Use bridge networks

Estimated reading time: 9 minutes

In terms of networking, a bridge network is a Link Layer device which forwards traffic between network segments. A bridge can be a hardware device or a software device running within a host machine's kernel.

In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network. The Docker bridge driver automatically installs rules in the host machine so that containers on different bridge networks cannot communicate directly with each other.

Bridge networks apply to containers running on the **same** Docker daemon host. For communication among containers running on different Docker daemon hosts, you can either manage routing at the OS level, or you can use an overlay network (https://docs.docker.com/network/overlay/).

When you start Docker, a default bridge network (/network/bridge/#use-the-default-bridge-network) (also called `bridge`) is created automatically, and newly-started containers connect to it unless otherwise specified. You can also create user-defined custom bridge networks. **User-defined bridge networks are superior to the default `bridge` network.**

Differences between user-defined bridges and the default bridge

- **User-defined bridges provide better isolation and interoperability between containerized applications**.

  Containers connected to the same user-defined bridge network automatically expose **all ports** to each other, and **no ports** to the outside world. This allows containerized applications to communicate with each other easily, without accidentally opening access to the outside world.

  Imagine an application with a web front-end and a database back-end. The outside world needs access to the web front-end (perhaps on port 80), but only the back-end itself needs access to the database host and port. Using a user-

defined bridge, only the web port needs to be opened, and the database application doesn't need any ports open, since the web front-end can reach it over the user-defined bridge.

If you run the same application stack on the default bridge network, you need to open both the web port and the database port, using the `-p` or `--publish` flag for each. This means the Docker host needs to block access to the database port by other means.

- **User-defined bridges provide automatic DNS resolution between containers**.

  Containers on the default bridge network can only access each other by IP addresses, unless you use the `--link` option (https://docs.docker.com/network/links/), which is considered legacy. On a user-defined bridge network, containers can resolve each other by name or alias.

  Imagine the same application as in the previous point, with a web front-end and a database back-end. If you call your containers `web` and `db`, the web container can connect to the db container at `db`, no matter which Docker host the application stack is running on.

  If you run the same application stack on the default bridge network, you need to manually create links between the containers (using the legacy `--link` flag). These links need to be created in both directions, so you can see this gets complex with more than two containers which need to communicate. Alternatively, you can manipulate the `/etc/hosts` files within the containers, but this creates problems that are difficult to debug.

- **Containers can be attached and detached from user-defined networks on the fly**.

  During a container's lifetime, you can connect or disconnect it from user-defined networks on the fly. To remove a container from the default bridge network, you need to stop the container and recreate it with different network options.

- **Each user-defined network creates a configurable bridge**.

  If your containers use the default bridge network, you can configure it, but all the containers use the same settings, such as MTU and `iptables` rules. In addition, configuring the default bridge network happens outside of Docker itself, and requires a restart of Docker.

  User-defined bridge networks are created and configured using `docker network create`. If different groups of applications have different network requirements, you can configure each user-defined bridge separately, as you create it.

- **Linked containers on the default bridge network share environment variables**.

Originally, the only way to share environment variables between two containers was to link them using the `--link` flag (https://docs.docker.com/network/links/). This type of variable sharing is not possible with user-defined networks. However, there are superior ways to share environment variables. A few ideas:

- Multiple containers can mount a file or directory containing the shared information, using a Docker volume.
- Multiple containers can be started together using `docker-compose` and the compose file can define the shared variables.
- You can use swarm services instead of standalone containers, and take advantage of shared secrets (https://docs.docker.com/engine/swarm/secrets/) and configs (https://docs.docker.com/engine/swarm/configs/).

Containers connected to the same user-defined bridge network effectively expose all ports to each other. For a port to be accessible to containers or non-Docker hosts on different networks, that port must be *published* using the `-p` or `--publish` flag.

Manage a user-defined bridge

Use the `docker network create` command to create a user-defined bridge network.

```
$ docker network create my-net
```

You can specify the subnet, the IP address range, the gateway, and other options. See the docker network create (https://docs.docker.com/engine/reference/commandline/network_create/#specify-advanced-options) reference or the output of `docker network create --help` for details.

Use the `docker network rm` command to remove a user-defined bridge network. If containers are currently connected to the network, disconnect them (/network/bridge/#disconnect-a-container-from-a-user-defined-bridge) first.

```
$ docker network rm my-net
```

> **What's really happening?**
>
> When you create or remove a user-defined bridge or connect or disconnect a container from a user-defined bridge, Docker uses tools specific to the operating system to manage the underlying network infrastructure (such as adding or removing bridge devices or configuring `iptables` rules on Linux). These details should be considered implementation details. Let Docker manage your user-defined networks for you.

Connect a container to a user-defined bridge

When you create a new container, you can specify one or more `--network` flags. This example connects a Nginx container to the `my-net` network. It also publishes port 80 in the container to port 8080 on the Docker host, so external clients can access that port. Any other container connected to the `my-net` network has access to all ports on the `my-nginx` container, and vice versa.

```
$ docker create --name my-nginx \
  --network my-net \
  --publish 8080:80 \
  nginx:latest
```

To connect a **running** container to an existing user-defined bridge, use the `docker network connect` command. The following command connects an already-running `my-nginx` container to an already-existing `my-net` network:

```
$ docker network connect my-net my-nginx
```

Disconnect a container from a user-defined bridge

To disconnect a running container from a user-defined bridge, use the `docker network disconnect` command. The following command disconnects the `my-nginx` container from the `my-net` network.

```
$ docker network disconnect my-net my-nginx
```

Use IPv6

If you need IPv6 support for Docker containers, you need to enable the option (https://docs.docker.com/config/daemon/ipv6/) on the Docker daemon and reload its configuration, before creating any IPv6 networks or assigning containers IPv6 addresses.

When you create your network, you can specify the `--ipv6` flag to enable IPv6. You can't selectively disable IPv6 support on the default `bridge` network.

Enable forwarding from Docker containers to the outside world

By default, traffic from containers connected to the default bridge network is **not** forwarded to the outside world. To enable forwarding, you need to change two settings. These are not Docker commands and they affect the Docker host's kernel.

1. Configure the Linux kernel to allow IP forwarding.

   ```
   $ sysctl net.ipv4.conf.all.forwarding=1
   ```

2. Change the policy for the `iptables` `FORWARD` policy from `DROP` to `ACCEPT` .

   ```
   $ sudo iptables -P FORWARD ACCEPT
   ```

These settings do not persist across a reboot, so you may need to add them to a start-up script.

Use the default bridge network

The default `bridge` network is considered a legacy detail of Docker and is not recommended for production use. Configuring it is a manual operation, and it has technical shortcomings (/network/bridge/#differences-between-user-defined-bridges-and-the-default-bridge).

Connect a container to the default bridge network

If you do not specify a network using the `--network` flag, and you do specify a network driver, your container is connected to the default `bridge` network by default. Containers connected to the default `bridge` network can communicate, but only by IP address, unless they are linked using the legacy `--link` flag (https://docs.docker.com/network/links/).

Configure the default bridge network

To configure the default `bridge` network, you specify options in `daemon.json`. Here is an example `daemon.json` with several options specified. Only specify the settings you need to customize.

```
{
  "bip": "192.168.1.5/24",
  "fixed-cidr": "192.168.1.5/25",
  "fixed-cidr-v6": "2001:db8::/64",
  "mtu": 1500,
  "default-gateway": "10.20.1.1",
  "default-gateway-v6": "2001:db8:abcd::89",
  "dns": ["10.20.1.2","10.20.1.3"]
}
```

Restart Docker for the changes to take effect.

Use IPv6 with the default bridge network

If you configure Docker for IPv6 support (see Use IPv6 (/network/bridge/#use-ipv6)), the default bridge network is also configured for IPv6 automatically. Unlike user-defined bridges, you can't selectively disable IPv6 on the default bridge.

Next steps

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

network (https://docs.docker.com/glossary/?term=network), bridge (https://docs.docker.com/glossary/?term=bridge), user-defined (https://docs.docker.com/glossary/?term=user-defined), standalone (https://docs.docker.com/glossary/?term=standalone)