SWARM using Virtual box in SWARM Mode

Create 5 machines

Managers- manager1, manager1,

Workers: worker1, worker2, worker3

1. Creating Machine
   1. Open docker terminal in your desktop

$docker-machine create -d virtualbox <machine name>

1. Login to Master1

$docker-machine ip manager1

$docker-machine ssh manager1

1. Run the following command to create a new swarm:

docker swarm init --advertise-addr <MANAGER-IP>

**Note**: If you are using Docker for Mac or Docker for Windows to test single-node swarm, simply run docker swarm init with no arguments. There is no need to specify --advertise-addr in this case. To learn more, see the topic on how to [Use Docker for Mac or Docker for Windows](https://docs.docker.com/engine/swarm/swarm-tutorial/#use-docker-for-mac-or-docker-for-windows) with Swarm.

In the tutorial, the following command creates a swarm on the manager1 machine:

$ docker swarm init --advertise-addr 192.168.99.100

Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a manager.

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

docker swarm join \

--token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \

192.168.99.100:2377

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

The --advertise-addr flag configures the manager node to publish its address as 192.168.99.100. The other nodes in the swarm must be able to access the manager at the IP address.

The output includes the commands to join new nodes to the swarm. Nodes will join as managers or workers depending on the value for the --token flag.

1. Run docker info to view the current state of the swarm:

$ docker info

Containers: 2

Running: 0

Paused: 0

Stopped: 2

...snip...

Swarm: active

NodeID: dxn1zf6l61qsb1josjja83ngz

Is Manager: true

Managers: 1

Nodes: 1

...snip...

1. Run the docker node ls command to view information about nodes:

$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

dxn1zf6l61qsb1josjja83ngz \* manager1 Ready Active Leader

1. Adding a Another Manager
   1. Login in to the Manager1

$docker-machine ssh manager1

* 1. Get Keys for Manager join

$docker swarm join-token manager

* 1. Login into the Manager2

docker-machine ssh manager2

* 1. Join the node as manager

Past the output of step b here

* 1. Check node status

$docker node ls

$ docker node inspect <node id or name>

1. Adding worker to the cluster

To retrieve the join command including the join token for worker nodes, run the following command on a manager node:

$ docker swarm join-token worker

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

docker swarm join \

--token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \

192.168.99.100:2377

Run the command from the output on the worker to join the swarm:

$ docker swarm join \

--token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \

192.168.99.100:2377

This node joined a swarm as a worker.

The docker swarm join command does the following:

* switches the Docker Engine on the current node into swarm mode.
* requests a TLS certificate from the manager.
* names the node with the machine hostname
* joins the current node to the swarm at the manager listen address based upon the swarm token.
* sets the current node to Active availability, meaning it can receive tasks from the scheduler.
* extends the ingress overlay network to the current node.

### **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>

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.

## 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

## 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 will not be able to 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 will need the unlock key in order 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 will be less predictable when restoring. While the manager is down, other nodes will continue generating swarm data that will not be 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.

1. Back up the entire /var/lib/docker/swarm directory.
2. Restart the manager.

To restore, see [Restore from a backup](https://docs.docker.com/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](https://docs.docker.com/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 where the swarm will be restored.
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 will use 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 will be needed to restore.

1. 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

1. 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.
2. If you use auto-lock,

## Rotate the unlock key

You should rotate the locked swarm’s unlock key on a regular schedule.

$ docker swarm unlock-key --rotate

Successfully rotated manager unlock key.

To unlock a swarm manager after it restarts, run the `docker swarm unlock`

command and provide the following key:

SWMKEY-1-8jDgbUNlJtUe5P/lcr9IXGVxqZpZUXPzd+qzcGp4ZYA

Please remember to store this key in a password manager, since without it you

will not be able to restart the manager.

1. Add manager and worker nodes to bring your new swarm up to operating capacity.
2. Reinstate your previous backup regimen on the new swarm.

Deploy a Service in SWARM

## Create a service

To create a single-replica service with no extra configuration, you only need to supply the image name. This command starts an Nginx service with a randomly-generated name and no published ports. This is a naive example, since you won’t be able to interact with the Nginx service.

$ docker service create nginx

The service is scheduled on an available node. To confirm that the service was created and started successfully, use the docker service ls command:

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

a3iixnklxuem quizzical\_lamarr replicated 1/1 docker.io/library/nginx@sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268

Created services do not always run right away. A service can be in a pending state if its image is unavailable, if no node meets the requirements you configure for the service, or other reasons. See [Pending services](https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/#pending-services) for more information.

To provide a name for your service, use the --name flag:

$ docker service create --name my\_web nginx

Just like with standalone containers, you can specify a command that the service’s containers should run, by adding it after the image name. This example starts a service called helloworld which uses an alpine image and runs the command ping docker.com:

$ docker service create --name helloworld alpine ping docker.com

You can also specify an image tag for the service to use. This example modifies the previous one to use the alpine:3.6 tag:

$ docker service create --name helloworld alpine:3.6 ping docker.com

## Update a service

You can change almost everything about an existing service using the docker service update command. When you update a service, Docker stops its containers and restarts them with the new configuration.

Since Nginx is a web service, it will work much better if you publish port 80 to clients outside the swarm. You can specify this when you create the service, using the -p or --publish flag. When updating an existing service, the flag is --publish-add. There is also a --publish-rm flag to remove a port that was previously published.

Assuming that the my\_web service from the previous section still exists, use the following command to update it to publish port 80.

$ docker service update --publish-add 80 my\_web

To verify that it worked, use docker service ls:

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

4nhxl7oxw5vz my\_web replicated 1/1 docker.io/library/nginx@sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268 \*:0->80/tcp

### **Configure the runtime environment**

You can configure the following options for the runtime environment in the container:

* environment variables using the --env flag
* the working directory inside the container using the --workdir flag
* the username or UID using the --user flag

The following service’s containers will have an environment variable $MYVAR set to myvalue, will run from the /tmp/ directory, and will run as the my\_user user.

$ docker service create --name helloworld \

--env MYVAR=myvalue \

--workdir /tmp \

--user my\_user \

alpine ping docker.com

## Remove a service

To remove a service, use the docker service remove command. You can remove a service by its ID or name, as shown in the output of the docker service ls command. The following command removes the my\_web service.

$ docker service remove my\_web

Create a Overlay Network

You can use overlay networks to connect one or more services within the swarm.

First, create overlay network on a manager node using the docker network create command with the --driver overlay flag.

$ docker network create --driver overlay my-network

After you create an overlay network in swarm mode, all manager nodes have access to the network.

You can create a new service and pass the --network flag to attach the service to the overlay network:

$ docker service create \

--replicas 3 \

--network my-network \

--name my-web \

nginx

The swarm extends my-network to each node running the service.

You can also connect an existing service to an overlay network using the --network-add flag.

$ docker service update --network-add my-network my-web

To disconnect a running service from a network, use the --network-rm flag.

$ docker service update --network-rm my-network my-web

Create Node metadata and deploying to Node Metadata

### **Add or remove label metadata**

Node labels provide a flexible method of node organization. You can also use node labels in service constraints. Apply constraints when you create a service to limit the nodes where the scheduler assigns tasks for the service.

Run docker node update --label-add on a manager node to add label metadata to a node. The --label-add flag supports either a <key> or a <key>=<value> pair.

Pass the --label-add flag once for each node label you want to add:

$ docker node update --label-add foo --label-add bar=baz node-1

node-1

#### **PLACEMENT CONSTRAINTS**

Use placement constraints to control the nodes a service can be assigned to. In the following example, the service only runs on nodes with the [label](https://docs.docker.com/engine/swarm/engine/swarm/manage-nodes/#add-or-remove-label-metadata) region set to east. If no appropriately-labelled nodes are available, deployment will fail. The --constraint flag uses an equality operator (== or !=). For replicated services, it is possible that all services will run on the same node, or each node will only run one replica, or that some nodes won’t run any replicas. For global services, the service will run on every node that meets the placement constraint and any [resource requirements](https://docs.docker.com/engine/swarm/services/#reserve-cpu-or-memory-for-a-service).

$ docker service create \

--name my-nginx \

--replicas 5 \

--constraint region==east \

nginx

You can also use the constraint service-level key in a docker-compose.yml file.

If you specify multiple placement constraints, the service will only deploy onto nodes where they are all met. The following example limits the service to run on all nodes where region is set to east and type is not set to devel:

$ docker service create \

--name my-nginx \

--global \

--constraint region==east \

--constraint type!=devel \

nginx

#### **REPLICATED OR GLOBAL SERVICES**

Swarm mode has two types of services: replicated and global. For replicated services, you specify the number of replica tasks for the swarm manager to schedule onto available nodes. For global services, the scheduler places one task on each available node that meets the service’s [placement constraints](https://docs.docker.com/engine/swarm/services/#placement-constraints) and [resource requirements](https://docs.docker.com/engine/swarm/services/#reserve-cpu-or-memory-for-a-service).

You control the type of service using the --mode flag. If you don’t specify a mode, the service defaults to replicated. For replicated services, you specify the number of replica tasks you want to start using the --replicas flag. For example, to start a replicated nginx service with 3 replica tasks:

$ docker service create \

--name my\_web \

--replicas 3 \

nginx

To start a global service on each available node, pass --mode global to docker service create. Every time a new node becomes available, the scheduler places a task for the global service on the new node. For example to start a service that runs alpine on every node in the swarm:

$ docker service create \

--name myservice \

--mode global \

alpine top

Service constraints let you set criteria for a node to meet before the scheduler deploys a service to the node. You can apply constraints to the service based upon node attributes and metadata or engine metadata. For more information on constraints, refer to the docker service create [CLI reference](https://docs.docker.com/engine/reference/commandline/service_create/).

#### **RESERVE MEMORY OR CPUS FOR A SERVICE**

To reserve a given amount of memory or number of CPUs for a service, use the --reserve-memory or --reserve-cpu flags. If no available nodes can satisfy the requirement (for instance, if you request 4 CPUs and no node in the swarm has 4 CPUs), the service remains in a pending state until a node is available to run its tasks.