Network Configuration

Contents

Proxmox VE is using the Linux network stack. This provides a lot of flexibility on how to set up the network on the Proxmox VE nodes. The configuration can be done either via the GUI, or by manually editing the file `/etc/network/interfaces`, which contains the whole network configuration. The `interfaces(5)` manual page contains the complete format description. All Proxmox VE tools try hard to keep direct user modifications, but using the GUI is still preferable, because it protects you from errors.

A Linux bridge interface (commonly called *vmbrX*) is needed to connect guests to the underlying physical network. It can be thought of as a virtual switch which the guests and physical interfaces are connected to. This section provides some examples on how the network can be set up to accomodate different use cases like redundancy with a *bond*, *vlans* or *routed* and *NAT* setups.

The Software Defined Network is an option for more complex virtual networks in Proxmox VE clusters.

It's discouraged to use the traditional Debian tools `ifup` and `ifdown` if unsure, as they have some pitfalls like interupting all guest traffic on `ifdown vmbrX` but not reconnecting those guest again when doing `ifup` on the same bridge later.

Apply Network Changes

Proxmox VE does not write changes directly to `/etc/network/interfaces`. Instead, we write into a temporary file called `/etc/network/interfaces.new`, this way you can do many related changes at once. This also allows to ensure your changes are correct before applying, as a wrong network configuration may render a node inaccessible.

Live-Reload Network with ifupdown2

With the recommended *ifupdown2* package (default for new installations since Proxmox VE 7.0), it is possible to apply network configuration changes without a reboot. If you change the network configuration via the GUI, you can click the *Apply Configuration* button. This will move changes from the staging `interfaces.new` file to `/etc/network/interfaces` and apply them live.

If you made manual changes directly to the `/etc/network/interfaces` file, you can apply them by running `ifreload -a`

> If you installed Proxmox VE on top of Debian, or upgraded to Proxmox VE 7.0 from an older Proxmox VE installation, make sure *ifupdown2* is installed: `apt install ifupdown2`

Reboot Node to Apply

Another way to apply a new network configuration is to reboot the node. In that case the systemd service `pvenetcommit` will activate the staging `interfaces.new` file before the `networking` service will apply that configuration.

Naming Conventions

We currently use the following naming conventions for device names:

- Ethernet devices: `en*`, systemd network interface names. This naming scheme is used for new Proxmox VE installations since version 5.0.
- Ethernet devices: `eth[N]`, where 0 ≤ N (`eth0`, `eth1`, …) This naming scheme is used for Proxmox VE hosts which were installed before the 5.0 release. When upgrading to 5.0, the names are kept as-is.
- Bridge names: Commonly `vmbr[N]`, where 0 ≤ N ≤ 4094 (`vmbr0` - `vmbr4094`), but you can use any alphanumeric string that starts with a character and is at most 10 characters long.
- Bonds: `bond[N]`, where 0 ≤ N (`bond0`, `bond1`, …)
- VLANs: Simply add the VLAN number to the device name, separated by a period (`eno1.50`, `bond1.30`)

This makes it easier to debug networks problems, because the device name implies the device type.

Systemd Network Interface Names

Systemd defines a versioned naming scheme for network device names. The scheme uses the two-character prefix `en` for Ethernet network devices. The next characters depends on the device driver, device location and other attributes. Some possible patterns are:

- `o<index>[n<phys_port_name>|d<dev_port>]` — devices on board
- `s<slot>[f<function>][n<phys_port_name>|d<dev_port>]` — devices by hotplug id
- `[P<domain>]p<bus>s<slot>[f<function>][n<phys_port_name>|d<dev_port>]` — devices by bus id
- `x<MAC>` — devices by MAC address

Some examples for the most common patterns are:

- `eno1` — is the first on-board NIC
- `enp3s0f1` — is function 1 of the NIC on PCI bus 3, slot 0

For a full list of possible device name patterns, see the systemd.net-naming-scheme(7) manpage.

A new version of systemd may define a new version of the network device naming scheme, which it then uses by default. Consequently, updating to a newer systemd version, for example during a major Proxmox VE upgrade, can change the names of network devices and require adjusting the network configuration. To avoid name changes due to a new version of the naming scheme, you can manually pin a particular naming scheme version (see below).

However, even with a pinned naming scheme version, network device names can still change due to kernel or driver updates. In order to avoid name changes for a particular network device altogether, you can manually override its name using a link file (see below).

For more information on network interface names, see Predictable Network Interface Names.

Pinning a specific naming scheme version

You can pin a specific version of the naming scheme for network devices by adding the `net.naming-scheme=<version>` parameter to the kernel command line. For a list of naming scheme versions, see the systemd.net-naming-scheme(7) manpage.

For example, to pin the version `v252`, which is the latest naming scheme version for a fresh Proxmox VE 8.0 installation, add the following kernel command-line parameter:

```
net.naming-scheme=v252
```

See also this section on editing the kernel command line. You need to reboot for the changes to take effect.

Overriding network device names

You can manually assign a name to a particular network device using a custom systemd.link file. This overrides the name that would be assigned according to the latest network device naming scheme. This way, you can avoid naming changes due to kernel updates, driver updates or newer versions of the naming scheme.

Custom link files should be placed in `/etc/systemd/network/` and named `<n>-<id>.link`, where `n` is a priority smaller than `99` and `id` is some identifier. A link file has two sections: `[Match]` determines which interfaces the file will apply to; `[Link]` determines how these interfaces should be configured, including their naming.

To assign a name to a particular network device, you need a way to uniquely and permanently identify that device in the `[Match]` section. One possibility is to match the device's MAC address using the `MACAddress` option, as it is unlikely to change. Then, you can assign a name using the `Name` option in the `[Link]` section.

For example, to assign the name `enwan0` to the device with MAC address `aa:bb:cc:dd:ee:ff`, create a file `/etc/systemd/network/10-enwan0.link` with the following contents:

```
[Match]
MACAddress=aa:bb:cc:dd:ee:ff

[Link]
Name=enwan0
```

Do not forget to adjust `/etc/network/interfaces` to use the new name. You need to reboot the node for the change to take effect.

> note: It is recommended to assign a name starting with `en` or `eth` so that Proxmox VE recognizes the interface as a physical network device which can then be configured via the GUI. Also, you should ensure that the name will not clash with other interface names in the future. One possibility is to assign a name that does not match any name pattern that systemd uses for network interfaces (see above), such as `enwan0` in the example above.

For more information on link files, see the systemd.link(5) manpage.

Choosing a network configuration

Depending on your current network organization and your resources you can choose either a bridged, routed, or masquerading networking setup.

Proxmox VE server in a private LAN, using an external gateway to reach the internet

The **Bridged** model makes the most sense in this case, and this is also the default mode on new Proxmox VE installations. Each of your Guest system will have a virtual interface attached to the Proxmox VE bridge. This is similar in effect to having the Guest network card directly connected to a new switch on your LAN, the Proxmox VE host playing the role of the switch.

Proxmox VE server at hosting provider, with public IP ranges for Guests

For this setup, you can use either a **Bridged** or **Routed** model, depending on what your provider allows.

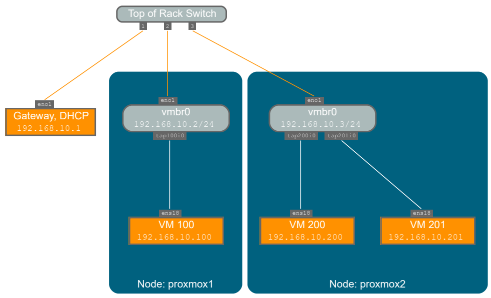
Proxmox VE server at hosting provider, with a single public IP address

In that case the only way to get outgoing network accesses for your guest systems is to use **Masquerading**. For incoming network access to your guests, you will need to configure **Port Forwarding**.

For further flexibility, you can configure VLANs (IEEE 802.1q) and network bonding, also known as "link aggregation". That way it is possible to build complex and flexible virtual networks.

Default Configuration using a Bridge

Bridges are like physical network switches implemented in software. All virtual guests can share a single bridge, or you can create multiple bridges to separate network domains. Each host can have up to 4094 bridges.



The installation program creates a single bridge named `vmbr0`, which is connected to the first Ethernet card. The corresponding configuration in `/etc/network/interfaces` might look like this:

```
auto lo
iface lo inet loopback

iface eno1 inet manual

auto vmbr0
iface vmbr0 inet static
        address 192.168.10.2/24
        gateway 192.168.10.1
        bridge-ports eno1
        bridge-stp off
        bridge-fd 0
```

Virtual machines behave as if they were directly connected to the physical network. The network, in turn, sees each virtual machine as having its own MAC, even though there is only one network cable connecting all of these VMs to the network.

Routed Configuration

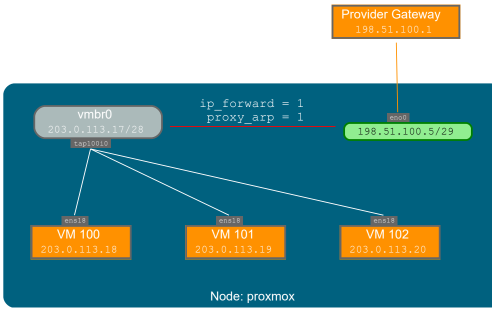
Most hosting providers do not support the above setup. For security reasons, they disable networking as soon as they detect multiple MAC addresses on a single interface.

> Some providers allow you to register additional MACs through their management interface. This avoids the problem, but can be clumsy to configure because you need to register a MAC for each of your VMs.

You can avoid the problem by “routing” all traffic via a single interface. This makes sure that all network packets use the same MAC address.

A common scenario is that you have a public IP (assume `198.51.100.5` for this example), and an

```
auto lo
iface lo inet loopback

auto eno0
iface eno0 inet static
        address  198.51.100.5/29
        gateway  198.51.100.1
        post-up echo 1 > /proc/sys/net/ipv4/ip_forward
        post-up echo 1 > /proc/sys/net/ipv4/conf/eno0/proxy_arp


auto vmbr0
iface vmbr0 inet static
        address  203.0.113.17/28
        bridge-ports none
        bridge-stp off
        bridge-fd 0
```



Masquerading (NAT) with `iptables`

Masquerading allows guests having only a private IP address to access the network by using the host IP address for outgoing traffic. Each outgoing packet is rewritten by `iptables` to appear as originating from the host, and responses are rewritten accordingly to be routed to the original sender.

```
auto lo
iface lo inet loopback

auto eno1
#real IP address
iface eno1 inet static
        address  198.51.100.5/24
        gateway  198.51.100.1

auto vmbr0
#private sub network
iface vmbr0 inet static
        address  10.10.10.1/24
        bridge-ports none
        bridge-stp off
        bridge-fd 0

        post-up   echo 1 > /proc/sys/net/ipv4/ip_forward
        post-up   iptables -t nat -A POSTROUTING -s '10.10.10.0/24' -o eno1 -j MASQUERADE
        post-down iptables -t nat -D POSTROUTING -s '10.10.10.0/24' -o eno1 -j MASQUERADE
```

Note: In some masquerade setups with firewall enabled, conntrack zones might be needed for outgoing connections. Otherwise the firewall could block outgoing connections since they will prefer the `POSTROUTING` of the VM bridge (and not `MASQUERADE`).

Adding these lines in the `/etc/network/interfaces` can fix this problem:

```
post-up   iptables -t raw -I PREROUTING -i fwbr+ -j CT --zone 1
post-down iptables -t raw -D PREROUTING -i fwbr+ -j CT --zone 1
```

For more information about this, refer to the following links:

Netfilter Packet Flow

Patch on netdev-list introducing conntrack zones

Blog post with a good explanation by using TRACE in the raw table

Linux Bond

Bonding (also called NIC teaming or Link Aggregation) is a technique for binding multiple NIC's to a single network device. It is possible to achieve different goals, like make the network fault-tolerant, increase the performance or both together.

High-speed hardware like Fibre Channel and the associated switching hardware can be quite expensive. By doing link aggregation, two NICs can appear as one logical interface, resulting in double speed. This is a native Linux kernel feature that is supported by most switches. If your nodes have multiple Ethernet ports, you can distribute your points of failure by running network cables to different switches and the bonded connection will failover to one cable or the other in case of network trouble.

Aggregated links can improve live-migration delays and improve the speed of replication of data between Proxmox VE Cluster nodes.

There are 7 modes for bonding:

- **Round-robin (balance-rr):** Transmit network packets in sequential order from the first available network interface (NIC) slave through the last. This mode provides load balancing and fault tolerance.

- **Active-backup (active-backup):** Only one NIC slave in the bond is active. A different slave becomes active if, and only if, the active slave fails. The single logical bonded interface's MAC address is externally visible on only one NIC (port) to avoid distortion in the network switch. This mode provides fault tolerance.

- **XOR (balance-xor):** Transmit network packets based on [(source MAC address XOR'd with destination MAC address) modulo NIC slave count]. This selects the same NIC slave for each destination MAC address. This mode provides load balancing and fault tolerance.

- **Broadcast (broadcast):** Transmit network packets on all slave network interfaces. This mode provides fault tolerance.

- **IEEE 802.3ad Dynamic link aggregation (802.3ad)(LACP):** Creates aggregation groups that share the same speed and duplex settings. Utilizes all slave network interfaces in the active aggregator group according to the 802.3ad specification.

- **Adaptive transmit load balancing (balance-tlb):** Linux bonding driver mode that does not require any special network-switch support. The outgoing network packet traffic is distributed according to the current load (computed relative to the speed) on each network interface slave. Incoming traffic is received by one currently designated slave network interface. If this receiving slave fails, another slave takes over the MAC address of the failed receiving slave.

- **Adaptive load balancing (balance-alb):** Includes balance-tlb plus receive load balancing (rlb) for IPV4 traffic, and does not require any special network switch support. The receive load balancing is achieved by ARP negotiation. The bonding driver intercepts the ARP Replies sent by the local system on their way out and overwrites the source hardware address with the unique hardware address of one of the NIC slaves in the single logical bonded interface such that different network-peers use different MAC addresses for their network packet traffic.

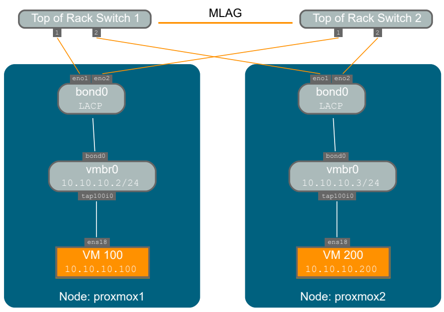
If your switch support the LACP (IEEE 802.3ad) protocol then we recommend using the corresponding bonding mode (802.3ad). Otherwise you should generally use the active-backup mode.

For the cluster network (Corosync) we recommend configuring it with multiple networks. Corosync does not need a bond for network reduncancy as it can switch between networks by itself, if one becomes unusable.

The following bond configuration can be used as distributed/shared storage network. The benefit would be that you get more speed and the network will be fault-tolerant.

Example: Use bond with fixed IP address

```
auto lo
iface lo inet loopback

iface eno1 inet manual

iface eno2 inet manual

iface eno3 inet manual

auto bond0
iface bond0 inet static
      bond-slaves eno1 eno2
      address  192.168.1.2/24
      bond-miimon 100
      bond-mode 802.3ad
      bond-xmit-hash-policy layer2+3

auto vmbr0
iface vmbr0 inet static
        address  10.10.10.2/24
        gateway  10.10.10.1
        bridge-ports eno3
        bridge-stp off
        bridge-fd 0
```



Another possibility it to use the bond directly as bridge port. This can be used to make the guest network fault-tolerant.

Example: Use a bond as bridge port

```
auto lo
iface lo inet loopback

iface eno1 inet manual

iface eno2 inet manual

auto bond0
iface bond0 inet manual
      bond-slaves eno1 eno2
      bond-miimon 100
      bond-mode 802.3ad
      bond-xmit-hash-policy layer2+3
```

```
        address  10.10.10.2/24
        gateway  10.10.10.1
        bridge-ports bond0
        bridge-stp off
        bridge-fd 0
```

VLAN 802.1Q

A virtual LAN (VLAN) is a broadcast domain that is partitioned and isolated in the network at layer two. So it is possible to have multiple networks (4096) in a physical network, each independent of the other ones.

Each VLAN network is identified by a number often called *tag*. Network packages are then *tagged* to identify which virtual network they belong to.

VLAN for Guest Networks

Proxmox VE supports this setup out of the box. You can specify the VLAN tag when you create a VM. The VLAN tag is part of the guest network configuration. The networking layer supports different modes to implement VLANs, depending on the bridge configuration:

- **VLAN awareness on the Linux bridge:** In this case, each guest's virtual network card is assigned to a VLAN tag, which is transparently supported by the Linux bridge. Trunk mode is also possible, but that makes configuration in the guest necessary.
- **"traditional" VLAN on the Linux bridge:** In contrast to the VLAN awareness method, this method is not transparent and creates a VLAN device with associated bridge for each VLAN. That is, creating a guest on VLAN 5 for example, would create two interfaces eno1.5 and vmbr0v5, which would remain until a reboot occurs.
- **Open vSwitch VLAN:** This mode uses the OVS VLAN feature.
- **Guest configured VLAN:** VLANs are assigned inside the guest. In this case, the setup is completely done inside the guest and can not be influenced from the outside. The benefit is that you can use more than one VLAN on a single virtual NIC.

VLAN on the Host

To allow host communication with an isolated network. It is possible to apply VLAN tags to any network device (NIC, Bond, Bridge). In general, you should configure the VLAN on the interface with the least abstraction layers between itself and the physical NIC.

For example, in a default configuration where you want to place the host management address on a separate VLAN.

Example: Use VLAN 5 for the Proxmox VE management IP with traditional Linux bridge

```
iface eno1 inet manual

iface eno1.5 inet manual

auto vmbr0v5
iface vmbr0v5 inet static
        address  10.10.10.2/24
        gateway  10.10.10.1
        bridge-ports eno1.5
        bridge-stp off
        bridge-fd 0

auto vmbr0
iface vmbr0 inet manual
        bridge-ports eno1
        bridge-stp off
        bridge-fd 0
```

Example: Use VLAN 5 for the Proxmox VE management IP with VLAN aware Linux bridge

```
auto lo
iface lo inet loopback

iface eno1 inet manual


auto vmbr0.5
iface vmbr0.5 inet static
        address  10.10.10.2/24
        gateway  10.10.10.1

auto vmbr0
iface vmbr0 inet manual
        bridge-ports eno1
        bridge-stp off
        bridge-fd 0
        bridge-vlan-aware yes
        bridge-vids 2-4094
```

The next example is the same setup but a bond is used to make this network fail-safe.

Example: Use VLAN 5 with bond0 for the Proxmox VE management IP with traditional Linux bridge

```
auto lo
iface lo inet loopback

iface eno1 inet manual

iface eno2 inet manual

auto bond0
iface bond0 inet manual
      bond-slaves eno1 eno2
      bond-miimon 100
      bond-mode 802.3ad
      bond-xmit-hash-policy layer2+3

iface bond0.5 inet manual

auto vmbr0v5
iface vmbr0v5 inet static
        address  10.10.10.2/24
        gateway  10.10.10.1
        bridge-ports bond0.5
        bridge-stp off
        bridge-fd 0

auto vmbr0
iface vmbr0 inet manual
        bridge-ports bond0
        bridge-stp off
```

Disabling IPv6 on the Node

Proxmox VE works correctly in all environments, irrespective of whether IPv6 is deployed or not. We recommend leaving all settings at the provided defaults.

Should you still need to disable support for IPv6 on your node, do so by creating an appropriate `sysctl.conf (5)` snippet file and setting the proper sysctls, for example adding `/etc/sysctl.d/disable-ipv6.conf` with content:

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
```

This method is preferred to disabling the loading of the IPv6 module on the kernel commandline.

Disabling MAC Learning on a Bridge

By default, MAC learning is enabled on a bridge to ensure a smooth experience with virtual guests and their networks.

But in some environments this can be undesired. Since Proxmox VE 7.3 you can disable MAC learning on the bridge by setting the ‘bridge-disable-mac-learning 1` configuration on a bridge in `/etc/network/interfaces’, for example:

```
# ...

auto vmbr0
iface vmbr0 inet static
        address  10.10.10.2/24
        gateway  10.10.10.1
        bridge-ports ens18
        bridge-stp off
        bridge-fd 0
        bridge-disable-mac-learning 1
```

Once enabled, Proxmox VE will manually add the configured MAC address from VMs and Containers to the bridges forwarding database to ensure that guest can still use the network - but only when they are using their actual MAC address.