# systemd-networkd

*systemd-networkd* is a system daemon that manages network configurations. It detects and configures network devices as they appear; it can also create virtual network devices. This service can be especially useful to set up complex network configurations for a container managed by **systemd-nspawn** or for virtual machines. It also works fine on simple connections.

## Contents



## Related articles

systemd

**systemd-nspawn**

Network bridge

Network configuration

Wireless network configuration

**Category:Network configuration**

# Basic usage

The `systemd (https://www.archlinux.org/packages/?name=systemd)` package is part of the default Arch installation and contains all needed files to operate a wired network. Wireless adapters can be setup by other services, such as **wpa_supplicant**, which are covered later in this article.

## Required services and setup

To use *systemd-networkd*, **start/enable** both `systemd-networkd.service` and `systemd-resolved.service`.

**Tip:** *systemd-resolved* is required only if you are specifying DNS entries in *.network* files or if you want to obtain DNS addresses from the network DHCP client. Alternatively you may manually manage `/etc/resolv.conf`.

For compatibility with **resolv.conf**, and redirect software which directly read this file to the local *systemd-resolved* stub DNS resolver, delete or rename the existing file and create the following symbolic link when using *systemd-resolved*:

```
# ln -s /run/systemd/resolve/stub-resolv.conf /etc/resolv.conf
```

For more information, see **systemd-resolved(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-resolved.8)**, **resolved.conf(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/resolved.conf.5)**, and **Systemd README (https://github.com/systemd/systemd/blob/master/README#L205)**.

## Configuration examples

All configurations in this section are stored as `foo.network` in `/etc/systemd/network`. For a full listing of options and processing order, see **#Configuration files** and **systemd.network(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.network.5)**.

Systemd/udev automatically assigns predictable, stable network interface names for all local Ethernet, WLAN, and WWAN interfaces. Use `networkctl list` to list the devices on the system.

After making changes to a configuration file, **restart** `systemd-networkd.service`.

**Note:**

- In the examples below, `enp1s0` is the wired adapter and `wlp2s0` is the wireless adapter. These names can be different on different systems. It is also possible to use a wildcard, e.g. `Name=en*`.
- If you want to disable IPv6, see **IPv6#systemd-networkd**.
- Set `DHCP=yes` to accept an IPv4 **and** IPv6 DHCP request to the `[Network]` section.

## Wired adapter using DHCP

`/etc/systemd/network/50-wired.network`

```
[Match]
Name=enp1s0

[Network]
DHCP=ipv4
```

## Wired adapter using a static IP

`/etc/systemd/network/25-wired.network`

```
[Match]
Name=enp1s0

[Network]
Address=10.1.10.9/24
Gateway=10.1.10.1
DNS=10.1.10.1
#DNS=8.8.8.8
```

`Address=` can be used more than once to configure multiple IPv4 or IPv6 addresses. See **#network files** or **systemd.network(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.network.5)** for more options.

## Wireless adapter

In order to connect to a wireless network with *systemd-networkd*, a wireless adapter configured with another service such as **wpa_supplicant** is required. In this example, the corresponding systemd service file that needs to be enabled is `wpa_supplicant@wlp2s0.service`. This service will run *wpa_supplicant* with the configuration file `/etc/wpa_supplicant/wpa_supplicant-wlp2s0.conf`. If this file does not exist, the service will not start.

/etc/systemd/network/25-wireless.network

```
[Match]
Name=wlp2s0

[Network]
DHCP=ipv4
```

If the wireless adapter has a static IP address, the configuration is the same (except for the interface name) as in a **wired adapter**.

## Wired and wireless adapters on the same machine

This setup will enable a DHCP IP for both a wired and wireless connection making use of the metric directive to allow the kernel to decide on-the-fly which one to use. This way, no connection downtime is observed when the wired connection is unplugged.

The kernel's route metric (same as configured with *ip*) decides which route to use for outgoing packets, in cases when several match. This will be the case when both wireless and wired devices on the system have active connections. To break the tie, the kernel uses the metric. If one of the connections is terminated, the other automatically wins without there being a gap with nothing configured (ongoing transfers may still not deal with this nicely but that is at a different OSI layer).

**Note:** The **Metric** option is for static routes while the **RouteMetric** option is for setups not using static routes.

```
/etc/systemd/network/20-wired.network
```

```
[Match]
Name=enp1s0

[Network]
DHCP=ipv4

[DHCP]
RouteMetric=10
```

```
/etc/systemd/network/25-wireless.network
```

```
[Match]
Name=wlp2s0

[Network]
DHCP=ipv4
```

```
[DHCP]
RouteMetric=20
```

### Renaming an interface

Instead of **editing udev rules**, a .link file can be used to rename an interface. A useful example is to set a predictable interface name for a USB-to-Ethernet adapter based on its MAC address, as those adapters are usually given different names depending on which USB port they are plugged into.

```
/etc/systemd/network/10-ethusb0.link
```

```
[Match]
MACAddress=12:34:56:78:90:ab

[Link]
Description=USB to Ethernet Adapter
Name=ethusb0
```

**Note:** Any user-supplied .link **must** have a lexically earlier file name than the default config `99-default.link` in order to be considered at all. For example, name the file `10-ethusb0.link` and not `ethusb0.link`.

# Configuration files

Configuration files are located in `/usr/lib/systemd/network`, the volatile runtime network directory `/run/systemd/network` and the local administration network directory `/etc/systemd/network`. Files in `/etc/systemd/network` have the highest priority.

There are three types of configuration files. They all use a format similar to **systemd unit files**.

- **.network** files. They will apply a network configuration for a *matching* device
- **.netdev** files. They will create a *virtual network device* for a *matching* environment
- **.link** files. When a network device appears, **udev** will look for the first *matching* **.link** file

They all follow the same rules:

- If **all** conditions in the `[Match]` section are matched, the profile will be activated
- an empty `[Match]` section means the profile will apply in any case (can be compared to the `*` joker)
- all configuration files are collectively sorted and processed in lexical order, regardless of the directory in which they live
- files with identical name replace each other

**Tip:**

- to override a system-supplied file in `/usr/lib/systemd/network` in a permanent manner (i.e even after upgrade), place a file with same name in `/etc/systemd/network` and symlink it to `/dev/null`

- the `*` joker can be used in `VALUE` (e.g `en*` will match any Ethernet device)
- following this **Arch-general thread (https://mailman.archlinux.org/pipermail/arch-general/2014-March/035381.html)**, the best practice is to setup specific container network settings *inside the container* with **networkd** configuration files.

## network files

These files are aimed at setting network configuration variables, especially for servers and containers.

`.network` files have the following sections: `[Match]`, `[Link]`, `[Network]`, `[Address]`, `[Route]`, and `[DHCP]`. Below are commonly configured keys for each section. See `systemd.network(5)` (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.network.5) for more information and examples.

### [Match]

- `Name=` the device name
- `Host=` the machine hostname
- `Virtualization=` check whether the system is executed in a virtualized environment or not. A `Virtualization=no` key will only apply on your host machine, while `Virtualization=yes` apply to any container or VM.

## [Link]

- `MACAddress=` useful for **MAC address spoofing**
- `MTUBytes=` setting a larger MTU value (**jumbo frames**) can significantly speed up your network transfers

## [Network]

- `DHCP=` enables the DHCP client
- `DHCPServer=` enables the DHCP server
- `DNS=` DNS server address
- `Bridge=` the bridge name
- `IPForward=` enables IP packet forwarding
- `Domains=` a list of domains to be resolved on this link

## [Address]

- `Address=` this option is **mandatory** unless DHCP is used

## [Route]

- `Gateway=` this option is **mandatory** unless DHCP is used

- `Destination=` the destination prefix of the route, possibly followed by a slash and the prefix length

If `Destination` is not present in `[Route]` section this section is treated as a default route.

**Tip:** You can put the `Address=` and `Gateway=` keys in the `[Network]` section as a short-hand if `[Address]` section contains only an Address key and `[Route]` section contains only a Gateway key.

### [DHCP]

- `UseDomains=true` can sometimes fix local name resolving when using systemd-resolved

## netdev files

These files will create virtual network devices. They have two sections: `[Match]` and `[NetDev]`. Below are commonly configured keys for each section. See **systemd.netdev(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.netdev.5)** for more information and examples.

### [Match] section

- `Host=` the hostname

- `Virtualization=` check if running in a VM

### [NetDev] section

Most common keys are:

- `Name=` the interface name. **mandatory**
- `Kind=` e.g. *bridge*, *bond*, *vlan*, *veth*, *sit*, etc. **mandatory**

## link files

These files are an alternative to custom udev rules and will be applied by **udev** as the device appears. They have two sections: `[Match]` and `[Link]`. Below are commonly configured keys for each section. See **systemd.link(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.link.5)** for more information and examples.

**Tip:** Use `udevadm test-builtin net_setup_link /sys/path/to/network/device` to diagnose problems with `.link` files.

### [Match] section

- `MACAddress=` the MAC address
- `Host=` the host name

- `Virtualization=`
- `Type=` the device type e.g. vlan

### [Link] section

- `MACAddressPolicy=` persistent or random addresses, or
- `MACAddress=` a specific address

**Note:** the system `/usr/lib/systemd/network/99-default.link` is generally sufficient for most of the basic cases.

# Usage with containers

The service is available with **systemd (https://www.archlinux.org/packages/?name=systemd)**. You will want to **enable** and **start** the `systemd-networkd.service` unit on the host and container.

For debugging purposes, it is strongly advised to **install** the **bridge-utils (https://www.archlinux.org/packages/?name=bridge-utils)**, **net-tools (https://www.archlinux.org/packages/?name=net-tools)**, and **iproute2 (https://www.archlinux.org/packages/?name=iproute2)** packages.

If you are using **systemd-nspawn**, you may need to modify the `systemd-nspawn@.service` and append boot options to the `ExecStart` line. Please refer to **systemd-nspawn(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-nspawn.1)** for an exhaustive list of options.

Note that if you want to take advantage of automatic DNS configuration from DHCP, you need to enable `systemd-resolved` and symlink `/run/systemd/resolve/resolv.conf` to `/etc/resolv.conf`. See **systemd-resolved.service(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-resolved.service.8)** for more details.

Before you start to configure your container network, it is useful to:

- disable all your **netctl** (host and container), **dhcpcd** (host and container), **systemd-networkd** (container only) and `systemd-nspawn@.service` (host only) services to avoid potential conflicts and to ease debugging
- make sure **packet forwarding** is enabled if you want to let containers access the internet. Make sure that your `.network` file does not accidentally turn off forwarding because if you do not have a `IPForward=1` setting in it, `systemd-networkd` will turn off forwarding on this interface, even if you have it enabled globally.
- make sure you do not have any **iptables** rules which can block traffic
- when the daemon is started the systemd `networkctl` command displays the status of network interfaces.

For the set-up described below,

- we will limit the output of the `ip a` command to the concerned interfaces
- we assume the *host* is your main OS you are booting to and the *container* is your guest virtual machine
- all interface names and IP addresses are only examples

## Basic DHCP network

This setup will enable a DHCP IP for host and container. In this case, both systems will share the same IP as they share the same interfaces.

```
/etc/systemd/network/MyDhcp.network
```

```
[Match]
Name=en*

[Network]
DHCP=ipv4
```

Then, **enable** and start `systemd-networkd.service` on your container.

You can of course replace `en*` by the full name of your ethernet device given by the output of the `ip link` command.

- on host and container:

```
$ ip a
```

```
2: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 14:da:e9:b5:7a:88 brd ff:ff:ff:ff:ff:ff
```

```
    inet 192.168.1.72/24 brd 192.168.1.255 scope global enp7s0
       valid_lft forever preferred_lft forever
    inet6 fe80::16da:e9ff:feb5:7a88/64 scope link
       valid_lft forever preferred_lft forever
```

By default, hostname received from the DHCP server will be used as the transient hostname.

To change it add `UseHostname=false` in section `[DHCPv4]`

```
/etc/systemd/network/MyDhcp.network
```

```
[DHCPv4]
UseHostname=false
```

If you did not want to configure a DNS in `/etc/resolv.conf` and want to rely on DHCP for setting it up, you need to **enable** `systemd-resolved.service` and symlink `/run/systemd/resolve/resolv.conf` to `/etc/resolv.conf`

```
# ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

See **systemd-resolved.service(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-resolved.service.8)** for more details.

**Note:** Users accessing a system partition via `/usr/bin/arch-chroot` from **arch-install-scripts (https://www.archlinux.org/packages/?name=arch-install-scripts)**, will need to create the symlink outside of the chroot, on the mounted partition. This is due to arch-chroot linking the file to the live environment.

# DHCP with two distinct IP

## Bridge interface

Create a virtual bridge interface

```
/etc/systemd/network/MyBridge.netdev
```

```
[NetDev]
Name=br0
Kind=bridge
```

**Restart** `systemd-networkd.service` to have systemd create the bridge.

On host and container:

```
$ ip a
```

```
3: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
    link/ether ae:bd:35:ea:0c:c9 brd ff:ff:ff:ff:ff:ff
```

Note that the interface br0 is listed but is DOWN.

## Bind ethernet to bridge

Modify the `/etc/systemd/network/MyDhcp.network` to remove the DHCP, as the bridge requires an interface to bind to with no IP, and add a key to bind this device to br0. Let us change its name to a more relevant one.

/etc/systemd/network/*MyEth*.network

```
[Match]
Name=en*

[Network]
Bridge=br0
```

## Bridge network

Create a network profile for the Bridge

/etc/systemd/network/*MyBridge*.network

```
[Match]
Name=br0

[Network]
DHCP=ipv4
```

## Add option to boot the container

As we want to give a separate IP for host and container, we need to *Disconnect* networking of the container from the host. To do this, add this option `--network-bridge=br0` to your container boot command.

```
# systemd-nspawn --network-bridge=br0 -bD /path_to/my_container
```

# Result

- on host

```
$ ip a
```

```
3: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 14:da:e9:b5:7a:88 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.87/24 brd 192.168.1.255 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::16da:e9ff:feb5:7a88/64 scope link
       valid_lft forever preferred_lft forever
6: vb-MyContainer: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br0 state UP group default qlen 1000
    link/ether d2:7c:97:97:37:25 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::d07c:97ff:fe97:3725/64 scope link
       valid_lft forever preferred_lft forever
```

- on container

```
$ ip a
```

```
2: host0: <BROADCAST,MULTICAST,ALLMULTI,AUTOMEDIA,NOTRAILERS,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 5e:96:85:83:a8:5d brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.73/24 brd 192.168.1.255 scope global host0
       valid_lft forever preferred_lft forever
    inet6 fe80::5c96:85ff:fe83:a85d/64 scope link
       valid_lft forever preferred_lft forever
```

# Notice

- we have now one IP address for `br0` on the host, and one for `host0` in the container
- two new interfaces have appeared: `vb-MyContainer` in the host and `host0` in the container. This comes as a result of the `--network-bridge=br0` option. This option *implies* another option, `--network-veth`. This means a *virtual Ethernet link* has been created between host and container.
- the DHCP address on `host0` comes from the system `/usr/lib/systemd/network/80-container-host0.network` file.
- on host

```
$ brctl show
```

```
bridge name     bridge id               STP enabled     interfaces
br0             8000.14dae9b57a88       no              enp7s0
                                                        vb-MyContainer
```

the above command output confirms we have a bridge with two interfaces binded to.

- on host

```
$ ip route
```

```
default via 192.168.1.254 dev br0
192.168.1.0/24 dev br0  proto kernel  scope link  src 192.168.1.87
```

- on container

```
$ ip route
```

```
default via 192.168.1.254 dev host0
192.168.1.0/24 dev host0  proto kernel  scope link  src 192.168.1.73
```

the above command outputs confirm we have activated `br0` and `host0` interfaces with an IP address and Gateway 192.168.1.254. The gateway address has been automatically grabbed by *systemd-networkd*

```
$ cat /run/systemd/resolve/resolv.conf
```

```
nameserver 192.168.1.254
```

## Static IP network

Setting a static IP for each device can be helpful in case of deployed web services (e.g FTP, http, SSH). Each device will keep the same MAC address across reboots if your system `/usr/lib/systemd/network/99-default.link` file has the `MACAddressPolicy=persistent` option (it has by default). Thus, you will easily route any service on your Gateway to the desired device.

The following configuration needs to be done for this setup:

- on host

The configuration is very similar to that of **#DHCP with two distinct IP**. First, a virtual bridge interface needs to be created and the main physical interface needs to be bound to it. This task can be accomplished with the following two files, with contents equal to those available at the DHCP section.

```
/etc/systemd/network/MyBridge.netdev
/etc/systemd/network/MyEth.network
```

Next, you need to configure the IP and DNS of the newly created virtual bridge interface. The following *MyBridge*.network provides an example configuration:

```
/etc/systemd/network/MyBridge.network
```

```
[Match]
Name=br0

[Network]
DNS=192.168.1.254
Address=192.168.1.87/24
Gateway=192.168.1.254
```

- on container

First, we shall get rid of the system `/usr/lib/systemd/network/80-container-host0.network` file, which provides a DHCP configuration for the default network interface of the container. To do it in a permanent way (e.g. even after **systemd (https://www.archlinux.org/packages/?name=systemd)** upgrades), do the following on the container. This will mask the file

`/usr/lib/systemd/network/80-container-host0.network` since files of the same name in `/etc/systemd/network` take priority over `/usr/lib/systemd/network`. Keep in mind that this file can be kept if you only want a static IP on the host, and want the IP address of your containers to be assigned via DHCP.

```
# ln -sf /dev/null /etc/systemd/network/80-container-host0.network
```

Then, configure an static IP for the default `host0` network interface and **enable and start** `systemd-networkd` on your container. An example configuration is provided below:

/etc/systemd/network/*MyVeth*.network

```
[Match]
Name=host0

[Network]
DNS=192.168.1.254
Address=192.168.1.94/24
Gateway=192.168.1.254
```

# Management, status information, and desktop integration

*systemd-networkd* doesn't have a proper interactive management interface via either command-line or GUI. *networkctl* (via CLI) just offers a simple dump of the network interface states.

When *networkd* is configured with **wpa_supplicant**, both *wpa_cli* and *wpa_gui* offer the ability to associate and reconfigure WLAN interfaces dynamically.

**networkd-notify (https://github.com/wavexx/networkd-notify)** can generate simple notifications in response to network interface state changes (such as connection/disconnection and re-association).

The `networkd-dispatcher (https://aur.archlinux.org/packages/networkd-dispatcher/)`[AUR] daemon allows executing scripts in response to network interface state changes, similar to NetworkManager-dispatcher.

For the DNS resolver *systemd-resolved*, information about current DNS servers can be visualized with `systemd-resolve --status`.

# Troubleshooting

## Mount services at boot fail

If running services like **Samba**/**NFS** which fail if they are started before the network is up, you may want to **enable** the `systemd-networkd-wait-online.service`. This is, however, rarely necessary because most networked daemons start up okay, even if the network has not been configured yet.

### systemd-resolve not searching the local domain

systemd-resolved may not search the local domain when given just the hostname, even when `UseDomains=yes` or `Domains=[domain-list]` is present in the appropriate `.network` file, and that file produces the expected `search [domain-list]` in `resolv.conf`. If you run into this problem:

- Trim `/etc/nsswitch.conf`'s `hosts` database (e.g., by removing `[!UNAVAIL=return]` option after `resolve` service)
- Switch to using fully-qualified domain names
- Use `/etc/hosts` to resolve hostnames
- Fall back to using glibc's `dns` instead of using systemd's `resolve`

## See also

- **systemd.networkd man page (http://www.freedesktop.org/software/systemd/man/systemd-networkd.service.html)**
- **Tom Gundersen, main systemd-networkd developer, G+ home page (https://plus.google.com/u/0/+TomGundersen/posts)**
- **Tom Gundersen posts on Core OS blog (https://coreos.com/blog/intro-to-systemd-networkd/)**
- **How to set up systemd-networkd with wpa_supplicant (https://bbs.archlinux.org/viewtopic.php?pid=1393759#p1393759)** (WonderWoofy's walkthrough on Arch forums)

Retrieved from "https://wiki.archlinux.org/index.php?title=Systemd-networkd&oldid=509911"