anoBytes | TECHNOLOGY SOLUTIONS

Docker Swarm cheat sheet

The commands to set up, run, and recover a Docker Swarm cluster, with what each one does and when to reach for it. Swarm mode is built into Docker Engine.

DOCKER ENGINE 29 · UBUNTU SERVER 26.04 LTS · SETUP · SERVICES · RECOVERY · TROUBLESHOOTING

General educational guidance, not affiliated with or endorsed by Docker, Inc. or Canonical. Try commands on a test cluster before production. The security items follow the CIS Docker Benchmark v1.8.0, section 7 (Docker Swarm). Published 2026-08-13.

HOW A SWARM FITS TOGETHER

A swarm is a group of Docker hosts acting as one. You declare the services you want; the managers keep the cluster matched to that declaration and restart or move containers when a node fails.

```
+- managers: keep an odd number ------------+
|  mgr-1 (leader)    mgr-2    mgr-3         |
+--------------------+--------------------+
                     | schedules tasks
+- workers -----------+--------------------+
|  wkr-1 [web.1] [api.1]     wkr-2 [web.2]  |
+-------------------------------------------+
client -> any node:80 -> routing mesh -> a web task
```

manager schedules work and stores the cluster state; changes need a majority of managers (Raft), which is why you run an odd number.

worker runs containers and holds no cluster state; lose one and its tasks restart elsewhere.

service the state you declare: image, replica count, ports, secrets, limits.

task one running container of a service. `web.2` is the second task of service `web`.

stack a group of services deployed together from one Compose file.

INSTALL AND OPEN PORTS, UBUNTU SERVER 26.04

Run the same steps on every node, managers and workers alike.

```
# add Docker's apt repository first:
# docs.docker.com/engine/install/ubuntu
sudo apt install docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
# open between the nodes only, never at a perimeter firewall
sudo ufw allow 2377/tcp  # cluster management (managers)
sudo ufw allow 7946      # node discovery, tcp and udp
sudo ufw allow 4789/udp  # overlay data (VXLAN, unauthenticated)
# encrypted overlay networks also need IP protocol 50 (ESP)
```

CREATE AND LOCK THE CLUSTER

One init on the first manager; every other node joins with a token. Treat the token like a password.

```
# on the first manager
docker swarm init --advertise-addr <manager-ip>
# harden: --listen-addr <ip>:2377 --data-path-addr <ip2>
# print the join command, token included, for each role
docker swarm join-token worker
docker swarm join-token manager
# on each new node, paste what it printed
docker swarm join --token <token> <manager-ip>:2377
# new token; the old one stops working
docker swarm join-token --rotate worker
```

Without autolock, the state's encryption keys sit readable on the manager's disk. With it, Docker asks for the unlock key after every restart.

```
docker swarm update --autolock=true  # save the key it prints
docker swarm unlock          # needed after every Docker restart
docker swarm unlock-key --rotate  # new key; update your copy
```

SERVICES

A service is the unit you operate. You create it once; scaling, upgrades, and rollbacks are edits to its declared state, and the managers roll every edit out task by task.

```
docker service create \
  --name web \      # the name every other command uses
  --replicas 3 \    # run three copies
  -p 80:80 \        # reachable on port 80 of every node
  nginx:stable
```

COMMAND	WHAT IT DOES
`docker service ls`	every service; 3/3 means all replicas run
`docker service ps web`	web's tasks, the node each runs on, and errors
`docker service logs -f web`	follow the logs of all web's tasks at once
`docker service inspect web --pretty`	the full declared state, readable
`docker service scale web=5`	change the replica count
`docker service rm web`	remove it, tasks and all

```
# rolling update: replace 2 tasks at a time, 10s apart
docker service update --image <image:tag> \
  --update-parallelism 2 --update-delay 10s web
# start each new task before stopping the old one
docker service update --update-order start-first web
# roll back automatically if an update fails
docker service update --update-failure-action rollback web
# put back the state before the last update
docker service update --rollback web
# recreate every task, config unchanged (rolling restart)
docker service update --force web
```

PLACEMENT AND LIMITS

Create and update flags that control where tasks land and what they may consume.

```
--mode global             # one task on every node (agents)
--constraint node.role==worker    # keep off the managers
--constraint node.labels.zone==a  # only nodes you labeled
--replicas-max-per-node 1  # spread copies across nodes
--reserve-memory 256M      # scheduler plans around this
--limit-memory 512M        # hard cap at runtime
```

SECRETS AND CONFIGS

A secret is a value the swarm stores encrypted and mounts as a file into only the services you grant it to; it stays out of the image, the environment, and the shell history. A config is the same mechanism for values that are not sensitive.

```
# from a file (or pipe stdin to: docker secret create db_pass -)
docker secret create db_pass ./db_pass.txt
# grant it; the task reads /run/secrets/db_pass
docker service create --secret db_pass ...
# swap a credential without rebuilding anything
docker service update --secret-add new_pass \
  --secret-rm db_pass web
docker config create app_conf ./app.conf
docker service create \
  --config source=app_conf,target=/etc/app.conf ...
```

STACKS: SERVICES FROM A COMPOSE FILE

You write service flags once, under `deploy:` in a Compose file, and deploy the whole application as a stack. Underneath, a stack is the same services, so every command above still works.

```
# compose.yml
services:
  web:
    image: registry.example.com/web:1.4
    ports: ["80:80"]
    secrets: [db_pass]
    deploy:
      replicas: 3
      update_config:
        parallelism: 2
        delay: 10s
        order: start-first
      restart_policy: { condition: on-failure }
      placement: { constraints: [node.role==worker] }
secrets:
  db_pass:
    external: true    # created with docker secret create
```

```
# create, and later update, the whole stack
docker stack deploy -c compose.yml app --with-registry-auth
docker stack services app  # its services, with replica counts
docker stack ps app        # every task in the stack
docker stack rm app
```

`docker stack deploy` does not build images: build, push to a registry the nodes can reach, then deploy. Re-run it after editing the file; `--prune` also removes services you deleted.

NETWORKS AND PUBLISHING

Services on the same overlay network reach each other by name: `web` resolves to a virtual IP that balances across web's tasks, on any node. Published ports go cluster-wide through the routing mesh: every node answers, then forwards to a task.

```
# app network; --attachable lets plain docker run join it
docker network create -d overlay --attachable appnet
# encrypt task-to-task traffic between nodes (ESP)
docker network create -d overlay --opt encrypted private
# publish on the task's own node only, skip the mesh
--publish published=8080,target=80,mode=host
# DNS round robin, for use behind your own load balancer
--endpoint-mode dnsrr
```

RETIRE A NODE, DISMANTLE THE CLUSTER

```
# retire a node: drain it first (Nodes), then
docker swarm leave            # on the leaving node
docker node rm <node>         # then on a manager
# dissolve: rm stacks; workers leave; demote+leave managers;
docker swarm leave --force    # last manager; close the ports
```

NODES

COMMAND	WHAT IT DOES
`docker node ls`	every node, its state, and which manager leads
`docker node inspect <node> --pretty`	labels, resources, and engine version, readable
`docker node promote <node>`	make a worker a manager
`docker node demote <node>`	make a manager a worker; always demote before removing
`docker node update --label-add zone=a <node>`	tag a node, for placement constraints
`docker node rm <node>`	remove a node that left or failed

```
# node maintenance, in this order
docker node update --availability drain <node>
#   its service tasks move to other nodes; patch and reboot
docker node update --availability active <node>
# pause: the node keeps its tasks but takes no new ones
docker node update --availability pause <node>
```

OPERATE AND RECOVER

The cluster state (services, secrets, certificates, membership) lives in `/var/lib/docker/swarm` on the managers. Back it up from one; without it a dead cluster cannot be rebuilt.

```
# back up on a manager; the others keep quorum meanwhile
sudo systemctl stop docker.socket docker
sudo tar -czf swarm-backup.tgz -C /var/lib/docker swarm
sudo systemctl start docker
# restore: unpack there on a fresh node, start Docker, then:
# (with autolock on, the restore also needs the unlock key)
docker swarm init --force-new-cluster --advertise-addr <ip>
# the same command recovers a swarm that lost quorum
docker swarm ca --rotate     # new CA; join tokens change
docker swarm update --cert-expiry 2160h  # default 90 days
```

Run an odd number of managers: 3 tolerate the loss of 1, 5 tolerate 2. Changes need a quorum (a majority of managers); without one, running services keep running but nothing can be changed until quorum returns or you force a new cluster.

TROUBLESHOOTING

COMMAND	WHAT IT ANSWERS
`docker info`	is this node in a swarm, its role, manager count
`docker node ls`	a node Down or Drain? no leader elected?
`docker service ps web --no-trunc`	full task errors: pull failures, "no suitable node"
`docker service logs web`	is the application itself failing?
`sudo journalctl -u docker`	the Docker daemon's own log (systemd)

A task stuck in Pending means no node qualifies: drained, short of reserved memory, or excluded by a constraint. A service at 0/3 with a pull error means the nodes cannot fetch the image; check the registry and `--with-registry-auth`.