Docker Swarm Basics

Objective: This tutorial guides you through, initializing a cluster of Docker Engines in swarm mode, adding nodes to the swarm, deploying application services to the swarm, and managing the swarm once you have everything running

Preparation: To run this tutorial, you need three networked host machines, Docker Engine 1.12 or later installed, he IP address of the manager machine, and open ports between the hosts.

Outcome: xx

Data Files: Ask Instructor

You can create a Docker Swarm cluster using the swarm executable image from a container or using an executable swarm binary you install on your system. This page introduces the two methods and discusses their pros and cons. Also, you see how Swarm and Docker Compose can work together.

Step 1. Set up for this lab

1. Three networked host machines-This tutorial uses three networked host machines as nodes in the swarm. These can be virtual machines on your PC, in a data center, or on a cloud service provider. This tutorial uses the following machine names:

```
manager1
worker1
worker2
```

> *NOTE: You can follow many of the tutorial steps to test single-node swarm as well, in which case you need only one host. Multi-node commands will not work, but you can initialize a swarm, create services, and scale them.*

2. Installing Docker Engine 1.12 or newer. This tutorial requires Docker Engine 1.12 or newer on each of the host machines. Install Docker Engine and verify that the Docker Engine daemon is running on each of the machines. You can get the latest version of Docker Engine as follows:

INSTALL DOCKER ENGINE ON LINUX MACHINES

If you are using Linux based physical computers or cloud-provided computers as hosts, simply follow the Linux install instructions for your platform. Spin up the three machines, and you are ready. You can test both single-node and multi-node swarm scenarios on Linux machines.

USE DOCKER FOR MAC OR DOCKER FOR WINDOWS

Alternatively, install the latest Docker for Mac or Docker for Windows application on one computer. You can test both single-node and multi-node swarm from this computer, but you will need to use Docker Machine to test the multi-node scenarios.

- You can use Docker for Mac or Windows to test single-node features of swarm mode, including initializing a swarm with a single node, creating services, and scaling services. Docker “Moby” on Hyperkit (Mac) or Hyper-V (Windows) will serve as the single swarm node.
- Currently, you cannot use Docker for Mac or Windows alone to test a multi-node swarm. However, you can use the included version of Docker Machine to create the swarm nodes (see Get started with Docker Machine and a local VM), then follow the tutorial for all multi-node features. For this scenario, you run commands from a Docker for Mac or Docker for Windows host, but that Docker host itself is not participating in the swarm (i.e., it will not be `manager1`, `worker1`, or `worker2` in our example). After you create the nodes, you can run all swarm commands as shown from the Mac terminal or Windows PowerShell with Docker for Mac or Docker for Windows running.

3. The IP address of the manager machine. The IP address must be assigned to a network interface available to the host operating system. All nodes in the swarm must be able to access the manager at the IP address.

Because other nodes contact the manager node on its IP address, you should use a fixed IP address.

You can run `ifconfig` on Linux or macOS to see a list of the available network interfaces.

If you are using Docker Machine, you can get the manager IP with either `docker-machine ls` or `docker-machine ip <MACHINE-NAME>` — for example, `docker-machine ip manager1`.

The tutorial uses `manager1` : `192.168.99.100`.

Open protocols and ports between the hosts. The following ports must be available. On some systems, these ports are open by default.

- **TCP port 2377** for cluster management communications
- **TCP** and **UDP port 7946** for communication among nodes
- **UDP port 4789** for overlay network traffic

If you are planning on creating an overlay network with encryption (–opt encrypted), you will also need to ensure ip protocol 50 (ESP) traffic is allowed.

```
> NOTE: If you are planning on creating an overlay network wi
th encryption (```--opt encrypted```), you will also need to
ensure **ip protocol 50 (ESP)** traffic is allowed.
```

Step 2. Create a Swarm

After you complete the setup steps, you’re ready to create a swarm. Make sure the Docker Engine daemon is started on the host machines.

1. Open a terminal and ssh into the machine where you want to run your manager node. This tutorial uses a machine named manager1. If you use Docker Machine, you can connect to it via SSH using the following command:

   ```
   $ docker-machine ssh manager1
   ```

2. 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.*

3. 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 (dxn1zf6l61qsb1josjja83
   ngz) is now a manager.

   To add a worker to this swarm, run the following comman
   d:
   ```

```

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0
xx14ie39trti4wxv-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.

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

```
$ docker info
```

Output:

```
Containers: 2
Running: 0
Paused: 0
```

```
Stopped: 2
  ...snip...
Swarm: active
  NodeID: dxn1zf6l61qsb1josjja83ngz
  Is Manager: true
  Managers: 1
  Nodes: 1
  ...snip...
```

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

```
$ docker node ls
```

Output:

```
ID                           HOSTNAME  STATUS  AVAILABI
LITY  MANAGER STATUS
dxn1zf6l61qsb1josjja83ngz *  manager1  Ready   Active
      Leader
```

The `*` next to the node ID indicates that you’re currently connected on this node.
Docker Engine swarm mode automatically names the node for the machine host name. The tutorial covers other columns in later steps.

Step 3. Add Nodes with Swarm

Once you’ve created a swarm with a manager node, you’re ready to add worker nodes.

1. Open a terminal and ssh into the machine where you want to run a worker node. This tutorial uses the name `worker1`.

2. Run the command produced by the `docker swarm init` output from the previous step to create a worker node joined to the existing swarm:

   ```
   $ docker swarm join --token  SWMTKN-1-49nj1cmql0jkz5s95
   4yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr
   2e7c 192.168.99.100:2377
   ```

   Output:

   ```
   This node joined a swarm **as** a worker.
   ```

3. If you don’t have the command available, you can run the following command on a manager node to retrieve the join command for a worker:

   ```
   $ docker swarm join-token worker
   ```

   Output:

```
To **add** a worker to this swarm, **run** the followin
g command:

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0
xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
    192.168.99.100:2377
```

4. Open a terminal and ssh into the machine where you want to run a second worker node. This tutorial uses the name `worker2`.

5. Run the command produced by the `docker swarm init` output from the previous step to create a second worker node joined to the existing swarm:

   ```
   $ docker swarm join \
     --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx
   14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
     192.168.99.100:2377
   ```

   Output:

   ```
   This node joined a swarm **as** a worker.
   ```

6. Open a terminal and ssh into the machine where the manager node

runs and run the `docker node ls` command to see the worker nodes:

```
ID                            HOSTNAME  STATUS  AVAILABI
LITY  MANAGER STATUS
03g1y59jwfg7cf99w4lt0f662     worker2   Ready   Active
9j68exjopxe7wfl6yuxml7a7j     worker1   Ready   Active
dxn1zf6l61qsb1josjja83ngz *   manager1  Ready   Active
      Leader
```

The `MANAGER` column identifies the manager nodes in the swarm. The empty status in this column for `worker1` and `worker2` identifies them as worker nodes.

Swarm management commands like `docker node ls` only work on manager nodes.

Step 4. Deploy Swarm Service

After you create a swarm, you can deploy a service to the swarm. For this tutorial, you also added worker nodes, but that is not a requirement to deploy a service.

1. Open a terminal and ssh into the machine where you run your manager node. For example, the tutorial uses a machine named `manager1`.

2. Run the following command:

```
$ docker service create --replicas 1 --name helloworld
alpine ping docker.com
```

Output:

```
9uk4639qpg7npwf3fn2aasksr
```

  - The `docker service create` command creates the service.
  - The `--name` flag names the service `helloworld`.
  - The `--replicas` flag specifies the desired state of 1 running instance.
  - The arguments `alpine ping docker.com` define the service as an Alpine Linux container that executes the command `ping docker.com`.

3. Run docker service ls to see the list of running services:

```
$ docker service ls
```

Output:

```
ID            NAME        SCALE  IMAGE   COMMAND
9uk4639qpg7n  helloworld  1/1    alpine  ping docker.co
m
```

Step 5. [OPTIONAL] Inspect a Swarm Service

When you have deployed a service to your swarm, as we just did in the previous step, you can use the Docker CLI to see details about the service running in the swarm.

1. If you haven't already, open a terminal and ssh into the machine where you run your manager node. For example, the tutorial uses a machine named `manager1`.

2. Run `docker service inspect --pretty <SERVICE-ID>` to display the details about a service in an easily readable format. To see the details on the `helloworld` service:

   ```
   $ docker service inspect --pretty helloworld
   ```

   Output:

   ```
   ID:		9uk4639qpg7npwf3fn2aasksr
   Name:		helloworld
   Service Mode:	REPLICATED
    Replicas:		1
   Placement:
   UpdateConfig:
   ```

```
 Parallelism:   1
ContainerSpec:
 Image:     alpine
 Args:  ping docker.com
Resources:
Endpoint Mode:  vip
```

> *NOTE: To return the service details in json format, run the same command without the `--pretty` flag.*

3. Run `docker service ps <SERVICE-ID>` to see which nodes are running the service:

   ```
   $ docker service ps helloworld
   ```

   Output:

   ```
   NAME                                    IMAGE   NODE
     DESIRED STATE  LAST STATE
   helloworld.1.8p1vev3fq5zm0mi8g0as41w35  alpine  worker2
     Running        Running 3 minutes
   ```

In this case, the one instance of the `helloworld` service is running on the `worker2` node. You may see the service running on your manager node. By default, manager nodes in a swarm can execute tasks just like worker nodes.

Swarm also shows you the `DESIRED STATE` and `LAST STATE` of the service task so you can see if tasks are running according to the service definition.

4. Run docker ps on the node where the task is running to see details about the container for the task.
   Tip: If `helloworld` is running on a node other than your manager node, you must ssh to that node.

   ```
   $docker ps
   ```

   Output:

   ```
   CONTAINER ID      IMAGE                COMMAND                C
   REATED            STATUS               PORTS        NAMES
   e609dde94e47      alpine:latest       "ping docker.com"    3
    minutes ago      Up 3 minutes          helloworld.1.8p1vev3
   fq5zm0mi8g0as41w35
   ```

Conclusion

Docker Swarm is an iceberg and this was just the tip. This course goes light on Swarm because of the centeralized theme of Kubernetes. You have however, still managed to learn the basics and that will help you learn Kubernetes faster.