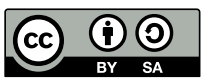
Cloud computing

Orchestration: swarm

Aleix Llusà Serra

Master's degree in Machine Learning and Cybersecurity for Internet Connected Systems
Universitat Politècnica de Catalunya
https://epsem.upc.edu

29 de setembre de 2026



This work is licensed under a Creative Commons "Attribution-ShareAlike 4.0 International" license.

Orchestration tools

The automated configuration, coordination, and management of computer systems and software.
The two most relevant:

- Kubernetes: the most popular
- Docker compose+swarm: already native into Docker

Docker swarm

- Cluster of docker hosts
- Orchestrator: docker stack

Swarm cluster

First manager: `docker swarm init`

- Manager: `docker swarm join-token manager`
- Worker: `docker swarm join-token worker`

Ports:

- 2377/tcp: secure swarm connection
- 7946/tcp+udp: control plane gossip
- 4789/udp: VXLAN overlay networks

Swarm cluster

List nodes:

```
docker node ls
```

Manager status:

- Leader: The actual manager. One Ring to rule them all
- Reachable: Manager
- Nothing: Worker

Only the leader can issue commands to the workers.

Raft consensus

- Raft Concensus Algorithm [https://raft.github.io/] [http://thesecretlivesofdata.com/raft/]
- Raft consensus in swarm mode [https://docs.docker.com/engine/swarm/raft/]
- The implementation of the consensus algorithm in swarm mode means it features the properties inherent to distributed systems, fault tolerant systems.

Swarm cluster command

Prevent working:

```
docker node update --availability drain NODENAME
```

Label nodes:

```
docker node update --label-add ‘‘l=manresa’’ NODENAME
```

Connect to remote docker:

```
DOCKER_HOST=ssh://user@example.test docker ps
```

Transfer local images to another node:

```
docker save IMATGE:latest | gzip |
DOCKER_HOST=ssh://user@example.test docker load
```

Swarm orchestrator

Docker stack service: high-level abstraction.

- Each stack is the set of services (micro) for an application/service (you may think a compose file).
- Each service (micro) defines how many replicas (containers) are needed. This is the desired state.
- Swarm observes and keeps into line the desired state. This is reconciliation.

Swarm orchestrator

- Compose file: now a Stack file
- And now we can publish ports regardless replication! They get published on all swarm nodes: ingress mode)

Let us deploy the last exercise `filemanager/compose.yml`:

```
docker stack deploy -c compose.yml STACKNAME
```

Swarm commands

List stacks: `docker stack ls`
List stack services: `docker stack services filemanager`
List stack replicas: `docker stack ps filemanager`

List all running services: `docker service ls`
List a services' replicas: `docker service ps filemanager_web`
List a services' logs: `docker service logs filemanager_web`

Restart a service:

```
docker service update --force filemanager_web
```

Scaling and rollout

Scaling (changing desired state):

```
docker service scale filemanager_web=20
```

Rollout (or rollback):

```
docker service update --image nginx:25 filemanager_web
```

Controlling scaled rollout:

```
docker service update --image nginx:26 --update-delay
30s --update-parallelism 2 nginx:27 filemanager_web
```

Recall: you can also define replicas or change the image tag in the compose file and then

```
docker stack deploy -c compose.yml STACKNAME
```

Compose

Recall: infrastructure as code and reproducibility. Work on the compose.

compose.yml

```
services:
  web:
    [...]
    deploy:
      replicas: 4
      update_config:
        parallelism: 2
        delay: 30s
        failure_action: rollback
    placement:
      constraints:
        - 'node.labels.l == manresa'
```

Docker Stack/Swarm TLDR

Yes! Finally! I can update deployments when I'm at the café.

Yes! Hot update. On live running services.

Yes! With zero downtime.