MENU ☰

OCTOBER 7, 2019

Docker Swarm Mode

Docker Swarm

- The cluster maanagement & Orchestration features are embedded inside Docker Engine.
- Docker swarm consists of multiple docker hosts which run in swarm mode.
- Two Roles **managers** and **workers** exist in Docker swarm
- **Manager** is responsible for membership & delegation
- **Worker** is responsible for running swarm services
- Each Docker Host can be a manager, a worker or both.
- In Docker Swarm **Desired State** is maintained. For instance if you are running one container in swarm on a particular node (Worker) and that node goes down, then Swarm schedules this nodes task on other node to maintain the state.
- **Task** is a running container which is part of swarm service managed by Swarm Manager

Nodes

- It is instance of the docker engine participating in Swarm.
- There are two kinds of nodes
  - Manager nodes:
    - You communicate to manager node to deploy applications in the form of Service Definitions.
    - Manager nodes dispatch unit of work called as tasks to the Worker ndoes
  - Worker nodes:
    - They receive & execute the tasks dispatched from manager nodes.
    - An agent runs on the worker node & reports on the tasks assigned to it

Services and tasks

- Service is the definition of the task to be executed.
- Typically it would be the application to be deployed.
- Two kinds of Service models are available
  - **Replicated Services model**: In this case swarm manager distributes a specific number of replica task among the nodes based upon the scale you set in the desired state
  - **Global Services Model**: In this case swarm runs one task for the service on every available node in the cluster.
- **Task**
  - carries a Docker container and the commands to run inside the container.
  - It is the atomic secheduling unit of swarm.
  - Once a task is assigned to node, it cannot move to another node.
  - It can only run on the assigned node or fail.

Swarm Setup

- In this series, I would be using 3 ubuntu 18 machines.
- One would be manager & other two would be workers.
- Install docker on all the machines by following instructions from here
- Login into ssh session of the machie which would be manager.
- Ensure all the machines can be communicated (or pingable from manager)
- Make a note of private ip address of the manager (In this example the managers ip address would be 172.31.42.125) and then exec

```
docker swarm init --advertise-addr <Manager-ip>

# In my case this is
docker swarm init --advertise-addr 172.31.42.125
```

Book Now

*Terms & conditions apply

- Execute `docker info` on the manager and observe the output should consists of Swarm: active and other info about Docker Swarm.
- Execute command `docker node ls` and you should see the status of the manager node
- Now login into other nodes and execute docker swarm join command which is output of the docker swarm init command as mentioned above.

```
docker swarm join --token SWMTKN-1-1w51ouq6zrmts85l71z53ruqcc1pivzprpigdodspu58o7dp3z-172

##Output##
This node joined a swarm as a worker.
```

- Now ssh into the manager and execute `docker node ls` and you should be able to see three nodes information.

- Lets create a tomcat service by using the following command

```
docker service create --replicas 2 --name tomcat tomcat:8
```

- This command leads to creation of tasks and output would be like

```
tvdml6nt5dryszozaydr8sv8o

overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]

2/2: running   [==================================================>]
verify: Service converged
```

- Execute `docker service ls` and the possible output would look like

```
ID                  NAME                MODE                REPLICAS            IMAGE
tvdml6nt5dry        tomcat              replicated          2/2                 tomcat:8
```

- Lets inspect the service using `docker service inspect --pretty tomcat` and the output of the command would be

```
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         tomcat:8@sha256:bb4ceffaf5aa2eba6c3ee0db46d863c8b23b263cb547dec0942e75
 Init:          false
Resources:
Endpoint Mode:  vip
```

- Execute `docker service ps tomcat` command to findout on which node the tasks are executed.
- Lets scale the number of containers running tomcat by using the following command `docker service scale tomcat=4` and the output would be

```
tomcat scaled to 4

overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged
```

- Service can be deleted using the follwing command `docker service rm tomcat`

Rolling updates to docker swarm

- Now navigate to ipaddress of any node and http://<nodip&gt;:8081
- Now lets try to update to the newer version of jenkins

```
docker service update --image jenkins:latest jenkins
```

Relevance to Docker Networking

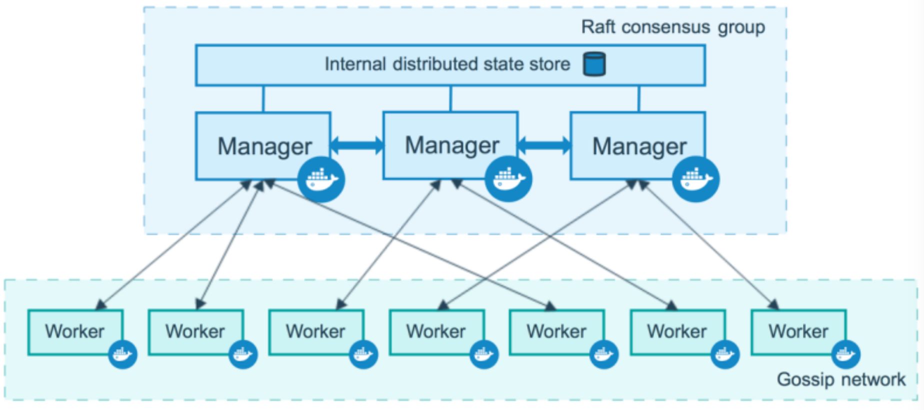
- In this series so far we were able to run docker containers on different nodes from swarm manager.
- Now execute this command `docker network ls` on the manager and you should see the output which would look like

```
NETWORK ID          NAME                DRIVER              SCOPE
4fa602b7a4ec        bridge              bridge              local
c3ba8230b575        docker_gwbridge     bridge              local
436956ab2dd0        host                host                local
m6zbhh5cn5ag        ingress             overlay             swarm
dfa793d07248        none                null                local
```

- In this overlay network driver is used and scope for that driver is swarm.
- So we can conclude that Docker swarm uses overlay and bridge (docker_gwbridge) to enable multiple Docker Host Communications.

How Docker Swarm Works?

- Docker swarm uses **RAFT** Consensus Algorithm to maintain a consistent internal state of the entire swarm and all the services running on it.



Manager Nodes

- Manager nodes handle cluster management tasks
  - cluster state management
  - service scheduling
  - serving Swarm mode

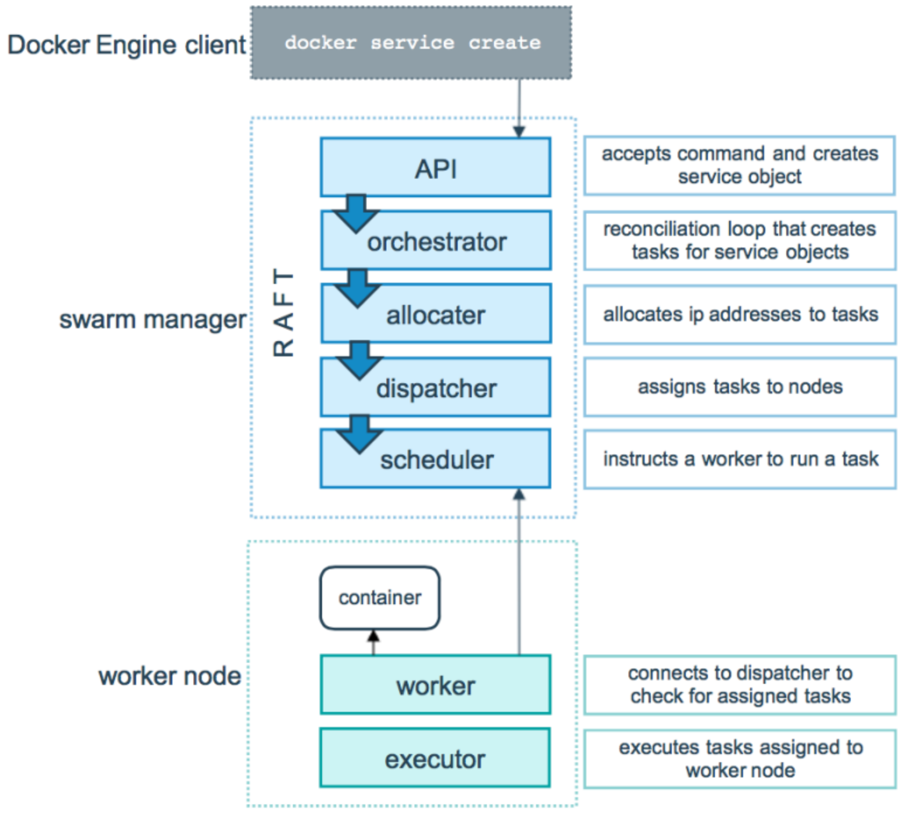
To take advantage of swarm mode's fault-tolerance features, Docker recommends you implement an odd number of nodes according to your organization's high-availability requirements. When you have multiple managers you can recover from the failure of a manager node without downtime.

```
* A three-manager swarm tolerates a maximum loss of one manager.
* A five-manager swarm tolerates a maximum simultaneous loss of two manager nodes.
* An N manager cluster tolerates the loss of at most (N-1)/2 managers.
* Docker recommends a maximum of seven manager nodes for a swarm.
```

Worker Nodes

- Instance of Docker Engine whose purpose is to execute containers
- They dont participate in Raft distributed state or any of the managers tasks.
- Worker Node can be made Manager node by using `docker node promote`. note: This has to be executed by Manager Node

Docker Service LifeCycle


Docker Engine client
docker service create
swarm manager
RAFT
API
orchestrator
allocater
dispatcher
scheduler
accepts command and creates service object
reconciliation loop that creates tasks for service objects
allocates ip addresses to tasks
assigns tasks to nodes
instructs a worker to run a task
worker node
container
worker
executor
connects to dispatcher to check for assigned tasks
executes tasks assigned to worker node

External Access For Docker Services

Ingress Mode Service Publishing

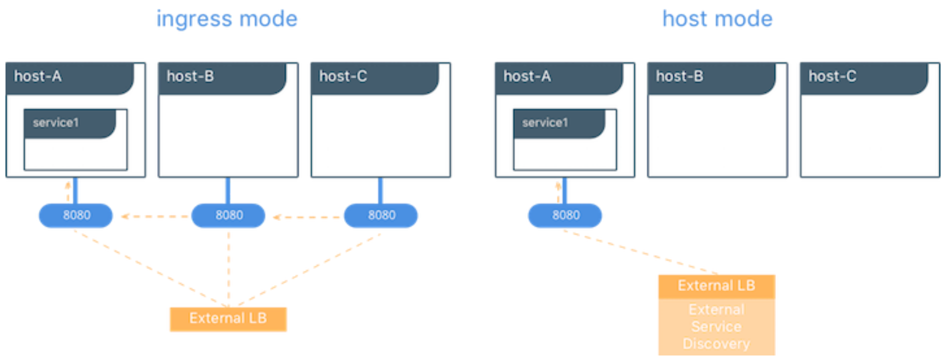
```
docker service create --replicas 2 --publish mode=ingress,target=80,published=8080 nginx
```

- This mode publishes the exposed port on every Swarm node.
- Load balancing happens in this mode

Host Mode Service Publishing

```
docker service create --replicas 2 --publish mode=host,target=80,published=8080 nginx
```

- In this mode the published port is exposed on the host where this service is running
- Load balancing doesn't happen



3 thoughts on "Docker Swarm Mode"

Ahammad Shaik says:
October 17, 2019 at 3:56 pm

Pingback: Docker Classroom Series 09/Oct/2019 – Direct DevOps from QualityThought

Pingback: Docker Networking Series – II Overlay Networks – Direct DevOps from QualityThought

Leave a Reply

Enter your comment here...

About continuous learner

devops & cloud enthusiastic learner

VIEW ALL POSTS

PREVIOUS POST

Docker Networking Series - I

NEXT POST

Docker Networking Series - II Overlay Networks