Use overlay networks

Estimated reading time: 11 minutes

The `overlay` network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely. Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.

When you initialize a swarm or join a Docker host to an existing swarm, two new networks are created on that Docker host:

- an overlay network called `ingress`, which handles control and data traffic related to swarm services. When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the `ingress` network by default.
- a bridge network called `docker_gwbridge`, which connects the individual Docker daemon to the other daemons participating in the swarm.

You can create user-defined `overlay` networks using `docker network create`, in the same way that you can create user-defined `bridge` networks. Services or containers can be connected to more than one network at a time. Services or containers can only communicate across networks they are each connected to.

Although you can connect both swarm services and standalone containers to an overlay network, the default behaviors and configuration concerns are different. For that reason, the rest of this topic is divided into operations that apply to all overlay networks, those that apply to swarm service networks, and those that apply to overlay networks used by standalone containers.

Operations for all overlay networks

Create an overlay network

Prerequisites:

- Firewall rules for Docker daemons using overlay networks

  You need the following ports open to traffic to and from each Docker host participating on an overlay network:

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

- Before you can create an overlay network, you need to either initialize your Docker daemon as a swarm manager using `docker swarm init` or join it to an existing swarm using `docker swarm join`. Either of these creates the default `ingress` overlay network which is used by swarm services by default. You need to do this even if you never plan to use swarm services. Afterward, you can create additional user-defined overlay networks.

To create an overlay network for use with swarm services, use a command like the following:

```
$ docker network create -d overlay my-overlay
```

To create an overlay network which can be used by swarm services **or** standalone containers to communicate with other standalone containers running on other Docker daemons, add the `--attachable` flag:

```
$ docker network create -d overlay --attachable my-attachable-overlay
```

You can specify the IP address range, subnet, gateway, and other options. See `docker network create --help` for details.

Encrypt traffic on an overlay network

All swarm service management traffic is encrypted by default, using the AES algorithm (https://en.wikipedia.org/wiki/Galois/Counter_Mode) in GCM mode. Manager nodes in the swarm rotate the key used to encrypt gossip data every 12 hours.

To encrypt application data as well, add `--opt encrypted` when creating the overlay network. This enables IPSEC encryption at the level of the vxlan. This encryption imposes a non-negligible performance penalty, so you should test this option before using it in production.

When you enable overlay encryption, Docker creates IPSEC tunnels between all the nodes where tasks are scheduled for services attached to the overlay network. These tunnels also use the AES algorithm in GCM mode and manager nodes automatically rotate the keys every 12 hours.

> **Do not attach Windows nodes to encrypted overlay networks.**
>
> Overlay network encryption is not supported on Windows. If a Windows node attempts to connect to an encrypted overlay network, no error is detected but the node cannot communicate.

SWARM MODE OVERLAY NETWORKS AND STANDALONE CONTAINERS

You can use the overlay network feature with both `--opt encrypted --attachable` and attach unmanaged containers to that network:

```
$ docker network create --opt encrypted --driver overlay --attachable
```

Customize the default ingress network

Most users never need to configure the `ingress` network, but Docker 17.05 and higher allow you to do so. This can be useful if the automatically-chosen subnet conflicts with one that already exists on your network, or you need to

customize other low-level network settings such as the MTU.

Customizing the `ingress` network involves removing and recreating it. This is usually done before you create any services in the swarm. If you have existing services which publish ports, those services need to be removed before you can remove the `ingress` network.

During the time that no `ingress` network exists, existing services which do not publish ports continue to function but are not load-balanced. This affects services which publish ports, such as a WordPress service which publishes port 80.

1. Inspect the `ingress` network using `docker network inspect ingress`, and remove any services whose containers are connected to it. These are services that publish ports, such as a WordPress service which publishes port 80. If all such services are not stopped, the next step fails.

2. Remove the existing `ingress` network:

```
$ docker network rm ingress

WARNING! Before removing the routing-mesh network, make sure all
in your swarm run the same docker engine version. Otherwise, remo
be effective and functionality of newly created ingress networks
impaired.
Are you sure you want to continue? [y/N]
```

3. Create a new overlay network using the `--ingress` flag, along with the custom options you want to set. This example sets the MTU to 1200, sets the subnet to `10.11.0.0/16`, and sets the gateway to `10.11.0.2`.

```
$ docker network create \
  --driver overlay \
  --ingress \
  --subnet=10.11.0.0/16 \
  --gateway=10.11.0.2 \
  --opt com.docker.network.driver.mtu=1200 \
  my-ingress
```

> **Note**: You can name your `ingress` network something other than `ingress`, but you can only have one. An attempt to create a second one fails.

4. Restart the services that you stopped in the first step.

Customize the docker_gwbridge interface

The `docker_gwbridge` is a virtual bridge that connects the overlay networks (including the `ingress` network) to an individual Docker daemon's physical network. Docker creates it automatically when you initialize a swarm or join a Docker host to a swarm, but it is not a Docker device. It exists in the kernel of the Docker host. If you need to customize its settings, you must do so before joining the Docker host to the swarm, or after temporarily removing the host from the swarm.

1. Stop Docker.
2. Delete the existing `docker_gwbridge` interface.

```
$ sudo ip link set docker_gwbridge down
```

```
$ sudo ip link del dev docker_gwbridge
```

3. Start Docker. Do not join or initialize the swarm.
4. Create or re-create the `docker_gwbridge` bridge manually with your custom settings, using the `docker network create` command. This example uses the subnet `10.11.0.0/16`. For a full list of customizable

options, see Bridge driver options (https://docs.docker.com/engine/reference/commandline/network_create/#bridge-driver-options).

```
$ docker network create \
--subnet 10.11.0.0/16 \
--opt com.docker.network.bridge.name=docker_gwbridge \
--opt com.docker.network.bridge.enable_icc=false \
--opt com.docker.network.bridge.enable_ip_masquerade=true \
docker_gwbridge
```

5. Initialize or join the swarm. Since the bridge already exists, Docker does not create it with automatic settings.

Operations for swarm services

Publish ports on an overlay network

Swarm services connected to the same overlay network effectively expose all ports to each other. For a port to be accessible outside of the service, that port must be *published* using the `-p` or `--publish` flag on `docker service create` or `docker service update`. Both the legacy colon-separated syntax and the newer comma-separated value syntax are supported. The longer syntax is preferred because it is somewhat self-documenting.

Flag value	Description
`-p 8080:80` or `-p published=8080,target=80`	Map TCP port 80 on the service to port 8080 on the routing mesh.
`-p 8080:80/udp` or `-p published=8080,target=80,protocol=udp`	Map UDP port 80 on the service to port 8080 on the routing mesh.

Flag value	Description
`-p 8080:80/tcp -p 8080:80/udp` or `-p published=8080,target=80,protocol=tcp -p published=8080,target=80,protocol=udp`	Map TCP port 80 on the service to TCP port 8080 on the routing mesh, and map UDP port 80 on the service to UDP port 8080 on the routing mesh.

Bypass the routing mesh for a swarm service

By default, swarm services which publish ports do so using the routing mesh. When you connect to a published port on any swarm node (whether it is running a given service or not), you are redirected to a worker which is running that service, transparently. Effectively, Docker acts as a load balancer for your swarm services. Services using the routing mesh are running in *virtual IP (VIP) mode*. Even a service running on each node (by means of the `--mode global` flag) uses the routing mesh. When using the routing mesh, there is no guarantee about which Docker node services client requests.

To bypass the routing mesh, you can start a service using *DNS Round Robin (DNSRR) mode*, by setting the `--endpoint-mode` flag to `dnsrr`. You must run your own load balancer in front of the service. A DNS query for the service name on the Docker host returns a list of IP addresses for the nodes running the service. Configure your load balancer to consume this list and balance the traffic across the nodes.

Separate control and data traffic

By default, control traffic relating to swarm management and traffic to and from your applications runs over the same network, though the swarm control traffic is encrypted. You can configure Docker to use separate network interfaces for handling the two different types of traffic. When you initialize or join the swarm, specify `--advertise-addr` and `--datapath-addr` separately. You must do this for each node joining the swarm.

Operations for standalone containers on overlay networks

Attach a standalone container to an overlay network

The `ingress` network is created without the `--attachable` flag, which means that only swarm services can use it, and not standalone containers. You can connect standalone containers to user-defined overlay networks which are created with the `--attachable` flag. This gives standalone containers running on different Docker daemons the ability to communicate without the need to set up routing on the individual Docker daemon hosts.

Publish ports

Flag value	Description
`-p 8080:80`	Map TCP port 80 in the container to port 8080 on the overlay network.
`-p 8080:80/udp`	Map UDP port 80 in the container to port 8080 on the overlay network.
`-p 8080:80/sctp`	Map SCTP port 80 in the container to port 8080 on the overlay network.
`-p 8080:80/tcp -p 8080:80/udp`	Map TCP port 80 in the container to TCP port 8080 on the overlay network, and map UDP port 80 in the container to UDP port 8080 on the overlay network.

Container discovery

For most situations, you should connect to the service name, which is load-balanced and handled by all containers ("tasks") backing the service. To get a list of all tasks backing the service, do a DNS lookup for `tasks.<service-name>.`

Next steps

- Go through the overlay networking tutorial (https://docs.docker.com/network/network-tutorial-overlay/)
- Learn about networking from the container's point of view (https://docs.docker.com/config/containers/container-networking/)
- Learn about standalone bridge networks (https://docs.docker.com/network/bridge/)
- Learn about Macvlan networks (https://docs.docker.com/network/macvlan/)

network (https://docs.docker.com/glossary/?term=network), overlay (https://docs.docker.com/glossary/?term=overlay), user-defined (https://docs.docker.com/glossary/?term=user-defined), swarm (https://docs.docker.com/glossary/?term=swarm), service (https://docs.docker.com/glossary/?term=service)