaDB Galera Cluster is a synchronous multi-master cluster for MariaDB.
Overview:https://mariadb.com/kb/en/library/what-is-mariadb-galera-cluster/
Features:

- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Direct client connections, native MariaDB look & feel

Benefits DBMS clustering:

- No slave lag
- No lost transactions
- Both read and write scalability
- Smaller client latencies

Step: https://mariadb.com/kb/en/library/getting-started-with-mariadb-galera-cluster/
Docker Hub: https://hub.docker.com/r/colinmollenhour/mariadb-galera-swarm

2.1. Setup galera node

```
$ docker-machine ssh1 manager1       # enter ssh at manager1
docker@manager1 $
```

```
docker pull colinmollenhour/mariadb-galera-swarm
wget https://raw.githubusercontent.com/colinmollenhour/mariadb-galera-
    swarm/master/examples/swarm/docker-compose.yml
mkdir -p .secrets
openssl rand -base64 32 > .secrets/xtrabackup_password
openssl rand -base64 32 > .secrets/mysql_password
openssl rand -base64 32 > .secrets/mysql_root_password
docker stack deploy -c docker-compose.yml galera
```

```
# (wait for `galera_seed` to be healthy)
$ docker service ls
# (wait for both `galera_node` instances to be healthy)
$ docker service scale galera_seed=0
$ docker service scale galera_node=2
    # Add 2 gallera node
Use galera_seed too add/remove seeder.
```

Notes:

Seed : Used only to initialize a new cluster and after initialization and other nodes are joined the "seed" container should be stopped and replaced with a "node" container using the same volume.

Node : Join an existing cluster. Takes as a second argument a comma-separated list of IPs or hostnames to resolve which are used to build the --wsrep_cluster_address option for joining a cluster.

In mariaDB galera Cluster, every node is seeder (has information to other nodes).

3. Setup Minio service:

Docker Hub:https://hub.docker.com/r/minio/minio

Minio is an object storage server released under Apache License v2.0. It is compatible with Amazon S3 cloud storage service. It is best suited for storing unstructured data such as photos, videos, log files, backups and container / VM images. Size of an object can range from a few KBs to a maximum of 5TB.

3.1. Setup minio-swarm server

Docs: https://docs.minio.io/docs/deploy-minio-on-docker-swarm

```
$ docker-machine ssh1 manager1       # ssh to manager1
At: docker@manager1 $
```

```
docker node update --label-add minio1=true worker1
docker node update --label-add minio2=true worker1
docker node update --label-add minio3=true worker2
docker node update --label-add minio4=true worker2
echo "AKIAIOSFODNN7EXAMPLE" | docker secret create access_key -
echo "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" | docker secret create
    secret_key -
wget
https://github.com/minio/minio/blob/master/docs/orchestration/docker-
swarm/docker-compose-secrets.yaml?raw=true
docker stack deploy --compose-file=docker-compose-secrets.yaml
    minio_stack
docker stack ls
```

Note: **docker note** can be used for configure allocation of which service will be assigned. Followed echo [key] is default key configured in this tutorial.

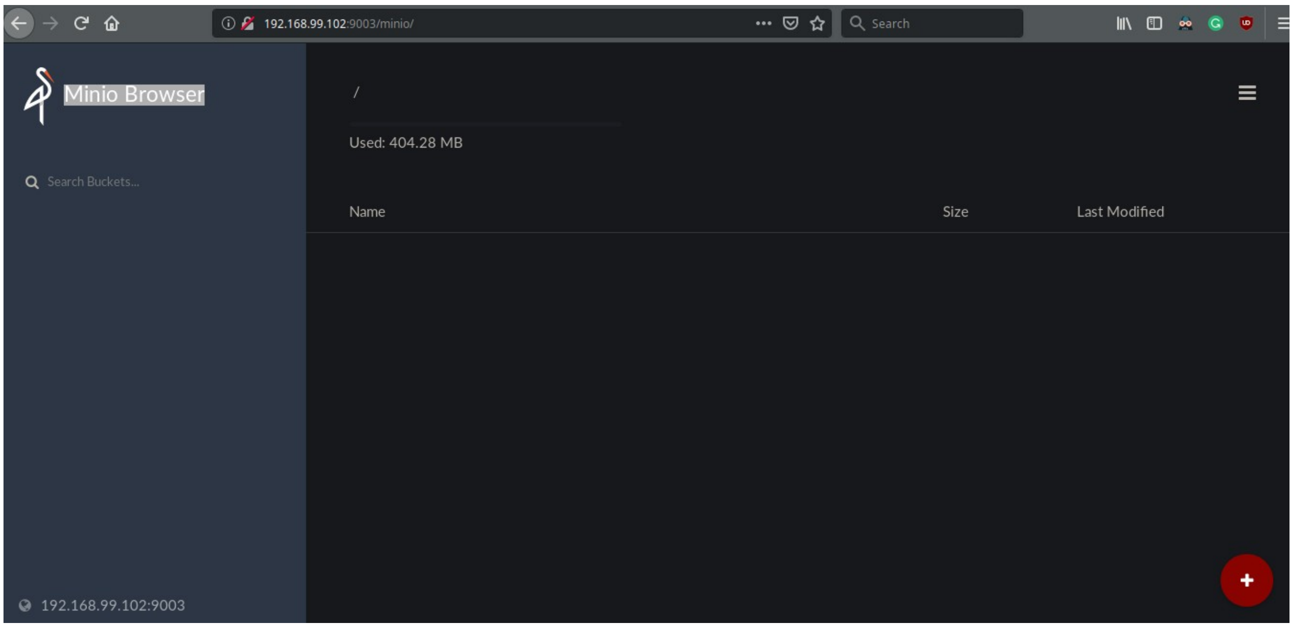
3.2 Open minio browser

Open [worker1 IP]:[port assigned]
in this example, the following link can access the minio:

- http://192.168.99.101:9001 # using worker1 trough stack minio1
- http://192.168.99.101:9002 # using worker1 trough stack minio2
- http://192.168.99.102:9003 # using worker2 trough stack minio3
- http://192.168.99.103:9004 # using worker2 trough stack minio4

Enter:

access_key : `AKIAIOSFODNN7EXAMPLE`
secret_key : `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`



Data protection

Distributed Minio provides protection against multiple node/drive failures and bit rot using erasure code. As the minimum disks required for distributed Minio is 4 (same as minimum disks required for erasure coding), erasure code automatically kicks in as you launch distributed Minio.

Important command

```
$ docker node ls          # Swarm node status
$ docker-machine ls       # List all docker swarm (manager and worker)
$ docker-machine ssh [Node Name] “[command]”
    #Execute [command] inside [Node Name]
$ docker service ls       # List active service
$ docker ps               # List running container
$ docker ps -a            # List all container
$ docker stack ls         # List SWARM service / stack
$ docker node ps [node]   # List serivce in selected node
$ docker secret --help
```

Other References

https://rominirani.com/docker-swarm-tutorial-b67470cf8872

https://medium.com/@Grigorkh/docker-swarm-tutorial-c5d5cf4b4de

https://semaphoreci.com/community/tutorials/running-applications-on-a-docker-swarm-mode-cluster

https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/

https://blog.alexellis.io/openfaas-storage-for-your-functions/

https://18pct.com/zero-to-mariadb-cluster-in-docker-swarm-in-15-minutes-part-1/

https://sysdig.com/blog/monitor-docker-swarm/

https://www.slideshare.net/MariaDB/getting-started-with-mariadb-with-docker-84370473