Administer and maintain a swarm of Docker Engines

Estimated reading time: 16 minutes

When you run a swarm of Docker Engines, **manager nodes** are the key components for managing the swarm and storing the swarm state. It is important to understand some key features of manager nodes to properly deploy and maintain the swarm.

Refer to How nodes work (https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/) for a brief overview of Docker Swarm mode and the difference between manager and worker nodes.

Operate manager nodes in a swarm

Swarm manager nodes use the Raft Consensus Algorithm (https://docs.docker.com/engine/swarm/raft/) to manage the swarm state. You only need to understand some general concepts of Raft in order to manage a swarm.

There is no limit on the number of manager nodes. The decision about how many manager nodes to implement is a trade-off between performance and fault-tolerance. Adding manager nodes to a swarm makes the swarm more fault-tolerant. However, additional manager nodes reduce write performance because more nodes must acknowledge proposals to update the swarm state. This means more network round-trip traffic.

Raft requires a majority of managers, also called the quorum, to agree on proposed updates to the swarm, such as node additions or removals. Membership operations are subject to the same constraints as state replication.

Maintain the quorum of managers

If the swarm loses the quorum of managers, the swarm cannot perform management tasks. If your swarm has multiple managers, always have more than two. To maintain quorum, a majority of managers must be available. An odd number of managers is recommended, because the next even number

does not make the quorum easier to keep. For instance, whether you have 3 or 4 managers, you can still only lose 1 manager and maintain the quorum. If you have 5 or 6 managers, you can still only lose two.

Even if a swarm loses the quorum of managers, swarm tasks on existing worker nodes continue to run. However, swarm nodes cannot be added, updated, or removed, and new or existing tasks cannot be started, stopped, moved, or updated.

See Recovering from losing the quorum (/engine/swarm/admin_guide/#recovering-from-losing-the-quorum) for troubleshooting steps if you do lose the quorum of managers.

Configure the manager to advertise on a static IP address

When initiating a swarm, you must specify the `--advertise-addr` flag to advertise your address to other manager nodes in the swarm. For more information, see Run Docker Engine in swarm mode (https://docs.docker.com/engine/swarm/swarm-mode/#configure-the-advertise-address). Because manager nodes are meant to be a stable component of the infrastructure, you should use a *fixed IP address* for the advertise address to prevent the swarm from becoming unstable on machine reboot.

If the whole swarm restarts and every manager node subsequently gets a new IP address, there is no way for any node to contact an existing manager. Therefore the swarm is hung while nodes try to contact one another at their old IP addresses.

Dynamic IP addresses are OK for worker nodes.

Add manager nodes for fault tolerance

You should maintain an odd number of managers in the swarm to support manager node failures. Having an odd number of managers ensures that during a network partition, there is a higher chance that the quorum remains available to process requests if the network is partitioned into two sets. Keeping the quorum is not guaranteed if you encounter more than two network partitions.

Swarm Size	Majority	Fault Tolerance

Swarm Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	**1**
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

For example, in a swarm with *5 nodes*, if you lose *3 nodes*, you don't have a quorum. Therefore you can't add or remove nodes until you recover one of the unavailable manager nodes or recover the swarm with disaster recovery commands. See Recover from disaster (/engine/swarm/admin_guide/#recover-from-disaster).

While it is possible to scale a swarm down to a single manager node, it is impossible to demote the last manager node. This ensures you maintain access to the swarm and that the swarm can still process requests. Scaling down to a single manager is an unsafe operation and is not recommended. If the last node leaves the swarm unexpectedly during the demote operation, the swarm becomes unavailable until you reboot the node or restart with `--force-new-cluster` .

You manage swarm membership with the `docker swarm` and `docker node` subsystems. Refer to Add nodes to a swarm (https://docs.docker.com/engine/swarm/join-nodes/) for more information on how to add worker nodes and promote a worker node to be a manager.

Distribute manager nodes

In addition to maintaining an odd number of manager nodes, pay attention to datacenter topology when placing managers. For optimal fault-tolerance, distribute manager nodes across a minimum of 3 availability-zones to support

failures of an entire set of machines or common maintenance scenarios. If you suffer a failure in any of those zones, the swarm should maintain the quorum of manager nodes available to process requests and rebalance workloads.

Swarm manager nodes	Repartition (on 3 Availability zones)
3	1-1-1
5	2-2-1
7	3-2-2
9	3-3-3

Run manager-only nodes

By default manager nodes also act as a worker nodes. This means the scheduler can assign tasks to a manager node. For small and non-critical swarms assigning tasks to managers is relatively low-risk as long as you schedule services using **resource constraints** for *cpu* and *memory*.

However, because manager nodes use the Raft consensus algorithm to replicate data in a consistent way, they are sensitive to resource starvation. You should isolate managers in your swarm from processes that might block swarm operations like swarm heartbeat or leader elections.

To avoid interference with manager node operation, you can drain manager nodes to make them unavailable as worker nodes:

```
docker node update --availability drain <NODE>
```

When you drain a node, the scheduler reassigns any tasks running on the node to other available worker nodes in the swarm. It also prevents the scheduler from assigning tasks to the node.

Add worker nodes for load balancing

Add nodes to the swarm (https://docs.docker.com/engine/swarm/join-nodes/) to balance your swarm's load. Replicated service tasks are distributed across the swarm as evenly as possible over time, as long as the worker nodes are matched to the requirements of the services. When limiting a service to run on

only specific types of nodes, such as nodes with a specific number of CPUs or amount of memory, remember that worker nodes that do not meet these requirements cannot run these tasks.

Monitor swarm health

You can monitor the health of manager nodes by querying the docker `nodes` API in JSON format through the `/nodes` HTTP endpoint. Refer to the nodes API documentation (https://docs.docker.com/engine/api/v1.25/#tag/Node) for more information.

From the command line, run `docker node inspect <id-node>` to query the nodes. For instance, to query the reachability of the node as a manager:

```
docker node inspect manager1 --format "{{ .ManagerStatus.Reachabil
ity }}"
reachable
```

To query the status of the node as a worker that accept tasks:

```
docker node inspect manager1 --format "{{ .Status.State }}"
ready
```

From those commands, we can see that `manager1` is both at the status `reachable` as a manager and `ready` as a worker.

An `unreachable` health status means that this particular manager node is unreachable from other manager nodes. In this case you need to take action to restore the unreachable manager:

- Restart the daemon and see if the manager comes back as reachable.
- Reboot the machine.
- If neither restarting or rebooting work, you should add another manager node or promote a worker to be a manager node. You also need to cleanly remove the failed node entry from the manager set with `docker node demote <NODE>` and `docker node rm <id-node>` .

Alternatively you can also get an overview of the swarm health from a manager node with `docker node ls` :

```
docker node ls
ID                            HOSTNAME  MEMBERSHIP  STATUS  AVAILAB
ILITY  MANAGER STATUS
1mhtdwhvsgr3c26xxbnzdc3yp    node05    Accepted    Ready   Active
516pacagkqp2xc3fk9t1dhjor    node02    Accepted    Ready   Active
       Reachable
9ifojw8of78kkusuc4a6c23fx *  node01    Accepted    Ready   Active
       Leader
ax11wdpwrrb6db3mfjydscgk7    node04    Accepted    Ready   Active
bb1nrq2cswhtbg4mrsqnlx1ck    node03    Accepted    Ready   Active
       Reachable
di9wxgz8dtuh9d2hn089ecqkf    node06    Accepted    Ready   Active
```

Troubleshoot a manager node

You should never restart a manager node by copying the `raft` directory from another node. The data directory is unique to a node ID. A node can only use a node ID once to join the swarm. The node ID space should be globally unique.

To cleanly re-join a manager node to a cluster:

1. To demote the node to a worker, run `docker node demote <NODE>`.
2. To remove the node from the swarm, run `docker node rm <NODE>`.
3. Re-join the node to the swarm with a fresh state using `docker swarm join`.

For more information on joining a manager node to a swarm, refer to Join nodes to a swarm (https://docs.docker.com/engine/swarm/join-nodes/).

Forcibly remove a node

In most cases, you should shut down a node before removing it from a swarm with the `docker node rm` command. If a node becomes unreachable, unresponsive, or compromised you can forcefully remove the node without shutting it down by passing the `--force` flag. For instance, if `node9` becomes compromised:

```
$ docker node rm node9

Error response from daemon: rpc error: code = 9 desc = node node9
is not down and can't be removed

$ docker node rm --force node9

Node node9 removed from swarm
```

Before you forcefully remove a manager node, you must first demote it to the worker role. Make sure that you always have an odd number of manager nodes if you demote or remove a manager.

Back up the swarm

Docker manager nodes store the swarm state and manager logs in the `/var/lib/docker/swarm/` directory. In 1.13 and higher, this data includes the keys used to encrypt the Raft logs. Without these keys, you cannot restore the swarm.

You can back up the swarm using any manager. Use the following procedure.

1. If the swarm has auto-lock enabled, you need the unlock key to restore the swarm from backup. Retrieve the unlock key if necessary and store it in a safe location. If you are unsure, read Lock your swarm to protect its encryption key (https://docs.docker.com/engine/swarm/swarm_manager_locking/).
2. Stop Docker on the manager before backing up the data, so that no data is being changed during the backup. It is possible to take a backup while the manager is running (a “hot” backup), but this is not recommended and your results are less predictable when restoring. While the manager is down, other nodes continue generating swarm data that is not part of this backup.

   > **Note**: Be sure to maintain the quorum of swarm managers. During the time that a manager is shut down, your swarm is more vulnerable to losing the quorum if further nodes are lost. The number of managers you run is a trade-off. If you regularly take down managers to do backups, consider running a 5-manager swarm, so that you can lose an additional manager while the backup is running, without disrupting your services.

3. Back up the entire `/var/lib/docker/swarm` directory.
4. Restart the manager.

To restore, see Restore from a backup (/engine/swarm/admin_guide/#restore-from-a-backup).

Recover from disaster

Restore from a backup

After backing up the swarm as described in Back up the swarm (/engine/swarm/admin_guide/#back-up-the-swarm), use the following procedure to restore the data to a new swarm.

1. Shut down Docker on the target host machine for the restored swarm.
2. Remove the contents of the `/var/lib/docker/swarm` directory on the new swarm.
3. Restore the `/var/lib/docker/swarm` directory with the contents of the backup.

   > **Note: The new node uses the same encryption key for on-disk storage as the old one. It is not possible to change the on-disk storage encryption keys at this time.**
   >
   > In the case of a swarm with auto-lock enabled, the unlock key is also the same as on the old swarm, and the unlock key is needed to restore the swarm.

4. Start Docker on the new node. Unlock the swarm if necessary. Re-initialize the swarm using the following command, so that this node does not attempt to connect to nodes that were part of the old swarm, and presumably no longer exist.

   ```
   $ docker swarm init --force-new-cluster
   ```

5. Verify that the state of the swarm is as expected. This may include application-specific tests or simply checking the output of `docker service ls` to be sure that all expected services are present.

6. If you use auto-lock, rotate the unlock key (https://docs.docker.com/engine/swarm/swarm_manager_locking/#rotate-the-unlock-key).
7. Add manager and worker nodes to bring your new swarm up to operating capacity.
8. Reinstate your previous backup regimen on the new swarm.

Recover from losing the quorum

Swarm is resilient to failures and the swarm can recover from any number of temporary node failures (machine reboots or crash with restart) or other transient errors. However, a swarm cannot automatically recover if it loses a quorum. Tasks on existing worker nodes continue to run, but administrative tasks are not possible, including scaling or updating services and joining or removing nodes from the swarm. The best way to recover is to bring the missing manager nodes back online. If that is not possible, continue reading for some options for recovering your swarm.

In a swarm of `N` managers, a quorum (a majority) of manager nodes must always be available. For example, in a swarm with 5 managers, a minimum of 3 must be operational and in communication with each other. In other words, the swarm can tolerate up to `(N-1)/2` permanent failures beyond which requests involving swarm management cannot be processed. These types of failures include data corruption or hardware failures.

If you lose the quorum of managers, you cannot administer the swarm. If you have lost the quorum and you attempt to perform any management operation on the swarm, an error occurs:

```
Error response from daemon: rpc error: code = 4 desc = context dea
dline exceeded
```

The best way to recover from losing the quorum is to bring the failed nodes back online. If you can't do that, the only way to recover from this state is to use the `--force-new-cluster` action from a manager node. This removes all managers except the manager the command was run from. The quorum is achieved because there is now only one manager. Promote nodes to be managers until you have the desired number of managers.

```
# From the node to recover
docker swarm init --force-new-cluster --advertise-addr node01:2377
```

When you run the `docker swarm init` command with the `--force-new-cluster` flag, the Docker Engine where you run the command becomes the manager node of a single-node swarm which is capable of managing and running services. The manager has all the previous information about services and tasks, worker nodes are still part of the swarm, and services are still running. You need to add or re-add manager nodes to achieve your previous task distribution and ensure that you have enough managers to maintain high availability and prevent losing the quorum.

Force the swarm to rebalance

Generally, you do not need to force the swarm to rebalance its tasks. When you add a new node to a swarm, or a node reconnects to the swarm after a period of unavailability, the swarm does not automatically give a workload to the idle node. This is a design decision. If the swarm periodically shifted tasks to different nodes for the sake of balance, the clients using those tasks would be disrupted. The goal is to avoid disrupting running services for the sake of balance across the swarm. When new tasks start, or when a node with running tasks becomes unavailable, those tasks are given to less busy nodes. The goal is eventual balance, with minimal disruption to the end user.

In Docker 1.13 and higher, you can use the `--force` or `-f` flag with the `docker service update` command to force the service to redistribute its tasks across the available worker nodes. This causes the service tasks to restart. Client applications may be disrupted. If you have configured it, your service uses a rolling update (https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/).

If you use an earlier version and you want to achieve an even balance of load across workers and don't mind disrupting running tasks, you can force your swarm to re-balance by temporarily scaling the service upward. Use `docker service inspect --pretty <servicename>` to see the configured scale of a service. When you use `docker service scale`, the nodes with the lowest number of tasks are targeted to receive the new workloads. There may be multiple under-loaded nodes in your swarm. You may need to scale the service up by modest increments a few times to achieve the balance you want across all the nodes.

When the load is balanced to your satisfaction, you can scale the service back down to the original scale. You can use `docker service ps` to assess the current balance of your service across nodes.

See also `docker service scale` (https://docs.docker.com/engine/reference/commandline/service_scale/) and `docker service ps` (https://docs.docker.com/engine/reference/commandline/service_ps/).

docker (https://docs.docker.com/glossary/?term=docker), container (https://docs.docker.com/glossary/?term=container), swarm (https://docs.docker.com/glossary/?term=swarm), manager (https://docs.docker.com/glossary/?term=manager), raft (https://docs.docker.com/glossary/?term=raft)