Get started with multi-host networking

Estimated reading time: 11 minutes

This article uses an example to explain the basics of creating a multi-host network. Docker Engine supports multi-host networking out-of-the-box through the `overlay` network driver. Unlike `bridge` networks, overlay networks require some pre-existing conditions before you can create one:

- Docker Engine running in swarm mode

OR

- A cluster of hosts using a key value store

Overlay networking and swarm mode

Using docker engine running in swarm mode, you can create an overlay network on a manager node.

The swarm makes the overlay network available only to nodes in the swarm that require it for a service. When you create a service that uses an overlay network, the manager node automatically extends the overlay network to nodes that run service tasks.

To learn more about running Docker Engine in swarm mode, refer to the Swarm mode overview.

The example below shows how to create a network and use it for a service from a manager node in the swarm:

```
# Create an overlay network `my-multi-host-network`.
$ docker network create \
```

```
  --driver overlay \
  --subnet 10.0.9.0/24 \
  my-multi-host-network

400g6bwzd68jizzdx5pgyoe95

# Create an nginx service and extend the my-multi-host-network to node
# the service's tasks run.
$ docker service create --replicas 2 --network my-multi-host-network -

716thylsndqma81j6kkkb5aus
```

Overlay networks for a swarm are not available to unmanaged containers. For more information refer to Docker swarm mode overlay network security model.

See also Attach services to an overlay network.

Overlay networking with an external key-value store

To use an Docker engine with an external key-value store, you need the following:

- Access to the key-value store. Docker supports Consul, Etcd, and ZooKeeper (Distributed store) key-value stores.
- A cluster of hosts with connectivity to the key-value store.
- A properly configured Engine `daemon` on each host in the cluster.
- Hosts within the cluster must have unique hostnames because the key-value store uses the hostnames to identify cluster members.

Though Docker Machine and Docker Swarm are not mandatory to experience Docker multi-host networking with a key-value store, this example uses them to illustrate how they are integrated. You'll use Machine to create both the key-value store server and the host cluster. This example creates a swarm cluster.

> **Note:** Docker Engine running in swarm mode is not compatible with

> networking with an external key-value store.

Prerequisites

Before you begin, make sure you have a system on your network with the latest version of Docker Engine and Docker Machine installed. The example also relies on VirtualBox. If you installed on a Mac or Windows with Docker Toolbox, you have all of these installed already.

If you have not already done so, make sure you upgrade Docker Engine and Docker Machine to the latest versions.

Set up a key-value store

An overlay network requires a key-value store. The key-value store holds information about the network state which includes discovery, networks, endpoints, IP addresses, and more. Docker supports Consul, Etcd, and ZooKeeper key-value stores. This example uses Consul.

1. Log into a system prepared with the prerequisite Docker Engine, Docker Machine, and VirtualBox software.

2. Provision a VirtualBox machine called `mh-keystore`.

   ```
   $ docker-machine create -d virtualbox mh-keystore
   ```

   When you provision a new machine, the process adds Docker Engine to the host. This means rather than installing Consul manually, you can create an instance using the consul image from Docker Hub. You'll do this in the next step.

3. Set your local environment to the `mh-keystore` machine.

   ```
   $  eval "$(docker-machine env mh-keystore)"
   ```

4. Start a `progrium/consul` container running on the `mh-keystore` machine.

```
$  docker run -d \
    -p "8500:8500" \
    -h "consul" \
    progrium/consul -server -bootstrap
```

   The client starts a `progrium/consul` image running in the `mh-keystore` machine. The server is called `consul` and is listening on port `8500`.

5. Run the `docker ps` command to see the `consul` container.

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND
4d51392253b3        progrium/consul     "/bin/start -server -"
```

Keep your terminal open and move onto the next step.

Create a Swarm cluster

In this step, you use `docker-machine` to provision the hosts for your network. At this point, you won't actually create the network. You'll create several machines in VirtualBox. One of the machines will act as the swarm master; you'll create that first. As you create each host, you'll pass the Engine on that machine options that are needed by the `overlay` network driver.

1. Create a swarm master.

```
$ docker-machine create \
-d virtualbox \
--swarm --swarm-master \
--swarm-discovery="consul://$(docker-machine ip mh-keystore):850(
```

```
--engine-opt="cluster-store=consul://$(docker-machine ip mh-keyst
--engine-opt="cluster-advertise=eth1:2376" \
mhs-demo0
```

At creation time, you supply the Engine `daemon` with the `--cluster-store` option. This option tells the Engine the location of the key-value store for the `overlay` network. The bash expansion `$(docker-machine ip mh-keystore)` resolves to the IP address of the Consul server you created in "STEP 1". The `--cluster-advertise` option advertises the machine on the network.

2. Create another host and add it to the swarm cluster.

```
$ docker-machine create -d virtualbox \
    --swarm \
    --swarm-discovery="consul://$(docker-machine ip mh-keystore):
    --engine-opt="cluster-store=consul://$(docker-machine ip mh-k
    --engine-opt="cluster-advertise=eth1:2376" \
  mhs-demo1
```

3. List your machines to confirm they are all up and running.

```
$ docker-machine ls

NAME          ACTIVE   DRIVER       STATE     URL
default       -        virtualbox   Running   tcp://192.168.99.10(
mh-keystore   *        virtualbox   Running   tcp://192.168.99.10:
mhs-demo0     -        virtualbox   Running   tcp://192.168.99.10
mhs-demo1     -        virtualbox   Running   tcp://192.168.99.10
```

At this point you have a set of hosts running on your network. You are ready to create a multi-host network for containers using these hosts.

Leave your terminal open and go onto the next step.

Create the overlay Network

To create an overlay network

1. Set your docker environment to the swarm master.

```
$ eval $(docker-machine env --swarm mhs-demo0)
```

   Using the `--swarm` flag with `docker-machine` restricts the `docker` commands to swarm information alone.

2. Use the `docker info` command to view the swarm.

```
$ docker info

Containers: 3
Images: 2
Role: primary
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 2
mhs-demo0: 192.168.99.104:2376
└ Containers: 2
└ Reserved CPUs: 0 / 1
└ Reserved Memory: 0 B / 1.021 GiB
└ Labels: executiondriver=native-0.2, kernelversion=4.1.10-boot2d
mhs-demo1: 192.168.99.105:2376
└ Containers: 1
└ Reserved CPUs: 0 / 1
└ Reserved Memory: 0 B / 1.021 GiB
└ Labels: executiondriver=native-0.2, kernelversion=4.1.10-boot2d
CPUs: 2
Total Memory: 2.043 GiB
Name: 30438ece0915
```

   From this information, you can see that you are running three containers and two images on the Master.

3. Create your `overlay` network.

```
$ docker network create --driver overlay --subnet=10.0.9.0/24 my-
```

You only need to create the network on a single host in the cluster. In this case, you used the swarm master but you could easily have run it on any host in the cluster.

> **Note** : It is highly recommended to use the `--subnet` option when creating a network. If the `--subnet` is not specified, the docker daemon automatically chooses and assigns a subnet for the network and it could overlap with another subnet in your infrastructure that is not managed by docker. Such overlaps can cause connectivity issues or failures when containers are connected to that network.

1. Check that the network is running:

```
$ docker network ls

NETWORK ID          NAME                DRIVER
412c2496d0eb        mhs-demo1/host      host
dd51763e6dd2        mhs-demo0/bridge    bridge
6b07d0be843f        my-net              overlay
b4234109bd9b        mhs-demo0/none      null
1aeead6dd890        mhs-demo0/host      host
d0bb78cbe7bd        mhs-demo1/bridge    bridge
1c0eb8f69ebb        mhs-demo1/none      null
```

As you are in the swarm master environment, you see all the networks on all the swarm agents: the default networks on each engine and the single overlay network. Notice that each `NETWORK ID` is unique.

2. Switch to each swarm agent in turn and list the networks.

```
$ eval $(docker-machine env mhs-demo0)

$ docker network ls
```

```
NETWORK ID          NAME                DRIVER
6b07d0be843f        my-net              overlay
dd51763e6dd2        bridge              bridge
b4234109bd9b        none                null
1aeead6dd890        host                host

$ eval $(docker-machine env mhs-demo1)

$ docker network ls

NETWORK ID          NAME                DRIVER
d0bb78cbe7bd        bridge              bridge
1c0eb8f69ebb        none                null
412c2496d0eb        host                host
6b07d0be843f        my-net              overlay
```

Both agents report they have the `my-net` network with the `6b07d0be843f` ID. You now have a multi-host container network running!

Run an application on your Network

Once your network is created, you can start a container on any of the hosts and it automatically is part of the network.

1. Point your environment to the swarm master.

```
$ eval $(docker-machine env --swarm mhs-demo0)
```

2. Start an Nginx web server on the `mhs-demo0` instance.

```
$ docker run -itd --name=web --network=my-net --env="constraint:r
```

3. Run a BusyBox instance on the `mhs-demo1` instance and get the contents of the Nginx server's home page.

```
$ docker run -it --rm --network=my-net --env="constraint:node==mh
```

```
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
ab2b8a86ca6c: Pull complete
2c5ac3f849df: Pull complete
Digest: sha256:5551dbdfc48d66734d0f01cafee0952cb6e8eeecd1e249224(
Status: Downloaded newer image for busybox:latest
Connecting to web (10.0.0.2:80)
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully ins
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
-                    100% |********************************|   612
```

Check external connectivity

As you've seen, Docker's built-in overlay network driver provides out-of-the-box connectivity between the containers on multiple hosts within the same network. Additionally, containers connected to the multi-host network are automatically connected to the `docker_gwbridge` network. This network allows the containers to have external connectivity outside of

their cluster.

1. Change your environment to the swarm agent.

```
$ eval $(docker-machine env mhs-demo1)
```

2. View the `docker_gwbridge` network, by listing the networks.

```
$ docker network ls

NETWORK ID          NAME                DRIVER
6b07d0be843f        my-net              overlay
dd51763e6dd2        bridge              bridge
b4234109bd9b        none                null
1aeead6dd890        host                host
e1dbd5dff8be        docker_gwbridge     bridge
```

3. Repeat steps 1 and 2 on the swarm master.

```
$ eval $(docker-machine env mhs-demo0)

$ docker network ls

NETWORK ID          NAME                DRIVER
6b07d0be843f        my-net              overlay
d0bb78cbe7bd        bridge              bridge
1c0eb8f69ebb        none                null
412c2496d0eb        host                host
97102a22e8d2        docker_gwbridge     bridge
```

4. Check the Nginx container's network interfaces.

```
$ docker exec web ip addr

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNC
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
```

```
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
22: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueu
link/ether 02:42:0a:00:09:03 brd ff:ff:ff:ff:ff:ff
inet 10.0.9.3/24 scope global eth0
    valid_lft forever preferred_lft forever
inet6 fe80::42:aff:fe00:903/64 scope link
    valid_lft forever preferred_lft forever
24: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueu
link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
inet 172.18.0.2/16 scope global eth1
    valid_lft forever preferred_lft forever
inet6 fe80::42:acff:fe12:2/64 scope link
    valid_lft forever preferred_lft forever
```

The `eth0` interface represents the container interface that is connected to the `my-net` overlay network. While the `eth1` interface represents the container interface that is connected to the `docker_gwbridge` network.

Extra Credit with Docker Compose

Please refer to the Networking feature introduced in Compose V2 format and execute the multi-host networking scenario in the swarm cluster used above.

Related information

- Understand Docker container networks
- Work with network commands
- Docker Swarm overview
- Docker Machine overview

Feedback? Suggestions? Can't find something in the docs?
Edit this page ● Request docs changes ● Get support
Rate this page: