Swarm Introduction

06 May 2025 21:17

Problem :-

① There are too many manual steps for zero-downtime deployment in docker compose.

②
There is possibility of single point failure, because the docker compose run in a single device, so if this device fails everything will stop.

③ Docker compose doesn't provide rolling updates, load balancing. This is too complex to implement with docker compose.

Solution :-

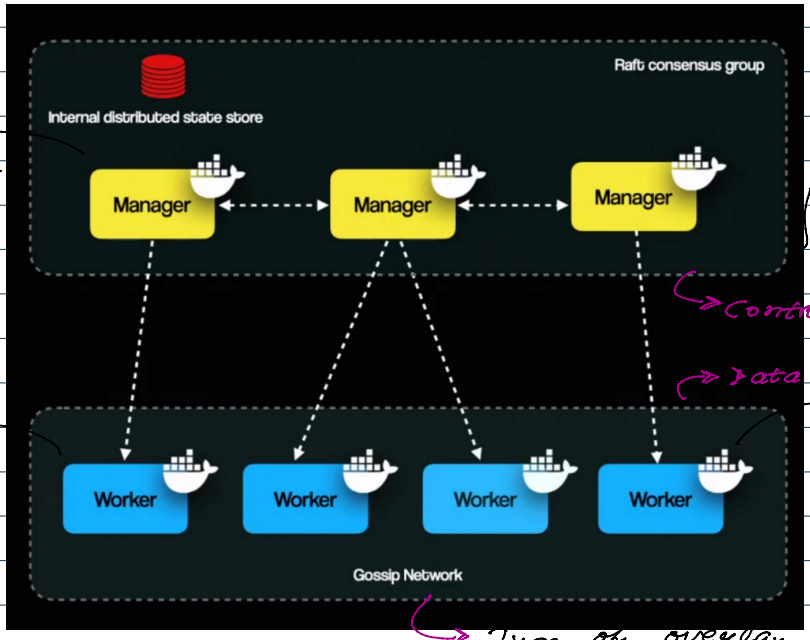
→ Docker swarm
→ it orchestrate (manage) too many containers, there load balancing, rolling updates scaling.

→ it avoids the single point of failure.

✱ Docker swarm is for horizontal scaling
load balancing
avoiding single time of failure.

✱ It manges very smoothly.

✱ It creates cluster of multiple machines (Docker engine)
✱ Native feature of Docker.



(Node) Machine which have Docker Engine

All three machine are connected.

Control plane

Data plane → Docker Engine

Machine (Node)

Type of overlay n/w

↳ Type of overlay n/w

- If we have to deploy, then we say to manager, further manager sends command to worker & the task is done.
- Multiple managers resolve "single time of failure" problem.
- Number of workers dependens on the application popularity (no. of request). It is for the "horizontal scaling."
- Docker sworm also handles the load and equally distribute the request to different workers.
- It also regularly checks the health of the worker, if some goes docon it spin other to maintain requirement.
- Managers all synced & they interally share data with each other.
- Docker sworm is for small to medium level application. for enterprise level we should use "kubernets."

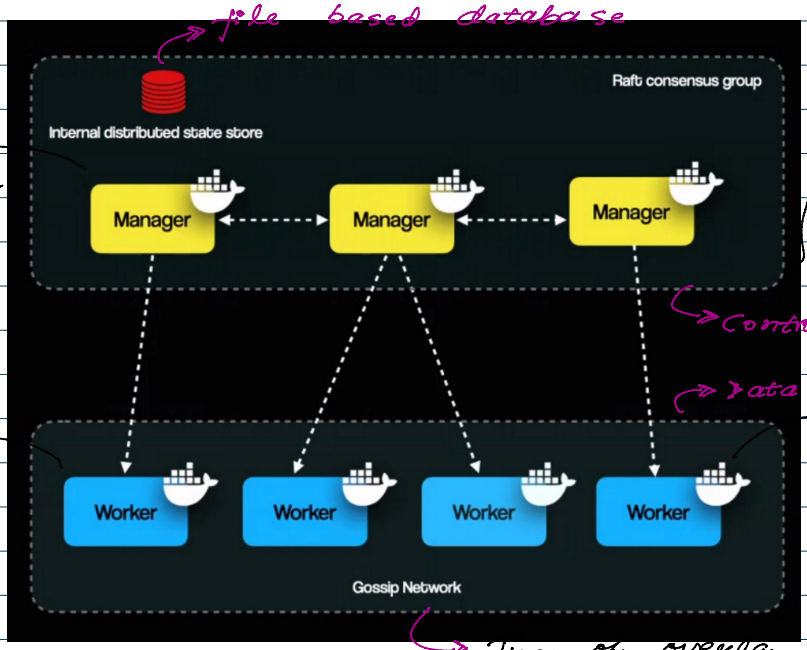
Architecture and Voting

12 May 2025 14:06

Sworm is so much powerful to orchestrate large no. of containers. It manage containers in multiple machines.

It works in less resource.

Architecture of Docker sworm :-



(Node) Machine which have Docker Engine

All these machin are connected.

Control plane

Data plane

Docker Engine

Machine (Node)

Type of overlay n/w

Control plane :- It has mangers which controls containers. Every manage is a seperate machine which has Docker engine.

Data plane :- It has real containers. Every container is in a machine.

{In this image we have total 7 machines}

** All these 7 machines (Nodes) participate in deployment}

Manager :- Manager not just order to create or run containers, if needed, then it also runs containers on these machine.

Raft consensus group

Algorithm — Rule

↳ it chooses the leader in docker plane

↳ it chooses the leader in docker plane among all managers.
→ Leader gives order.
→ Other than leader, rest are followers.
→ followers regularly sync data with leader so that in case of leader failure new leader continue the work.

How to choose leader :-

① As manager gets online, they set a "random" timer. (Range : 150 to 300 ms)

② Now manger will send signal to leader, if he get's feedback within time interval then nothing will change.
↳ But if he didn't get the response then
→ He thinks leader crashed
→ So, he will initiate a election.
→ increment term number of election.
→ He elects him self as a candidate
→ He votes himself
→ Now, he sends request to other up manager, and every manger will vote if the machine up.
↳ (who came first)
→ Now when Quorum reached, manager change his state to "leader" from candidate
→ Now it takeovers. & lead.

{Quorum → minimum number of voter required}
↳ Not just used while electing the leader.
→ Also used when a leader is commiting the info to database.
↳ It is necessary to sync data between followers
↳ it avoid the "split brain" conflict.
{one brain different data}

※ This "Raft" algorithm is example of debounce with random time.

Q. How is the first leader decided ?

↳ The manager who has the least number of timer, he will elect the election and send vote req. to other & get elected as leader.

→ The timer is randomize so there is no conflict.

Q. what happend when a leader again start ?

→ He will send req & get the election term which is updated, so now he will be a follower.

Q. why 3 managers are ideal ? {we should always keep in odd number}

N = 1 :- If there is only 1 manager, then the system will be not fault tolerant.

N = 2 :- This is even, so there can be tie while voting. That's why we don't keep even number.

N = 3 :- This is odd number, so no conflict while voting & system failure tolerance.

(N) No of managers	(N - Quorum) failure tolerance	((N/2) + 1) Quorum
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	2	4

** → failure tolerance :- No. of system crash we can tolerate.

compose & Dockerfile setup

12 May 2025 23:58

Docker is for production, not for development.
So, we need images & we will run containers on different machines.

Changes required in Docker file :-

↳ ① Depends on :
↳ it will not work in swarm, because of multiple machine.

Solution :-
we have to move this wait logic in our application.

→ ① we need a wait-for-it.sh file. ↳ popular git repo
② it takes the server IP (name) : port
③ Copy it in /usr/local/bin
④ Give full access to execute.
⑤ Run this first before any command
↳ as entrypoint

✱ Now we have to build image & push it to docker hub.

build :- docker build --platform linux/amd64 -t <username>/blog_app:v1
↳ image is build for this platform. it is same where you will deploy this.

login :- docker login -u <username>

push :- docker push <username>/blog_app:v1

✱ Now, we have to do some changes on compose.yaml file.
Remove name property.

① image :- we will our pushed image name.

② Container name :- Swarm decides it, so we don't need to give it.

③ Add this code :-

deploy :
  replicas : 1 { No of containers, we want to run }
  update_config :
    order : start-first { New container run, if it is healthy then prev container stops }
    { By default : stop-first }

  failure_action : rollback { run prev version }
  delay : 10s { switch time delay }

  rollback_config :
    parallelism : 1 { At one time rollback to only 1 container }

    order : start-first ~~rest~~ -
    ~~her wil a~~

healthcheck :
  test : command to test health of your application
  interval : 10s { run on every 10s }
  timeout : 5s { self explanatory }
  retries : 3

```
test :
    [
        'CMD',
        'wget',
        '--no-verbose',
        '--tries=1',
        '--spider',
        'http://localhost:3000/health',
    ]
```

→ test command
→ using wget to test
→ location where our test code is written.

This is for mysql database :-

```
test :
    [
        'CMD',
        'mysqladmin',
        'ping',
        '-h',
        'localhost'
    ]
```

Also we have to provide environment variables :-

environment :
  - MYSQL_ROOT_PASSWORD : root { This is not a good practice }

④ Changes for database containers :-

→ Bind mount will not work due to several machine
→ we should keep one replica of database container & reference other container to it
→ we had already seen health test command for mysql DB.
→ volumes is necessary for database data persistency.

Docker swarm implementation

13 May 2025 13:50

Docker should be installed on all machines.

① Init docker swarm

↳ docker swarm init

→ This will make the machine manager

→ Give you a code to add worker. We have to run this code from worker node

{To make this work you have to enable TCP port 2377.}

you also have to open these ports :-

1. TCP 3001 :- To expose our application
2. UDP 4789 :- for overlay connection
3. TCP & UDP 7946 :- for node discovery

② Run the given command to machines you want to join.

Command to get the join code :-

↳ docker swarm join-token worker

Here you can also provide manager if you want to join some node as manager.

Run this code on swarm initialize machine.

docker info → This command can be used to see the info of manager, workers.

There will be two n/w created = {in all machine}

→ bridge & overlay type

docker node ls → used to watch the nodes status.

inside the manager create a folder & make a compose.yaml file & paste the compose file code.

Go to that folder.

code.

Go to that folder.

(**) when we deploy multiple services, those are known as stack.

(*) Command to create stack & run services :-

↳ docker stack deploy -c compose.yaml blogstack
(stack name)

↳ docker stack ls → list all the stacks. And also number of services in the stack

↳ docker service ls → To list all services details.

↳ docker service ps <service_name>
↳ To check in which node service is deployed.

Now you know, where is the service running. So goto that machine & do whatever you want.

we can also add property that where we want to deploy our service.

↳ inside compose :

deploy :
placement :
constraints :
- node.role = worker
- node.label.storage == fast

Also need to give label to nodes.

After this deployment one more n/w will be created of type overlay.

↳ This is the same n/w creates in development mode to connect containers.

but this time it is overlay type (not bridge) because smooth connection between machines

Due to this n/w we can access the service from any node.

↳ Also there is proxy works using ingress network.

This is a hidden n/w run by dockers.

→ This proxy uses VIP n/w & tunneling using VXLAN to pass the packets & receive.

Zero downtime

13 May 2025 14:56

To make our application scalable :-

(i) Add number of replica you want

(ii) Do dynamically by running a command.

command to run multiple service :-

↳ docker service scale <service_name / id> = 10

no. of services you want to run.

↳ docker service ps <service_name>

↳ To list all the services running at which node.

Dynamically the services are devided by swarm.

To reduce run same code with less number.