freedesktop.org

www/ Software/ systemd/ NetworkTarget

Edit | Page History | Repo Info

Back to systemd

Running Services After the Network is up

So you have configured your service to run after `network.target` but it still gets run before your network is up? And now you are wondering why that is and what you can do about it?

LSB init scripts know the `$network` facility. As this facility is defined only very unprecisely people tend to have different ideas what it is supposed to mean. Here are a couple of ideas people came up with so far:

- The network management software is up
- All "configured" network interfaces are up and an IP address has been assigned to each
- All discovered local hardware interfaces that have a link beat have an IP address assigned, independently whether there is actually any explicit local configuration for them
- The network has been set up precisely to the level that a DNS server is reachable
- Similar, but some specific site-specific server is reachable
- Similar, but "the Internet" is reachable
- All "configured" ethernet devices are up, but all "configured" PPP links which are supposed to also start at boot don't have to be yet
- A certain "profile" is enabled and some condition of the above holds. If another "profile" is enabled a different condition would have to be checked
- Based on the location of the system a different set of configuration should be up or checked for
- At least one global IPv4 address is configured
- At least one global IPv6 address is configured
- At least one global IPv4 or IPv6 address is configured

And so on and so on. All these are valid approaches to the question "When is the network up?", but none of them would be useful to be good

as generic default.

Modern networking tends to be highly dynamic: machines are moved between networks, network configuration changes, hardware is added and removed, virtual networks are set up, reconfigured and shut down again. Network connectivity is not unconditionally and continuously available, and a machine is connected to different networks at different times. This is particularly true for mobile hardware such as handsets, tablets and laptops, but also for embedded and servers. Software that is written under the assumption that network connectivity is available continuously and never changes is hence not up-to-date with modern standards. Well-written software should be able to handle dynamic configuration changes. It should react to changing network configuration and make the best of it. If it cannot reach a server it must retry. If network configuration connectivity is lost it must react. Reacting to local network configuration changes in daemon code is not particularly hard. In fact many well-known network-facing services running on Linux have been doing this for decades. A service written like this is robust, can be started at any time, and will always do the best of the circumstances it is running in.

`$network` is a mechanism that is required only to deal with software that assumes continuous network is available (i.e. of the simple non-well-written kind). Which facet of it it requires is undefined. An IMAP server might just require a certain IP to be assigned so that it can listen on it. OTOH a network file system client might need DNS up, and the service to contact up, as well. What precisely is required for `$network` is not obvious and can be different things depending on local configuration.

A robust system boots up independently of external services. More specifically, if a network DHCP server does not react this should not slow down boot on most setups, but only for those where network connectivity is strictly needed (for example, because the host actually boots from the network).

Concepts in systemd

In systemd, three target units take the role of `$network`:

- `network.target` has very little meaning during start-up. It only indicates that the network management stack is up after it has been reached. Whether any network interfaces are already configured when it is reached is undefined. It's primary purpose is for ordering things properly at shutdown: since the shutdown ordering of units in systemd is the reverse of the startup ordering, any unit that is order After=network.target can be sure that it is stopped before the network is shut down if the system is powered off. This allows services to cleanly terminate connections before going down, instead of abruptly losing connectivity for ongoing connections, leaving them in an undefined state. Note that `network.target` is a *passive* unit: you cannot start it directly and it is not pulled in by any services that want to make use of the network. Instead, it is pulled in by the network management service itself. Services using the network should hence simply place an After=network.target dependency in their unit files, and avoid any Wants=network.target or even Requires=network.target.

- `network-online.target` is a target that actively waits until the nework is "up", where the definition of "up" is defined by the network management software. Usually it indicates a configured, routable IP address of some kind. It's primary purpose is to actively delay

activation of services until the network is set up. It is an *active* target, meaning that is may be pulled in by the services requiring the network to be up, but is not pulled in by the network management service itself. By default all remote mounts defined in `/etc/fstab` pull this service in, in order to make sure the network is up before it is attempted to connect to a network share. Note that normally, if no service requires it, and if not remote mount point is configured this target is not pulled into the boot, thus avoiding any delays during boot should the network not be available. It is strongly recommended not to pull in this target too liberally: for example network server software should generally not pull this in (since server software generally is happy to accept local connections even before any routable network interface is up), it's primary purpose is network client software that cannot operate without network.

- `network-pre.target` is a target that may be used to order services before any network interface is configured. It's primary purpose is for usage with firewall services that want to establish a firewall before any network interface is up. It's a *passive* unit: you cannot start it directly and it is not pulled in by the the network management service, but by the service that wants to run before it. Network management services hence should set After=network-pre.target, but avoid any Wants=network-pre.target or even Requires=network-pre.target. Services that want to be run before the network is configured should place Before=network-pre.target and also set Wants=network-pre.target to pull it in. This way, unless there's actually a service that needs to be ordered before the network is up the target is not pulled in, hence avoiding any unnecessary synchronization point.

Whenever systemd encounters a `$network` dependency in LSB headers of init scripts it will translate this to a Wants= and After= dependency on `network-online.target` hence staying relatively close to traditional LSB behaviour.

For more details, see the systemd.special(7) man page.

Cut the crap! How do I make `network.target` work for me?

Well, that depends on your setup and the services you plan to run after it (see above). Many network management solutions provide a way to unconditionally pull in network-online.target, and thus upgrading the effect of network.target to the effect of network-online.target.

If you use NetworkManager you can do this by enabling `NetworkManager-wait-online.service`:

```
systemctl enable NetworkManager-wait-online.service
```

If you use systemd-networkd you can do this by enabling `systemd-networkd-wait-online.service`:

```
systemctl enable systemd-networkd-wait-online.service
```

This will ensure that all configured network devices are up and have an IP address assigned before boot continues. This service will time out after 90s. Enabling this service might considerably delay your boot even if the timeout is not reached. Both services are disabled by default.

Alternatively, you can change your service that needs the network to be up, to include After=network-online.target and Wants=network-online.target.

What does this mean for me, a Developer?

If you are a developer, instead of wondering what to do about `network.target`, please just fix your program to be friendly to dynamically changing network configuration. That way you will make your users happy because things just start to work, and you will get fewer bug reports as your stuff is just rock solid. You also make the boot faster for your users, as they don't have to delay arbitrary services for the network anymore (which is particularly annoying for folks with slow address assignment replies from a DHCP server).

Here are a couple of possible approaches:

- Watch rtnetlink and react properly to network configuration changes as they happen. This is usually the nicest solution, but not always the easiest.
- If you write a server: listen on [::], [::1], 0.0.0.0 and 127.0.0.1 only. These pseudo-addresses are unconditionally available. If you always bind to these addresses you will have code that doesn't have to react to network changes, as all you listen on is catch-all and private addresses.
- If you write a server: if you want to listen on other, explicitly configured addresses, consider using the IP_FREEBIND sockopt functionality of the Linux kernel. This allows your code to bind to an address even if it is not actually (yet or ever) configured locally. This also makes your code robust towards network configuration changes.

Last edited Wed Jun 11 06:22:03 2014