Swarm !

Πίνακας περιεχομένων

1. Swarm - Intro

Docker Swarm is a **clustering** and **scheduling** tool for Docker containers. With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.

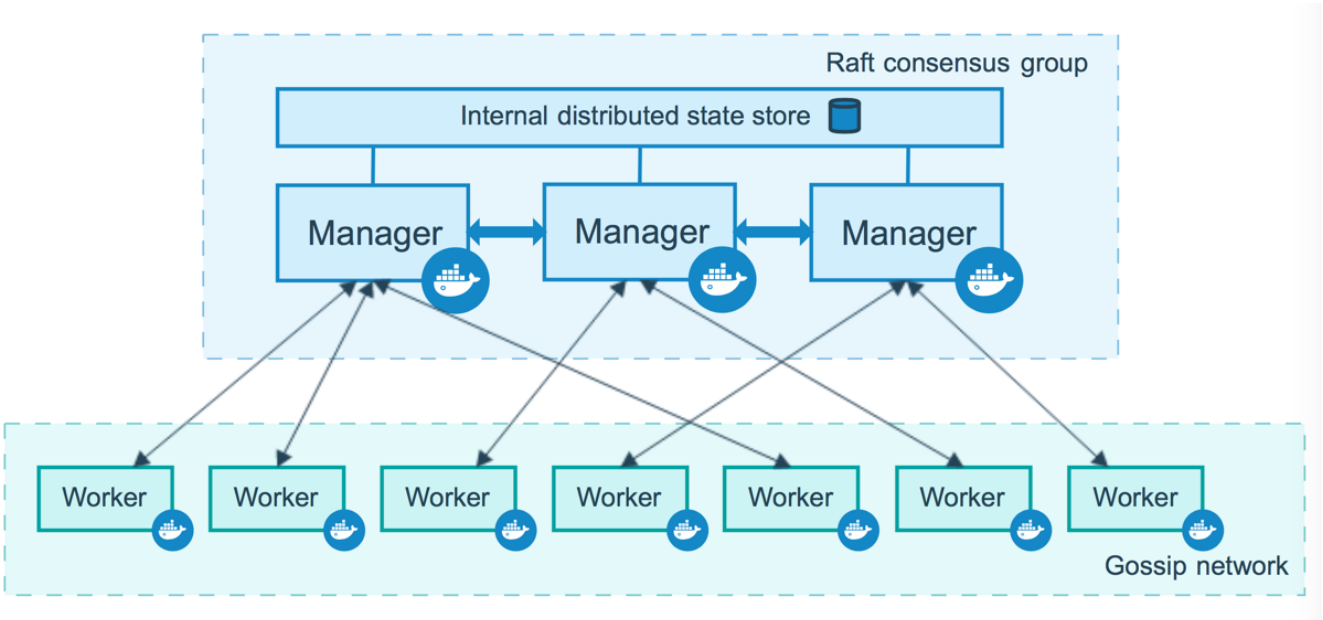
Clustering is an important feature for container technology, because it creates a cooperative group of systems that can provide redundancy, enabling Docker Swarm failover if one or more nodes experience an outage.

Features:

- Decentralized design: Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine.
- Scaling: For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

- 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. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager creates two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
- Multi-host networking: You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
- Service discovery: Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.
- 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.
- Secure by default: Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
- Rolling updates: At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll back to a previous version of the service.

2. Swarm architecture



There are two types of nodes: **managers** and **workers.**

2.1. 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.

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.

2.2. 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.

3. Create swarm

Initialize a swarm. The docker engine targeted by this command becomes a manager in the newly created single-node swarm.

create swarm

```
docker swarm init --advertise-addr ip
Swarm initialized: current node (bvz81updecsj6wjz393c09vti) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
1awxwuwd3z9j1z3puu7rcgdbx \
    172.17.0.2:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

3.1. Join token

Join tokens are secrets that allow a node to join the swarm.

There are two different join tokens available,

- one for the worker role and
- one for the manager role.

join token

```
docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
1awxwuwd3z9j1z3puu7rcgdbx \
    172.17.0.2:2377

docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
7p73s1dx5in4tatdymyhg9hu2 \
    172.17.0.2:2377
```

3.2. Join swarm

Join a node to a swarm. The node joins as a manager node or worker node based upon the token you pass with the --token flag. If you pass a manager token, the node joins as a manager. If you pass a worker token, the node joins as a worker.

join worker

```
docker swarm join --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
1awxwuwd3z9j1z3puu7rcgdbx --advertise-addr eth1:2377   192.168.99.121:2377
This node joined a swarm as a worker.
```

join manager

```
docker swarm join --token SWMTKN-1-3pu6hszjas19xyp7ghgosyx9k8atbfcr8p2is99znpy26u2lkl-
7p73s1dx5in4tatdymyhg9hu2  --advertise-addr eth1:2377  192.168.99.121:2377
This node joined a swarm as a manager.
```

3.3. Leave swarm

When you run this command on a worker, that worker leaves the swarm.

```
docker swarm leave
```

You can use the **--force** option on a manager to remove it from the swarm.

INFO

The safe way to remove a manager from a swarm is to **demote** it to a worker and then direct it to leave the quorum without using --force

```
docker node demote id
```

4. Manage nodes

INFO

This is a cluster management command, and must be executed on a swarm manager node.

- **docker node ls** List nodes in the swarm

```
docker node ls
```

- **docker node demote** _ Demote one or more nodes from manager in the swarm_
- **docker node inspect** _ Display detailed information on one or more nodes_
- **docker node promote** _ Promote one or more nodes to manager in the swarm_
- **docker node ps** _ List tasks running on one or more nodes, defaults to current node_
- **docker node rm** _ Remove one or more nodes from the swarm_
- **docker node update** _ Update a node_
- **docker node inspect** _ Display detailed information on one or more nodes_

```
docker node inspect id   # from ls
```

```
[
  {
    "ID": "e216jshn25ckzbvmwlnh5jr3g",
    "Version": {
      "Index": 10
    },
    "CreatedAt": "2017-05-16T22:52:44.9910662Z",
    "UpdatedAt": "2017-05-16T22:52:45.230878043Z",
    "Spec": {
      "Role": "manager",
      "Availability": "active"
    },
    "Description": {
      "Hostname": "swarm-manager",
      "Platform": {
        "Architecture": "x86_64",
        "OS": "linux"
      },
      "Resources": {
        "NanoCPUs": 1000000000,
        "MemoryBytes": 1039843328
      },
      "Engine": {
        "EngineVersion": "17.06.0-ce",
        "Plugins": [
          {
            "Type": "Volume",
    },
    "Status": {
      "State": "ready",
      "Addr": "168.0.32.137"
    },
    "ManagerStatus": {
      "Leader": true,
      "Reachability": "reachable",
      "Addr": "168.0.32.137:2377"
    }
  }
]
```


INFO

Filtering output with jq

jq

```
apt update
apt install jq
docker node inspect id | jq -r '.[]["Status"]["State"]'
```

5. Deploy services and Tasks

To deploy an application image when Docker Engine is in swarm mode, you **create a service.**

A **service** is the image for a microservice within the context of some larger application.

Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.

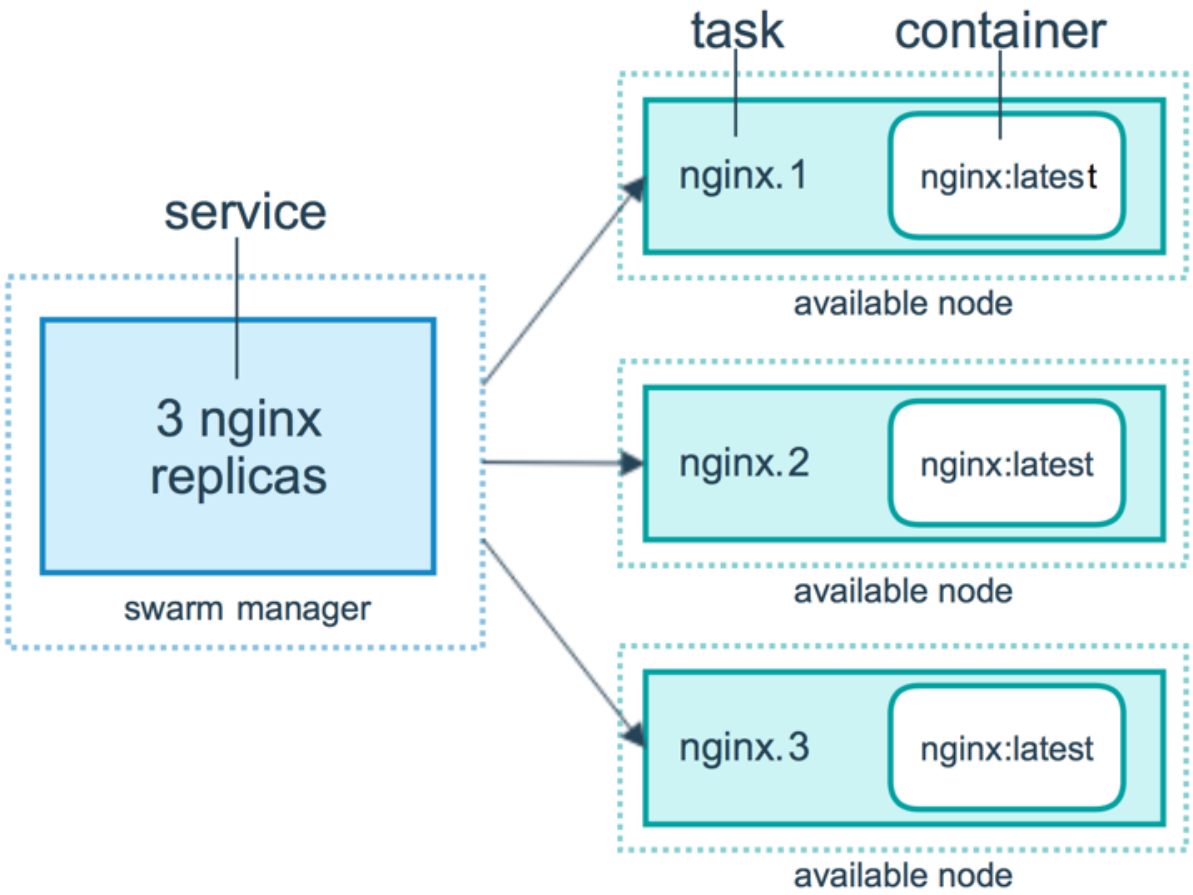
A **task** is the atomic unit of scheduling within a swarm.

When you declare a desired service state by creating or updating a service, the orchestrator realizes the desired state by scheduling tasks.

For instance, you define a service that instructs the orchestrator to keep three instances of an HTTP listener running at all times.

The orchestrator responds by creating three tasks.

Each task is a slot that the scheduler fills by spawning a container.



5.1. service vs stack

- A **Service** defines one or more instances of a single image deployed on one or more machines (described by one entry in the services part of **yaml** files).
- A **Stack** defines a group of heterogeneous services (described by the whole yaml file).

The **docker service** command is used when managing individual service on a docker swarm cluster.

The **docker stack** command can be used to manage a multi-service application.

5.2. Build

INFO

This is a cluster management command, and must be executed on a swarm manager node.

5.2.1. Create yaml

yml example MPI (save as run.yml)

```
version: "3.8"

services:

  master:
    image: image
    user: root
    environment:
     # Pass environment variables to containers   CUSTOM
     - PASSWORD=dgergergergerrfgwehrtsger
     - PASSWORDVIEW=rtyrwtyrwftertgueteyserfy5e6ytrg
     - SERVERROLE=master
     - SERVERWEB=no
     # Pass environment variables to containers  FROM ENGINE see inspect
     - NODENAME={{.Node.Hostname}}
     - NODEID={{.Node.ID}}
     - SERVICEID={{.Service.ID}}
     - SERVICENAME={{.Service.Name}}
     - TASKID={{.Task.ID}}
     - TASKNAME={{.Task.Name}}
     - TASKREPID={{.Task.Slot}}
    deploy:
      # mode: global # see image  replica-vs-global
      replicas: 1
      placement:
        max_replicas_per_node: 1
        constraints:
          - node.role == worker
          #- node.labels.region == region1
      resources:
        limits:
          cpus: '0.50'
          memory: 500M
        reservations:
```

```
          cpus: '0.25'
          memory: 200M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 5
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    networks:
      mpi2-net:
    ports:
      - "5580:80"
      - "5588:8088"

  slave:
    image: image
    user: root
    environment:
     - PASSWORD=rtyrthrthyrthyrtyrtyrty
     - PASSWORDVIEW=rtyrtuyrtuyrt
     - SERVERROLE=slave
     - SERVERWEB=no
     - NODENAME={{.Node.Hostname}}
     - NODEID={{.Node.ID}}
     - SERVICEID={{.Service.ID}}
     - SERVICENAME={{.Service.Name}}
     - TASKID={{.Task.ID}}
     - TASKNAME={{.Task.Name}}
     - TASKREPID={{.Task.Slot}}
    deploy:
      # mode: global # see image  replica-vs-global
      replicas: 9
      placement:
        max_replicas_per_node: 1
        constraints:
          - node.role == worker
          - node.labels.region == region2
      resources:
        limits:
          cpus: '0.50'
          memory: 500M
        reservations:
          cpus: '0.25'
          memory: 200M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 5
```

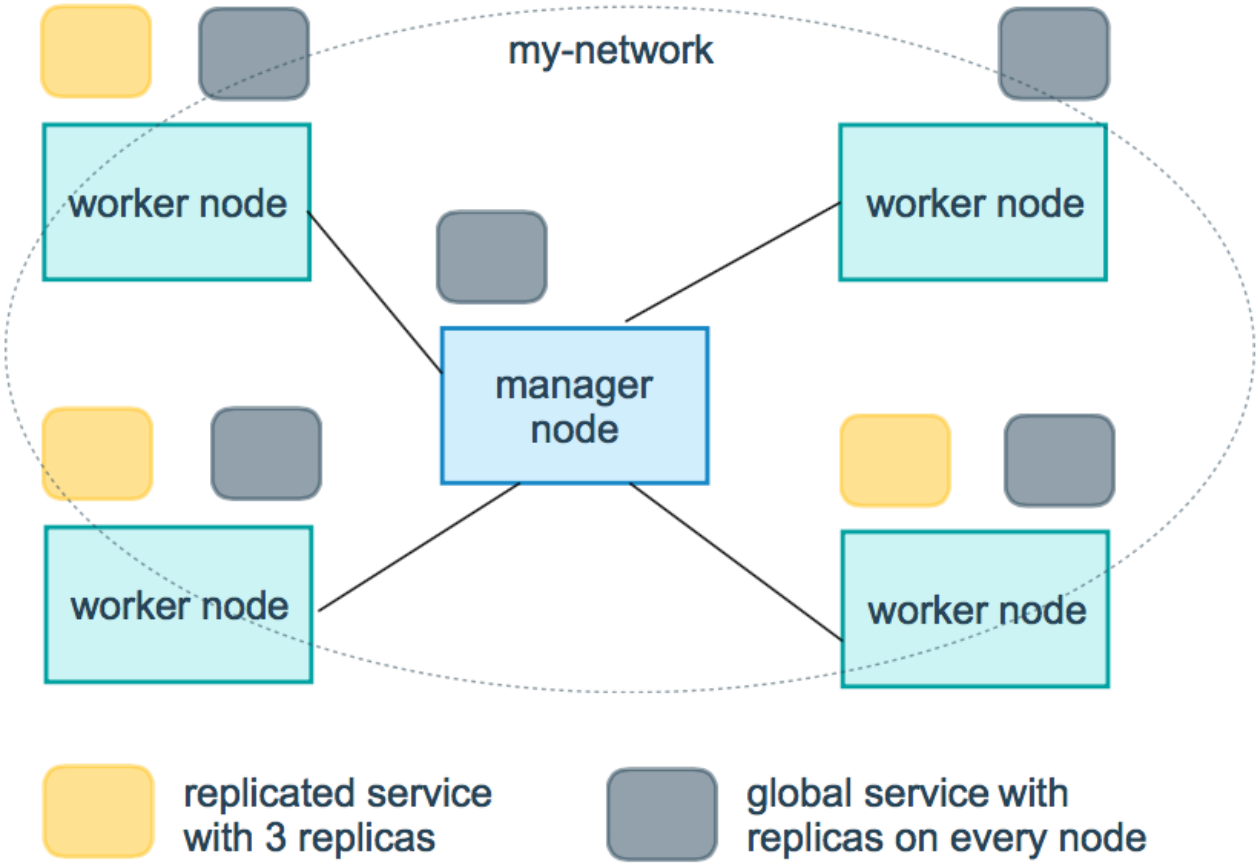
```
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    networks:
      mpi2-net:
    ports:
      - "5581:80"
      - "5590:8088"


  web:
    image: image
    user: root
    environment:
     - SERVERROLE=web
     - SERVERWEB=yes
     - NODENAME={{.Node.Hostname}}
     - NODEID={{.Node.ID}}
     - SERVICEID={{.Service.ID}}
     - SERVICENAME={{.Service.Name}}
     - TASKID={{.Task.ID}}
     - TASKNAME={{.Task.Name}}
     - TASKREPID={{.Task.Slot}}
    deploy:
      # mode: global # see image  replica-vs-global
      replicas: 1
      placement:
        constraints:
          - node.role == worker
          - node.labels.region == region3
      resources:
        limits:
          cpus: '0.50'
          memory: 500M
        reservations:
          cpus: '0.25'
          memory: 200M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 5
        window: 120s
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    networks:
      mpi2-net:
    ports:
```

```
      - "5598:80"
      - "5599:8088"


networks:
  mpi2-net:
```

replicas-vs-global



INFO

YAML (a recursive acronym for "YAML Ain't Markup Language") is a human-readable data-serialization language.

It is commonly used for configuration files and in applications where data is being stored or transmitted.

YAML targets many of the same communications applications as Extensible Markup Language (XML) but has a minimal syntax which intentionally differs from SGML

It uses Python-style indentation to indicate nesting

From Wikipedia, the free encyclopedia

more: Learn YAML in five minutes! https://www.codeproject.com/Articles/1214409/Learn-YAML-in-five-minutes

5.2.2. Build

build

```
docker stack deploy -c run.yml my_service
```

5.2.3. List services

List the services that are running as part of the specified stack.

```
docker stack services
```

List **ALL** *services are running in the swarm.*

```
docker service ls
```

5.2.4. Remove one or more stacks

Remove the stack from the swarm.

```
docker stack rm
```

Removes the specified services from the swarm.

```
docker service rm
```

5.2.5. List tasks

List the tasks that are running as part of the specified services.

```
docker service ps
```

List the tasks in the stack

```
docker stack ps
```

INFO

Command-line completion (also tab completion) is a common feature of command-line interpreters, in which the program automatically fills in partially typed commands.

more info: https://en.wikipedia.org/wiki/Command-line_completion