Get $250 in credits for Cloud Sandboxes. Claim credits

Get started Guides Manuals Reference

Networking overview

Table of contents

- User-defined networks
  - Drivers
  - Connecting to multiple networks
- Published ports
- IP address and hostname
  - Subnet allocation
  - DNS services
  - Custom hosts
- Container networks

Container networking refers to the ability for containers to connect to and communicate with each other, and with non-Docker network services.

Containers have networking enabled by default, and they can make outgoing connections. A container has no information about what kind of network it's attached to, or whether its network peers are also Docker containers. A container only sees a network interface with an IP address, a gateway, a routing table, DNS services, and other networking details.

This page describes networking from the point of view of the container, and the concepts around container networking.

When Docker Engine on Linux starts for the first time, it has a single built-in network called the "default bridge" network. When you run a container without the `--network` option, it is connected to the default bridge.

Containers attached to the default bridge have access to network services outside the Docker host. They use "masquerading" which means, if the Docker host has Internet access, no additional configuration is needed for the container to have Internet access.

For example, to run a container on the default bridge network, and ping an IP address:

```
$ docker run --rm -ti busybox ping -c1 docker.c
PING docker.com (23.185.0.4): 56 data bytes
64 bytes from 23.185.0.4: seq=0 ttl=62 time=6.5

--- docker.com ping statistics ---
1 packets transmitted, 1 packets received, 0% p
round-trip min/avg/max = 6.564/6.564/6.564 ms
```

User-defined networks

With the default configuration, containers attached to the default bridge network have unrestricted network access to each other using container IP addresses. They cannot refer to each other by name.

It can be useful to separate groups of containers that should have full access to each other, but restricted access to containers in other groups.

You can create custom, user-defined networks, and connect groups of containers to the same network. Once connected to a user-defined network, containers can communicate with each other using container IP addresses or container names.

The following example creates a network using the `bridge` network driver and runs a container in that network:

```
$ docker network create -d bridge my-net
$ docker run --network=my-net -it busybox
```

Drivers

Docker Engine has a number of network drivers, as well as the default "bridge". On Linux, the following built-in network drivers are available:

Driver	Description
bridge	The default network driver.
host	Remove network isolation between the container and the Docker host.
none	Completely isolate a container from the host and other containers.
overlay	Swarm Overlay networks connect multiple Docker daemons together.
ipvlan	Connect containers to external VLANs.
macvlan	Containers appear as devices on the host's network.

More information can be found in the network driver specific pages, including their configuration options and details about their functionality.

Native Windows containers have a different set of drivers, see Windows container network drivers.

Connecting to multiple networks

Connecting a container to a network can be compared to connecting an Ethernet cable to a physical host. Just as a host can be connected to multiple Ethernet networks, a container can be connected to multiple Docker networks.

For example, a frontend container may be connected to a bridge network with external access, and a `--internal` network to communicate with containers running backend services that do not need external access.

A container may also be connected to different types of network. For example, an `ipvlan` network to provide internet access, and a `bridge` network for access to local services.

Containers can also share networking stacks, see Container networks.

When sending packets, if the destination is an address in a directly connected network, packets are sent to that network. Otherwise, packets are sent to a default gateway for routing to their destination. In the example above, the `ipvlan` network's gateway must be the default gateway.

The default gateway is selected by Docker, and may change whenever a container's network connections change. To make Docker choose a specific default gateway when creating the container or connecting a new network, set a gateway priority. See option `gw-priority` for the `docker run` and `docker network connect` commands.

The default `gw-priority` is `0` and the gateway in the network with the highest priority is the default gateway. So, when a network should always be the default gateway, it is enough to set its `gw-priority` to `1` .

```
$ docker run --network name=gw=net,gw-priority=1
$ docker network connect net2 myctr
```

Published ports

When you create or run a container using `docker create` or `docker run`, all ports of containers on bridge networks are accessible from the Docker host and other containers connected to the same network. Ports are not accessible from outside the host or, with the default configuration, from containers in other networks.

Use the `--publish` or `-p` flag to make a port available outside the host, and to containers in other bridge networks.

For more information about port mapping, including how to disable it and use direct routing to containers, see port publishing.

IP address and hostname

When creating a network, IPv4 address allocation is enabled by default, it can be disabled using `--ipv4=false`. IPv6 address allocation can be enabled using `--ipv6`.

```
$ docker network create --ipv6 --ipv4=false v6n
```

By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.

You can connect a running container to multiple networks, either by passing the `--network` flag multiple times when creating the container, or using the `docker network connect` command for already running containers. In both cases, you can use the `--ip` or `--ip6` flags to specify the container's IP address on that particular network.

In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using `--hostname`. When connecting to an existing network using `docker network connect`, you can use the `--alias` flag to specify an additional network alias for the container on that network.

Subnet allocation

Docker networks can use either explicitly configured subnets or automatically allocated ones from default pools.

Explicit subnet configuration

You can specify exact subnets when creating a network:

```
$ docker network create --ipv6 --subnet 192.0.2
```

Automatic subnet allocation

When no `--subnet` option is provided, Docker automatically selects a subnet from predefined "default address pools". These pools can be configured in `/etc/docker/daemon.json`. Docker's built-in default is equivalent to:

```
{
  "default-address-pools": [
    {"base":"172.17.0.0/16", "size": 16 },
    {"base":"172.18.0.0/16", "size": 16 },
    {"base":"172.19.0.0/16", "size": 16 },
    {"base":"172.20.0.0/14", "size": 16 },
    {"base":"172.24.0.0/14", "size": 16 },
    {"base":"172.28.0.0/14", "size": 16 },
    {"base":"192.168.0.0/16", "size": 20 }
  ]
}
```

- `base`: The subnet that can be allocated from.
- `size`: The prefix length used for each allocated subnet.

When an IPv6 subnet is required and there are no IPv6 addresses in `default-address-pools`, Docker allocates subnets from a Unique Local Address (ULA) prefix. To use specific IPv6 subnets instead, add them to your `default-address-pools`. See Dynamic IPv6 subnet allocation for more information.

Docker attempts to avoid address prefixes already in use on the host. However, you may need to customize `default-address-pools` to prevent routing conflicts in some network environments.

The default pools use large subnets, which limits the number of networks you can create. You can divide base subnets into smaller pools to support more networks.

For example, this configuration allows Docker to create 256 networks from `172.17.0.0/16`. Docker will allocate subnets `172.17.0.0/24`, `172.17.1.0/24`, and so on, up to `172.17.255.0/24`:

```
{
  "default-address-pools": [{ "base": "172.17.0
}
```

You can also request a subnet with a specific prefix length from the default pools by using unspecified addresses in the `--subnet` option:

```
$ docker network create --ipv6 --subnet ::/56 -
6686a074bb1722df38f525281130ba0e6dd8a2e298fe6d4b
$ docker network inspect mynet -f '{{json .IPAM
{
  "Subnet": "172.19.0.0/24",
```

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.

Cookies Settings | Accept All Cookies | Reject All

AI AND AGENTS
Overview
Docker Agentic Platform Experimental
Docker Sandboxes
Sandboxes API and SDK
AI governance
Model Runner
Gordon
Docker Agent
Docker Skills

APPLICATION DEVELOPMENT
Docker Desktop
Docker Engine
Install
Storage
Networking
Docker with iptables
Docker with nftables
Packet filtering and firewalls
Port publishing and mapping
Network drivers
CA certificates
Legacy container links
Containers
CLI
Daemon
Manage resources
Logs and metrics
Security
Swarm mode
Deprecated features
Docker Engine plugins
Release notes
Docker Build

Docker Build Cloud
Docker Offload
Docker Compose
Docker Extensions
Deprecated products and features
Release lifecycle

SUPPLY CHAIN SECURITY
Docker Hardened Images
Docker Hub
Docker Scout

ACCOUNTS AND ADMIN
Accounts
Subscription and billing
Security
FAQs
Support
Release notes

```
    }
]
```

> **Note**
>
> Support for unspecified addresses in `--subnet` was introduced in Docker 29.0.0. If Docker is downgraded to an older version, networks created in this way will become unusable. They can be removed and re-created, or will function again if the daemon is restored to 29.0.0 or later.

DNS services

Containers use the same DNS servers as the host by default, but you can override this with `--dns`.

By default, containers inherit the DNS settings as defined in the `/etc/resolv.conf` configuration file. Containers that attach to the default `bridge` network receive a copy of this file. Containers that attach to a custom network use Docker's embedded DNS server. The embedded DNS server forwards external DNS lookups to the DNS servers configured on the host. The embedded DNS server address is `127.0.0.11`. There is no IPv6 equivalent; the IPv4 address works even in IPv6-only containers. If an application requires an explicit DNS server address, use `127.0.0.11`.

> **Note**
>
> When you configure multiple DNS servers for a container on the default `bridge` network, the container's resolver library determines how it queries them. Some resolvers query the servers in order. Others query them in parallel and use the first response received, even if that response is `NXDOMAIN`. On a custom network, Docker's embedded DNS server queries upstream servers in order and stops after a successful response or an `NXDOMAIN` response.

You can configure DNS resolution on a per-container basis, using flags for the `docker run` or `docker create` command used to start the container. The following table describes the available `docker run` flags related to DNS configuration.

Flag	Description
`--dns`	The IP address of a DNS server. To specify multiple DNS servers, use multiple `--dns` flags. DNS requests will be forwarded from the container's network namespace so, for example, `--dns=127.0.0.1` refers to the container's own loopback address.
`--dns-search`	A DNS search domain to search non-fully qualified hostnames. To specify multiple DNS search prefixes, use multiple `--dns-search` flags.
`--dns-opt`	A key-value pair representing a DNS option and its value. See your operating system's documentation for `resolv.conf` for valid options.
`--hostname`	The hostname a container uses for itself. Defaults to the container's ID if not specified.

Custom hosts

Your container will have lines in `/etc/hosts` which define the hostname of the container itself, as well as `localhost` and a few other common things. Custom hosts, defined in `/etc/hosts` on the host machine, aren't inherited by containers. To pass additional hosts into a container, refer to add entries to container hosts file in the `docker run` reference documentation.

Container networks

In addition to user-defined networks, you can attach a container to another container's networking stack directly, using the `--network container:<name|id>` flag format.

The following flags aren't supported for containers using the `container:` networking mode:

- `--add-host`
- `--hostname`
- `--dns`
- `--dns-search`
- `--dns-option`
- `--mac-address`
- `--publish`
- `--publish-all`
- `--expose`

The following example runs a Redis container, with Redis binding to 127.0.0.1, then running the `redis-cli` command and connecting to the Redis server over 127.0.0.1.

```
$ docker run -d --name redis redis --bind 127.0
$ docker run --rm -it --network container:redis
```


Copyright © 2013-2026 Docker Inc. All rights reserved.

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.