



Red Hat Enterprise Linux 10-beta

Configuring and managing networking

Managing network interfaces and advanced networking features

Red Hat Enterprise Linux 10-beta Configuring and managing networking

Managing network interfaces and advanced networking features

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Using the networking capabilities of Red Hat Enterprise Linux (RHEL), you can configure your host to meet your organization's network and security requirements. This includes different network types and advanced networking features, such as policy-based routing and Multipath TCP (MPTCP).

Table of Contents

RHEL BETA RELEASE	7
CHAPTER 1. IMPLEMENTING CONSISTENT NETWORK INTERFACE NAMING	8
1.1. HOW THE UDEV DEVICE MANAGER RENAMES NETWORK INTERFACES	8
1.2. NETWORK INTERFACE NAMING POLICIES	9
1.3. NETWORK INTERFACE NAMING SCHEMES	10
1.4. SWITCHING TO A DIFFERENT NETWORK INTERFACE NAMING SCHEME	10
1.5. CUSTOMIZING THE PREFIX FOR ETHERNET INTERFACES DURING INSTALLATION	12
1.6. CONFIGURING USER-DEFINED NETWORK INTERFACE NAMES BY USING UDEV RULES	13
1.7. CONFIGURING USER-DEFINED NETWORK INTERFACE NAMES BY USING SYSTEMD LINK FILES	14
1.8. ASSIGNING ALTERNATIVE NAMES TO A NETWORK INTERFACE BY USING SYSTEMD LINK FILES	16
CHAPTER 2. CONFIGURING AN ETHERNET CONNECTION	19
2.1. CONFIGURING AN ETHERNET CONNECTION BY USING NMCLI	19
2.2. CONFIGURING AN ETHERNET CONNECTION BY USING NMTUI	22
2.3. CONFIGURING AN ETHERNET CONNECTION BY USING CONTROL-CENTER	25
2.4. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING NMSTATECTL	27
2.5. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME	30
2.6. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH	32
2.7. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING NMSTATECTL	35
2.8. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH AN INTERFACE NAME	37
2.9. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE NETWORK RHEL SYSTEM ROLE WITH A DEVICE PATH	39
2.10. CONFIGURING MULTIPLE ETHERNET INTERFACES BY USING A SINGLE CONNECTION PROFILE BY INTERFACE NAME	41
2.11. CONFIGURING A SINGLE CONNECTION PROFILE FOR MULTIPLE ETHERNET INTERFACES USING PCI IDS	42
CHAPTER 3. CONFIGURING A NETWORK BOND	44
3.1. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES	44
3.2. UPSTREAM SWITCH CONFIGURATION DEPENDING ON THE BONDING MODES	44
3.3. CONFIGURING A NETWORK BOND BY USING NMCLI	45
3.4. CONFIGURING A NETWORK BOND BY USING THE RHEL WEB CONSOLE	48
3.5. CONFIGURING A NETWORK BOND BY USING NMTUI	51
3.6. CONFIGURING A NETWORK BOND BY USING NMSTATECTL	54
3.7. CONFIGURING A NETWORK BOND BY USING THE NETWORK RHEL SYSTEM ROLE	56
3.8. THE DIFFERENT NETWORK BONDING MODES	58
3.9. THE XMIT_HASH_POLICY BONDING PARAMETER	60
CHAPTER 4. CONFIGURING VLAN TAGGING	63
4.1. CONFIGURING VLAN TAGGING BY USING NMCLI	63
4.2. CONFIGURING NESTED VLANS BY USING NMCLI	65
4.3. CONFIGURING VLAN TAGGING BY USING THE RHEL WEB CONSOLE	67
4.4. CONFIGURING VLAN TAGGING BY USING NMTUI	69
4.5. CONFIGURING VLAN TAGGING BY USING NMSTATECTL	73
4.6. CONFIGURING VLAN TAGGING BY USING THE NETWORK RHEL SYSTEM ROLE	75
CHAPTER 5. CONFIGURING A NETWORK BRIDGE	78
5.1. CONFIGURING A NETWORK BRIDGE BY USING NMCLI	78

5.2. CONFIGURING A NETWORK BRIDGE BY USING THE RHEL WEB CONSOLE	81
5.3. CONFIGURING A NETWORK BRIDGE BY USING NMTUI	83
5.4. CONFIGURING A NETWORK BRIDGE BY USING NMSTATECTL	87
5.5. CONFIGURING A NETWORK BRIDGE BY USING THE NETWORK RHEL SYSTEM ROLE	89
CHAPTER 6. SETTING UP AN IPSEC VPN	92
6.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION	92
6.2. AUTHENTICATION METHODS IN LIBRESWAN	93
6.3. INSTALLING LIBRESWAN	94
6.4. CREATING A HOST-TO-HOST VPN	95
6.5. CONFIGURING A SITE-TO-SITE VPN	96
6.6. CONFIGURING A REMOTE ACCESS VPN	97
6.7. CONFIGURING A MESH VPN	98
6.8. DEPLOYING A FIPS-COMPLIANT IPSEC VPN	101
6.9. PROTECTING THE IPSEC NSS DATABASE BY A PASSWORD	103
6.10. CONFIGURING AN IPSEC VPN TO USE TCP	104
6.11. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL SYSTEM ROLE	105
6.11.1. Creating a host-to-host VPN with IPsec by using the vpn RHEL system role	105
6.11.2. Creating an opportunistic mesh VPN connection with IPsec by using the vpn RHEL system role	108
6.12. CONFIGURING IPSEC CONNECTIONS THAT OPT OUT OF THE SYSTEM-WIDE CRYPTO POLICIES	109
6.13. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS	110
6.14. CONFIGURING A VPN CONNECTION WITH CONTROL-CENTER	114
6.15. CONFIGURING AN IPSEC BASED VPN CONNECTION BY USING NMSTATECTL	118
6.15.1. Configuring a host-to-subnet IPSec VPN with PKI authentication and tunnel mode by using nmstatectl	118
6.15.2. Configuring a host-to-subnet IPSec VPN with RSA authentication and tunnel mode by using nmstatectl	120
6.15.3. Configuring a host-to-subnet IPSec VPN with PSK authentication and tunnel mode by using nmstatectl	122
6.15.4. Configuring a host-to-host IPsec VPN with PKI authentication and tunnel mode by using nmstatectl	124
6.15.5. Configuring a host-to-host IPsec VPN with PSK authentication and transport mode by using nmstatectl	126
6.16. ADDITIONAL RESOURCES	128
CHAPTER 7. SETTING UP A WIREGUARD VPN	130
7.1. PROTOCOLS AND PRIMITIVES USED BY WIREGUARD	130
7.2. HOW WIREGUARD USES TUNNEL IP ADDRESSES, PUBLIC KEYS, AND REMOTE ENDPOINTS	131
7.3. USING A WIREGUARD CLIENT BEHIND NAT AND FIREWALLS	131
7.4. CREATING PRIVATE AND PUBLIC KEYS TO BE USED IN WIREGUARD CONNECTIONS	131
7.5. CONFIGURING A WIREGUARD SERVER BY USING NMCLI	132
7.6. CONFIGURING A WIREGUARD SERVER BY USING NMTUI	135
7.7. CONFIGURING A WIREGUARD SERVER BY USING THE RHEL WEB CONSOLE	138
7.8. CONFIGURING FIREWALLD ON A WIREGUARD SERVER BY USING THE COMMAND LINE	140
7.9. CONFIGURING FIREWALLD ON A WIREGUARD SERVER BY USING THE RHEL WEB CONSOLE	141
7.10. CONFIGURING A WIREGUARD CLIENT BY USING NMCLI	142
7.11. CONFIGURING A WIREGUARD CLIENT BY USING NMTUI	145
7.12. CONFIGURING A WIREGUARD CLIENT BY USING THE RHEL WEB CONSOLE	149
CHAPTER 8. CONFIGURING IP TUNNELS	153
8.1. CONFIGURING AN IPIP TUNNEL TO ENCAPSULATE IPV4 TRAFFIC IN IPV4 PACKETS	153
8.2. CONFIGURING A GRE TUNNEL TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS	156
8.3. CONFIGURING A GRE-TAP TUNNEL TO TRANSFER ETHERNET FRAMES OVER IPV4	158
CHAPTER 9. MANAGING WIFI CONNECTIONS	162

9.1. SUPPORTED WIFI SECURITY TYPES	162
9.2. CONNECTING TO A WIFI NETWORK BY USING NMCLI	162
9.3. CONNECTING TO A WIFI NETWORK BY USING THE GNOME SETTINGS APPLICATION	164
9.4. CONFIGURING A WIFI CONNECTION BY USING NMTUI	165
9.5. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION IN AN EXISTING PROFILE BY USING NMCLI	167
9.6. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE	168
9.7. MANUALLY SETTING THE WIRELESS REGULATORY DOMAIN	171
CHAPTER 10. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK ..	172
10.1. HOW MACSEC INCREASES SECURITY	172
10.2. CONFIGURING A MACSEC CONNECTION BY USING NMCLI	172
10.3. CONFIGURING A MACSEC CONNECTION BY USING NMSTATECTL	174
CHAPTER 11. CONFIGURING NETWORKMANAGER TO IGNORE CERTAIN DEVICES	177
11.1. PERMANENTLY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER	177
11.2. TEMPORARILY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER	178
CHAPTER 12. MANAGING THE DEFAULT GATEWAY	180
12.1. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING NMCLI	180
12.2. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING NMSTATECTL	181
12.3. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE	182
12.4. HOW NETWORKMANAGER MANAGES MULTIPLE DEFAULT GATEWAYS	183
12.5. CONFIGURING NETWORKMANAGER TO AVOID USING A SPECIFIC PROFILE TO PROVIDE A DEFAULT GATEWAY	185
12.6. FIXING UNEXPECTED ROUTING BEHAVIOR DUE TO MULTIPLE DEFAULT GATEWAYS	185
CHAPTER 13. CONFIGURING A STATIC ROUTE	188
13.1. EXAMPLE OF A NETWORK THAT REQUIRES STATIC ROUTES	188
13.2. HOW TO USE THE NMCLI UTILITY TO CONFIGURE A STATIC ROUTE	190
13.3. CONFIGURING A STATIC ROUTE BY USING NMCLI	191
13.4. CONFIGURING A STATIC ROUTE BY USING NMTUI	192
13.5. CONFIGURING A STATIC ROUTE BY USING NMSTATECTL	194
13.6. CONFIGURING A STATIC ROUTE BY USING THE NETWORK RHEL SYSTEM ROLE	195
CHAPTER 14. CONFIGURING POLICY-BASED ROUTING TO DEFINE ALTERNATIVE ROUTES	198
14.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY BY USING NMCLI	198
14.2. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY BY USING THE NETWORK RHEL SYSTEM ROLE	201
CHAPTER 15. CHANGING A HOSTNAME BY USING HOSTNAMECTL	206
CHAPTER 16. CONFIGURING THE DHCP TIMEOUT BEHAVIOR OF A NETWORKMANAGER CONNECTION ...	207
CHAPTER 17. CONFIGURING THE ORDER OF DNS SERVERS	209
17.1. HOW NETWORKMANAGER ORDERS DNS SERVERS IN /ETC/RESOLV.CONF	209
Default values of DNS priority parameters	209
Valid DNS priority values:	209
17.2. SETTING A NETWORKMANAGER-WIDE DEFAULT DNS SERVER PRIORITY VALUE	210
17.3. SETTING THE DNS PRIORITY OF A NETWORKMANAGER CONNECTION	211
CHAPTER 18. USING DNSMASQ IN NETWORKMANAGER TO SEND DNS REQUESTS FOR A SPECIFIC DOMAIN TO A SELECTED DNS SERVER	212

CHAPTER 19. AUTHENTICATING A RHEL CLIENT TO THE NETWORK BY USING THE 802.1X STANDARD WITH A CERTIFICATE STORED ON THE FILE SYSTEM	215
19.1. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING ETHERNET CONNECTION BY USING NMCLI	215
19.2. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING NMSTATECTL	216
19.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE	218
19.4. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE	220
CHAPTER 20. SETTING UP AN 802.1X NETWORK AUTHENTICATION SERVICE FOR LAN CLIENTS BY USING HOSTAPD WITH FREERADIUS BACKEND	224
20.1. PREREQUISITES	224
20.2. SETTING UP THE BRIDGE ON THE AUTHENTICATOR	224
20.3. CONFIGURING FREERADIUS TO AUTHENTICATE NETWORK CLIENTS SECURELY BY USING EAP	225
20.4. CONFIGURING HOSTAPD AS AN AUTHENTICATOR IN A WIRED NETWORK	230
20.5. TESTING EAP-TTLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR	232
20.6. BLOCKING AND ALLOWING TRAFFIC BASED ON HOSTAPD AUTHENTICATION EVENTS	234
CHAPTER 21. NETWORKMANAGER CONNECTION PROFILES IN KEYFILE FORMAT	237
21.1. THE KEYFILE FORMAT OF NETWORKMANAGER PROFILES	237
21.2. USING NMCLI TO CREATE KEYFILE CONNECTION PROFILES IN OFFLINE MODE	238
21.3. MANUALLY CREATING A NETWORKMANAGER PROFILE IN KEYFILE FORMAT	240
CHAPTER 22. SYSTEMD NETWORK TARGETS AND SERVICES	242
22.1. DIFFERENCES BETWEEN THE NETWORK AND NETWORK-ONLINE SYSTEMD TARGET	242
22.2. OVERVIEW OF NETWORKMANAGER-WAIT-ONLINE	242
22.3. CONFIGURING A SYSTEMD SERVICE TO START AFTER THE NETWORK HAS BEEN STARTED	243
CHAPTER 23. INTRODUCTION TO NMSTATE	244
23.1. USING THE LIBNMSTATE LIBRARY IN A PYTHON APPLICATION	244
23.2. UPDATING THE CURRENT NETWORK CONFIGURATION BY USING NMSTATECTL	245
23.3. THE NMSTATE SYSTEMD SERVICE	245
23.4. NETWORK STATES FOR THE NETWORK RHEL SYSTEM ROLE	246
CHAPTER 24. GETTING STARTED WITH MULTIPATH TCP	248
24.1. UNDERSTANDING MPTCP	248
24.2. PERMANENTLY CONFIGURING MULTIPLE PATHS FOR MPTCP APPLICATIONS	248
24.3. CONFIGURING MPTCPD	250
CHAPTER 25. LINUX TRAFFIC CONTROL	252
25.1. OVERVIEW OF QUEUING DISCIPLINES	252
25.2. INSPECTING QDISCS OF A NETWORK INTERFACE BY USING THE TC UTILITY	252
25.3. UPDATING THE DEFAULT QDISC	253
25.4. TEMPORARILY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE BY USING THE TC UTILITY	254
25.5. PERMANENTLY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE BY USING NETWORKMANAGER	254
25.6. CONFIGURING THE RATE LIMITING OF PACKETS BY USING THE TC-CTINFO UTILITY	255
25.7. AVAILABLE QDISCS IN RHEL	259
CHAPTER 26. CONFIGURING INFINIBAND AND RDMA NETWORKS	261
26.1. INTRODUCTION TO INFINIBAND AND RDMA	261
26.2. CONFIGURING THE CORE RDMA SUBSYSTEM	262
26.3. CONFIGURING IPOIB	263

26.3.1. The iPoB communication modes	264
26.3.2. Understanding iPoB hardware addresses	264
26.3.3. Renaming iPoB devices by using systemd link file	265
26.3.4. Configuring an iPoB connection by using nmcli	266
26.3.5. Configuring an iPoB connection by using the network RHEL system role	267
26.3.6. Configuring an iPoB connection by using nmstatectl	269
26.3.7. Testing an RDMA network by using iperf3 after iPoB is configured	271
26.4. CONFIGURING ROCE	272
26.4.1. Overview of RoCE protocol versions	272
26.4.2. Temporarily changing the default RoCE version	273
26.4.3. Configuring Soft-RoCE	273
26.5. INCREASING THE AMOUNT OF MEMORY THAT USERS ARE ALLOWED TO PIN IN THE SYSTEM	275
26.6. ENABLING NFS OVER RDMA ON AN NFS SERVER	275
26.7. INFINIBAND SUBNET MANAGER	277

RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.
- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see the Red Hat Knowledgebase solution [What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability \(GA\) release?](#).

CHAPTER 1. IMPLEMENTING CONSISTENT NETWORK INTERFACE NAMING

The **udev** device manager implements consistent device naming in Red Hat Enterprise Linux. The device manager supports different naming schemes and, by default, assigns fixed names based on firmware, topology, and location information.

Without consistent device naming, the Linux kernel assigns names to network interfaces by combining a fixed prefix and an index. The index increases as the kernel initializes the network devices. For example, **eth0** represents the first Ethernet device being probed on start-up. If you add another network interface controller to the system, the assignment of the kernel device names is no longer fixed because, after a reboot, the devices can initialize in a different order. In that case, the kernel can name the devices differently.

To solve this problem, **udev** assigns consistent device names. This has the following advantages:

- Device names are stable across reboots.
- Device names stay fixed even if you add or remove hardware.
- Defective hardware can be seamlessly replaced.
- The network naming is stateless and does not require explicit configuration files.



WARNING

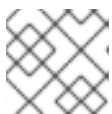
Generally, Red Hat does not support systems where consistent device naming is disabled. For exceptions, see the Red Hat Knowledgebase solution [Is it safe to set net.ifnames=0](#).

1.1. HOW THE UDEV DEVICE MANAGER RENAMES NETWORK INTERFACES

To implement a consistent naming scheme for network interfaces, the **udev** device manager processes the following rule files in the listed order:

1. Only on Dell systems: **/usr/lib/udev/rules.d/71-biosdevname.rules**

This file exists only if the **biosdevname** package is installed, and the rules file defines that the **biosdevname** utility renames the interface according to its naming policy, if it was not renamed in the previous step.



NOTE

Install and use **biosdevname** only on Dell systems.

2. **/usr/lib/udev/rules.d/75-net-description.rules**

This file defines how **udev** examines the network interface and sets the properties in **udev**-internal variables. These variables are then processed in the next step by the **/usr/lib/udev/rules.d/80-net-setup-link.rules** file. Some of the properties can be undefined.

3. `/usr/lib/udev/rules.d/80-net-setup-link.rules`

This file calls the `net_setup_link` builtin of the `udev` service, and `udev` renames the interface based on the order of the policies in the `NamePolicy` parameter in the `/usr/lib/systemd/network/99-default.link` file. For further details, see [Network interface naming policies](#).

If none of the policies applies, `udev` does not rename the interface.

Additional resources

- [Why are systemd network interface names different between major RHEL versions](#) (Red Hat Knowledgebase)

1.2. NETWORK INTERFACE NAMING POLICIES

By default, the `udev` device manager uses the `/usr/lib/systemd/network/99-default.link` file to determine which device naming policies to apply when it renames interfaces. The `NamePolicy` parameter in this file defines which policies `udev` uses and in which order:

`NamePolicy=keep kernel database onboard slot path`

The following table describes the different actions of `udev` based on which policy matches first as specified by the `NamePolicy` parameter:

Policy	Description	Example name
keep	If the device already has a name that was assigned in the user space, <code>udev</code> does not rename this device. For example, this is the case if the name was assigned during device creation or by a rename operation.	
kernel	If the kernel indicates that a device name is predictable, <code>udev</code> does not rename this device.	lo
database	This policy assigns names based on mappings in the <code>udev</code> hardware database. For details, see the <code>hwdb(7)</code> man page.	idrac
onboard	Device names incorporate firmware or BIOS-provided index numbers for onboard devices.	eno1
slot	Device names incorporate firmware or BIOS-provided PCI Express (PCIe) hot-plug slot-index numbers.	ens1
path	Device names incorporate the physical location of the connector of the hardware.	enp1s0
mac	Device names incorporate the MAC address. By default, Red Hat Enterprise Linux does not use this policy, but administrators can enable it.	enx525400d5e0fb

Additional resources

- [How the **udev** device manager renames network interfaces](#)
- **systemd.link(5)** man page on your system

1.3. NETWORK INTERFACE NAMING SCHEMES

The **udev** device manager uses certain stable interface attributes that device drivers provide to generate consistent device names.

If a new **udev** version changes how the service creates names for certain interfaces, Red Hat adds a new scheme version and documents the details in the **systemd.net-naming-scheme(7)** man page on your system. By default, Red Hat Enterprise Linux (RHEL) 10 uses the **rhel-10.0** naming scheme, even if you install or update to a later minor version of RHEL.



NOTE

On RHEL 10 you can also use all **rhel-8.*** and **rhel-9.*** naming schemes.

If you want to use a scheme other than the default, you can [switch the network interface naming scheme](#).

For further details about the naming schemes for different device types and platforms, see the **systemd.net-naming-scheme(7)** man page on your system.

1.4. SWITCHING TO A DIFFERENT NETWORK INTERFACE NAMING SCHEME

By default, Red Hat Enterprise Linux (RHEL) 10 uses the **rhel-10.0** naming scheme, even if you install or update to a later minor version of RHEL. While the default naming scheme fits in most scenarios, there might be reasons to switch to a different scheme version, for example:

- A new scheme can help to better identify a device if it adds additional attributes, such as a slot number, to an interface name.
- An new scheme can prevent **udev** from falling back to the kernel-assigned device names (**eth***). This happens if the driver does not provide enough unique attributes for two or more interfaces to generate unique names for them.

Prerequisites

- You have access to the console of the server.

Procedure

1. List the network interfaces:

```
# ip link show
2: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

Record the MAC addresses of the interfaces.

- Optional: Display the **ID_NET_NAMING_SCHEME** property of a network interface to identify the naming scheme that RHEL currently uses:

```
# udevadm info --query=property --property=ID_NET_NAMING_SCHEME
/sys/class/net/eno1'
ID_NET_NAMING_SCHEME=rhel-10.0
```

Note that the property is not available on the **lo** loopback device.

- Append the **net.naming-scheme=<scheme>** option to the command line of all installed kernels, for example:

```
# grubby --update-kernel=ALL --args=net.naming-scheme=rhel-10.1
```

- Reboot the system.

```
# reboot
```

- Based on the MAC addresses you recorded, identify the new names of network interfaces that have changed due to the different naming scheme:

```
# ip link show
2: eno1np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

After switching the scheme, **udev** names the device with the specified MAC address **eno1np0**, whereas it was named **eno1** before.

- Identify which NetworkManager connection profile uses an interface with the previous name:

```
# nmcli -f device,name connection show
DEVICE NAME
eno1 example_profile
...
```

- Set the **connection.interface-name** property in the connection profile to the new interface name:

```
# nmcli connection modify example_profile connection.interface-name "eno1np0"
```

- Reactivate the connection profile:

```
# nmcli connection up example_profile
```

Verification

- Identify the naming scheme that RHEL now uses by displaying the **ID_NET_NAMING_SCHEME** property of a network interface:

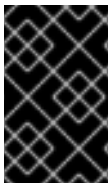
```
# udevadm info --query=property --property=ID_NET_NAMING_SCHEME  
/sys/class/net/eno1np0'  
ID_NET_NAMING_SCHEME=_rhel-10.1
```

Additional resources

- [Network interface naming schemes](#)

1.5. CUSTOMIZING THE PREFIX FOR ETHERNET INTERFACES DURING INSTALLATION

If you do not want to use the default device-naming policy for Ethernet interfaces, you can set a custom device prefix during the Red Hat Enterprise Linux (RHEL) installation.



IMPORTANT

Red Hat supports systems with customized Ethernet prefixes only if you set the prefix during the RHEL installation. Using the **prefixdevname** utility on already deployed systems is not supported.

If you set a device prefix during the installation, the **udev** service uses the **<prefix><index>** format for Ethernet interfaces after the installation. For example, if you set the prefix **net**, the service assigns the names **net0**, **net1**, and so on to the Ethernet interfaces.

The **udev** service appends the index to the custom prefix, and preserves the index values of known Ethernet interfaces. If you add an interface, **udev** assigns an index value that is one greater than the previously-assigned index value to the new interface.

Prerequisites

- The prefix consists of ASCII characters.
- The prefix is an alphanumeric string.
- The prefix is shorter than 16 characters.
- The prefix does not conflict with any other well-known network interface prefix, such as **eth**, **eno**, **ens**, and **em**.

Procedure

1. Boot the Red Hat Enterprise Linux installation media.
2. In the boot manager, follow these steps:
 - a. Select the **Install Red Hat Enterprise Linux <version>** entry.
 - b. Press **Tab** to edit the entry.
 - c. Append **net.ifnames.prefix=<prefix>** to the kernel options.
 - d. Press **Enter** to start the installation program.
3. Install Red Hat Enterprise Linux.

Verification

- To verify the interface names, display the network interfaces:

```
# ip link show
...
2: net0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

1.6. CONFIGURING USER-DEFINED NETWORK INTERFACE NAMES BY USING UDEV RULES

You can use **udev** rules to implement custom network interface names that reflect your organization's requirements.

Procedure

1. Identify the network interface that you want to rename:

```
# ip link show
...
enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

Record the MAC address of the interface.

2. Display the device type ID of the interface:

```
# cat /sys/class/net/enp1s0/type
1
```

3. Create the **/etc/udev/rules.d/70-persistent-net.rules** file, and add a rule for each interface that you want to rename:

```
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="<MAC_address>",ATTR{type}=="<
device_type_id>",NAME="<new_interface_name>"
```



IMPORTANT

Use only **70-persistent-net.rules** as a file name if you require consistent device names during the boot process. The **dracut** utility adds a file with this name to the **initrd** image if you regenerate the RAM disk image.

For example, use the following rule to rename the interface with MAC address **00:00:5e:00:53:1a** to **provider0**:

```
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="00:00:5e:00:53:1a",ATTR{type}=="
1",NAME="provider0"
```

- Optional: Regenerate the **initrd** RAM disk image:

```
# dracut -f
```

You require this step only if you need networking capabilities in the RAM disk. For example, this is the case if the root file system is stored on a network device, such as iSCSI.

- Identify which NetworkManager connection profile uses the interface that you want to rename:

```
# nmcli -f device,name connection show
DEVICE NAME
enp1s0 example_profile
...
```

- Unset the **connection.interface-name** property in the connection profile:

```
# nmcli connection modify example_profile connection.interface-name ""
```

- Temporarily, configure the connection profile to match both the new and the previous interface name:

```
# nmcli connection modify example_profile match.interface-name "provider0 enp1s0"
```

- Reboot the system:

```
# reboot
```

- Verify that the device with the MAC address that you specified in the link file has been renamed to **provider0**:

```
# ip link show
provider0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

- Configure the connection profile to match only the new interface name:

```
# nmcli connection modify example_profile match.interface-name "provider0"
```

You have now removed the old interface name from the connection profile.

- Reactivate the connection profile:

```
# nmcli connection up example_profile
```

Additional resources

- udev(7)** man page on your system

1.7. CONFIGURING USER-DEFINED NETWORK INTERFACE NAMES BY USING SYSTEMD LINK FILES

You can use **systemd** link files to implement custom network interface names that reflect your organization's requirements.

Prerequisites

- NetworkManager does not manage this interface.

Procedure

1. Identify the network interface that you want to rename:

```
# ip link show
...
enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

Record the MAC address of the interface.

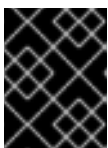
2. If it does not already exist, create the **/etc/systemd/network/** directory:

```
# mkdir -p /etc/systemd/network/
```

3. For each interface that you want to rename, create a **70-*.link** file in the **/etc/systemd/network/** directory with the following content:

```
[Match]
MACAddress=<MAC_address>

[Link]
Name=<new_interface_name>
```



IMPORTANT

Use a file name with a **70-** prefix to keep the file names consistent with the **udev** rules-based solution.

For example, create the **/etc/systemd/network/70-provider0.link** file with the following content to rename the interface with MAC address **00:00:5e:00:53:1a** to **provider0**:

```
[Match]
MACAddress=00:00:5e:00:53:1a

[Link]
Name=provider0
```

4. Optional: Regenerate the **initrd** RAM disk image:

```
# dracut -f
```

You require this step only if you need networking capabilities in the RAM disk. For example, this is the case if the root file system is stored on a network device, such as iSCSI.

5. Identify which NetworkManager connection profile uses the interface that you want to rename:

```
# nmcli -f device,name connection show
DEVICE NAME
enp1s0 example_profile
...
```

6. Unset the **connection.interface-name** property in the connection profile:

```
# nmcli connection modify example_profile connection.interface-name ""
```

7. Temporarily, configure the connection profile to match both the new and the previous interface name:

```
# nmcli connection modify example_profile match.interface-name "provider0 enp1s0"
```

8. Reboot the system:

```
# reboot
```

9. Verify that the device with the MAC address that you specified in the link file has been renamed to **provider0**:

```
# ip link show
provider0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

10. Configure the connection profile to match only the new interface name:

```
# nmcli connection modify example_profile match.interface-name "provider0"
```

You have now removed the old interface name from the connection profile.

11. Reactivate the connection profile.

```
# nmcli connection up example_profile
```

Additional resources

- **systemd.link(5)** man page on your system

1.8. ASSIGNING ALTERNATIVE NAMES TO A NETWORK INTERFACE BY USING SYSTEMD LINK FILES

With alternative interface naming, the kernel can assign additional names to network interfaces. You can use these alternative names in the same way as the normal interface names in commands that require a network interface name.

Prerequisites

- You must use ASCII characters for the alternative name.
- The alternative name must be shorter than 128 characters.

Procedure

1. Display the network interface names and their MAC addresses:

```
# ip link show
...
enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
...
```

Record the MAC address of the interface to which you want to assign an alternative name.

2. If it does not already exist, create the `/etc/systemd/network/` directory:

```
# mkdir -p /etc/systemd/network/
```

3. For each interface that must have an alternative name, create a copy of the `/usr/lib/systemd/network/99-default.link` file with a unique name and `.link` suffix in the `/etc/systemd/network/` directory, for example:

```
# cp /usr/lib/systemd/network/99-default.link /etc/systemd/network/98-lan.link
```

4. Modify the file you created in the previous step. Rewrite the **[Match]** section as follows, and append the **AlternativeName** entries to the **[Link]** section:

```
[Match]
MACAddress=<MAC_address>

[Link]
...
AlternativeName=<alternative_interface_name_1>
AlternativeName=<alternative_interface_name_2>
AlternativeName=<alternative_interface_name_n>
```

For example, create the `/etc/systemd/network/70-altname.link` file with the following content to assign **provider** as an alternative name to the interface with MAC address **00:00:5e:00:53:1a**:

```
[Match]
MACAddress=00:00:5e:00:53:1a

[Link]
NamePolicy=keep kernel database onboard slot path
AlternativeNamesPolicy=database onboard slot path mac
MACAddressPolicy=none
AlternativeName=provider
```

5. Regenerate the **initrd** RAM disk image:

```
# dracut -f
```

6. Reboot the system:

```
# reboot
```

Verification

- Use the alternative interface name. For example, display the IP address settings of the device with the alternative name **provider**:

```
# ip address show provider
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:00:5e:00:53:1a brd ff:ff:ff:ff:ff:ff
    altname provider
...
```

Additional resources

- [What is AlternativeNamesPolicy in Interface naming scheme?](#) (Red Hat Knowledgebase)

CHAPTER 2. CONFIGURING AN ETHERNET CONNECTION

NetworkManager creates a connection profile for each Ethernet adapter that is installed in a host. By default, this profile uses DHCP for both IPv4 and IPv6 connections. Modify this automatically-created profile or add a new one in the following cases:

- The network requires custom settings, such as a static IP address configuration.
- You require multiple profiles because the host roams among different networks.

Red Hat Enterprise Linux provides administrators different options to configure Ethernet connections. For example:

- Use **nmcli** to configure connections on the command line.
- Use **nmtui** to configure connections in a text-based user interface.
- Use the GNOME Settings menu to configure connections in a graphical interface.
- Use **nmstatectl** to configure connections through the Nmstate API.
- Use RHEL system roles to automate the configuration of connections on one or multiple hosts.



NOTE

If you want to manually configure Ethernet connections on hosts running in the Microsoft Azure cloud, disable the **cloud-init** service or configure it to ignore the network settings retrieved from the cloud environment. Otherwise, **cloud-init** will override on the next reboot the network settings that you have manually configured.

2.1. CONFIGURING AN ETHERNET CONNECTION BY USING **NMCLI**

If you connect a host to the network over Ethernet, you can manage the connection's settings on the command line by using the **nmcli** utility.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server's configuration.

Procedure

1. List the NetworkManager connection profiles:

```
# nmcli connection show
NAME                UUID                                  TYPE    DEVICE
Wired connection 1  a5eb6490-cc20-3668-81f8-0314a27f3f75 ethernet enp1s0
```

By default, NetworkManager creates a profile for each NIC in the host. If you plan to connect this NIC only to a specific network, adapt the automatically-created profile. If you plan to connect this NIC to networks with different settings, create individual profiles for each network.

2. If you want to create an additional connection profile, enter:

```
# nmcli connection add con-name <connection-name> ifname <device-name> type ethernet
```

Skip this step to modify an existing profile.

- Optional: Rename the connection profile:

```
# nmcli connection modify "Wired connection 1" connection.id "Internal-LAN"
```

On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.

- Display the current settings of the connection profile:

```
# nmcli connection show Internal-LAN
...
connection.interface-name: enp1s0
connection.autoconnect:   yes
ipv4.method:              auto
ipv6.method:              auto
...
```

- Configure the IPv4 settings:

- To use DHCP, enter:

```
# nmcli connection modify Internal-LAN ipv4.method auto
```

Skip this step if **ipv4.method** is already set to **auto** (default).

- To set a static IPv4 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify Internal-LAN ipv4.method manual ipv4.addresses 192.0.2.1/24 ipv4.gateway 192.0.2.254 ipv4.dns 192.0.2.200 ipv4.dns-search example.com
```

- Configure the IPv6 settings:

- To use stateless address autoconfiguration (SLAAC), enter:

```
# nmcli connection modify Internal-LAN ipv6.method auto
```

Skip this step if **ipv6.method** is already set to **auto** (default).

- To set a static IPv6 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify Internal-LAN ipv6.method manual ipv6.addresses 2001:db8:1::fffe/64 ipv6.gateway 2001:db8:1::fffe ipv6.dns 2001:db8:1::ffbb ipv6.dns-search example.com
```

- To customize other settings in the profile, use the following command:


```
# nmcli connection modify <connection-name> <setting> <value>
```

Enclose values with spaces or semicolons in quotes.

8. Activate the profile:

```
# nmcli connection up Internal-LAN
```

Verification

1. Display the IP settings of the NIC:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::fffe/64 scope global noprefixroute
        valid_lft forever preferred_lft forever
```

2. Display the IPv4 default gateway:

```
# ip route show default
default via 192.0.2.254 dev enp1s0 proto static metric 102
```

3. Display the IPv6 default gateway:

```
# ip -6 route show default
default via 2001:db8:1::fffe dev enp1s0 proto static metric 102 pref medium
```

4. Display the DNS settings:

```
# cat /etc/resolv.conf
search example.com
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb
```

If multiple connection profiles are active at the same time, the order of **nameserver** entries depend on the DNS priority values in these profiles and the connection types.

5. Use the **ping** utility to verify that this host can send packets to other hosts:

```
# ping <host-name-or-IP-address>
```

Troubleshooting

- Verify that the network cable is plugged-in to the host and a switch.
- Check whether the link failure exists only on this host or also on other hosts connected to the same switch.

- Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defective cables and network interface cards.
- If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see the Red Hat Knowledgebase solution [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- **nm-settings(5)** man page on your system

2.2. CONFIGURING AN ETHERNET CONNECTION BY USING **NMTUI**

If you connect a host to the network over Ethernet, you can manage the connection's settings in a text-based user interface by using the **nmtui** application. Use **nmtui** to create new profiles and to update existing ones on a host without a graphical interface.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server's configuration.

Procedure

1. If you do not know the network device name you want to use in the connection, display the available devices:

```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp1s0  ethernet unavailable --
...
```

2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Choose whether to add a new connection profile or to modify an existing one:
 - To create a new profile:

- i. Press **Add**.
 - ii. Select **Ethernet** from the list of network types, and press **Enter**.
- To modify an existing profile, select the profile from the list, and press **Enter**.
5. Optional: Update the name of the connection profile.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.
6. If you create a new connection profile, enter the network device name into the **Device** field.
7. Depending on your environment, configure the IP address settings in the **IPv4 configuration** and **IPv6 configuration** areas accordingly. For this, press the button next to these areas, and select:
 - **Disabled**, if this connection does not require an IP address.
 - **Automatic**, if a DHCP server dynamically assigns an IP address to this NIC.
 - **Manual**, if the network requires static IP address settings. In this case, you must fill further fields:
 - i. Press **Show** next to the protocol you want to configure to display additional fields.
 - ii. Press **Add** next to **Addresses**, and enter the IP address and the subnet mask in Classless Inter-Domain Routing (CIDR) format.
If you do not specify a subnet mask, NetworkManager sets a **/32** subnet mask for IPv4 addresses and **/64** for IPv6 addresses.
 - iii. Enter the address of the default gateway.
 - iv. Press **Add** next to **DNS servers**, and enter the DNS server address.
 - v. Press **Add** next to **Search domains**, and enter the DNS search domain.

Figure 2.1. Example of an Ethernet connection with static IP address settings

The screenshot shows the 'Edit Connection' window in nmcli. The profile name is 'Example-Connection' and the device is 'enp7s0'. The connection type is 'ETHERNET'. Under 'IPv4 CONFIGURATION', the mode is set to '<Manual>'. The IP address is '192.0.2.1/24', the gateway is '192.0.2.254', and the DNS server is '192.0.2.200'. The search domain is 'example.com'. Under 'IPv6 CONFIGURATION', the mode is also '<Manual>'. The IP address is '2001:db8:1::1/64', the gateway is '2001:db8:1::fffe', and the DNS server is '2001:db8:1::ffbb'. The search domain is 'example.com'. Both IPv4 and IPv6 sections have routing options that are currently unchecked. At the bottom, 'Automatically connect' and 'Available to all users' are checked. The window has '<Cancel>' and '<OK>' buttons at the bottom right.

```

Edit Connection

Profile name Example-Connection
Device      enp7s0

= ETHERNET <Show>

= IPv4 CONFIGURATION <Manual> <Hide>
  Addresses 192.0.2.1/24 <Remove>
             <Add...>
  Gateway   192.0.2.254
  DNS servers 192.0.2.200 <Remove>
             <Add...>
  Search domains example.com <Remove>
             <Add...>

  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv4 addressing for this connection

= IPv6 CONFIGURATION <Manual> <Hide>
  Addresses 2001:db8:1::1/64 <Remove>
             <Add...>
  Gateway   2001:db8:1::fffe
  DNS servers 2001:db8:1::ffbb <Remove>
             <Add...>
  Search domains example.com <Remove>
             <Add...>

  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv6 addressing for this connection

[X] Automatically connect
[X] Available to all users

<Cancel> <OK>

```

8. Press **OK** to create and automatically activate the new connection.
9. Press **Back** to return to the main menu.
10. Select **Quit**, and press **Enter** to close the **nmcli** application.

Verification

1. Display the IP settings of the NIC:

```

# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff

```

```

inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0
    valid_lft forever preferred_lft forever
inet6 2001:db8:1::fffe/64 scope global noprefixroute
    valid_lft forever preferred_lft forever

```

2. Display the IPv4 default gateway:

```

# ip route show default
default via 192.0.2.254 dev enp1s0 proto static metric 102

```

3. Display the IPv6 default gateway:

```

# ip -6 route show default
default via 2001:db8:1::ffee dev enp1s0 proto static metric 102 pref medium

```

4. Display the DNS settings:

```

# cat /etc/resolv.conf
search example.com
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb

```

If multiple connection profiles are active at the same time, the order of **nameserver** entries depend on the DNS priority values in these profiles and the connection types.

5. Use the **ping** utility to verify that this host can send packets to other hosts:

```

# ping <host-name-or-IP-address>

```

Troubleshooting

- Verify that the network cable is plugged-in to the host and a switch.
- Check whether the link failure exists only on this host or also on other hosts connected to the same switch.
- Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defective cables and network interface cards.
- If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see the Red Hat Knowledgebase solution [NetworkManager duplicates a connection after restart of NetworkManager service](#).

2.3. CONFIGURING AN ETHERNET CONNECTION BY USING CONTROL-CENTER

If you connect a host to the network over Ethernet, you can manage the connection's settings with a graphical interface by using the GNOME Settings menu.

Note that **control-center** does not support as many configuration options as the **nmcli** utility.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server's configuration.
- GNOME is installed.

Procedure

1. Press the **Super** key, enter **Settings**, and press **Enter**.
2. Select **Network** in the navigation on the left.
3. Choose whether to add a new connection profile or to modify an existing one:
 - To create a new profile, click the **+** button next to the **Ethernet** entry.
 - To modify an existing profile, click the gear icon next to the profile entry.
4. Optional: On the **Identity** tab, update the name of the connection profile.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.
5. Depending on your environment, configure the IP address settings on the **IPv4** and **IPv6** tabs accordingly:
 - To use DHCP or IPv6 stateless address autoconfiguration (SLAAC), select **Automatic (DHCP)** as method (default).
 - To set a static IP address, network mask, default gateway, DNS servers, and search domain, select **Manual** as method, and fill the fields on the tabs:

6. Depending on whether you add or modify a connection profile, click the **Add** or **Apply** button to save the connection.
The GNOME **control-center** automatically activates the connection.

Verification

1. Display the IP settings of the NIC:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0
```

```
valid_lft forever preferred_lft forever
inet6 2001:db8:1::fffe/64 scope global noprefixroute
valid_lft forever preferred_lft forever
```

2. Display the IPv4 default gateway:

```
# ip route show default
default via 192.0.2.254 dev enp1s0 proto static metric 102
```

3. Display the IPv6 default gateway:

```
# ip -6 route show default
default via 2001:db8:1::ffee dev enp1s0 proto static metric 102 pref medium
```

4. Display the DNS settings:

```
# cat /etc/resolv.conf
search example.com
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb
```

If multiple connection profiles are active at the same time, the order of **nameserver** entries depend on the DNS priority values in these profiles and the connection types.

5. Use the **ping** utility to verify that this host can send packets to other hosts:

```
# ping <host-name-or-IP-address>
```

Troubleshooting steps

- Verify that the network cable is plugged-in to the host and a switch.
- Check whether the link failure exists only on this host or also on other hosts connected to the same switch.
- Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defective cables and network interface cards.
- If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see the Red Hat Knowledgebase solution [NetworkManager duplicates a connection after restart of NetworkManager service](#).

2.4. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING NMSTATECTL

Use the **nmstatectl** utility to configure an Ethernet connection through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server's configuration.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example `~/create-ethernet-profile.yml`, with the following content:

```
---
interfaces:
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
      - ip: 2001:db8:1::1
        prefix-length: 64
    autoconf: false
    dhcp: false
routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-address: 192.0.2.254
      next-hop-interface: enp1s0
    - destination: ::0
      next-hop-address: 2001:db8:1::fffe
      next-hop-interface: enp1s0
dns-resolver:
  config:
    search:
      - example.com
    server:
      - 192.0.2.200
      - 2001:db8:1::ffbb
```

These settings define an Ethernet connection profile for the **enp1s0** device with the following settings:

- A static IPv4 address – **192.0.2.1** with the **/24** subnet mask
- A static IPv6 address – **2001:db8:1::1** with the **/64** subnet mask
- An IPv4 default gateway – **192.0.2.254**
- An IPv6 default gateway – **2001:db8:1::fffe**
- An IPv4 DNS server – **192.0.2.200**

- An IPv6 DNS server – **2001:db8:1::ffbb**
 - A DNS search domain – **example.com**
- Optional: You can define the **identifier: mac-address** and **mac-address: <mac_address>** properties in the **interfaces** property to identify the network interface card by its MAC address instead of its name, for example:

```
---
interfaces:
- name: <profile_name>
  type: ethernet
  identifier: mac-address
  mac-address: <mac_address>
...
```

- Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification

- Display the current state in YAML format:

```
# nmstatectl show enp1s0
```

- Display the IP settings of the NIC:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::fffe/64 scope global noprefixroute
        valid_lft forever preferred_lft forever
```

- Display the IPv4 default gateway:

```
# ip route show default
default via 192.0.2.254 dev enp1s0 proto static metric 102
```

- Display the IPv6 default gateway:

```
# ip -6 route show default
default via 2001:db8:1::fffe dev enp1s0 proto static metric 102 pref medium
```

- Display the DNS settings:

```
# cat /etc/resolv.conf
search example.com
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb
```

If multiple connection profiles are active at the same time, the order of **nameserver** entries depend on the DNS priority values in these profiles and the connection types.

6. Use the **ping** utility to verify that this host can send packets to other hosts:

```
# ping <host-name-or-IP-address>
```

Additional resources

- **nmstatectl(8)** man page on your system
- **/usr/share/doc/nmstate/examples/** directory

2.5. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE **network** RHEL SYSTEM ROLE WITH AN INTERFACE NAME

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection with static IP addresses, gateways, and DNS settings, and assign them to a specified interface name.

Typically, administrators want to reuse a playbook and not maintain individual playbooks for each host to which Ansible should assign static IP addresses. In this case, you can use variables in the playbook and maintain the settings in the inventory. As a result, you need only one playbook to dynamically assign individual settings to multiple hosts.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- A physical or virtual Ethernet device exists in the server configuration.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Edit the **~/inventory** file, and append the host-specific settings to the host entries:

```
managed-node-01.example.com interface=enp1s0 ip_v4=192.0.2.1/24
ip_v6=2001:db8:1::1/64 gateway_v4=192.0.2.254 gateway_v6=2001:db8:1::fffe

managed-node-02.example.com interface=enp1s0 ip_v4=192.0.2.2/24
ip_v6=2001:db8:1::2/64 gateway_v4=192.0.2.254 gateway_v6=2001:db8:1::fffe
```

2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
```

```

- name: Configure the network
  hosts: managed-node-01.example.com,managed-node-02.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: "{{ interface }}"
            interface_name: "{{ interface }}"
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - "{{ ip_v4 }}"
                - "{{ ip_v6 }}"
              gateway4: "{{ gateway_v4 }}"
              gateway6: "{{ gateway_v6 }}"
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            state: up

```

This playbook reads certain values dynamically for each host from the inventory file and uses static values in the playbook for settings which are the same for all hosts.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```

# ansible managed-node-01.example.com -m ansible.builtin.setup
...
"ansible_default_ipv4": {
  "address": "192.0.2.1",
  "alias": "enp1s0",
  "broadcast": "192.0.2.255",
  "gateway": "192.0.2.254",
  "interface": "enp1s0",
  "macaddress": "52:54:00:17:b8:b6",

```

```

        "mtu": 1500,
        "netmask": "255.255.255.0",
        "network": "192.0.2.0",
        "prefix": "24",
        "type": "ether"
    },
    "ansible_default_ipv6": {
        "address": "2001:db8:1::1",
        "gateway": "2001:db8:1::fffe",
        "interface": "enp1s0",
        "macaddress": "52:54:00:17:b8:b6",
        "mtu": 1500,
        "prefix": "64",
        "scope": "global",
        "type": "ether"
    },
    ...
    "ansible_dns": {
        "nameservers": [
            "192.0.2.1",
            "2001:db8:1::ffbb"
        ],
        "search": [
            "example.com"
        ]
    },
    ...

```

Additional resources

- [/usr/share/ansible/roles/rhel-system-roles.network/README.md](#) file
- [/usr/share/doc/rhel-system-roles/network/](#) directory

2.6. CONFIGURING AN ETHERNET CONNECTION WITH A STATIC IP ADDRESS BY USING THE `network` RHEL SYSTEM ROLE WITH A DEVICE PATH

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection with static IP addresses, gateways, and DNS settings, and assign them to a device based on its path instead of its name.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

- A physical or virtual Ethernet device exists in the server's configuration.
- The managed nodes use NetworkManager to configure the network.
- You know the path of the device. You can display the device path by using the **udevadm info /sys/class/net/<device_name> | grep ID_PATH=** command.

Procedure

1. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: example
            match:
              path:
                - pci-0000:00:0[1-3].0
                - &!pci-0000:00:02.0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            state: up
```

The settings specified in the example playbook include the following:

match

Defines that a condition must be met in order to apply the settings. You can only use this variable with the **path** option.

path

Defines the persistent path of a device. You can set it as a fixed path or an expression. Its value can contain modifiers and wildcards. The example applies the settings to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
```

```
...
  "ansible_default_ipv4": {
    "address": "192.0.2.1",
    "alias": "enp1s0",
    "broadcast": "192.0.2.255",
    "gateway": "192.0.2.254",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "192.0.2.0",
    "prefix": "24",
    "type": "ether"
  },
  "ansible_default_ipv6": {
    "address": "2001:db8:1::1",
    "gateway": "2001:db8:1::fffe",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "prefix": "64",
    "scope": "global",
    "type": "ether"
  },
  ...
  "ansible_dns": {
    "nameservers": [
      "192.0.2.1",
      "2001:db8:1::ffbb"
    ],
    "search": [
      "example.com"
    ]
  },
  ...
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file

- `/usr/share/doc/rhel-system-roles/network/` directory

2.7. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING NMSTATECTL

Use the **nmstatectl** utility to configure an Ethernet connection through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server's configuration.
- A DHCP server is available in the network.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example `~/create-ethernet-profile.yml`, with the following content:

```
---
interfaces:
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    dhcp: true
  ipv6:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    autoconf: true
    dhcp: true
```

These settings define an Ethernet connection profile for the **enp1s0** device. The connection retrieves IPv4 addresses, IPv6 addresses, default gateway, routes, DNS servers, and search domains from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC).

2. Optional: You can define the **identifier: mac-address** and **mac-address: <mac_address>** properties in the **interfaces** property to identify the network interface card by its MAC address instead of its name, for example:

```
---
interfaces:
- name: <profile_name>
  type: ethernet
```

```

identifier: mac-address
mac-address: <mac_address>
...

```

3. Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification

1. Display the current state in YAML format:

```
# nmstatectl show enp1s0
```

2. Display the IP settings of the NIC:

```

# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::fffe/64 scope global noprefixroute
        valid_lft forever preferred_lft forever

```

3. Display the IPv4 default gateway:

```

# ip route show default
default via 192.0.2.254 dev enp1s0 proto static metric 102

```

4. Display the IPv6 default gateway:

```

# ip -6 route show default
default via 2001:db8:1::fffe dev enp1s0 proto static metric 102 pref medium

```

5. Display the DNS settings:

```

# cat /etc/resolv.conf
search example.com
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb

```

If multiple connection profiles are active at the same time, the order of **nameserver** entries depend on the DNS priority values in these profiles and the connection types.

6. Use the **ping** utility to verify that this host can send packets to other hosts:

```
# ping <host-name-or-IP-address>
```

Additional resources

- **nmstatectl(8)** man page on your system

- `/usr/share/doc/nmstate/examples/` directory

2.8. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE `network` RHEL SYSTEM ROLE WITH AN INTERFACE NAME

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection that retrieves its IP addresses, gateways, and DNS settings from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC). With this role you can assign the connection profile to the specified interface name.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- A physical or virtual Ethernet device exists in the server's configuration.
- A DHCP server and SLAAC are available in the network.
- The managed nodes use the NetworkManager service to configure the network.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            interface_name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              dhcp4: yes
              auto6: yes
            state: up
```

The settings specified in the example playbook include the following:

dhcp4: yes

Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes

Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify that the interface received IP addresses and DNS settings:

```
# ansible managed-node-01.example.com -m ansible.builtin.setup
```

```
...
  "ansible_default_ipv4": {
    "address": "192.0.2.1",
    "alias": "enp1s0",
    "broadcast": "192.0.2.255",
    "gateway": "192.0.2.254",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "192.0.2.0",
    "prefix": "24",
    "type": "ether"
  },
  "ansible_default_ipv6": {
    "address": "2001:db8:1::1",
    "gateway": "2001:db8:1::fffe",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "prefix": "64",
    "scope": "global",
    "type": "ether"
  },
  ...
  "ansible_dns": {
    "nameservers": [
      "192.0.2.1",
```

```

        "2001:db8:1::ffbb"
    ],
    "search": [
        "example.com"
    ]
  },
  ...

```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

2.9. CONFIGURING AN ETHERNET CONNECTION WITH A DYNAMIC IP ADDRESS BY USING THE `network` RHEL SYSTEM ROLE WITH A DEVICE PATH

To connect a Red Hat Enterprise Linux host to an Ethernet network, create a NetworkManager connection profile for the network device. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure an Ethernet connection that retrieves its IP addresses, gateways, and DNS settings from a DHCP server and IPv6 stateless address autoconfiguration (SLAAC). The role can assign the connection profile to a device based on its path instead of an interface name.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- A physical or virtual Ethernet device exists in the server's configuration.
- A DHCP server and SLAAC are available in the network.
- The managed hosts use NetworkManager to configure the network.
- You know the path of the device. You can display the device path by using the **udevadm info /sys/class/net/<device_name> | grep ID_PATH=** command.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```

---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with dynamic IP address settings
      ansible.builtin.include_role:

```

```

name: rhel-system-roles.network
vars:
  network_connections:
    - name: example
      match:
        path:
          - pci-0000:00:0[1-3].0
          - &!pci-0000:00:02.0
      type: ethernet
      autoconnect: yes
      ip:
        dhcp4: yes
        auto6: yes
      state: up

```

The settings specified in the example playbook include the following:

match: path

Defines that a condition must be met in order to apply the settings. You can only use this variable with the **path** option.

path: <path_and_expressions>

Defines the persistent path of a device. You can set it as a fixed path or an expression. Its value can contain modifiers and wildcards. The example applies the settings to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**.

dhcp4: yes

Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes

Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify that the interface received IP addresses and DNS settings:

```

# ansible managed-node-01.example.com -m ansible.builtin.setup
...
"ansible_default_ipv4": {

```

```

    "address": "192.0.2.1",
    "alias": "enp1s0",
    "broadcast": "192.0.2.255",
    "gateway": "192.0.2.254",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "192.0.2.0",
    "prefix": "24",
    "type": "ether"
  },
  "ansible_default_ipv6": {
    "address": "2001:db8:1::1",
    "gateway": "2001:db8:1::fffe",
    "interface": "enp1s0",
    "macaddress": "52:54:00:17:b8:b6",
    "mtu": 1500,
    "prefix": "64",
    "scope": "global",
    "type": "ether"
  },
  ...
  "ansible_dns": {
    "nameservers": [
      "192.0.2.1",
      "2001:db8:1::ffbb"
    ],
    "search": [
      "example.com"
    ]
  },
  ...

```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

2.10. CONFIGURING MULTIPLE ETHERNET INTERFACES BY USING A SINGLE CONNECTION PROFILE BY INTERFACE NAME

In most cases, one connection profile contains the settings of one network device. However, NetworkManager also supports wildcards when you set the interface name in connection profiles. If a host roams between Ethernet networks with dynamic IP address assignment, you can use this feature to create a single connection profile that you can use for multiple Ethernet interfaces.

Prerequisites

- Multiple physical or virtual Ethernet devices exist in the server's configuration.
- A DHCP server is available in the network.
- No connection profile exists on the host.

Procedure

1. Add a connection profile that applies to all interface names starting with **enp**:

```
# nmcli connection add con-name "Wired connection 1" connection.multi-connect
multiple match.interface-name enp* type ethernet
```

Verification

1. Display all settings of the single connection profile:

```
# nmcli connection show "Wired connection 1"
connection.id:          Wired connection 1
...
connection.multi-connect: 3 (multiple)
match.interface-name:   enp*
...
```

3 indicates that the interface can be active multiple times at a particular moment. The connection profile uses all devices that match the pattern in the **match.interface-name** parameter and, therefore, the connection profiles have the same Universally Unique Identifier (UUID).

2. Display the status of the connections:

```
# nmcli connection show
NAME                UUID                                TYPE    DEVICE
...
Wired connection 1  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp7s0
Wired connection 1  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp8s0
Wired connection 1  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp9s0
```

Additional resources

- **nmcli(1)** and **nm-settings(5)** man page on your system
- [NetworkManager duplicates a connection after restart of NetworkManager service](#) (Red Hat Knowledgebase)

2.11. CONFIGURING A SINGLE CONNECTION PROFILE FOR MULTIPLE ETHERNET INTERFACES USING PCI IDS

The PCI ID is a unique identifier of the devices connected to the system. The connection profile adds multiple devices by matching interfaces based on a list of PCI IDs. You can use this procedure to connect multiple device PCI IDs to the single connection profile.

Prerequisites

- Multiple physical or virtual Ethernet devices exist in the server's configuration.
- A DHCP server is available in the network.
- No connection profile exists on the host.

Procedure

1. Identify the device path. For example, to display the device paths of all interfaces starting with **enp**, enter :

```
# udevadm info /sys/class/net/enp* | grep ID_PATH=
...
E: ID_PATH=pci-0000:07:00.0
E: ID_PATH=pci-0000:08:00.0
```

2. Add a connection profile that applies to all PCI IDs matching the **0000:00:0[7-8].0** expression:

```
# nmcli connection add type ethernet connection.multi-connect multiple match.path
"pci-0000:07:00.0 pci-0000:08:00.0" con-name "Wired connection 1"
```

Verification

1. Display the status of the connection:

```
# nmcli connection show
NAME                UUID                                TYPE    DEVICE
Wired connection 1  9cee0958-512f-4203-9d3d-b57af1d88466 ethernet enp7s0
Wired connection 1  9cee0958-512f-4203-9d3d-b57af1d88466 ethernet enp8s0
...
```

2. To display all settings of the connection profile:

```
# nmcli connection show "Wired connection 1"
connection.id:      Wired connection 1
...
connection.multi-connect: 3 (multiple)
match.path:         pci-0000:07:00.0,pci-0000:08:00.0
...
```

This connection profile uses all devices with a PCI ID which match the pattern in the **match.path** parameter and, therefore, the connection profiles have the same Universally Unique Identifier (UUID).

Additional resources

- **nmcli(1)** and **nm-settings(5)** man pages on your system

CHAPTER 3. CONFIGURING A NETWORK BOND

A network bond is a method to combine or aggregate physical and virtual network interfaces to provide a logical interface with higher throughput or redundancy. In a bond, the kernel handles all operations exclusively. You can create bonds on different types of devices, such as Ethernet devices or VLANs.

Red Hat Enterprise Linux provides administrators different options to configure team devices. For example:

- Use **nmcli** to configure bond connections using the command line.
- Use the RHEL web console to configure bond connections using a web browser.
- Use **nmtui** to configure bond connections in a text-based user interface.
- Use **nmstatectl** to configure bond connections through the Nmstate API.
- Use RHEL system roles to automate the bond configuration on one or multiple hosts.

3.1. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES

Consider the following default behavior when managing or troubleshooting bond port interfaces using the **NetworkManager** service:

- Starting the controller interface does not automatically start the port interfaces.
- Starting a port interface always starts the controller interface.
- Stopping the controller interface also stops the port interface.
- A controller without ports can start static IP connections.
- A controller without ports waits for ports when starting DHCP connections.
- A controller with a DHCP connection waiting for ports completes when you add a port with a carrier.
- A controller with a DHCP connection waiting for ports continues waiting when you add a port without a carrier.

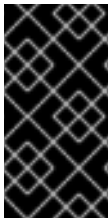
3.2. UPSTREAM SWITCH CONFIGURATION DEPENDING ON THE BONDING MODES

Depending on the bonding mode you want to use, you must configure the ports on the switch:

Bonding mode	Configuration on the switch
0 - balance-rr	Requires static EtherChannel enabled, not Link Aggregation Control Protocol (LACP)-negotiated.
1 - active-backup	No configuration required on the switch.

Bonding mode	Configuration on the switch
2 - balance-xor	Requires static EtherChannel enabled, not LACP-negotiated.
3 - broadcast	Requires static EtherChannel enabled, not LACP-negotiated.
4 - 802.3ad	Requires LACP-negotiated EtherChannel enabled.
5 - balance-tlb	No configuration required on the switch.
6 - balance-alb	No configuration required on the switch.
balance-slb	No configuration required on the switch.

For details how to configure your switch, see the documentation of the switch.



IMPORTANT

Certain network bonding features, such as the fail-over mechanism, do not support direct cable connections without a network switch. For further details, see the Red Hat Knowledgebase solution [Is bonding supported with direct connection using crossover cables](#).

3.3. CONFIGURING A NETWORK BOND BY USING `nmcli`

To configure a network bond on the command line, use the **`nmcli`** utility.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use bridge or VLAN devices as ports of the bond, you can either create these devices while you create the bond or you can create them in advance as described in:
 - [Configuring a network bridge by using `nmcli`](#)
 - [Configuring VLAN tagging by using `nmcli`](#)

Procedure

1. Create a bond interface:

```
# nmcli connection add type bond con-name bond0 ifname bond0 bond.options
"mode=active-backup"
```

This command creates a bond named **`bond0`** that uses the **`active-backup`** mode.

To additionally set a Media Independent Interface (MII) monitoring interval, add the **miimon=interval** option to the **bond.options** property, for example:

```
# nmcli connection add type bond con-name bond0 ifname bond0 bond.options
"mode=active-backup,miimon=1000"
```

2. Display the network interfaces, and note names of interfaces you plan to add to the bond:

```
# nmcli device status
DEVICE TYPE    STATE    CONNECTION
enp7s0 ethernet disconnected --
enp8s0 ethernet disconnected --
bridge0 bridge    connected bridge0
bridge1 bridge    connected bridge1
...
```

In this example:

- **enp7s0** and **enp8s0** are not configured. To use these devices as ports, add connection profiles in the next step.
- **bridge0** and **bridge1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

3. Assign interfaces to the bond:

- a. If the interfaces you want to assign to the bond are not configured, create new connection profiles for them:

```
# nmcli connection add type ethernet port-type bond con-name bond0-port1
ifname enp7s0 controller bond0
# nmcli connection add type ethernet port-type bond con-name bond0-port2
ifname enp8s0 controller bond0
```

These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **bond0** connection.

- b. To assign an existing connection profile to the bond:
 - i. Set the **controller** parameter of these connections to **bond0**:

```
# nmcli connection modify bridge0 controller bond0
# nmcli connection modify bridge1 controller bond0
```

These commands assign the existing connection profiles named **bridge0** and **bridge1** to the **bond0** connection.

- ii. Reactivate the connections:

```
# nmcli connection up bridge0
# nmcli connection up bridge1
```

4. Configure the IPv4 settings:

- If you plan to use this bond device as a port of other devices, enter:

—

```
# nmcli connection modify bond0 ipv4.method disabled
```

- To use DHCP, no action is required.
- To set a static IPv4 address, network mask, default gateway, and DNS server to the **bond0** connection, enter:

```
# nmcli connection modify bond0 ipv4.addresses '192.0.2.1/24' ipv4.gateway
'192.0.2.254' ipv4.dns '192.0.2.253' ipv4.dns-search 'example.com' ipv4.method
manual
```

5. Configure the IPv6 settings:

- If you plan to use this bond device as a port of other devices, enter:

```
# nmcli connection modify bond0 ipv6.method disabled
```

- To use stateless address autoconfiguration (SLAAC), no action is required.
- To set a static IPv6 address, network mask, default gateway, and DNS server to the **bond0** connection, enter:

```
# nmcli connection modify bond0 ipv6.addresses '2001:db8:1::1/64' ipv6.gateway
'2001:db8:1::ffe' ipv6.dns '2001:db8:1::fffd' ipv6.dns-search 'example.com'
ipv6.method manual
```

6. Optional: If you want to set any parameters on the bond ports, use the following command:

```
# nmcli connection modify bond0-port1 bond-port.<parameter> <value>
```

7. Activate the connection:

```
# nmcli connection up bond0
```

8. Verify that the ports are connected, and the **CONNECTION** column displays the port's connection name:

```
# nmcli device
DEVICE  TYPE    STATE    CONNECTION
...
enp7s0  ethernet connected bond0-port1
enp8s0  ethernet connected bond0-port2
```

When you activate any port of the connection, NetworkManager also activates the bond, but not the other ports of it. You can configure that Red Hat Enterprise Linux enables all ports automatically when the bond is enabled:

- Enable the **connection.autoconnect-ports** parameter of the bond's connection:

```
# nmcli connection modify bond0 connection.autoconnect-ports 1
```

- Reactivate the bridge:

```
# nmcli connection up bond0
```

Verification

1. Temporarily remove the network cable from one of the network devices and check if the other device in the bond is handling the traffic.
Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.
2. Display the status of the bond:

```
# cat /proc/net/bonding/bond0
```

3.4. CONFIGURING A NETWORK BOND BY USING THE RHEL WEB CONSOLE

Use the RHEL web console to configure a network bond if you prefer to manage network settings using a web browser-based interface.

Prerequisites

- You are logged in to the RHEL web console.
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as members of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use bridge or VLAN devices as members of the bond, create them in advance as described in:
 - [Configuring a network bridge by using the RHEL web console](#)
 - [Configuring VLAN tagging by using the RHEL web console](#)

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add bond** in the **Interfaces** section.
3. Enter the name of the bond device you want to create.
4. Select the interfaces that should be members of the bond.
5. Select the mode of the bond.
If you select **Active backup**, the web console shows the additional field **Primary** in which you can select the preferred active device.
6. Set the link monitoring mode. For example, when you use the **Adaptive load balancing** mode, set it to **ARP**.
7. Optional: Adjust the monitoring interval, link up delay, and link down delay settings. Typically, you only change the defaults for troubleshooting purposes.

Bond settings

Name

bond0

Interfaces

☒ enp7s0
☒ enp8s0

MAC

Mode

Active backup

Primary

enp7s0

Link monitoring

MII (recommended)

Monitoring interval

100

Link up delay

0

Link down delay

0

Apply

Cancel

8. Click **Apply**.
9. By default, the bond uses a dynamic IP address. If you want to set a static IP address:
 - a. Click the name of the bond in the **Interfaces** section.
 - b. Click **Edit** next to the protocol you want to configure.
 - c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
 - d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
 - e. In the **DNS search domains** section, click the **+** button, and enter the search domain.

- f. If the interface requires static routes, configure them in the **Routes** section.

IPv4 settings

Addresses

Manual

+

Address	Prefix length or netmask	Gateway
192.0.2.1	24	192.0.2.254

-

DNS

Automatic

+

Server

192.0.2.253

-

DNS search domains

Automatic

+

Search domain

example.com

-

Routes

Automatic

+

Apply

Cancel

- g. Click **Apply**

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:

Interfaces			
<div> <div>Add bond</div> <div>Add team</div> <div>Add bridge</div> <div>Add VLAN</div> </div>			
Name	IP address	Sending	Receiving
bond0	192.0.2.1/24	1.11 Mbps	61.2 Mbps

2. Temporarily remove the network cable from one of the network devices and check if the other device in the bond is handling the traffic.
Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as the web console, show only the bonding driver's ability to handle member configuration changes and not actual link failure events.
3. Display the status of the bond:

```
# cat /proc/net/bonding/bond0
```

3.5. CONFIGURING A NETWORK BOND BY USING **NMTUI**

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to configure a network bond on a host without a graphical interface.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bond, the physical or virtual Ethernet devices must be installed on the server.

Procedure

1. If you do not know the network device names on which you want configure a network bond, display the available devices:

```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

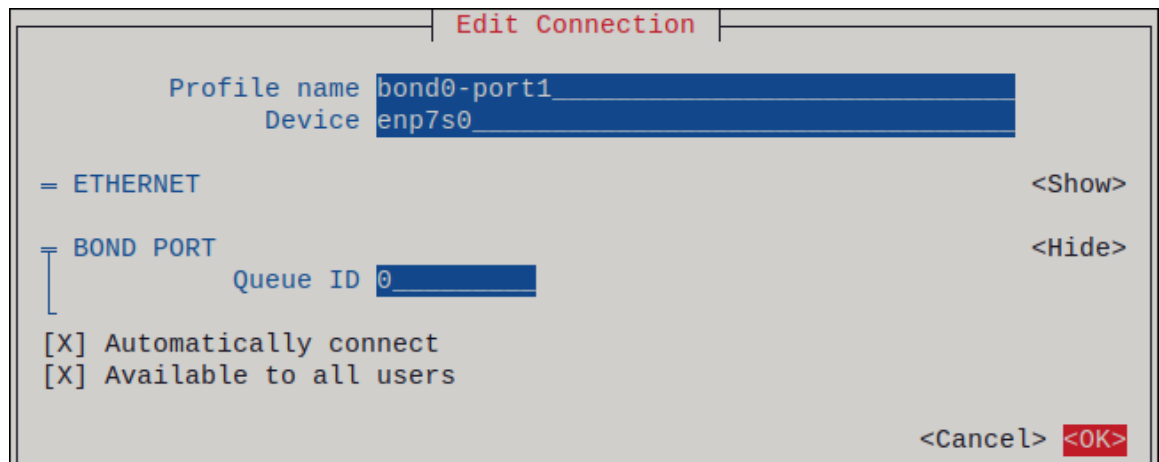
2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press **Add**.
5. Select **Bond** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.
7. Enter the bond device name to be created into the **Device** field.
8. Add ports to the bond to be created:
 - a. Press **Add** next to the **Slaves** list.

- b. Select the type of the interface you want to add as port to the bond, for example, **Ethernet**.
- c. Optional: Enter a name for the NetworkManager profile to be created for this bond port.
- d. Enter the port's device name into the **Device** field.
- e. Press **OK** to return to the window with the bond settings.

Figure 3.1. Adding an Ethernet device as port to a bond



- f. Repeat these steps to add more ports to the bond.
9. Set the bond mode. Depending on the value you set, **nmtui** displays additional fields for settings that are related to the selected mode.
 10. Depending on your environment, configure the IP address settings in the **IPv4 configuration** and **IPv6 configuration** areas accordingly. For this, press the button next to these areas, and select:
 - **Disabled**, if the bond does not require an IP address.
 - **Automatic**, if a DHCP server or stateless address autoconfiguration (SLAAC) dynamically assigns an IP address to the bond.
 - **Manual**, if the network requires static IP address settings. In this case, you must fill further fields:
 - i. Press **Show** next to the protocol you want to configure to display additional fields.
 - ii. Press **Add** next to **Addresses**, and enter the IP address and the subnet mask in Classless Inter-Domain Routing (CIDR) format.
If you do not specify a subnet mask, NetworkManager sets a **/32** subnet mask for IPv4 addresses and **/64** for IPv6 addresses.
 - iii. Enter the address of the default gateway.
 - iv. Press **Add** next to **DNS servers**, and enter the DNS server address.
 - v. Press **Add** next to **Search domains**, and enter the DNS search domain.

Figure 3.2. Example of a bond connection with static IP address settings

Edit Connection

Profile name
Device

BOND Slaves <Hide>

bond0-port1

bond0-port2

<Add>

<Edit...>

<Delete>

Mode

Primary

Link monitoring

Monitoring frequency ms

Link up delay ms

Link down delay ms

Cloned MAC address

IPv4 CONFIGURATION <Manual> <Hide>

Addresses <Remove>

<Add...>

Gateway

DNS servers <Remove>

<Add...>

Search domains <Add...>

Routing (No custom routes) <Edit...>

☐ Never use this network for default route
☐ Ignore automatically obtained routes
☐ Ignore automatically obtained DNS parameters

☐ Require IPv4 addressing for this connection

IPv6 CONFIGURATION <Manual> <Hide>

Addresses <Remove>

<Add...>

Gateway

DNS servers <Remove>

<Add...>

Search domains <Add...>

Routing (No custom routes) <Edit...>

☐ Never use this network for default route
☐ Ignore automatically obtained routes
☐ Ignore automatically obtained DNS parameters

☐ Require IPv6 addressing for this connection

☒ Automatically connect
☒ Available to all users

<Cancel> OK

11. Press **OK** to create and automatically activate the new connection.

12. Press **Back** to return to the main menu.
13. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

1. Temporarily remove the network cable from one of the network devices and check if the other device in the bond is handling the traffic.
Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.
2. Display the status of the bond:

```
# cat /proc/net/bonding/bond0
```

3.6. CONFIGURING A NETWORK BOND BY USING **NMSTATECTL**

Use the **nmstatectl** utility to configure a network bond through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Depending on your environment, adjust the YAML file accordingly. For example, to use different devices than Ethernet adapters in the bond, adapt the **base-iface** attribute and **type** attributes of the ports you use in the bond.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports in the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bridge, or VLAN devices as ports in the bond, set the interface name in the **port** list, and define the corresponding interfaces.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-bond.yml**, with the following content:

```
---
interfaces:
- name: bond0
  type: bond
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
```

```

enabled: true
address:
  - ip: 2001:db8:1::1
    prefix-length: 64
  autoconf: false
  dhcp: false
link-aggregation:
  mode: active-backup
  port:
    - enp1s0
    - enp7s0
  - name: enp1s0
    type: ethernet
    state: up
  - name: enp7s0
    type: ethernet
    state: up

routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-address: 192.0.2.254
      next-hop-interface: bond0
    - destination: ::0
      next-hop-address: 2001:db8:1::fffe
      next-hop-interface: bond0

dns-resolver:
  config:
    search:
      - example.com
    server:
      - 192.0.2.200
      - 2001:db8:1::ffbb

```

These settings define a network bond with the following settings:

- Network interfaces in the bond: **enp1s0** and **enp7s0**
- Mode: **active-backup**
- Static IPv4 address: **192.0.2.1** with a **/24** subnet mask
- Static IPv6 address: **2001:db8:1::1** with a **/64** subnet mask
- IPv4 default gateway: **192.0.2.254**
- IPv6 default gateway: **2001:db8:1::fffe**
- IPv4 DNS server: **192.0.2.200**
- IPv6 DNS server: **2001:db8:1::ffbb**
- DNS search domain: **example.com**

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-bond.yml
```

Verification

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
bond0     bond    connected bond0
```

2. Display all settings of the connection profile:

```
# nmcli connection show bond0
connection.id:      bond0
connection.uuid:    79cbc3bd-302e-4b1f-ad89-f12533b818ee
connection.stable-id: --
connection.type:    bond
connection.interface-name: bond0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show bond0
```

Additional resources

- **nmstatectl(8)** man page on your system
- `/usr/share/doc/nmstate/examples/` directory

3.7. CONFIGURING A NETWORK BOND BY USING THE `network` RHEL SYSTEM ROLE

You can combine network interfaces in a bond to provide a logical interface with higher throughput or redundancy. To configure a bond, create a NetworkManager connection profile. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure a network bond and, if a connection profile for the bond's parent device does not exist, the role can create it as well.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- Two or more physical or virtual network devices are installed on the server.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Bond connection profile with two Ethernet ports
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # Bond profile
          - name: bond0
            type: bond
            interface_name: bond0
            ip:
              dhcp4: yes
              auto6: yes
            bond:
              mode: active-backup
            state: up

          # Port profile for the 1st Ethernet device
          - name: bond0-port1
            interface_name: enp7s0
            type: ethernet
            controller: bond0
            state: up

          # Port profile for the 2nd Ethernet device
          - name: bond0-port2
            interface_name: enp8s0
            type: ethernet
            controller: bond0
            state: up
```

The settings specified in the example playbook include the following:

type: `<profile_type>`

Sets the type of the profile to create. The example playbook creates three connection profiles: One for the bond and two for the Ethernet devices.

dhcp4: `yes`

Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: `yes`

Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

mode: `<bond_mode>`

Sets the bonding mode. Possible values are:

- **balance-rr** (default)
- **active-backup**

- **balance-xor**
- **broadcast**
- **802.3ad**
- **balance-tlb**
- **balance-alb**

Depending on the mode you set, you need to set additional variables in the playbook.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Temporarily remove the network cable from one of the network devices and check if the other device in the bond is handling the traffic.
Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory

3.8. THE DIFFERENT NETWORK BONDING MODES

The Linux bonding driver provides link aggregation. Bonding is the process of aggregating multiple network interfaces in parallel to provide a single logical bonded interface. The actions of a bonded interface depend on the bonding policy that is also known as mode. The different modes provide either load-balancing or hot standby services.

The Linux bonding driver supports the following modes:

Balance-rr (Mode 0)

Balance-rr uses the round-robin algorithm that sequentially transmits packets from the first available port to the last one. This mode provides load balancing and fault tolerance.

This mode requires switch configuration of a port aggregation group, also called EtherChannel or similar port grouping. An EtherChannel is a port link aggregation technology to group multiple physical Ethernet links to one logical Ethernet link.

The drawback of this mode is that it is not suitable for heavy workloads and if TCP throughput or ordered packet delivery is essential.

Active-backup (Mode 1)

Active-backup uses the policy that determines that only one port is active in the bond. This mode provides fault tolerance and does not require any switch configuration.

If the active port fails, an alternate port becomes active. The bond sends a gratuitous address resolution protocol (ARP) response to the network. The gratuitous ARP forces the receiver of the ARP frame to update their forwarding table. The **Active-backup** mode transmits a gratuitous ARP to announce the new path to maintain connectivity for the host.

The **primary** option defines the preferred port of the bonding interface.

Balance-xor (Mode 2)

Balance-xor uses the selected transmit hash policy to send the packets. This mode provides load balancing, fault tolerance, and requires switch configuration to set up an Etherchannel or similar port grouping.

To alter packet transmission and balance transmit, this mode uses the **xmit_hash_policy** option. Depending on the source or destination of traffic on the interface, the interface requires an additional load-balancing configuration. See description [xmit_hash_policy bonding parameter](#).

Broadcast (Mode 3)

Broadcast uses a policy that transmits every packet on all interfaces. This mode provides fault tolerance and requires a switch configuration to set up an EtherChannel or similar port grouping. The drawback of this mode is that it is not suitable for heavy workloads and if TCP throughput or ordered packet delivery is essential.

802.3ad (Mode 4)

802.3ad uses the same-named IEEE standard dynamic link aggregation policy. This mode provides fault tolerance. This mode requires switch configuration to set up a Link Aggregation Control Protocol (LACP) port grouping.

This mode creates aggregation groups that share the same speed and duplex settings and utilizes all ports in the active aggregator. Depending on the source or destination of traffic on the interface, this mode requires an additional load-balancing configuration.

By default, the port selection for outgoing traffic depends on the transmit hash policy. Use the **xmit_hash_policy** option of the transmit hash policy to change the port selection and balance transmit.

The difference between the **802.3ad** and the **Balance-xor** is compliance. The **802.3ad** policy negotiates LACP between the port aggregation groups. See description [xmit_hash_policy bonding parameter](#)

Balance-tlb (Mode 5)

Balance-tlb uses the transmit load balancing policy. This mode provides fault tolerance, load balancing, and establishes channel bonding that does not require any switch support.

The active port receives the incoming traffic. In case of failure of the active port, another one takes over the MAC address of the failed port. To decide which interface processes the outgoing traffic, use one of the following modes:

- Value **0**: Uses the hash distribution policy to distribute traffic without load balancing
- Value **1**: Distributes traffic to each port by using load balancing
With the bonding option **tlb_dynamic_lb=0**, this bonding mode uses the **xmit_hash_policy** bonding option to balance transmit. The **primary** option defines the preferred port of the bonding interface.

See description [xmit_hash_policy bonding parameter](#).

Balance-alb (Mode 6)

Balance-alb uses an adaptive load balancing policy. This mode provides fault tolerance, load balancing, and does not require any special switch support.

This mode Includes balance-transmit load balancing (**balance-tlb**) and receive-load balancing for IPv4 and IPv6 traffic. The bonding intercepts ARP replies sent by the local system and overwrites the source hardware address of one of the ports in the bond. ARP negotiation manages the receive-load balancing. Therefore, different ports use different hardware addresses for the server.

The **primary** option defines the preferred port of the bonding interface. With the bonding option **tlb_dynamic_lb=0**, this bonding mode uses the **xmit_hash_policy** bonding option to balance transmit. See description [xmit_hash_policy bonding parameter](#).

Additionally, you can use NetworkManager to configure the following mode:

Balance-slb

The source load balancing (SLB) bonding mode distributes outgoing data streams across multiple network interfaces based on the source address of the traffic and a VLAN hash. This mode does not require any switch configuration.

NetworkManager uses the **balance-xor** mode in combination with **nftables** rules to provide SLB. For details about configuring this mode, see [Configuring a network bond on RHEL with source load balancing](#).

Additional resources

- [/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.rst](#) provided by the **kernel-doc** package
- [/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.txt](#) provided by the **kernel-doc** package
- [Which bonding modes work when used with a bridge that virtual machine guests or containers connect to?](#) (Red Hat Knowledgebase)
- [How are the values for different policies in "xmit_hash_policy" bonding parameter calculated?](#) (Red Hat Knowledgebase)

3.9. THE XMIT_HASH_POLICY BONDING PARAMETER

The **xmit_hash_policy** load balancing parameter selects the transmit hash policy for a node selection in the **balance-xor**, **802.3ad**, **balance-alb**, and **balance-tlb** modes. It is only applicable to mode 5 and 6 if the **tlb_dynamic_lb** parameter is 0. The possible values of this parameter are **layer2**, **layer2+3**, **layer3+4**, **encap2+3**, **encap3+4**, and **vlan+srcmac**.

Refer the table for details:

Policy or Network layers	Layer2	Layer2+3	Layer3+4	encap2+3	encap3+4	VLAN+src mac
Uses	XOR of source and destination MAC addresses and Ethernet protocol type	XOR of source and destination MAC addresses and IP addresses	XOR of source and destination ports and IP addresses	XOR of source and destination MAC addresses and IP addresses inside a supported tunnel, for example, Virtual Extensible LAN (VXLAN). This mode relies on skb_flow_dissect() function to obtain the header fields	XOR of source and destination ports and IP addresses inside a supported tunnel, for example, VXLAN. This mode relies on skb_flow_dissect() function to obtain the header fields	XOR of VLAN ID and source MAC vendor and source MAC device
Placement of traffic	All traffic to a particular network peer on the same underlying network interface	All traffic to a particular IP address on the same underlying network interface	All traffic to a particular IP address and port on the same underlying network interface			

Primary choice	If network traffic is between this system and multiple other systems in the same broadcast domain	If network traffic between this system and multiple other systems goes through a default gateway	If network traffic between this system and another system uses the same IP addresses but goes through multiple ports	The encapsulated traffic is between the source system and multiple other systems using multiple IP addresses	The encapsulated traffic is between the source system and other systems using multiple port numbers	If the bond carries network traffic, from multiple containers or virtual machines (VM), that expose their MAC address directly to the external network such as the bridge network, and you can not configure a switch for Mode 2 or Mode 4
Secondary choice	If network traffic is mostly between this system and multiple other systems behind a default gateway	If network traffic is mostly between this system and another system				
Compliant	802.3ad	802.3ad	Not 802.3ad			
Default policy	This is the default policy if no configuration is provided	For non-IP traffic, the formula is the same as for the layer2 transmit policy	For non-IP traffic, the formula is the same as for the layer2 transmit policy			

CHAPTER 4. CONFIGURING VLAN TAGGING

A Virtual Local Area Network (VLAN) is a logical network within a physical network. The VLAN interface tags packets with the VLAN ID as they pass through the interface, and removes tags of returning packets. You create VLAN interfaces on top of another interface, such as Ethernet, bond, team, or bridge devices. These interfaces are called the **parent interface**.

Red Hat Enterprise Linux provides administrators different options to configure VLAN devices. For example:

- Use **nmcli** to configure VLAN tagging using the command line.
- Use the RHEL web console to configure VLAN tagging using a web browser.
- Use **nmtui** to configure VLAN tagging in a text-based user interface.
- Use **nmstatectl** to configure connections through the Nmstate API.
- Use RHEL system roles to automate the VLAN configuration on one or multiple hosts.

4.1. CONFIGURING VLAN TAGGING BY USING **NMCLI**

You can configure Virtual Local Area Network (VLAN) tagging on the command line using the **nmcli** utility.

Prerequisites

- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
 - The ports of the bond are up.
 - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the incorrect source MAC address.
 - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by setting the **ipv4.method=disable** and **ipv6.method=ignore** options while creating the bond. Otherwise, if DHCP or IPv6 auto-configuration fails after some time, the interface might be brought down.
- The switch, the host is connected to, is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Display the network interfaces:

```
# nmcli device status
DEVICE  TYPE    STATE    CONNECTION
enp1s0  ethernet disconnected enp1s0
bridge0 bridge   connected bridge0
bond0   bond     connected bond0
...
```

2. Create the VLAN interface. For example, to create a VLAN interface named **vlan10** that uses **enp1s0** as its parent interface and that tags packets with VLAN ID **10**, enter:

```
# nmcli connection add type vlan con-name vlan10 ifname vlan10 vlan.parent enp1s0
vlan.id 10
```

Note that the VLAN must be within the range from **0** to **4094**.

3. By default, the VLAN connection inherits the maximum transmission unit (MTU) from the parent interface. Optionally, set a different MTU value:

```
# nmcli connection modify vlan10 ethernet.mtu 2000
```

4. Configure the IPv4 settings:

- If you plan to use this VLAN device as a port of other devices, enter:

```
# nmcli connection modify vlan10 ipv4.method disabled
```

- To use DHCP, no action is required.
- To set a static IPv4 address, network mask, default gateway, and DNS server to the **vlan10** connection, enter:

```
# nmcli connection modify vlan10 ipv4.addresses '192.0.2.1/24' ipv4.gateway
'192.0.2.254' ipv4.dns '192.0.2.253' ipv4.method manual
```

5. Configure the IPv6 settings:

- If you plan to use this VLAN device as a port of other devices, enter:

```
# nmcli connection modify vlan10 ipv6.method disabled
```

- To use stateless address autoconfiguration (SLAAC), no action is required.
- To set a static IPv6 address, network mask, default gateway, and DNS server to the **vlan10** connection, enter:

```
# nmcli connection modify vlan10 ipv6.addresses '2001:db8:1::1/32' ipv6.gateway
'2001:db8:1::fffe' ipv6.dns '2001:db8:1::fffd' ipv6.method manual
```

6. Activate the connection:

```
# nmcli connection up vlan10
```

Verification

- Verify the settings:

```
# ip -d addr show vlan10
4: vlan10@enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 52:54:00:72:2f:6e brd ff:ff:ff:ff:ff:ff promiscuity 0
```

```

vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute vlan10
    valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/32 scope global noprefixroute
    valid_lft forever preferred_lft forever
inet6 fe80::8dd7:9030:6f8e:89e6/64 scope link noprefixroute
    valid_lft forever preferred_lft forever

```

Additional resources

- **nm-settings(5)** man page on your system

4.2. CONFIGURING NESTED VLANS BY USING NMCLI

802.1ad is a protocol used for Virtual Local Area Network (VLAN) tagging. It is also known as Q-in-Q tagging. You can use this technology to create multiple VLAN tags within a single Ethernet frame to achieve the following benefits:

- Increased network scalability by creating multiple isolated network segments within a VLAN. This enables you to segment and organize large networks into smaller, manageable units.
- Improved traffic management by isolating and controlling different types of network traffic. This can improve the network performance and reduce network congestion.
- Efficient resource utilization by enabling the creation of smaller, more targeted network segments.
- Enhanced security by isolating network traffic and reducing the risk of unauthorized access to sensitive data.

Prerequisites

- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
 - The ports of the bond are up.
 - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the incorrect source MAC address.
 - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by setting the **ipv4.method=disable** and **ipv6.method=ignore** options while creating the bond. Otherwise, if DHCP or IPv6 auto-configuration fails after some time, the interface might be brought down.
- The switch, the host is connected to, is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Display the physical network devices:

```
# nmcli device status
DEVICE  TYPE    STATE    CONNECTION
enp1s0  ethernet connected enp1s0
...
```

2. Create the base VLAN interface. For example, to create a base VLAN interface named **vlan10** that uses **enp1s0** as its parent interface and that tags packets with VLAN ID **10**, enter:

```
# nmcli connection add type vlan con-name vlan10 dev enp1s0 vlan.id 10
```

Note that the VLAN must be within the range from **0** to **4094**.

3. By default, the VLAN connection inherits the maximum transmission unit (MTU) from the parent interface. Optionally, set a different MTU value:

```
# nmcli connection modify vlan10 ethernet.mtu 2000
```

4. Create the nested VLAN interface on top of the base VLAN interface:

```
# nmcli connection add type vlan con-name vlan10.20 dev enp1s0.10 id 20
vlan.protocol 802.1ad
```

This command creates a new VLAN connection with a name of **vlan10.20** and a VLAN ID of **20** on the parent VLAN connection **vlan10**. The **dev** option specifies the parent network device. In this case it is **enp1s0.10**. The **vlan.protocol** option specifies the VLAN encapsulation protocol. In this case it is **802.1ad** (Q-in-Q).

5. Configure the IPv4 settings of the nested VLAN interface:

- To use DHCP, no action is required.
- To set a static IPv4 address, network mask, default gateway, and DNS server to the **vlan10.20** connection, enter:

```
# nmcli connection modify vlan10.20 ipv4.method manual ipv4.addresses
192.0.2.1/24 ipv4.gateway 192.0.2.254 ipv4.dns 192.0.2.200
```

6. Configure the IPv6 settings of the nested VLAN interface:

- To use stateless address autoconfiguration (SLAAC), no action is required.
- To set a static IPv4 address, network mask, default gateway, and DNS server to the **vlan10** connection, enter:

```
# nmcli connection modify vlan10 ipv4.addresses '192.0.2.1/24' ipv4.gateway
'192.0.2.254' ipv4.dns '192.0.2.253' ipv4.method manual
```

7. Activate the profile:

```
# nmcli connection up vlan10.20
```

Verification

1. Verify the configuration of the nested VLAN interface:

ip -d addr show enp1s0.10.20

```

10: enp1s0.10.20@enp1s0.10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc noqueue state UP group default qlen 1000
    link/ether 52:54:00:d2:74:3e brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 0 maxmtu 65535
    vlan protocol 802.1ad id 20 <REORDER_HDR> numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535 tso_max_size 65536 tso_max_segs 65535
gro_max_size 65536
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute enp1s0.10.20
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::ce3b:84c5:9ef8:d0e6/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

Additional resources

- **nm-settings(5)** man page on your system

4.3. CONFIGURING VLAN TAGGING BY USING THE RHEL WEB CONSOLE

Use the RHEL web console to configure VLAN tagging if you prefer to manage network settings using a web browser-based interface.

Prerequisites

- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
 - The ports of the bond are up.
 - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the incorrect source MAC address.
 - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by disabling the IPv4 and IPv6 protocol creating the bond. Otherwise, if DHCP or IPv6 auto-configuration fails after some time, the interface might be brought down.
- The switch, the host is connected to, is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add VLAN** in the **Interfaces** section.
3. Select the parent device.
4. Enter the VLAN ID.
5. Enter the name of the VLAN device or keep the automatically-generated name.

VLAN settings ✕

Parent	enp1s0 ▼
VLAN ID	10
Name	enp1s0.10

Apply Cancel

6. Click **Apply**.
7. By default, the VLAN device uses a dynamic IP address. If you want to set a static IP address:
 - a. Click the name of the VLAN device in the **Interfaces** section.
 - b. Click **Edit** next to the protocol you want to configure.
 - c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
 - d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
 - e. In the **DNS search domains** section, click the **+** button, and enter the search domain.
 - f. If the interface requires static routes, configure them in the **Routes** section.

IPv4 settings

Addresses

Manual

+

Address	Prefix length or netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS

Automatic

+

Server

192.0.2.253

-

DNS search domains

Automatic

+

Search domain

example.com

-

Routes

Automatic

+

Apply

Cancel

g. Click **Apply**

Verification

- Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:

Interfaces			
<div> <div>Add bond</div> <div>Add team</div> <div>Add bridge</div> <div>Add VLAN</div> </div>			
Name	IP address	Sending	Receiving
enp1s0.10	192.0.2.1/24	1.11 Mbps	61.2 Mbps

4.4. CONFIGURING VLAN TAGGING BY USING NMTUI

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to configure VLAN tagging on a host without a graphical interface.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Prerequisites

- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
 - The ports of the bond are up.
 - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the then incorrect source MAC address.
 - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by setting the **ipv4.method=disable** and **ipv6.method=ignore** options while creating the bond. Otherwise, if DHCP or IPv6 auto-configuration fails after some time, the interface might be brought down.
- The switch the host is connected to is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. If you do not know the network device name on which you want configure VLAN tagging, display the available devices:

```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp1s0  ethernet unavailable --
...
```

2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press **Add**.
5. Select **VLAN** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.

7. Enter the VLAN device name to be created into the **Device** field.
8. Enter the name of the device on which you want to configure VLAN tagging into the **Parent** field.
9. Enter the VLAN ID. The ID must be within the range from **0** to **4094**.
10. Depending on your environment, configure the IP address settings in the **IPv4 configuration** and **IPv6 configuration** areas accordingly. For this, press the button next to these areas, and select:
 - **Disabled**, if this VLAN device does not require an IP address or you want to use it as a port of other devices.
 - **Automatic**, if a DHCP server or stateless address autoconfiguration (SLAAC) dynamically assigns an IP address to the VLAN device.
 - **Manual**, if the network requires static IP address settings. In this case, you must fill further fields:
 - i. Press **Show** next to the protocol you want to configure to display additional fields.
 - ii. Press **Add** next to **Addresses**, and enter the IP address and the subnet mask in Classless Inter-Domain Routing (CIDR) format.
If you do not specify a subnet mask, NetworkManager sets a **/32** subnet mask for IPv4 addresses and **/64** for IPv6 addresses.
 - iii. Enter the address of the default gateway.
 - iv. Press **Add** next to **DNS servers**, and enter the DNS server address.
 - v. Press **Add** next to **Search domains**, and enter the DNS search domain.

Figure 4.1. Example of a VLAN connection with static IP address settings

The screenshot shows the 'Edit Connection' window in the nmtui application. The window is titled 'Edit Connection' in red text at the top right. It contains several sections for configuring a network connection:

- Profile name:** vlan10
- Device:** vlan10
- VLAN section:**
 - Parent:** enp1s0
 - VLAN id:** 10
 - Cloned MAC address:** (empty field)
 - MTU:** (empty field) (default)
- IPv4 CONFIGURATION <Manual>** (with a <Hide> button):
 - Addresses:** 192.0.2.1/24 (with <Remove> and <Add...> buttons)
 - Gateway:** 192.0.2.254
 - DNS servers:** 192.0.2.253 (with <Remove> and <Add...> buttons)
 - Search domains:** <Add...>
 - Routing (No custom routes) <Edit...>**
 - ☐ Never use this network for default route
 - ☐ Ignore automatically obtained routes
 - ☐ Ignore automatically obtained DNS parameters
 - ☐ Require IPv4 addressing for this connection
- IPv6 CONFIGURATION <Manual>** (with a <Hide> button):
 - Addresses:** 2001:db8:1::1/32 (with <Remove> and <Add...> buttons)
 - Gateway:** 2001:db8:1::fffe
 - DNS servers:** 2001:db8:1::fffd (with <Remove> and <Add...> buttons)
 - Search domains:** <Add...>
 - Routing (No custom routes) <Edit...>**
 - ☐ Never use this network for default route
 - ☐ Ignore automatically obtained routes
 - ☐ Ignore automatically obtained DNS parameters
 - ☐ Require IPv6 addressing for this connection
- Global options:**
 - ☒ Automatically connect
 - ☒ Available to all users

At the bottom right, there are buttons for '<Cancel>' and '<OK>'.

11. Press **OK** to create and automatically activate the new connection.
12. Press **Back** to return to the main menu.
13. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

- Verify the settings:

```
# ip -d addr show vlan10
4: vlan10@enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 52:54:00:72:2f:6e brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute vlan10
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::8dd7:9030:6f8e:89e6/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

4.5. CONFIGURING VLAN TAGGING BY USING NMSTATECTL

Use the **nmstatectl** utility to configure Virtual Local Area Network VLAN through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Depending on your environment, adjust the YAML file accordingly. For example, to use different devices than Ethernet adapters in the VLAN, adapt the **base-iface** attribute and **type** attributes of the ports you use in the VLAN.

Prerequisites

- To use Ethernet devices as ports in the VLAN, the physical or virtual Ethernet devices must be installed on the server.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-vlan.yml**, with the following content:

```
---
interfaces:
- name: vlan10
  type: vlan
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
      - ip: 2001:db8:1::1
        prefix-length: 64
    autoconf: false
    dhcp: false
  vlan:
```

```

    base-iface: enp1s0
    id: 10
- name: enp1s0
  type: ethernet
  state: up

routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-address: 192.0.2.254
      next-hop-interface: vlan10
    - destination: ::0
      next-hop-address: 2001:db8:1::fffe
      next-hop-interface: vlan10

dns-resolver:
  config:
    search:
      - example.com
    server:
      - 192.0.2.200
      - 2001:db8:1::ffbb

```

These settings define a VLAN with ID 10 that uses the **enp1s0** device. As the child device, the VLAN connection has the following settings:

- A static IPv4 address – **192.0.2.1** with the **/24** subnet mask
- A static IPv6 address – **2001:db8:1::1** with the **/64** subnet mask
- An IPv4 default gateway – **192.0.2.254**
- An IPv6 default gateway – **2001:db8:1::fffe**
- An IPv4 DNS server – **192.0.2.200**
- An IPv6 DNS server – **2001:db8:1::ffbb**
- A DNS search domain – **example.com**

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-vlan.yml
```

Verification

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE  TYPE  STATE  CONNECTION
vlan10  vlan  connected  vlan10
```

2. Display all settings of the connection profile:

```
# nmcli connection show vlan10
```

```

connection.id:          vlan10
connection.uuid:        1722970f-788e-4f81-bd7d-a86bf21c9df5
connection.stable-id:    --
connection.type:         vlan
connection.interface-name: vlan10
...

```

3. Display the connection settings in YAML format:

```
# nmstatectl show vlan0
```

Additional resources

- **nmstatectl(8)** man page on your system
- `/usr/share/doc/nmstate/examples/` directory

4.6. CONFIGURING VLAN TAGGING BY USING THE NETWORK RHEL SYSTEM ROLE

If your network uses Virtual Local Area Networks (VLANs) to separate network traffic into logical networks, create a NetworkManager connection profile to configure VLAN tagging. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure VLAN tagging and, if a connection profile for the VLAN's parent device does not exist, the role can create it as well.



NOTE

If the VLAN device requires an IP address, default gateway, and DNS settings, configure them on the VLAN device and not on the parent device.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```

---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: VLAN connection profile with Ethernet port
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:

```

```

network_connections:
  # Ethernet profile
  - name: enp1s0
    type: ethernet
    interface_name: enp1s0
    autoconnect: yes
    state: up
    ip:
      dhcp4: no
      auto6: no

  # VLAN profile
  - name: enp1s0.10
    type: vlan
    vlan:
      id: 10
    ip:
      dhcp4: yes
      auto6: yes
    parent: enp1s0
    state: up

```

e settings specified in the example playbook include the following:

type: <profile_type>

Sets the type of the profile to create. The example playbook creates two connection profiles: One for the parent Ethernet device and one for the VLAN device.

dhcp4: <value>

If set to **yes**, automatic IPv4 address assignment from DHCP, PPP, or similar services is enabled. Disable the IP address configuration on the parent device.

auto6: <value>

If set to **yes**, IPv6 auto-configuration is enabled. In this case, by default, NetworkManager uses Router Advertisements and, if the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server. Disable the IP address configuration on the parent device.

parent: <parent_device>

Sets the parent device of the VLAN connection profile. In the example, the parent is the Ethernet interface.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```


Verification

- Verify the VLAN settings:

```
# ansible managed-node-01.example.com -m command -a 'ip -d addr show enp1s0.10'
managed-node-01.example.com | CHANGED | rc=0 >>
4: vlan10@enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 52:54:00:72:2f:6e brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
    gso_max_size 65536 gso_max_segs 65535
    ...
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

CHAPTER 5. CONFIGURING A NETWORK BRIDGE

A Virtual Local Area Network (VLAN) is a logical network within a physical network. The VLAN interface tags packets with the VLAN ID as they pass through the interface, and removes tags of returning packets. You create VLAN interfaces on top of another interface, such as Ethernet, bond, team, or bridge devices. These interfaces are called the **parent interface**.

Red Hat Enterprise Linux provides administrators different options to configure VLAN devices. For example:

- Use **nmcli** to configure VLAN tagging using the command line.
- Use the RHEL web console to configure VLAN tagging using a web browser.
- Use **nmtui** to configure VLAN tagging in a text-based user interface.
- Use **nmstatectl** to configure connections through the Nmstate API.
- Use RHEL system roles to automate the VLAN configuration on one or multiple hosts.

5.1. CONFIGURING A NETWORK BRIDGE BY USING **NMCLI**

To configure a network bridge on the command line, use the **nmcli** utility.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use bond or VLAN devices as ports of the bridge, you can either create these devices while you create the bridge or you can create them in advance as described in:
 - [Configuring a network bond by using nmcli](#)
 - [Configuring VLAN tagging by using nmcli](#)

Procedure

1. Create a bridge interface:

```
# nmcli connection add type bridge con-name bridge0 ifname bridge0
```

This command creates a bridge named **bridge0**, enter:

2. Display the network interfaces, and note the names of the interfaces you want to add to the bridge:

```
# nmcli device status
DEVICE TYPE    STATE    CONNECTION
enp7s0 ethernet disconnected --
enp8s0 ethernet disconnected --
```

```
bond0 bond connected bond0
bond1 bond connected bond1
...
```

In this example:

- **enp7s0** and **enp8s0** are not configured. To use these devices as ports, add connection profiles in the next step.
- **bond0** and **bond1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

3. Assign the interfaces to the bridge.

- If the interfaces you want to assign to the bridge are not configured, create new connection profiles for them:

```
# nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
ifname enp7s0 master bridge0
# nmcli connection add type ethernet slave-type bridge con-name bridge0-port2
ifname enp8s0 master bridge0
```

These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **bridge0** connection.

- If you want to assign an existing connection profile to the bridge:

- Set the **master** parameter of these connections to **bridge0**:

```
# nmcli connection modify bond0 master bridge0
# nmcli connection modify bond1 master bridge0
```

These commands assign the existing connection profiles named **bond0** and **bond1** to the **bridge0** connection.

- Reactivate the connections:

```
# nmcli connection up bond0
# nmcli connection up bond1
```

4. Configure the IPv4 settings:

- If you plan to use this bridge device as a port of other devices, enter:

```
# nmcli connection modify bridge0 ipv4.method disabled
```

- To use DHCP, no action is required.
- To set a static IPv4 address, network mask, default gateway, and DNS server to the **bridge0** connection, enter:

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24' ipv4.gateway
'192.0.2.254' ipv4.dns '192.0.2.253' ipv4.dns-search 'example.com' ipv4.method
manual
```

5. Configure the IPv6 settings:

- If you plan to use this bridge device as a port of other devices, enter:

```
# nmcli connection modify bridge0 ipv6.method disabled
```

- To use stateless address autoconfiguration (SLAAC), no action is required.
- To set a static IPv6 address, network mask, default gateway, and DNS server to the **bridge0** connection, enter:

```
# nmcli connection modify bridge0 ipv6.addresses '2001:db8:1::1/64' ipv6.gateway
'2001:db8:1::fffe' ipv6.dns '2001:db8:1::fffd' ipv6.dns-search 'example.com'
ipv6.method manual
```

6. Optional: Configure further properties of the bridge. For example, to set the Spanning Tree Protocol (STP) priority of **bridge0** to **16384**, enter:

```
# nmcli connection modify bridge0 bridge.priority '16384'
```

By default, STP is enabled.

7. Activate the connection:

```
# nmcli connection up bridge0
```

8. Verify that the ports are connected, and the **CONNECTION** column displays the port's connection name:

```
# nmcli device
DEVICE  TYPE    STATE    CONNECTION
...
enp7s0  ethernet connected bridge0-port1
enp8s0  ethernet connected bridge0-port2
```

When you activate any port of the connection, NetworkManager also activates the bridge, but not the other ports of it. You can configure that Red Hat Enterprise Linux enables all ports automatically when the bridge is enabled:

- Enable the **connection.autoconnect-slaves** parameter of the bridge connection:

```
# nmcli connection modify bridge0 connection.autoconnect-slaves 1
```

- Reactivate the bridge:

```
# nmcli connection up bridge0
```

Verification

- Use the **ip** utility to display the link status of Ethernet devices that are ports of a specific bridge:

```
# ip link show master bridge0
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
```

```
bridge0 state UP mode DEFAULT group default qlen 1000
link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bridge0 state UP mode DEFAULT group default qlen 1000
link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
```

- Use the **bridge** utility to display the status of Ethernet devices that are ports of any bridge device:

```
# bridge link show
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
forwarding priority 32 cost 100
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
listening priority 32 cost 100
5: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
forwarding priority 32 cost 100
6: enp11s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
blocking priority 32 cost 100
...
```

To display the status for a specific Ethernet device, use the **bridge link show dev** *<ethernet_device_name>* command.

Additional resources

- **nm-settings(5)** and **bridge(8)** man pages on your system
- [NetworkManager duplicates a connection after restart of NetworkManager service](#) (Red Hat Knowledgebase)
- [How to configure a bridge with VLAN information?](#) (Red Hat Knowledgebase)

5.2. CONFIGURING A NETWORK BRIDGE BY USING THE RHEL WEB CONSOLE

Use the RHEL web console to configure a network bridge if you prefer to manage network settings using a web browser-based interface.

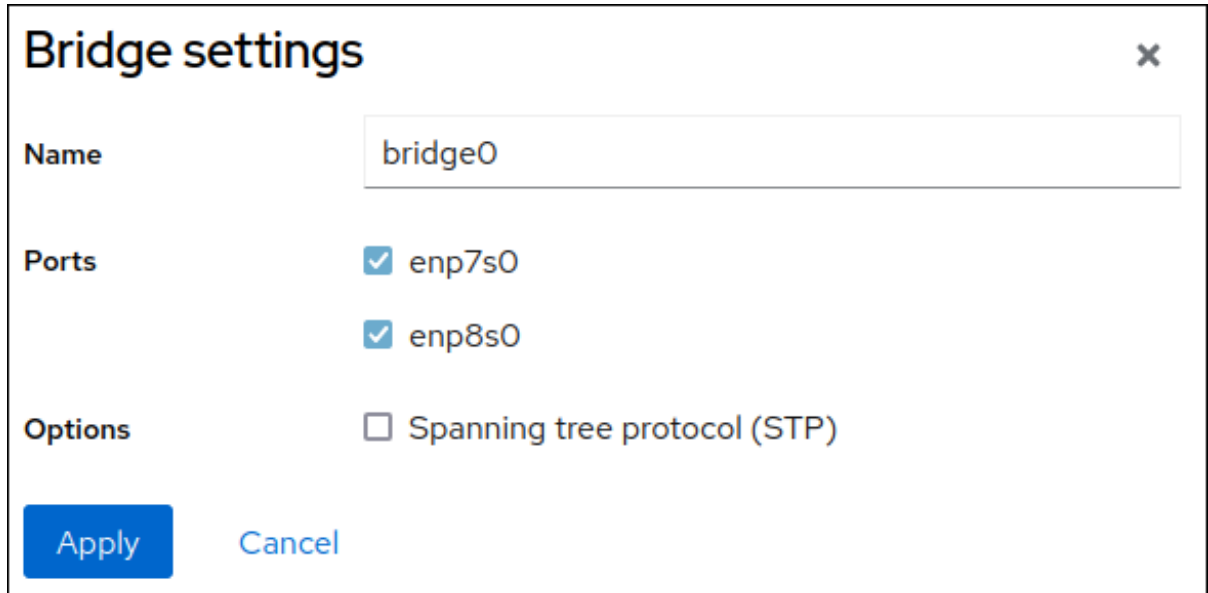
Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use bond or VLAN devices as ports of the bridge, you can either create these devices while you create the bridge or you can create them in advance as described in:
 - [Configuring a network bond by using the RHEL web console](#)
 - [Configuring VLAN tagging by using the RHEL web console](#)

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.

2. Click **Add bridge** in the **Interfaces** section.
3. Enter the name of the bridge device you want to create.
4. Select the interfaces that should be ports of the bridge.
5. Optional: Enable the **Spanning tree protocol (STP)** feature to avoid bridge loops and broadcast radiation.

A screenshot of the 'Bridge settings' dialog box. The dialog has a title bar with 'Bridge settings' and a close button (X). It contains three sections: 'Name' with a text input field containing 'bridge0'; 'Ports' with two checked checkboxes for 'enp7s0' and 'enp8s0'; and 'Options' with an unchecked checkbox for 'Spanning tree protocol (STP)'. At the bottom are two buttons: 'Apply' (blue) and 'Cancel' (light blue).

Bridge settings [X]

Name

Ports

- ☒ enp7s0
- ☒ enp8s0

Options

- ☐ Spanning tree protocol (STP)

Apply **Cancel**

6. Click **Apply**.
7. By default, the bridge uses a dynamic IP address. If you want to set a static IP address:
 - a. Click the name of the bridge in the **Interfaces** section.
 - b. Click **Edit** next to the protocol you want to configure.
 - c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
 - d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
 - e. In the **DNS search domains** section, click the **+** button, and enter the search domain.
 - f. If the interface requires static routes, configure them in the **Routes** section.

IPv4 settings

Addresses

Manual

+

Address	Prefix length or netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS

Automatic

+

Server

192.0.2.253

-

DNS search domains

Automatic

+

Search domain

example.com

-

Routes

Automatic

+

Apply

Cancel

g. Click **Apply**

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:

Interfaces			
<div> <div>Add bond</div> <div>Add team</div> <div>Add bridge</div> <div>Add VLAN</div> </div>			
Name	IP address	Sending	Receiving
bridge0	192.0.2.1/24	1.11 Mbps	61.2 Mbps

5.3. CONFIGURING A NETWORK BRIDGE BY USING `nmtui`

The `nmtui` application provides a text-based user interface for NetworkManager. You can use `nmtui` to configure a network bridge on a host without a graphical interface.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.

Procedure

1. If you do not know the network device names on which you want configure a network bridge, display the available devices:

```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

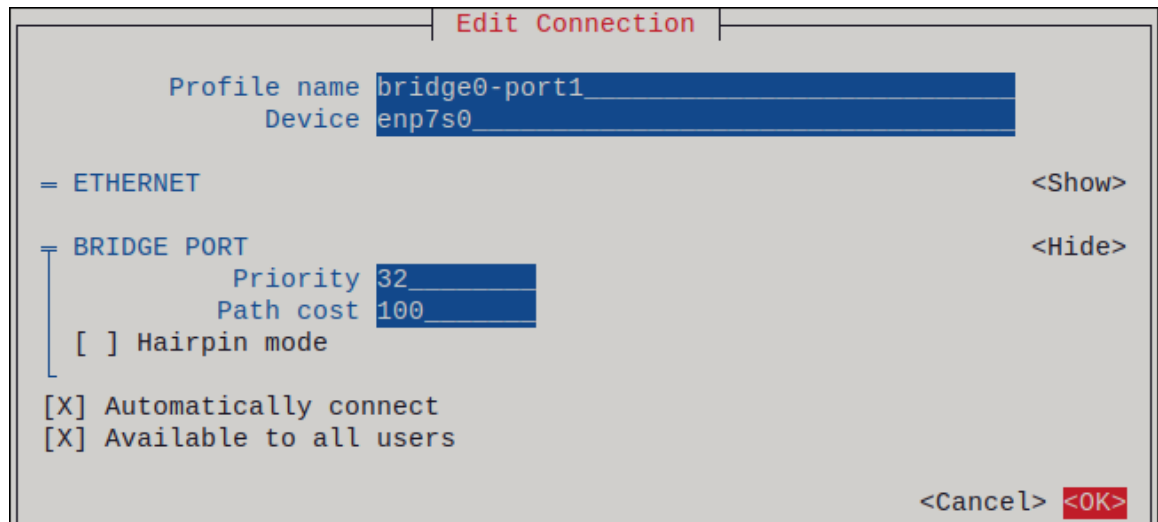
2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press **Add**.
5. Select **Bridge** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.
7. Enter the bridge device name to be created into the **Device** field.
8. Add ports to the bridge to be created:
 - a. Press **Add** next to the **Slaves** list.
 - b. Select the type of the interface you want to add as port to the bridge, for example, **Ethernet**.
 - c. Optional: Enter a name for the NetworkManager profile to be created for this bridge port.
 - d. Enter the port's device name into the **Device** field.

- e. Press **OK** to return to the window with the bridge settings.

Figure 5.1. Adding an Ethernet device as port to a bridge



- f. Repeat these steps to add more ports to the bridge.
9. Depending on your environment, configure the IP address settings in the **IPv4 configuration** and **IPv6 configuration** areas accordingly. For this, press the button next to these areas, and select:
- **Disabled**, if the bridge does not require an IP address.
 - **Automatic**, if a DHCP server or stateless address autoconfiguration (SLAAC) dynamically assigns an IP address to the bridge.
 - **Manual**, if the network requires static IP address settings. In this case, you must fill further fields:
 - i. Press **Show** next to the protocol you want to configure to display additional fields.
 - ii. Press **Add** next to **Addresses**, and enter the IP address and the subnet mask in Classless Inter-Domain Routing (CIDR) format.
If you do not specify a subnet mask, NetworkManager sets a **/32** subnet mask for IPv4 addresses and **/64** for IPv6 addresses.
 - iii. Enter the address of the default gateway.
 - iv. Press **Add** next to **DNS servers**, and enter the DNS server address.
 - v. Press **Add** next to **Search domains**, and enter the DNS search domain.

Figure 5.2. Example of a bridge connection without IP address settings

The screenshot shows the 'Edit Connection' window for a bridge named 'bridge0'. The 'Device' is also 'bridge0'. Under the 'BRIDGE Slaves' section, 'bridge0-port1' and 'bridge0-port2' are listed. The 'Aging time' is set to 300 seconds. The 'Enable IGMP snooping' and 'Enable STP (Spanning Tree Protocol)' checkboxes are both checked. The 'Priority' is 32768, 'Forward delay' is 15 seconds, 'Hello time' is 2 seconds, 'Max age' is 20 seconds, and 'Group forward mask' is 0. Both 'IPv4 CONFIGURATION' and 'IPv6 CONFIGURATION' are disabled. The 'Automatically connect' and 'Available to all users' checkboxes are also checked. The window has '<Cancel>' and '<OK>' buttons at the bottom right.

10. Press **OK** to create and automatically activate the new connection.
11. Press **Back** to return to the main menu.
12. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

1. Use the **ip** utility to display the link status of Ethernet devices that are ports of a specific bridge:

```
# ip link show master bridge0
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
```

2. Use the **bridge** utility to display the status of Ethernet devices that are ports of any bridge device:

```
# bridge link show
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
forwarding priority 32 cost 100
```

```
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
listening priority 32 cost 100
...
```

To display the status for a specific Ethernet device, use the **bridge link show dev** *<ethernet_device_name>* command.

5.4. CONFIGURING A NETWORK BRIDGE BY USING **NMSTATECTL**

Use the **nmstatectl** utility to configure a network bridge through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Depending on your environment, adjust the YAML file accordingly. For example, to use different devices than Ethernet adapters in the bridge, adapt the **base-iface** attribute and **type** attributes of the ports you use in the bridge.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports in the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports in the bridge, set the interface name in the **port** list, and define the corresponding interfaces.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-bridge.yml**, with the following content:

```
---
interfaces:
- name: bridge0
  type: linux-bridge
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
      - ip: 2001:db8:1::1
        prefix-length: 64
    autoconf: false
    dhcp: false
  bridge:
    options:
      stp:
        enabled: true
```

```

    port:
      - name: enp1s0
      - name: enp7s0
  - name: enp1s0
    type: ethernet
    state: up
  - name: enp7s0
    type: ethernet
    state: up

  routes:
    config:
      - destination: 0.0.0.0/0
        next-hop-address: 192.0.2.254
        next-hop-interface: bridge0
      - destination: ::0
        next-hop-address: 2001:db8:1::fffe
        next-hop-interface: bridge0
  dns-resolver:
    config:
      search:
        - example.com
    server:
      - 192.0.2.200
      - 2001:db8:1::ffbb

```

These settings define a network bridge with the following settings:

- Network interfaces in the bridge: **enp1s0** and **enp7s0**
- Spanning Tree Protocol (STP): Enabled
- Static IPv4 address: **192.0.2.1** with the **/24** subnet mask
- Static IPv6 address: **2001:db8:1::1** with the **/64** subnet mask
- IPv4 default gateway: **192.0.2.254**
- IPv6 default gateway: **2001:db8:1::fffe**
- IPv4 DNS server: **192.0.2.200**
- IPv6 DNS server: **2001:db8:1::ffbb**
- DNS search domain: **example.com**

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-bridge.yml
```

Verification

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE   TYPE   STATE   CONNECTION
bridge0  bridge connected bridge0
```

2. Display all settings of the connection profile:

```
# nmcli connection show bridge0
connection.id:      bridge0_
connection.uuid:    e2cc9206-75a2-4622-89cf-1252926060a9
connection.stable-id: --
connection.type:    bridge
connection.interface-name: bridge0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show bridge0
```

Additional resources

- **nmstatectl(8)** man page on your system
- `/usr/share/doc/nmstate/examples/` directory
- [How to configure a bridge with VLAN information?](#) (Red Hat Knowledgebase)

5.5. CONFIGURING A NETWORK BRIDGE BY USING THE `network` RHEL SYSTEM ROLE

You can connect multiple networks on layer 2 of the Open Systems Interconnection (OSI) model by creating a network bridge. To configure a bridge, create a connection profile in NetworkManager. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure a bridge and, if a connection profile for the bridge's parent device does not exist, the role can create it as well.



NOTE

If you want to assign IP addresses, gateways, and DNS settings to a bridge, configure them on the bridge and not on its ports.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- Two or more physical or virtual network devices are installed on the server.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Bridge connection profile with two Ethernet ports
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # Bridge profile
          - name: bridge0
            type: bridge
            interface_name: bridge0
            ip:
              dhcp4: yes
              auto6: yes
            state: up

          # Port profile for the 1st Ethernet device
          - name: bridge0-port1
            interface_name: enp7s0
            type: ethernet
            controller: bridge0
            port_type: bridge
            state: up

          # Port profile for the 2nd Ethernet device
          - name: bridge0-port2
            interface_name: enp8s0
            type: ethernet
            controller: bridge0
            port_type: bridge
            state: up
```

The settings specified in the example playbook include the following:

type: <profile_type>

Sets the type of the profile to create. The example playbook creates three connection profiles: One for the bridge and two for the Ethernet devices.

dhcp4: yes

Enables automatic IPv4 address assignment from DHCP, PPP, or similar services.

auto6: yes

Enables IPv6 auto-configuration. By default, NetworkManager uses Router Advertisements. If the router announces the **managed** flag, NetworkManager requests an IPv6 address and prefix from a DHCPv6 server.

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the link status of Ethernet devices that are ports of a specific bridge:

```
# ansible managed-node-01.example.com -m command -a 'ip link show master bridge0'
managed-node-01.example.com | CHANGED | rc=0 >>
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
```

2. Display the status of Ethernet devices that are ports of any bridge device:

```
# ansible managed-node-01.example.com -m command -a 'bridge link show'
managed-node-01.example.com | CHANGED | rc=0 >>
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state forwarding priority 32 cost 100
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state listening priority 32 cost 100
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

CHAPTER 6. SETTING UP AN IPSEC VPN

A virtual private network (VPN) is a way of connecting to a local network over the internet. **IPsec** provided by **Libreswan** is the preferred method for creating a VPN. **Libreswan** is a user-space **IPsec** implementation for VPN. A VPN enables the communication between your LAN, and another, remote LAN by setting up a tunnel across an intermediate network such as the internet. For security reasons, a VPN tunnel always uses authentication and encryption. For cryptographic operations, **Libreswan** uses the **NSS** library.

6.1. LIBRESWAN AS AN IPSEC VPN IMPLEMENTATION

In RHEL, you can configure a Virtual Private Network (VPN) by using the IPsec protocol, which is supported by the Libreswan application. Libreswan is a continuation of the Openswan application, and many examples from the Openswan documentation are interchangeable with Libreswan.

The IPsec protocol for a VPN is configured using the Internet Key Exchange (IKE) protocol. The terms IPsec and IKE are used interchangeably. An IPsec VPN is also called an IKE VPN, IKEv2 VPN, XAUTH VPN, Cisco VPN or IKE/IPsec VPN.

Libreswan is an open-source, user-space IKE implementation. IKE v1 and v2 are implemented as a user-level daemon. The IKE protocol is also encrypted. The IPsec protocol is implemented by the Linux kernel, and Libreswan configures the kernel to add and remove VPN tunnel configurations.

The IKE protocol uses UDP port 500 and 4500. The IPsec protocol consists of two protocols:

- Encapsulated Security Payload (ESP), which has protocol number 50.
- Authenticated Header (AH), which has protocol number 51.

The AH protocol is not recommended for use. Users of AH are recommended to migrate to ESP with null encryption.

The IPsec protocol provides two modes of operation:

- Tunnel Mode (the default)
- Transport Mode

You can configure the kernel with IPsec without IKE. This is called *manual keying*. You can also configure manual keying using the **ip xfrm** commands, however, this is strongly discouraged for security reasons. Libreswan communicates with the Linux kernel using the Netlink interface. The kernel performs packet encryption and decryption.

Libreswan uses the Network Security Services (NSS) cryptographic library.



IMPORTANT

IKE/IPsec VPNs, implemented by Libreswan and the Linux kernel, is the only VPN technology recommended for use in RHEL. Do not use any other VPN technology without understanding the risks of doing so.

In RHEL, Libreswan follows **system-wide cryptographic policies** by default. This ensures that Libreswan uses secure settings for current threat models including IKEv2 as a default protocol. See [Using system-wide crypto policies](#) for more information.

Libreswan does not use the terms "source" and "destination" or "server" and "client" because IKE/IPsec are peer to peer protocols. Instead, it uses the terms "left" and "right" to refer to end points (the hosts). This also allows you to use the same configuration on both end points in most cases. However, administrators usually choose to always use "left" for the local host and "right" for the remote host.

The **leftid** and **rightid** options serve as identification of the respective hosts in the authentication process. See the **ipsec.conf(5)** man page for more information.

6.2. AUTHENTICATION METHODS IN LIBRESWAN

Libreswan supports several authentication methods, each of which fits a different scenario.

Pre-Shared key (PSK)

Pre-Shared Key (PSK) is the simplest authentication method. For security reasons, do not use PSKs shorter than 64 random characters. In FIPS mode, PSKs must comply with a minimum-strength requirement depending on the integrity algorithm used. You can set PSK by using the **authby=secret** connection.

Raw RSA keys

Raw RSA keys are commonly used for static host-to-host or subnet-to-subnet IPsec configurations. Each host is manually configured with the public RSA keys of all other hosts, and Libreswan sets up an IPsec tunnel between each pair of hosts. This method does not scale well for large numbers of hosts.

You can generate a raw RSA key on a host using the **ipsec newhostkey** command. You can list generated keys by using the **ipsec showhostkey** command. The **leftrsasigkey=** line is required for connection configurations that use CKA ID keys. Use the **authby=rsasig** connection option for raw RSA keys.

X.509 certificates

X.509 certificates are commonly used for large-scale deployments with hosts that connect to a common IPsec gateway. A central *certificate authority* (CA) signs RSA certificates for hosts or users. This central CA is responsible for relaying trust, including the revocations of individual hosts or users.

For example, you can generate X.509 certificates using the **openssl** command and the NSS **certutil** command. Because Libreswan reads user certificates from the NSS database using the certificates' nickname in the **leftcert=** configuration option, provide a nickname when you create a certificate.

If you use a custom CA certificate, you must import it to the Network Security Services (NSS) database. You can import any certificate in the PKCS #12 format to the Libreswan NSS database by using the **ipsec import** command.



WARNING

Libreswan requires an Internet Key Exchange (IKE) peer ID as a subject alternative name (SAN) for every peer certificate as described in [section 3.1 of RFC 4945](#). Disabling this check by setting the **require-id-on-certificate=no** connection option can make the system vulnerable to man-in-the-middle attacks.

Use the **authby=rsasig** connection option for authentication based on X.509 certificates using RSA

with SHA-2. You can further limit it for ECDSA digital signatures using SHA-2 by setting **authby=ecdsha** and RSA Probabilistic Signature Scheme (RSASSA-PSS) digital signatures based authentication with SHA-2 through **authby=rsa-sha2**. The default value for IKEv2 peer authentication is **authby=rsasig,ecdsha**.

The certificates and the **authby=** signature methods should match. This increases interoperability and preserves authentication in one digital signature system.

NULL authentication

NULL authentication is used to gain mesh encryption without authentication. It protects against passive attacks but not against active attacks. However, because IKEv2 allows asymmetric authentication methods, NULL authentication can also be used for internet-scale opportunistic IPsec. In this model, clients authenticate the server, but servers do not authenticate the client. This model is similar to secure websites using TLS. Use **authby=null** for NULL authentication.

Protection against quantum computers

In addition to the previously mentioned authentication methods, you can use the *Post-quantum Pre-shared Key* (PPK) method to protect against possible attacks by quantum computers. Individual clients or groups of clients can use their own PPK by specifying a PPK ID that corresponds to an out-of-band configured pre-shared key.

Using IKEv1 with pre-shared keys protects against quantum attackers. The redesign of IKEv2 does not offer this protection natively. Libreswan offers the use of a *Post-quantum Pre-shared Key* (PPK) to protect IKEv2 connections against quantum attacks.

To enable optional PPK support, add **ppk=yes** to the connection definition. To require PPK, add **ppk=insist**. Then, each client can be given a PPK ID with a secret value that is communicated out-of-band (and preferably quantum-safe). The PPK's should be very strong in randomness and not based on dictionary words. The PPK ID and PPK data are stored in the **ipsec.secrets** file, for example:

```
@west @east : PPKS "user1" "thestringismeanttobearandomstr"
```

The **PPKS** option refers to static PPKs. This experimental function uses one-time-pad-based Dynamic PPKs. Upon each connection, a new part of the one-time pad is used as the PPK. When used, that part of the dynamic PPK inside the file is overwritten with zeros to prevent re-use. If there is no more one-time-pad material left, the connection fails. See the **ipsec.secrets(5)** man page for more information.

6.3. INSTALLING LIBRESWAN

Before you can set a VPN through the Libreswan IPsec/IKE implementation, you must install the corresponding packages, start the **ipsec** service, and allow the service in your firewall.

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **libreswan** packages:

```
# dnf install libreswan
```

2. If you are re-installing Libreswan, remove its old database files and create a new database:

–

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

3. Start the **ipsec** service, and enable the service to be started automatically on boot:

```
# systemctl enable ipsec --now
```

4. Configure the firewall to allow 500 and 4500/UDP ports for the IKE, ESP, and AH protocols by adding the **ipsec** service:

```
# firewall-cmd --add-service="ipsec"
# firewall-cmd --runtime-to-permanent
```

6.4. CREATING A HOST-TO-HOST VPN

You can configure Libreswan to create a host-to-host IPsec VPN between two hosts referred to as *left* and *right* using authentication by raw RSA keys.

Prerequisites

- Libreswan is installed and the **ipsec** service is started on each node.

Procedure

1. Generate a raw RSA key pair on each host:

```
# ipsec newhostkey
```

2. The previous step returned the generated key's **ckaid**. Use that **ckaid** with the following command on *left*, for example:

```
# ipsec showhostkey --left --ckaid 2d3ea57b61c9419dfd6cf43a1eb6cb306c0e857d
```

The output of the previous command generated the **leftrsasigkey=** line required for the configuration. Do the same on the second host (*right*):

```
# ipsec showhostkey --right --ckaid a9e1f6ce9ecd3608c24e8f701318383f41798f03
```

3. In the **/etc/ipsec.d/** directory, create a new **my_host-to-host.conf** file. Write the RSA host keys from the output of the **ipsec showhostkey** commands in the previous step to the new file. For example:

```
conn mytunnel
  leftid=@west
  left=192.1.2.23
  leftrsasigkey=0sAQOrlo+hOafUZDICQmXFrje/oZm [...] W2n417C/4urYHQkCvulQ==
  rightid=@east
  right=192.1.2.45
  rightrsasigkey=0sAQO3fwC6nSSGgt64DWiYZzuHbc4 [...] D/v8t5YTQ==
  authby=rsasig
```

4. After importing keys, restart the **ipsec** service:

```
# systemctl restart ipsec
```

5. Load the connection:

```
# ipsec add mytunnel
```

6. Establish the tunnel:

```
# ipsec up mytunnel
```

7. To automatically start the tunnel when the **ipsec** service is started, add the following line to the connection definition:

```
auto=start
```

6.5. CONFIGURING A SITE-TO-SITE VPN

To create a site-to-site IPsec VPN, by joining two networks, an IPsec tunnel between the two hosts is created. The hosts thus act as the end points, which are configured to permit traffic from one or more subnets to pass through. Therefore you can think of the host as gateways to the remote portion of the network.

The configuration of the site-to-site VPN only differs from the host-to-host VPN in that one or more networks or subnets must be specified in the configuration file.

Prerequisites

- A [host-to-host VPN](#) is already configured.

Procedure

1. Copy the file with the configuration of your host-to-host VPN to a new file, for example:

```
# cp /etc/ipsec.d/my_host-to-host.conf /etc/ipsec.d/my_site-to-site.conf
```

2. Add the subnet configuration to the file created in the previous step, for example:

```
conn mysubnet
    also=mytunnel
    leftsubnet=192.0.1.0/24
    rightsubnet=192.0.2.0/24
    auto=start

conn mysubnet6
    also=mytunnel
    leftsubnet=2001:db8:0:1::/64
    rightsubnet=2001:db8:0:2::/64
    auto=start
```

```
# the following part of the configuration file is the same for both host-to-host and site-to-site
connections:
```

```

conn mytunnel
  leftid=@west
  left=192.1.2.23
  lefttrsasigkey=0sAQOrlo+hOafUZDICQmXFrje/oZm [...] W2n417C/4urYHQkCvulQ==
  rightid=@east
  right=192.1.2.45
  righttrsasigkey=0sAQO3fwC6nSSGgt64DWiYZzuHbc4 [...] D/v8t5YTQ==
  authby=rsasig

```

6.6. CONFIGURING A REMOTE ACCESS VPN

Road warriors are traveling users with mobile clients and a dynamically assigned IP address. The mobile clients authenticate using X.509 certificates.

The following example shows configuration for **IKEv2**, and it avoids using the **IKEv1** XAUTH protocol.

On the server:

```

conn roadwarriors
  keyexchange=ikev2
  # support (roaming) MOBIKE clients (RFC 4555)
  mobike=yes
  fragmentation=yes
  left=1.2.3.4
  # if access to the LAN is given, enable this, otherwise use 0.0.0.0/0
  # leftsubnet=10.10.0.0/16
  leftsubnet=0.0.0.0/0
  leftcert=gw.example.com
  leftid=%fromcert
  leftxauthserver=yes
  leftmodecfgserver=yes
  right=%any
  # trust our own Certificate Agency
  rightca=%same
  # pick an IP address pool to assign to remote users
  # 100.64.0.0/16 prevents RFC1918 clashes when remote users are behind NAT
  rightaddresspool=100.64.13.100-100.64.13.254
  # if you want remote clients to use some local DNS zones and servers
  modecfgdns="1.2.3.4, 5.6.7.8"
  modecfgdomains="internal.company.com, corp"
  rightxauthclient=yes
  rightmodecfgclient=yes
  authby=rsasig
  # optionally, run the client X.509 ID through pam to allow or deny client
  # pam-authorize=yes
  # load connection, do not initiate
  auto=add
  # kill vanished roadwarriors

```

On the mobile client, the road warrior's device, use a slight variation of the previous configuration:

```

conn to-vpn-server
  keyexchange=ikev2
  # pick up our dynamic IP

```

```

left=%defaultroute
leftsubnet=0.0.0.0/0
leftcert=myname.example.com
leftid=%fromcert
leftmodecfgclient=yes
# right can also be a DNS hostname
right=1.2.3.4
# if access to the remote LAN is required, enable this, otherwise use 0.0.0.0/0
# rightsubnet=10.10.0.0/16
rightsubnet=0.0.0.0/0
fragmentation=yes
# trust our own Certificate Agency
rightca=%same
authby=rsasig
# allow narrowing to the server's suggested assigned IP and remote subnet
narrowing=yes
# support (roaming) MOBIKE clients (RFC 4555)
mobike=yes
# initiate connection
auto=start

```

6.7. CONFIGURING A MESH VPN

A mesh VPN network, which is also known as an *any-to-any* VPN, is a network where all nodes communicate using IPsec. The configuration allows for exceptions for nodes that cannot use IPsec. The mesh VPN network can be configured in two ways:

- To require IPsec.
- To prefer IPsec but allow a fallback to clear-text communication.

Authentication between the nodes can be based on X.509 certificates or on DNS Security Extensions (DNSSEC).

You can use any regular IKEv2 authentication method for *opportunistic IPsec*, because these connections are regular Libreswan configurations, except for the opportunistic IPsec that is defined by **right=%opportunisticgroup** entry. A common authentication method is for hosts to authenticate each other based on X.509 certificates using a commonly shared certification authority (CA). Cloud deployments typically issue certificates for each node in the cloud as part of the standard procedure.



IMPORTANT

Do not use PreSharedKey (PSK) authentication because one compromised host would result in the group PSK secret being compromised as well.

You can use NULL authentication to deploy encryption between nodes without authentication, which protects only against passive attackers.

The following procedure uses X.509 certificates. You can generate these certificates by using any kind of CA management system, such as the Dogtag Certificate System. Dogtag assumes that the certificates for each node are available in the PKCS #12 format (**.p12** files), which contain the private key, the node certificate, and the Root CA certificate used to validate other nodes' X.509 certificates.

Each node has an identical configuration with the exception of its X.509 certificate. This allows for adding new nodes without reconfiguring any of the existing nodes in the network. The PKCS #12 files

require a "friendly name", for which we use the name "node" so that the configuration files referencing the friendly name can be identical for all nodes.

Prerequisites

- Libreswan is installed, and the **ipsec** service is started on each node.
- A new NSS database is initialized.
 1. If you already have an old NSS database, remove the old database files:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
```

2. You can initialize a new database with the following command:

```
# ipsec initnss
```

Procedure

1. On each node, import PKCS #12 files. This step requires the password used to generate the PKCS #12 files:

```
# ipsec import nodeXXX.p12
```

2. Create the following three connection definitions for the **IPsec required** (private), **IPsec optional** (private-or-clear), and **No IPsec** (clear) profiles:

```
# cat /etc/ipsec.d/mesh.conf
conn clear
  auto=ondemand 1
  type=passthrough
  authby=never
  left=%defaultroute
  right=%group

conn private
  auto=ondemand
  type=transport
  authby=rsasig
  failurehunt=drop
  negotiationshunt=drop
  keyexchange=ikev2
  left=%defaultroute
  leftcert=nodeXXXX
  leftid=%fromcert 2
  rightid=%fromcert
  right=%opportunisticgroup

conn private-or-clear
  auto=ondemand
  type=transport
  authby=rsasig
  failurehunt=passthrough
```

```

negotiationshunt=passthrough
# left
left=%defaultroute
leftcert=nodeXXXX 3
leftid=%fromcert
leftrsasigkey=%cert
# right
rightrsasigkey=%cert
rightid=%fromcert
right=%opportunisticgroup

```

- 1 The **auto** variable has several options:

You can use the **ondemand** connection option with opportunistic IPsec to initiate the IPsec connection, or for explicitly configured connections that do not need to be active all the time. This option sets up a trap XFRM policy in the kernel, enabling the IPsec connection to begin when it receives the first packet that matches that policy.

You can effectively configure and manage your IPsec connections, whether you use Opportunistic IPsec or explicitly configured connections, by using the following options:

The **add** option

Loads the connection configuration and prepares it for responding to remote initiations. However, the connection is not automatically initiated from the local side. You can manually start the IPsec connection by using the command **ipsec up**.

The **start** option

Loads the connection configuration and prepares it for responding to remote initiations. Additionally, it immediately initiates a connection to the remote peer. You can use this option for permanent and always active connections.

- 2 The **leftid** and **rightid** variables identify the right and the left channel of the IPsec tunnel connection. You can use these variables to obtain the value of the local IP address or the subject DN of the local certificate, if you have configured one.

- 3 The **leftcert** variable defines the nickname of the NSS database that you want to use.

3. Add the IP address of the network to the corresponding category. For example, if all nodes reside in the **10.15.0.0/16** network, and all nodes must use IPsec encryption:

```
# echo "10.15.0.0/16" >> /etc/ipsec.d/policies/private
```

4. To allow certain nodes, for example, **10.15.34.0/24**, to work with and without IPsec, add those nodes to the private-or-clear group:

```
# echo "10.15.34.0/24" >> /etc/ipsec.d/policies/private-or-clear
```

5. To define a host, for example, **10.15.1.2**, which is not capable of IPsec into the clear group, use:

```
# echo "10.15.1.2/32" >> /etc/ipsec.d/policies/clear
```

You can create the files in the **/etc/ipsec.d/policies** directory from a template for each new node, or you can provision them by using Puppet or Ansible.

Note that every node has the same list of exceptions or different traffic flow expectations. Two nodes, therefore, might not be able to communicate because one requires IPsec and the other cannot use IPsec.

- Restart the node to add it to the configured mesh:

```
# systemctl restart ipsec
```

Verification

- Open an IPsec tunnel by using the **ping** command:

```
# ping <nodeYYY>
```

- Display the NSS database with the imported certification:

```
# certutil -L -d sql:/var/lib/ipsec/nss/
```

Certificate Nickname	Trust Attributes
	SSL,S/MIME,JAR/XPI
west	u,u,u
ca	CT,,

- See which tunnels are open on the node:

```
# ipsec trafficstatus
006 #2: "private#10.15.0.0/16"[1] ...<nodeYYY>, type=ESP, add_time=1691399301,
inBytes=512, outBytes=512, maxBytes=2^63B, id='C=US, ST=NC, O=Example
Organization, CN=east'
```

Additional resources

- ipsec.conf(5)** man page on your system.
- For more information about the **authby** variable, see [6.2. Authentication methods in Libreswan](#).

6.8. DEPLOYING A FIPS-COMPLIANT IPSEC VPN

You can deploy a FIPS-compliant IPsec VPN solution with Libreswan. To do so, you can identify which cryptographic algorithms are available and which are disabled for Libreswan in FIPS mode.

Prerequisites

- The **AppStream** repository is enabled.
- Your system has been installed in FIPS mode

Procedure

- Install the **libreswan** packages:

```
# dnf install libreswan
```

2. If you are re-installing Libreswan, remove its old NSS database:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
```

3. Start the **ipsec** service, and enable the service to be started automatically on boot:

```
# systemctl enable ipsec --now
```

4. Configure the firewall to allow **500** and **4500** UDP ports for the IKE, ESP, and AH protocols by adding the **ipsec** service:

```
# firewall-cmd --add-service="ipsec"
# firewall-cmd --runtime-to-permanent
```

Verification

1. Confirm Libreswan is running in FIPS mode:

```
# ipsec whack --fipsstatus
FIPS mode enabled
```

2. Alternatively, check entries for the **ipsec** unit in the **systemd** journal:

```
$ journalctl -u ipsec
...
Jan 22 11:26:50 localhost.localdomain pluto[3076]: FIPS Mode: ON
```

3. To see the available algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | head -6
Initializing NSS using read-only database "sql:/var/lib/ipsec/nss"
FIPS Mode: ON
NSS crypto library initialized
FIPS mode enabled for pluto daemon
NSS library is running in FIPS mode
FIPS HMAC integrity support [not required]
```

4. To query disabled algorithms in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | grep disabled
Encryption algorithm CAMELLIA_CTR disabled; not FIPS compliant
Encryption algorithm CAMELLIA_CBC disabled; not FIPS compliant
Encryption algorithm NULL disabled; not FIPS compliant
Encryption algorithm CHACHA20_POLY1305 disabled; not FIPS compliant
Hash algorithm MD5 disabled; not FIPS compliant
PRF algorithm HMAC_MD5 disabled; not FIPS compliant
PRF algorithm AES_XCBC disabled; not FIPS compliant
Integrity algorithm HMAC_MD5_96 disabled; not FIPS compliant
Integrity algorithm HMAC_SHA2_256_TRUNCBUG disabled; not FIPS compliant
Integrity algorithm AES_XCBC_96 disabled; not FIPS compliant
DH algorithm MODP1536 disabled; not FIPS compliant
DH algorithm DH31 disabled; not FIPS compliant
```

- To list all allowed algorithms and ciphers in FIPS mode:

```
# ipsec pluto --selftest 2>&1 | grep ESP | grep FIPS | sed "s/^.*FIPS//"
aes_ccm, aes_ccm_c
aes_ccm_b
aes_ccm_a
NSS(CBC) 3des
NSS(GCM) aes_gcm, aes_gcm_c
NSS(GCM) aes_gcm_b
NSS(GCM) aes_gcm_a
NSS(CTR) aesctr
NSS(CBC) aes
aes_gmac
NSS    sha, sha1, sha1_96, hmac_sha1
NSS    sha512, sha2_512, sha2_512_256, hmac_sha2_512
NSS    sha384, sha2_384, sha2_384_192, hmac_sha2_384
NSS    sha2, sha256, sha2_256, sha2_256_128, hmac_sha2_256
aes_cmac
null
NSS(MODP) null, dh0
NSS(MODP) dh14
NSS(MODP) dh15
NSS(MODP) dh16
NSS(MODP) dh17
NSS(MODP) dh18
NSS(ECP) ecp_256, ecp256
NSS(ECP) ecp_384, ecp384
NSS(ECP) ecp_521, ecp521
```

Additional resources

- [Using system-wide cryptographic policies](#).

6.9. PROTECTING THE IPSEC NSS DATABASE BY A PASSWORD

By default, the IPsec service creates its Network Security Services (NSS) database with an empty password during the first start. To enhance security, you can add password protection.

Prerequisites

- The `/var/lib/ipsec/nss/` directory contains NSS database files.

Procedure

- Enable password protection for the **NSS** database for Libreswan:

```
# certutil -N -d sql:/var/lib/ipsec/nss
Enter Password or Pin for "NSS Certificate DB":
Enter a password which will be used to encrypt your keys.
The password should be at least 8 characters long,
and should contain at least one non-alphabetic character.

Enter new password:
```

2. Create the **/etc/ipsec.d/nsspassword** file that contains the password you have set in the previous step, for example:

```
# cat /etc/ipsec.d/nsspassword
NSS Certificate DB:<password>
```

The **nsspassword** file use the following syntax:

```
<token_1>:<password1>
<token_2>:<password2>
```

The default NSS software token is **NSS Certificate DB**. If your system is running in FIPS mode, the name of the token is **NSS FIPS 140-2 Certificate DB**.

3. Depending on your scenario, either start or restart the **ipsec** service after you finish the **nsspassword** file:

```
# systemctl restart ipsec
```

Verification

1. Check that the **ipsec** service is running after you have added a non-empty password to its NSS database:

```
# systemctl status ipsec
● ipsec.service - Internet Key Exchange (IKE) Protocol Daemon for IPsec
   Loaded: loaded (/usr/lib/systemd/system/ipsec.service; enabled; vendor preset: disable>
   Active: active (running)...
```

Verification

- Check that the **Journal** log contains entries that confirm a successful initialization:

```
# journalctl -u ipsec
...
pluto[6214]: Initializing NSS using read-only database "sql:/var/lib/ipsec/nss"
pluto[6214]: NSS Password from file "/etc/ipsec.d/nsspassword" for token "NSS Certificate
DB" with length 20 passed to NSS
pluto[6214]: NSS crypto library initialized
...
```

Additional resources

- **certutil(1)** man page on your system
- [FIPS - Federal Information Processing Standards](#) section on the [Product compliance](#) Red Hat Customer Portal page

6.10. CONFIGURING AN IPSEC VPN TO USE TCP

Libreswan supports TCP encapsulation of IKE and IPsec packets as described in RFC 8229. With this feature, you can establish IPsec VPNs on networks that prevent traffic transmitted via UDP and Encapsulating Security Payload (ESP). You can configure VPN servers and clients to use TCP either as a

fallback or as the main VPN transport protocol. Because TCP encapsulation has bigger performance costs, use TCP as the main VPN protocol only if UDP is permanently blocked in your scenario.

Prerequisites

- A [remote-access VPN](#) is already configured.

Procedure

1. Add the following option to the `/etc/ipsec.conf` file in the **config setup** section:

```
listen-tcp=yes
```

2. To use TCP encapsulation as a fallback option when the first attempt over UDP fails, add the following two options to the client's connection definition:

```
enable-tcp=fallback
tcp-remoteport=4500
```

Alternatively, if you know that UDP is permanently blocked, use the following options in the client's connection configuration:

```
enable-tcp=yes
tcp-remoteport=4500
```

Additional resources

- [IETF RFC 8229: TCP Encapsulation of IKE and IPsec Packets](#)

6.11. CONFIGURING VPN CONNECTIONS WITH IPSEC BY USING THE RHEL SYSTEM ROLE

With the **vpn** system role, you can configure VPN connections on RHEL systems by using Red Hat Ansible Automation Platform. You can use it to set up host-to-host, network-to-network, VPN Remote Access Server, and mesh configurations.

For host-to-host connections, the role sets up a VPN tunnel between each pair of hosts in the list of **vpn_connections** using the default parameters, including generating keys as needed. Alternatively, you can configure it to create an opportunistic mesh configuration between all hosts listed. The role assumes that the names of the hosts under **hosts** are the same as the names of the hosts used in the Ansible inventory, and that you can use those names to configure the tunnels.



NOTE

The **vpn** RHEL system role currently supports only Libreswan, which is an IPsec implementation, as the VPN provider.

6.11.1. Creating a host-to-host VPN with IPsec by using the vpn RHEL system role

You can use the **vpn** system role to configure host-to-host connections by running an Ansible playbook on the control node, which configures all managed nodes listed in an inventory file.

Prerequisites

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
- name: Host to host VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - hosts:
          managed-node-01.example.com:
          managed-node-02.example.com:
        vpn_manage_firewall: true
        vpn_manage_selinux: true
```

This playbook configures the connection **managed-node-01.example.com-to-managed-node-02.example.com** by using pre-shared key authentication with keys auto-generated by the system role. Because **vpn_manage_firewall** and **vpn_manage_selinux** are both set to **true**, the **vpn** role uses the **firewall** and **selinux** roles to manage the ports used by the **vpn** role.

To configure connections from managed hosts to external hosts that are not listed in the inventory file, add the following section to the **vpn_connections** list of hosts:

```
vpn_connections:
  - hosts:
      managed-node-01.example.com:
      <external_node>:
        hostname: <IP_address_or_hostname>
```

This configures one additional connection: **managed-node-01.example.com-to-<external_node>**



NOTE

The connections are configured only on the managed nodes and not on the external node.

2. Optional: You can specify multiple VPN connections for the managed nodes by using additional sections within **vpn_connections**, for example, a control plane and a data plane:

```
- name: Multiple VPN
  hosts: managed-node-01.example.com, managed-node-02.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
```

```

- name: control_plane_vpn
  hosts:
    managed-node-01.example.com:
      hostname: 192.0.2.0 # IP for the control plane
    managed-node-02.example.com:
      hostname: 192.0.2.1
- name: data_plane_vpn
  hosts:
    managed-node-01.example.com:
      hostname: 10.0.0.1 # IP for the data plane
    managed-node-02.example.com:
      hostname: 10.0.0.2

```

3. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

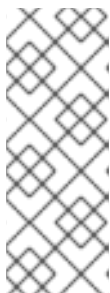
```
$ ansible-playbook ~/playbook.yml
```

Verification

1. On the managed nodes, confirm that the connection is successfully loaded:

```
# ipsec status | grep <connection_name>
```

Replace **<connection_name>** with the name of the connection from this node, for example **managed_node1-to-managed_node2**.



NOTE

By default, the role generates a descriptive name for each connection it creates from the perspective of each system. For example, when creating a connection between **managed_node1** and **managed_node2**, the descriptive name of this connection on **managed_node1** is **managed_node1-to-managed_node2** but on **managed_node2** the connection is named **managed_node2-to-managed_node1**.

2. On the managed nodes, confirm that the connection is successfully started:

```
# ipsec trafficstatus | grep <connection_name>
```

3. Optional: If a connection does not successfully load, manually add the connection by entering the following command. This provides more specific information indicating why the connection failed to establish:

```
# ipsec add <connection_name>
```



NOTE

Any errors that may occur during the process of loading and starting the connection are reported in the `/var/log/pluto.log` file. Because these logs are hard to parse, manually add the connection to obtain log messages from the standard output instead.

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.vpn/README.md` file
- `/usr/share/doc/rhel-system-roles/vpn/` directory

6.11.2. Creating an opportunistic mesh VPN connection with IPsec by using the `vpn` RHEL system role

You can use the `vpn` system role to configure an opportunistic mesh VPN connection that uses certificates for authentication by running an Ansible playbook on the control node, which will configure all the managed nodes listed in an inventory file.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The IPsec Network Security Services (NSS) crypto library in the `/var/lib/ipsec/nss/` directory contains the necessary certificates.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
- name: _Mesh VPN_
  hosts: managed-node-01.example.com, managed-node-02.example.com, managed-node-03.example.com
  roles:
    - rhel-system-roles.vpn
  vars:
    vpn_connections:
      - opportunistic: true
        auth_method: cert
        policies:
          - policy: private
            cidr: default
          - policy: private-or-clear
            cidr: 198.51.100.0/24
          - policy: private
            cidr: 192.0.2.0/24
          - policy: clear
            cidr: 192.0.2.7/32
    vpn_manage_firewall: true
    vpn_manage_selinux: true
```


Authentication with certificates is configured by defining the **auth_method: cert** parameter in the playbook. By default, the node name is used as the certificate nickname. In this example, this is **managed-node-01.example.com**. You can define different certificate names by using the **cert_name** attribute in your inventory.

In this example procedure, the control node, which is the system from which you will run the Ansible playbook, shares the same classless inter-domain routing (CIDR) number as both of the managed nodes (192.0.2.0/24) and has the IP address 192.0.2.7. Therefore, the control node falls under the private policy which is automatically created for CIDR 192.0.2.0/24.

To prevent SSH connection loss during the play, a clear policy for the control node is included in the list of policies. Note that there is also an item in the policies list where the CIDR is equal to default. This is because this playbook overrides the rule from the default policy to make it private instead of private-or-clear.

Because **vpn_manage_firewall** and **vpn_manage_selinux** are both set to **true**, the **vpn** role uses the **firewall** and **selinux** roles to manage the ports used by the **vpn** role.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.vpn/README.md** file
- **/usr/share/doc/rhel-system-roles/vpn/** directory

6.12. CONFIGURING IPSEC CONNECTIONS THAT OPT OUT OF THE SYSTEM-WIDE CRYPTO POLICIES

Overriding system-wide crypto-policies for a connection

The RHEL system-wide cryptographic policies create a special connection called **%default**. This connection contains the default values for the **ikev2**, **esp**, and **ike** options. However, you can override the default values by specifying the mentioned option in the connection configuration file.

For example, the following configuration allows connections that use IKEv1 with AES and SHA-1 or SHA-2, and IPsec (ESP) with either AES-GCM or AES-CBC:

```
conn MyExample
...
keyexchange=ikev1
ike=aes-sha2,aes-sha1;modp2048
esp=aes_gcm,aes-sha2,aes-sha1
...
```

Note that AES-GCM is available for IPsec (ESP) and for IKEv2, but not for IKEv1.

Disabling system-wide crypto policies for all connections

To disable system-wide crypto policies for all IPsec connections, comment out the following line in the `/etc/ipsec.conf` file:

```
include /etc/crypto-policies/back-ends/libreswan.config
```

Then add the **keyexchange=ikev1** option to your connection configuration file.

Additional resources

- [Using system-wide cryptographic policies](#).

6.13. TROUBLESHOOTING IPSEC VPN CONFIGURATIONS

Problems related to IPsec VPN configurations most commonly occur due to several main reasons. If you are encountering such problems, you can check if the cause of the problem corresponds to any of the following scenarios, and apply the corresponding solution.

Basic connection troubleshooting

Most problems with VPN connections occur in new deployments, where administrators configured endpoints with mismatched configuration options. Also, a working configuration can suddenly stop working, often due to newly introduced incompatible values. This could be the result of an administrator changing the configuration. Alternatively, an administrator may have installed a firmware update or a package update with different default values for certain options, such as encryption algorithms.

To confirm that an IPsec VPN connection is established:

```
# ipsec trafficstatus
006 #8: "vpn.example.com"[1] 192.0.2.1, type=ESP, add_time=1595296930, inBytes=5999,
outBytes=3231, id='@vpn.example.com', lease=100.64.13.5/32
```

If the output is empty or does not show an entry with the connection name, the tunnel is broken.

To check that the problem is in the connection:

1. Reload the `vpn.example.com` connection:

```
# ipsec add vpn.example.com
002 added connection description "vpn.example.com"
```

2. Next, initiate the VPN connection:

```
# ipsec up vpn.example.com
```

Firewall-related problems

The most common problem is that a firewall on one of the IPsec endpoints or on a router between the endpoints is dropping all Internet Key Exchange (IKE) packets.

- For IKEv2, an output similar to the following example indicates a problem with a firewall:

```
# ipsec up vpn.example.com
```

```
181 "vpn.example.com"[1] 192.0.2.2 #15: initiating IKEv2 IKE SA
181 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: sent v2I1, expected v2R1
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 0.5
seconds for response
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 1
seconds for response
010 "vpn.example.com"[1] 192.0.2.2 #15: STATE_PARENT_I1: retransmission; will wait 2
seconds for
...
```

- For IKEv1, the output of the initiating command looks like:

```
# ipsec up vpn.example.com
```

```
002 "vpn.example.com" #9: initiating Main Mode
102 "vpn.example.com" #9: STATE_MAIN_I1: sent MI1, expecting MR1
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 0.5 seconds for
response
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 1 seconds for
response
010 "vpn.example.com" #9: STATE_MAIN_I1: retransmission; will wait 2 seconds for
response
...
```

Because the IKE protocol, which is used to set up IPsec, is encrypted, you can troubleshoot only a limited subset of problems using the **tcpdump** tool. If a firewall is dropping IKE or IPsec packets, you can try to find the cause using the **tcpdump** utility. However, **tcpdump** cannot diagnose other problems with IPsec VPN connections.

- To capture the negotiation of the VPN and all encrypted data on the **eth0** interface:

```
# tcpdump -i eth0 -n -n esp or udp port 500 or udp port 4500 or tcp port 4500
```

Mismatched algorithms, protocols, and policies

VPN connections require that the endpoints have matching IKE algorithms, IPsec algorithms, and IP address ranges. If a mismatch occurs, the connection fails. If you identify a mismatch by using one of the following methods, fix it by aligning algorithms, protocols, or policies.

- If the remote endpoint is not running IKE/IPsec, you can see an ICMP packet indicating it. For example:

```
# ipsec up vpn.example.com
```

```
...
000 "vpn.example.com"[1] 192.0.2.2 #16: ERROR: asynchronous network error report on
wlp2s0 (192.0.2.2:500), complainant 198.51.100.1: Connection refused [errno 111, origin
ICMP type 3 code 3 (not authenticated)]
...
```

- Example of mismatched IKE algorithms:

```
# ipsec up vpn.example.com
```

```
...
003 "vpn.example.com"[1] 193.110.157.148 #3: dropping unexpected IKE_SA_INIT message
```

containing NO_PROPOSAL_CHOSEN notification; message payloads: N; missing payloads: SA,KE,Ni

- Example of mismatched IPsec algorithms:

```
# ipsec up vpn.example.com
...
182 "vpn.example.com"[1] 193.110.157.148 #5: STATE_PARENT_I2: sent v2I2, expected
v2R2 {auth=IKEv2 cipher=AES_GCM_16_256 integ=n/a prf=HMAC_SHA2_256
group=MODP2048}
002 "vpn.example.com"[1] 193.110.157.148 #6: IKE_AUTH response contained the error
notification NO_PROPOSAL_CHOSEN
```

A mismatched IKE version could also result in the remote endpoint dropping the request without a response. This looks identical to a firewall dropping all IKE packets.

- Example of mismatched IP address ranges for IKEv2 (called Traffic Selectors - TS):

```
# ipsec up vpn.example.com
...
1v2 "vpn.example.com" #1: STATE_PARENT_I2: sent v2I2, expected v2R2 {auth=IKEv2
cipher=AES_GCM_16_256 integ=n/a prf=HMAC_SHA2_512 group=MODP2048}
002 "vpn.example.com" #2: IKE_AUTH response contained the error notification
TS_UNACCEPTABLE
```

- Example of mismatched IP address ranges for IKEv1:

```
# ipsec up vpn.example.com
...
031 "vpn.example.com" #2: STATE_QUICK_I1: 60 second timeout exceeded after 0
retransmits. No acceptable response to our first Quick Mode message: perhaps peer likes
no proposal
```

- When using PreSharedKeys (PSK) in IKEv1, if both sides do not put in the same PSK, the entire IKE message becomes unreadable:

```
# ipsec up vpn.example.com
...
003 "vpn.example.com" #1: received Hash Payload does not match computed value
223 "vpn.example.com" #1: sending notification INVALID_HASH_INFORMATION to
192.0.2.23:500
```

- In IKEv2, the mismatched-PSK error results in an AUTHENTICATION_FAILED message:

```
# ipsec up vpn.example.com
...
002 "vpn.example.com" #1: IKE SA authentication request rejected by peer:
AUTHENTICATION_FAILED
```

Maximum transmission unit

Other than firewalls blocking IKE or IPsec packets, the most common cause of networking problems relates to an increased packet size of encrypted packets. Network hardware fragments packets larger than the maximum transmission unit (MTU), for example, 1500 bytes. Often, the fragments are lost and

the packets fail to re-assemble. This leads to intermittent failures, when a ping test, which uses small-sized packets, works but other traffic fails. In this case, you can establish an SSH session but the terminal freezes as soon as you use it, for example, by entering the 'ls -al /usr' command on the remote host.

To work around the problem, reduce MTU size by adding the **mtu=1400** option to the tunnel configuration file.

Alternatively, for TCP connections, enable an iptables rule that changes the MSS value:

```
# iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
```

If the previous command does not solve the problem in your scenario, directly specify a lower size in the **set-mss** parameter:

```
# iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1380
```

Network address translation (NAT)

When an IPsec host also serves as a NAT router, it could accidentally remap packets. The following example configuration demonstrates the problem:

```
conn myvpn
  left=172.16.0.1
  leftsubnet=10.0.2.0/24
  right=172.16.0.2
  rightsubnet=192.168.0.0/16
  ...
```

The system with address 172.16.0.1 have a NAT rule:

```
iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE
```

If the system on address 10.0.2.33 sends a packet to 192.168.0.1, then the router translates the source 10.0.2.33 to 172.16.0.1 before it applies the IPsec encryption.

Then, the packet with the source address 10.0.2.33 no longer matches the **conn myvpn** configuration, and IPsec does not encrypt this packet.

To solve this problem, insert rules that exclude NAT for target IPsec subnet ranges on the router, in this example:

```
iptables -t nat -I POSTROUTING -s 10.0.2.0/24 -d 192.168.0.0/16 -j RETURN
```

Kernel IPsec subsystem bugs

The kernel IPsec subsystem might fail, for example, when a bug causes a desynchronizing of the IKE user space and the IPsec kernel. To check for such problems:

```
$ cat /proc/net/xfrm_stat
XfrmInError          0
XfrmInBufferError    0
...
```

Any non-zero value in the output of the previous command indicates a problem. If you encounter this problem, open a new [support case](#), and attach the output of the previous command along with the corresponding IKE logs.

Libreswan logs

Libreswan logs using the **syslog** protocol by default. You can use the **journalctl** command to find log entries related to IPsec. Because the corresponding entries to the log are sent by the **pluto** IKE daemon, search for the "pluto" keyword, for example:

```
$ journalctl -b | grep pluto
```

To show a live log for the **ipsec** service:

```
$ journalctl -f -u ipsec
```

If the default level of logging does not reveal your configuration problem, enable debug logs by adding the **plutodebug=all** option to the **config setup** section in the **/etc/ipsec.conf** file.

Note that debug logging produces a lot of entries, and it is possible that either the **journald** or **syslogd** service rate-limits the **syslog** messages. To ensure you have complete logs, redirect the logging to a file. Edit the **/etc/ipsec.conf**, and add the **logfile=/var/log/pluto.log** in the **config setup** section.

Additional resources

- [Troubleshooting problems by using log files](#)
- **tcpdump(8)** and **ipsec.conf(5)** man pages on your system
- [Using and configuring firewalld](#)

6.14. CONFIGURING A VPN CONNECTION WITH CONTROL-CENTER

If you use Red Hat Enterprise Linux with a graphical interface, you can configure a VPN connection in the GNOME **control-center**.

Prerequisites

- The **NetworkManager-libreswan-gnome** package is installed.

Procedure

1. Press the **Super** key, type **Settings**, and press **Enter** to open the **control-center** application.
2. Select the **Network** entry on the left.
3. Click the **+** icon.
4. Select **VPN**.
5. Select the **Identity** menu entry to see the basic configuration options:
General

Gateway – The name or **IP** address of the remote VPN gateway.

Authentication

Type

- **IKEv2 (Certificate)**– client is authenticated by certificate. It is more secure (default).
- **IKEv1 (XAUTH)** – client is authenticated by user name and password, or a pre-shared key (PSK).

The following configuration settings are available under the **Advanced** section:

Figure 6.1. Advanced options of a VPN connection

IPsec Advanced Options [X]

Identification

Domain:

Security

Phase1 Algorithms:

Phase2 Algorithms:

☐ Disable PFS

Phase1 Lifetime:

Phase2 Lifetime:

☐ Disable rekeying

Connectivity

Remote Network:

☐ narrowing

Enable fragmentation

Enable MOBIKE

Apply

**WARNING**

When configuring an IPsec-based VPN connection using the **gnome-control-center** application, the **Advanced** dialog displays the configuration, but it does not allow any changes. As a consequence, users cannot change any advanced IPsec options. Use the **nm-connection-editor** or **nmcli** tools instead to perform configuration of the advanced properties.

Identification

- **Domain** – If required, enter the Domain Name.

Security

- **Phase1 Algorithms** – corresponds to the **ike** Libreswan parameter – enter the algorithms to be used to authenticate and set up an encrypted channel.
- **Phase2 Algorithms** – corresponds to the **esp** Libreswan parameter – enter the algorithms to be used for the **IPsec** negotiations.
Check the **Disable PFS** field to turn off Perfect Forward Secrecy (PFS) to ensure compatibility with old servers that do not support PFS.
- **Phase1 Lifetime** – corresponds to the **ikelifetime** Libreswan parameter – how long the key used to encrypt the traffic will be valid.
- **Phase2 Lifetime** – corresponds to the **salifetime** Libreswan parameter – how long a particular instance of a connection should last before expiring.
Note that the encryption key should be changed from time to time for security reasons.
- **Remote network** – corresponds to the **rightsubnet** Libreswan parameter – the destination private remote network that should be reached through the VPN.
Check the **narrowing** field to enable narrowing. Note that it is only effective in IKEv2 negotiation.
- **Enable fragmentation** – corresponds to the **fragmentation** Libreswan parameter – whether or not to allow IKE fragmentation. Valid values are **yes** (default) or **no**.
- **Enable Mobike** – corresponds to the **mobike** Libreswan parameter – whether or not to allow Mobility and Multihoming Protocol (MOBIKE, RFC 4555) to enable a connection to migrate its endpoint without needing to restart the connection from scratch. This is used on mobile devices that switch between wired, wireless, or mobile data connections. The values are **no** (default) or **yes**.

6. Select the **IPv4** menu entry:

IPv4 Method

- **Automatic (DHCP)** – Choose this option if the network you are connecting to uses a **DHCP** server to assign dynamic **IP** addresses.
- **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign **IP** addresses manually. Random addresses will be assigned as per [RFC 3927](#) with prefix **169.254/16**.

- **Manual** – Choose this option if you want to assign **IP** addresses manually.
- **Disable** – **IPv4** is disabled for this connection.
DNS

In the **DNS** section, when **Automatic** is **ON**, switch it to **OFF** to enter the IP address of a DNS server you want to use separating the IPs by comma.

Routes

Note that in the **Routes** section, when **Automatic** is **ON**, routes from DHCP are used, but you can also add additional static routes. When **OFF**, only static routes are used.

- **Address** – Enter the **IP** address of a remote network or host.
- **Netmask** – The netmask or prefix length of the **IP** address entered above.
- **Gateway** – The **IP** address of the gateway leading to the remote network or host entered above.
- **Metric** – A network cost, a preference value to give to this route. Lower values will be preferred over higher values.

Use this connection only for resources on its network

Select this check box to prevent the connection from becoming the default route. Selecting this option means that only traffic specifically destined for routes learned automatically over the connection or entered here manually is routed over the connection.

7. To configure **IPv6** settings in a **VPN** connection, select the **IPv6** menu entry:

IPv6 Method

- **Automatic** – Choose this option to use **IPv6** Stateless Address AutoConfiguration (SLAAC) to create an automatic, stateless configuration based on the hardware address and Router Advertisements (RA).
- **Automatic, DHCP only** – Choose this option to not use RA, but request information from **DHCPv6** directly to create a stateful configuration.
- **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign **IP** addresses manually. Random addresses will be assigned as per [RFC 4862](#) with prefix **FE80::0**.
- **Manual** – Choose this option if you want to assign **IP** addresses manually.
- **Disable** – **IPv6** is disabled for this connection.

Note that **DNS**, **Routes**, **Use this connection only for resources on its network** are common to **IPv4** settings.

8. Once you have finished editing the **VPN** connection, click the **Add** button to customize the configuration or the **Apply** button to save it for the existing one.
9. Switch the profile to **ON** to activate the **VPN** connection.

Additional resources

- **nm-settings-libreswan(5)** man page on your system

6.15. CONFIGURING AN IPSEC BASED VPN CONNECTION BY USING NMSTATECTL

IPsec (Internet Protocol Security) is a security protocol suite, provided by **Libreswan**, for implementation of VPN. IPsec includes protocols to initiate authentication at the time of connection establishment and manage keys during the data transfer. When an application deploys in a network and communicates by using the IP protocol, IPsec can protect data communication.

To manage an IPsec-based configuration for authenticating VPN connections, you can use the **nmstatectl** utility. This utility provides command line access to a declarative API for host network management. The following are the authentication types for the **host-to-subnet** and **host-to-host** communication modes:

- Host-to-subnet PKI authentication
- Host-to-subnet RSA authentication
- Host-to-subnet PSK authentication
- Host-to-host tunnel mode authentication
- Host-to-host transport mode authentication

6.15.1. Configuring a host-to-subnet IPsec VPN with PKI authentication and tunnel mode by using nmstatectl

If you want to use encryption based on the trusted entity authentication in IPsec, Public Key Infrastructure (PKI) provides secure communication by using cryptographic keys between two hosts. Both communicating hosts generate private and public keys where each host maintains a private key by sharing public key with the trusted entity Certificate Authority (CA). The CA generates a digital certificate after verifying the authenticity. In case of encryption and decryption, the host uses a private key for encryption and public key for decryption.

By using Nmstate, a declarative API for network management, you can configure a PKI authentication-based IPsec connection. After setting the configuration, the Nmstate API ensures that the result matches with the configuration file. If anything fails, **nmstate** automatically rolls back the changes to avoid an incorrect state of the system.

To establish encrypted communication in **host-to-subnet** configuration, remote IPsec end provides another IP to host by using parameter **dhcp: true**. In the case of defining systems for **IPsec** in nmstate, the **left**-named system is the local host while the **right**-named system is the remote host. The following procedure needs to run on both hosts.

Prerequisites

- By using a password, you have generated a PKCS #12 file that stores certificates and cryptographic keys.

Procedure

1. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

3. As **Libreswan** was already installed, remove its old database files and re-create them:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

4. Enable and start the **ipsec** service:

```
# systemctl enable --now ipsec
```

5. Import the PKCS#12 file:

```
# ipsec import node-example.p12
```

When importing the PKCS#12 file, enter the password that was used to create the file.

6. Create a YAML file, for example `~/create-pki-authentication.yml`, with the following content:

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: 'yes'         2
    left: '192.0.2.250'           3
    leftid: '%fromcert'           4
    leftcert: 'local-host.example.com' 5
    right: '192.0.2.150'          6
    rightid: '%fromcert'          7
    ikev2: 'insist'               8
    ikelifetime: '24h'            9
    salifetime: '24h'            10
```

The YAML file defines the following settings:

- 1 An IPsec connection name
- 2 The value **yes** means **libreswan** creates an IPsec **xfrm** virtual interface **ipsec<number>** and automatically finds the next available number
- 3 A static IPv4 address of public network interface for a local host
- 4 On a local host, the value of **%fromcert** sets the ID to a Distinguished Name (DN) that is fetched from a loaded certificate
- 5 A Distinguished Name (DN) of a local host's public key
- 6 A static IPv4 address of public network interface for a remote host

- 7 On a remote host, the value of **%fromcert** sets the ID to a Distinguished Name (DN) that is fetched from a loaded certificate.
- 8 **insist** value accepts and receives only the Internet Key Exchange (IKEv2) protocol
- 9 The duration of IKE protocol
- 10 The duration of IPsec security association (SA)

7. Apply settings to the system:

```
# nmstatectl apply ~/create-pki-authentication.yml
```

Verification

1. Verify IPsec status:

```
# ip xfrm status
```

2. Verify IPsec policies:

```
# ip xfrm policy
```

Additional resources

- **ipsec.conf(5)** man page on your system

6.15.2. Configuring a host-to-subnet IPSec VPN with RSA authentication and tunnel mode by using nmstatectl

If you want to use asymmetric cryptography-based key authentication in IPsec, the RSA algorithm provides secure communication by using either of private and public keys for encryption and decryption between two hosts. This method uses a private key for encryption, and a public key for decryption.

By using Nmstate, a declarative API for network management, you can configure RSA-based IPsec authentication. After setting the configuration, the Nmstate API ensures that the result matches with the configuration file. If anything fails, **nmstate** automatically rolls back the changes to avoid an incorrect state of the system.

To establish encrypted communication in **host-to-subnet** configuration, remote IPsec end provides another IP to host by using parameter **dhcp: true**. In the case of defining systems for **IPsec** in nmstate, the **left**-named system is the local host while the **right**-named system is the remote host. The following procedure needs to run on both hosts.

Procedure

1. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

3. If **Libreswan** was already installed, remove its old database files and re-create them:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

4. Generate a RSA key pair on each host:

```
# ipsec newhostkey --output
```

5. Display the public keys:

```
# ipsec showhostkey --list
```

6. The previous step returned the generated key **ckaid**. Use that **ckaid** with the following command on left, for example:

```
# ipsec showhostkey --left --ckaid <0sAwEAAesFfVZqFzRA9F>
```

7. The output of the previous command generated the **lefttrsasigkey=** line required for the configuration. Do the same on the second host (right):

```
# ipsec showhostkey --right --ckaid <0sAwEAAesFfVZqFzRA9E>
```

8. Enable the **ipsec** service to automatically start it on boot:

```
# systemctl enable --now ipsec
```

9. Create a YAML file, for example **~/create-rsa-authentication.yml**, with the following content:

```
---
interfaces:
- name: 'example_ipsec_conn1'
  type: ipsec
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: '99'
    lefttrsasigkey: '0sAwEAAesFfVZqFzRA9F'
    left: '192.0.2.250'
    leftid: 'local-host-rsa.example.com'
    right: '192.0.2.150'
    righttrsasigkey: '0sAwEAAesFfVZqFzRA9E'
    rightid: 'remote-host-rsa.example.com'
    ikev2: 'insist'
```

The YAML file defines the following settings:

- 1 An IPsec connection name
- 2 An interface name
- 3 The value **99** means that **libreswan** creates an IPsec **xfrm** virtual interface **ipsec<number>** and automatically finds the next available number
- 4 The RSA public key of a local host
- 5 A static IPv4 address of public network interface of a local host
- 6 A Distinguished Name (DN) for a local host
- 7 The RSA public key of a remote host
- 8 A static IPv4 address of public network interface of a remote host
- 9 A Distinguished Name(DN) for a remote host
- 10 **insist** value accepts and receives only the Internet Key Exchange (IKEv2) protocol

10. Apply the settings to the system:

```
# nmstatectl apply ~/create-rsa-authentication.yml
```

Verification

1. Display the IP settings of the network interface:

```
# ip addr show example_ipsec_conn1
```

2. Verify IPsec status:

```
# ip xfrm status
```

3. Verify IPsec policies:

```
# ip xfrm policy
```

Additional resources

- **ipsec.conf(5)** man page on your system

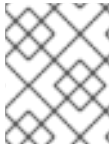
6.15.3. Configuring a host-to-subnet IPSec VPN with PSK authentication and tunnel mode by using nmstatectl

If you want to use encryption based on mutual authentication in IPsec, the Pre-Shared Key (PSK) method provides secure communication by using a secret key between two hosts. A file stores the secret key and the same key encrypts the data flowing through the tunnel.

By using Nmstate, a declarative API for network management, you can configure PSK-based IPsec authentication. After setting the configuration, the Nmstate API ensures that the result matches with the configuration file. If anything fails, **nmstate** automatically rolls back the changes to avoid incorrect

state of the system.

To establish encrypted communication in **host-to-subnet** configuration, remote IPsec end provides another IP to host by using parameter **dhcp: true**. In the case of defining systems for **IPsec** in nmstate, the **left**-named system is the local host while the **right**-named system is the remote host. The following procedure needs to run on both hosts.



NOTE

As this method uses static strings for authentication and encryption, use it only for testing/development purposes.

Procedure

1. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

3. If **Libreswan** was already installed, remove its old database files and re-create them:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

4. Enable the **ipsec** service to automatically start it on boot:

```
# systemctl enable --now ipsec
```

5. Create a YAML file, for example `~/create-pks-authentication.yml`, with the following content:

```
---
interfaces:
- name: 'example_ipsec_conn1'
  type: ipsec
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: 'no'
    right: '192.0.2.250'
    rightid: 'remote-host.example.org'
    left: '192.0.2.150'
    leftid: 'local-host.example.org'
    psk: "example_password"
    ikev2: 'insist'
```

The YAML file defines the following settings:

- 1 An IPsec connection name
- 2 Setting **no** value indicates that **libreswan** creates only **xfrm** policies, and not a virtual **xfrm** interface
- 3 A static IPv4 address of public network interface of a remote host
- 4 A Distinguished Name (DN) for a remote host
- 5 A static IPv4 address of public network interface of a local host
- 6 A Distinguished Name (DN) for a local host
- 7 **insist** value accepts and receives only the Internet Key Exchange (IKEv2) protocol

6. Apply the settings to the system:

```
# nmstatectl apply ~/create-pks-authentication.yml
```

Verification

1. Display the IP settings of network interface:

```
# ip addr show example_ipsec_conn1
```

2. Verify IPsec status:

```
# ip xfrm status
```

3. Verify IPsec policies:

```
# ip xfrm policy
```

6.15.4. Configuring a host-to-host IPsec VPN with PKI authentication and tunnel mode by using nmstatectl

IPsec (Internet Protocol Security) is a security protocol suite to authenticate and encrypt IP communications within networks and devices. The **Libreswan** software provides an IPsec implementation for VPNs.

In tunnel mode, the source and destination IP address of communication is encrypted in the IPsec tunnel. External network sniffers can only get left IP and right IP. In general, for tunnel mode, it supports **host-to-host**, **host-to-subnet**, and **subnet-to-subnet**. In this mode, a new IP packet encapsulates an existing packet along with its payload and header. Encapsulation in this mode protects IP data, source, and destination headers over an unsecure network. This mode is useful to transfer data in **subnet-to-subnet**, remote access connections, and untrusted networks, such as open public Wi-Fi networks. By default, IPsec establishes a secure channel between two sites in tunnel mode. With the following configuration, you can establish a VPN connection as a **host-to-host** architecture.

By using Nmstate, a declarative API for network management, you can configure an IPsec VPN connection. After setting the configuration, the Nmstate API ensures that the result matches with the configuration file. If anything fails, **nmstate** automatically rolls back the changes to avoid incorrect state of the system.

In **host-to-host** configuration, you need to set **leftmodecfgclient: no** so that it can't receive network configuration from the server, hence the value **no**. In the case of defining systems for **IPsec** in **nmstate**, the **left**-named system is the local host while the **right**-named system is the remote host. The following procedure needs to run on both hosts.

Prerequisites

- By using a password, you have generated a PKCS #12 file that stores certificates and cryptographic keys.

Procedure

1. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

3. As **Libreswan** was already installed, remove its old database files and re-create them:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

4. Import the PKCS#12 file:

```
# ipsec import node-example.p12
```

When importing the PKCS#12 file, enter the password that was used to generate the file.

5. Enable and start the **ipsec** service:

```
# systemctl enable --now ipsec
```

6. Create a YAML file, for example **~/create-p2p-vpn-authentication.yml**, with the following content:

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  libreswan:
    left: '192.0.2.250'            2
    leftid: 'local-host.example.com' 3
    leftcert: 'local-host.example.com' 4
    leftmodecfgclient: 'no'        5
    right: '192.0.2.150'           6
    rightid: 'remote-host.example.com' 7
    rightsubnet: '192.0.2.150/32'  8
    ikev2: 'insist'                9
```

The YAML file defines the following settings:

- 1 An IPsec connection name
- 2 A static IPv4 address of public network interface for a local host
- 3 A distinguished Name (DN) of a local host
- 4 A certificate name installed on a local host
- 5 The value for not to retrieve client configuration from a remote host
- 6 A static IPv4 address of public network interface for a remote host
- 7 A distinguished Name (DN) of a remote host
- 8 The subnet range of a remote host - **192.0.2.150** with **32** IPv4 addresses
- 9 The value to accept and receive only the Internet Key Exchange (IKEv2) protocol

7. Apply the settings to the system:

```
# nmstatectl apply ~/create-p2p-vpn-authentication.yml
```

Verification

1. Display the created P2P policy:

```
# ip xfrm policy
```

2. Verify IPsec status:

```
# ip xfrm status
```

Additional resources

- **ipsec.conf(5)** man page on your system

6.15.5. Configuring a host-to-host IPsec VPN with PSK authentication and transport mode by using nmstatectl

IPsec (Internet Protocol Security) is a security protocol suite to authenticate and encrypt IP communications within networks and devices. The **Libreswan** utility provides IPsec based implementation for VPN.

In transport mode, encryption works only for the payload of an IP packet. Also, a new IPsec header gets appended to the IP packet by keeping the original IP header as it is. Transport mode does not encrypt the source and destination IP of communication but copies them to an external IP header. Hence, encryption protects only IP data across the network. This mode is useful to transfer data in a **host-to-host** connection of a network. This mode is often used along with the GRE tunnel to save 20 bytes (IP header) of overheads. By default, the **IPsec** utility uses tunnel mode. To use transfer mode, set **type: transport** for **host-to-host** connection data transfer.

By using Nmstate, a declarative API for network management, you can configure an IPsec VPN

connection. After setting the configuration, the Nmstate API ensures that the result matches with the configuration file. If anything fails, **nmstate** automatically rolls back the changes to avoid incorrect state of the system. To override the default **tunnel** mode, specify **transport** mode.

In the case of defining systems for **IPsec** in nmstate, the **left**-named system is the local host while the **right**-named system is the remote host. The following procedure needs to run on both hosts.

Prerequisites

- By using a password, you have generated a PKCS #12 file that stores certificates and cryptographic keys.

Procedure

1. Install the required packages:

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. Restart the NetworkManager service:

```
# systemctl restart NetworkManager
```

3. As **Libreswan** was already installed, remove its old database files and re-create them:

```
# systemctl stop ipsec
# rm /var/lib/ipsec/nss/*db
# ipsec initnss
```

4. Import the PKCS#12 file:

```
# ipsec import node-example.p12
```

When importing the PKCS#12 file, enter the password that was used to create the file.

5. Enable and start the **ipsec** service:

```
# systemctl enable --now ipsec
```

6. Create a YAML file, for example `~/create-p2p-transport-authentication.yml`, with the following content:

```
---
interfaces:
- name: 'example_ipsec_conn1' 1
  type: ipsec
  libreswan:
    type: 'transport' 2
    ipsec-interface: '99' 3
    left: '192.0.2.250' 4
    leftid: '%fromcert' 5
    leftcert: 'local-host.example.org' 6
    right: '192.0.2.150' 7
    prefix-length: '32' 8
```

```

rightid: '%fromcert'
ikev2: 'insist'
ikelifetime: '24h'
salifetime: '24h'

```

9
10
11
12

The YAML file defines the following settings:

- 1 An IPsec connection name
- 2 An IPsec mode
- 3 The value **99** means that **libreswan** creates an IPsec **xfrm** virtual interface **ipsec<number>** and automatically finds the next available number
- 4 A static IPv4 address of public network interface for a local host
- 5 On a local host, the value of **%fromcert** sets the ID to a Distinguished Name (DN) which is fetched from a loaded certificate
- 6 A Distinguished Name (DN) of a local host's public key
- 7 A static IPv4 address of public network interface for a remote host
- 8 The subnet mask of a static IPv4 address of a local host
- 9 On a remote host, the value of **%fromcert** sets the ID to a Distinguished Name (DN) which is fetched from a loaded certificate
- 10 The value to accept and receive only the Internet Key Exchange (IKEv2) protocol
- 11 The duration of IKE protocol
- 12 The duration of IPsec security association (SA)

7. Apply the settings to the system:

```
# nmstatectl apply ~/create-p2p-transport-authentication.yml
```

Verification

1. Verify IPsec status:

```
# ip xfrm status
```

2. Verify IPsec policies:

```
# ip xfrm policy
```

Additional resources

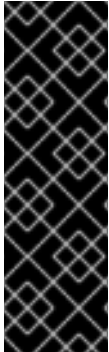
- [ipsec.conf\(5\)](#) man page on your system

6.16. ADDITIONAL RESOURCES

- **ipsec(8)**, **ipsec.conf(5)**, **ipsec.secrets(5)**, **ipsec_auto(8)**, and **ipsec_rasigkey(8)** man pages on your system
- **/usr/share/doc/libreswan-*version*/** directory

CHAPTER 7. SETTING UP A WIREGUARD VPN

WireGuard is a high-performance VPN solution that runs in the Linux kernel. It uses modern cryptography and is easier to configure than many other VPN solutions. Additionally, WireGuard's small codebase reduces the surface for attacks and, therefore, improves security. For authentication and encryption, WireGuard uses keys similar to SSH.



IMPORTANT

WireGuard is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See [Technology Preview Features Support Scope](#) on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

Note that all hosts that participate in a WireGuard VPN are peers. This documentation uses the terms **client** to describe hosts that establish a connection and **server** to describe the host with the fixed hostname or IP address that the clients connect to and optionally route all traffic through this server.

To set up a WireGuard VPN, you must complete the following steps:

1. Configure the server.
2. Open the WireGuard port in the local firewall.
3. Configure the clients.

WireGuard operates on the network layer (layer 3). Therefore, you cannot use DHCP and must assign static IP addresses or IPv6 global addresses to the tunnel devices on both the server and clients.



IMPORTANT

You can use WireGuard only if the Federal Information Processing Standard (FIPS) mode in RHEL is disabled.

7.1. PROTOCOLS AND PRIMITIVES USED BY WIREGUARD

WireGuard uses the following protocols and primitives:

- ChaCha20 for symmetric encryption, authenticated with Poly1305, using Authenticated Encryption with Associated Data (AEAD) construction as described in [RFC7539](#)
- Curve25519 for Elliptic-curve Diffie–Hellman (ECDH) key exchange
- BLAKE2s for hashing and keyed hashing, as described in [RFC7693](#)
- SipHash24 for hash table keys
- HKDF for key derivation, as described in [RFC5869](#)

7.2. HOW WIREGUARD USES TUNNEL IP ADDRESSES, PUBLIC KEYS, AND REMOTE ENDPOINTS

When WireGuard sends a network packet to a peer:

1. WireGuard reads the destination IP from the packet and compares it to the list of allowed IP addresses in the local configuration. If the peer is not found, WireGuard drops the packet.
2. If the peer is valid, WireGuard encrypts the packet using the peer's public key.
3. The sending host looks up the most recent Internet IP address of the host and sends the encrypted packet to it.

When WireGuard receives a packet:

1. WireGuard decrypts the packet using the private key of the remote host.
2. WireGuard reads the internal source address from the packet and looks up whether the IP is configured in the list of allowed IP addresses in the settings for the peer on the local host. If the source IP is on the allowlist, WireGuard accepts the packet. If the IP address is not on the list, WireGuard drops the packet.

The association of public keys and allowed IP addresses is called **Cryptokey Routing Table**. This means that the list of IP addresses behaves similar to a routing table when sending packets, and as a kind of access control list when receiving packets.

7.3. USING A WIREGUARD CLIENT BEHIND NAT AND FIREWALLS

WireGuard uses the UDP protocol and transmits data only when a peer sends packets. Stateful firewalls and network address translation (NAT) on routers track connections to enable a peer behind NAT or a firewall to receive packets.

To keep the connection active, WireGuard supports **persistent keepalives**. This means you can set an interval at which WireGuard sends keepalive packets. By default, the persistent keep-alive feature is disabled to reduce network traffic. Enable this feature on the client if you use the client in a network with NAT or if a firewall closes the connection after some time of inactivity.



NOTE

Note that you cannot configure keepalive packets in WireGuard connections by using the RHEL web console. To configure this feature, edit the connection profile by using the **nmcli** utility.

7.4. CREATING PRIVATE AND PUBLIC KEYS TO BE USED IN WIREGUARD CONNECTIONS

WireGuard uses base64-encoded private and public keys to authenticate hosts to each other. Therefore, you must create the keys on each host that participates in the WireGuard VPN.



IMPORTANT

For secure connections, create different keys for each host, and ensure that you only share the public key with the remote WireGuard host. Do not use the example keys used in this documentation.

If you plan to use the RHEL web console to create a WireGuard VPN connection, you can, alternatively, generate the public and private key pairs in the web console.

Procedure

1. Install the **wireguard-tools** package:

```
# dnf install wireguard-tools
```

2. Create a private key and a corresponding public key for the host:

```
# wg genkey | tee /etc/wireguard/$HOSTNAME.private.key | wg pubkey >
/etc/wireguard/$HOSTNAME.public.key
```

You will need the content of the key files, but not the files themselves. However, Red Hat recommends keeping the files in case that you need to remember the keys in future.

3. Set secure permissions on the key files:

```
# chmod 600 /etc/wireguard/$HOSTNAME.private.key
/etc/wireguard/$HOSTNAME.public.key
```

4. Display the private key:

```
# cat /etc/wireguard/$HOSTNAME.private.key
YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=
```

You will need the private key to configure the WireGuard connection on the local host. Do not share the private key.

5. Display the public key:

```
# cat /etc/wireguard/$HOSTNAME.public.key
UtiqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
```

You will need the public key to configure the WireGuard connection on the remote host.

Additional resources

- **wg(8)** man page on your system

7.5. CONFIGURING A WIREGUARD SERVER BY USING NMCLI

You can configure the WireGuard server by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Server:
 - Private key: **YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
 - Tunnel IPv4 address: **192.0.2.1/24**

- Tunnel IPv6 address: **2001:db8:1::1/32**
- Client:
 - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
 - Tunnel IPv4 address: **192.0.2.2/24**
 - Tunnel IPv6 address: **2001:db8:1::2/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
 - The private key of the server
 - The static tunnel IP addresses and subnet masks of the client
 - The public key of the client
 - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Add a NetworkManager WireGuard connection profile:

```
# nmcli connection add type wireguard con-name server-wg0 ifname wg0 autoconnect
no
```

This command creates a profile named **server-wg0** and assigns the virtual interface **wg0** to it. To prevent the connection from starting automatically after you add it without finalizing the configuration, disable the **autoconnect** parameter.

2. Set the tunnel IPv4 address and subnet mask of the server:

```
# nmcli connection modify server-wg0 ipv4.method manual ipv4.addresses
192.0.2.1/24
```

3. Set the tunnel IPv6 address and subnet mask of the server:

```
# nmcli connection modify server-wg0 ipv6.method manual ipv6.addresses
2001:db8:1::1/32
```

4. Add the server's private key to the connection profile:

```
# nmcli connection modify server-wg0 wireguard.private-key
"YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg="
```

5. Set the port for incoming WireGuard connections:

```
# nmcli connection modify server-wg0 wireguard.listen-port 51820
```

Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the **wg0** interface.

6. Add peer configurations for each client that you want to allow to communicate with this server. You must add these settings manually, because the **nmcli** utility does not support setting the corresponding connection properties.
 - a. Edit the **/etc/NetworkManager/system-connections/server-wg0.nmconnection** file, and append:

```
[wireguard-peer.bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=]
allowed-ips=192.0.2.2;2001:db8:1::2;
```

- The **[wireguard-peer.<public_key_of_the_client>]** entry defines the peer section of the client, and the section name contains the public key of the client.
 - The **allowed-ips** parameter sets the tunnel IP addresses of the client that are allowed to send data to this server.
- Add a section for each client.

- b. Reload the **server-wg0** connection profile:

```
# nmcli connection load /etc/NetworkManager/system-connections/server-
wg0.nmconnection
```

7. Optional: Configure the connection to start automatically, enter:

```
# nmcli connection modify server-wg0 autoconnect yes
```

8. Reactivate the **server-wg0** connection:

```
# nmcli connection up server-wg0
```

Next steps

- [Configure the firewalld service on the WireGuard server.](#)

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

2. Display the IP configuration of the **wg0** device:

■

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Additional resources

- **wg(8)** man page on your system
- The **WireGuard setting** section in the **nm-settings(5)** man page on your system

7.6. CONFIGURING A WIREGUARD SERVER BY USING **NMTUI**

You can configure the WireGuard server by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Server:
 - Private key: **YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
 - Tunnel IPv4 address: **192.0.2.1/24**
 - Tunnel IPv6 address: **2001:db8:1::1/32**
- Client:
 - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
 - Tunnel IPv4 address: **192.0.2.2/24**
 - Tunnel IPv6 address: **2001:db8:1::2/32**

Prerequisites

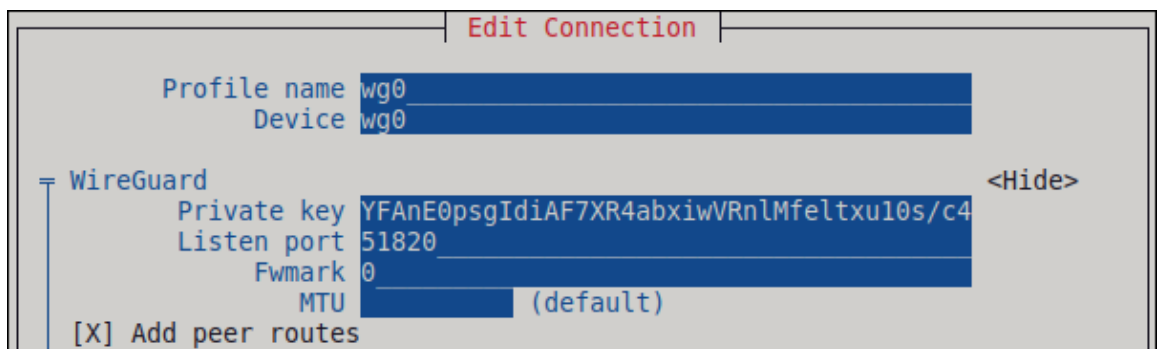
- You have generated the public and private key for both the server and client.
- You know the following information:
 - The private key of the server
 - The static tunnel IP addresses and subnet masks of the client
 - The public key of the client
 - The static tunnel IP addresses and subnet masks of the server
- You installed the **NetworkManager-tui** package.

Procedure

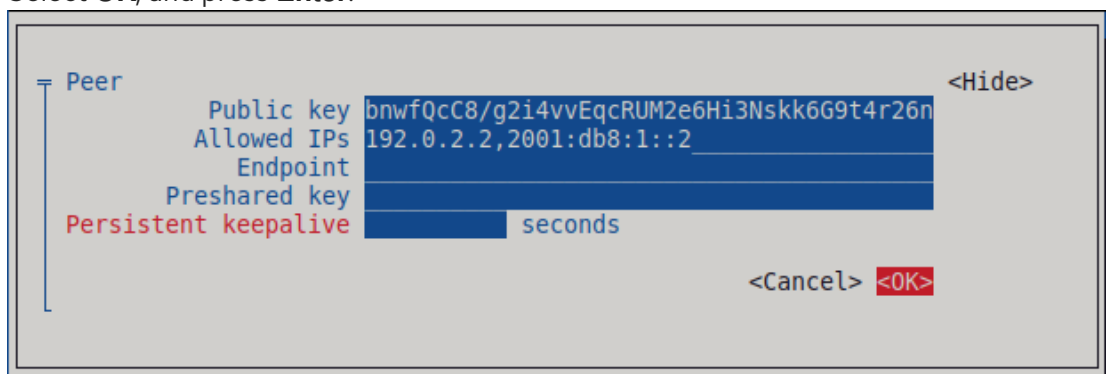
1. Start the **nmtui** application:

```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.
3. Select **Add**, and press **Enter**.
4. Select the **WireGuard** connection type in the list, and press **Enter**.
5. In the **Edit connection** window:
 - a. Enter the name of the connection and the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
 - b. Enter the private key of the server.
 - c. Set the listen port number, such as **51820**, for incoming WireGuard connections. Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the interface.



- d. Click **Add** next to the **Peers** pane:
 - i. Enter the public key of the client.
 - ii. Set the **Allowed IPs** field to the tunnel IP addresses of the client that are allowed to send data to this server.
 - iii. Select **OK**, and press **Enter**.



- e. Select **Show** next to ***IPv4 Configuration**, and press **Enter**.
 - i. Select the IPv4 configuration method **Manual**.

- ii. Enter the tunnel IPv4 address and the subnet mask. Leave the **Gateway** field empty.
- f. Select **Show** next to **IPv6 Configuration**, and press **Enter**.
 - i. Select the IPv6 configuration method **Manual**.
 - ii. Enter the tunnel IPv6 address and the subnet mask. Leave the **Gateway** field empty.
- g. Select **OK**, and press **Enter**

```

= IPv4 CONFIGURATION <Manual> <Hide>
  Addresses 192.0.2.1/24 <Remove>
             <Add...>
  Gateway   <Add...>
  DNS servers <Add...>
  Search domains <Add...>

  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv4 addressing for this connection

= IPv6 CONFIGURATION <Manual> <Hide>
  Addresses 2001:db8:1::1/32 <Remove>
             <Add...>
  Gateway   <Add...>
  DNS servers <Add...>
  Search domains <Add...>
  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv6 addressing for this connection

[X] Automatically connect
[X] Available to all users

<Cancel> <OK>
  
```

6. In the window with the list of connections, select **Back**, and press **Enter**.
7. In the **NetworkManager TUI** main window, select **Quit**, and press **Enter**.

Next steps

- [Configure the firewalld service on the WireGuard server.](#)

Verification

1. Display the interface configuration of the **wg0** device:

```

# wg show wg0
interface: wg0
public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
private key: (hidden)
listening port: 51820
  
```

```
peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 _2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Additional resources

- **wg(8)** man page on your system

7.7. CONFIGURING A WIREGUARD SERVER BY USING THE RHEL WEB CONSOLE

You can configure a WireGuard server by using the browser-based RHEL web console. Use this method to let NetworkManager manage the WireGuard connection.

Prerequisites

- You are logged in to the RHEL web console.
- You know the following information:
 - The static tunnel IP addresses and subnet masks of both the server and client
 - The public key of the client

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add VPN** in the **Interfaces** section.
3. If the **wireguard-tools** and **systemd-resolved** packages are not already installed, the web console displays a corresponding notification. Click **Install** to install these packages.
4. Enter the name of the WireGuard device that you want to create.
5. Configure the key pair of this host:
 - If you want use the keys that the web console has created:

- i. Keep the pre-selected **Generated** option in the **Private key** area.
 - ii. Note the **Public key** value. You require this information when you configure the client.
- If you want to use an existing private key:
 - i. Select **Paste existing key** in the **Private key** area.
 - ii. Paste the private key into the text field. The web console automatically calculates the corresponding public key.
6. Set a listen port number, such as **51820**, for incoming WireGuard connections.
Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the interface.
7. Set the tunnel IPv4 address and subnet mask of the server.
To also set an IPv6 address, you must edit the connection after you have created it.
8. Add peer configurations for each client that you want to allow to communicate with this server:
 - a. Click **Add peer**.
 - b. Enter the public key of the client.
 - c. Leave the **Endpoint** field empty.
 - d. Set the **Allowed IPs** field to the tunnel IP addresses of the clients that are allowed to send data to this server.

Add WireGuard VPN ×

Name


wg0

Private key

☒ Generated
 ☐ Paste existing key

YFAnE0psgIdiAF7XR4abxiwVRnLMfeltxu10s/c4JXg=

Public key

UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa406BE= 


Listen port

51820


IPv4 addresses

192.0.2.1/24

Multiple addresses can be specified using commas or spaces as delimiters.

Peers 

Add peer

Public key	Endpoint	Allowed IPs	
bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t ...		192.0.2.2	

Add

Cancel

9. Click **Add** to create the WireGuard connection.
10. If you want to also set a tunnel IPv6 address:
 - a. Click on the WireGuard connection's name in the **Interfaces** section.
 - b. Click **edit** next to **IPv6**.
 - c. Set the **Addresses** field to **Manual**, and enter the tunnel IPv6 address and prefix of the server.
 - d. Click **Save**.

Next steps

- [Configure the firewalld service on the WireGuard server.](#)

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWERUP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

7.8. CONFIGURING FIREWALLD ON A WIREGUARD SERVER BY USING THE COMMAND LINE

You must configure the **firewalld** service on the WireGuard server to allow incoming connections from clients. Additionally, if clients should be able to use the WireGuard server as the default gateway and route all traffic through the tunnel, you must enable masquerading.

Procedure

1. Open the WireGuard port for incoming connections in the **firewalld** service:

```
# firewall-cmd --permanent --add-port=51820/udp --zone=public
```

2. If clients should route all traffic through the tunnel and use the WireGuard server as the default gateway, enable masquerading for the **public** zone:

```
# firewall-cmd --permanent --zone=public --add-masquerade
```

3. Reload the **firewalld** rules.

```
# firewall-cmd --reload
```

Verification

- Display the configuration of the **public** zone:

```
# firewall-cmd --list-all
public (active)
...
ports: 51820/udp
masquerade: yes
...
```

Additional resources

- **firewall-cmd(1)** man page on your system

7.9. CONFIGURING FIREWALLD ON A WIREGUARD SERVER BY USING THE RHEL WEB CONSOLE

You must configure the **firewalld** service on the WireGuard server to allow incoming connections from clients. Additionally, if clients should be able to use the WireGuard server as the default gateway and route all traffic through the tunnel, you must enable masquerading.

Prerequisites

- You are logged in to the RHEL web console.

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Edit rules and zones** in the **Firewall** section.
3. Enter **wireguard** into the **Filter services** field.
4. Select the **wireguard** entry from the list.



5. Click **Add services**.
6. If clients should route all traffic through the tunnel and use the WireGuard server as the default gateway, enable masquerading for the **public** zone:

```
# firewall-cmd --permanent --zone=public --add-masquerade
# firewall-cmd --reload
```

Note that you cannot enable masquerading in **firewalld** zones in the web console.

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Edit rules and zones** in the **Firewall** section.
3. The list contains an entry for the **wireguard** service and displays the UDP port that you configured in the WireGuard connection profile.
4. To verify that masquerading is enabled in the **public** zone of **firewalld**, enter:

```
# firewall-cmd --list-all --zone=public
public (active)
...
ports: 51820/udp
masquerade: yes
...
```

7.10. CONFIGURING A WIREGUARD CLIENT BY USING NMCLI

You can configure a WireGuard client by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Client:
 - Private key: **aPUcp5vHz8yMLrzK8SsDyYnV33lhE/k20e52iKJFV0A=**
 - Tunnel IPv4 address: **192.0.2.2/24**

- Tunnel IPv6 address: **2001:db8:1::2/32**
- Server:
 - Public key: **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
 - Tunnel IPv4 address: **192.0.2.1/24**
 - Tunnel IPv6 address: **2001:db8:1::1/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
 - The private key of the client
 - The static tunnel IP addresses and subnet masks of the client
 - The public key of the server
 - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Add a NetworkManager WireGuard connection profile:

```
# nmcli connection add type wireguard con-name client-wg0 ifname wg0 autoconnect no
```

This command creates a profile named **client-wg0** and assigns the virtual interface **wg0** to it. To prevent the connection from starting automatically after you add it without finalizing the configuration, disable the **autoconnect** parameter.

2. Optional: Configure NetworkManager so that it does not automatically start the **client-wg** connection:

```
# nmcli connection modify client-wg0 autoconnect no
```

3. Set the tunnel IPv4 address and subnet mask of the client:

```
# nmcli connection modify client-wg0 ipv4.method manual ipv4.addresses 192.0.2.2/24
```

4. Set the tunnel IPv6 address and subnet mask of the client:

```
# nmcli connection modify client-wg0 ipv6.method manual ipv6.addresses 2001:db8:1::2/32
```

5. If you want to route all traffic through the tunnel, set the tunnel IP addresses of the server as the default gateway:

```
# nmcli connection modify client-wg0 ipv4.gateway 192.0.2.1 ipv6.gateway 2001:db8:1::1
```

Routing all traffic through the tunnel requires that you set, in a later step, the **allowed-ips** on this client to **0.0.0.0/0:::0**.

Note that routing all traffic through the tunnel can impact the connectivity to other hosts based on the server routing and firewall configuration.

6. Add the client's private key to the connection profile:

```
# nmcli connection modify client-wg0 wireguard.private-key
"aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A="
```

7. Add peer configurations for each server that you want to allow to communicate with this client. You must add these settings manually, because the **nmcli** utility does not support setting the corresponding connection properties.

- a. Edit the **/etc/NetworkManager/system-connections/client-wg0.nmconnection** file, and append:

```
[wireguard-peer.UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=]
endpoint=server.example.com:51820
allowed-ips=192.0.2.1;2001:db8:1::1;
persistent-keepalive=20
```

- The **[wireguard-peer.<public_key_of_the_server>]** entry defines the peer section of the server, and the section name has the public key of the server.
- The **endpoint** parameter sets the hostname or IP address and the port of the server. The client uses this information to establish the connection.
- The **allowed-ips** parameter sets a list of IP addresses that can send data to this client. For example, set the parameter to:
 - The tunnel IP addresses of the server to allow only the server to communicate with this client. The value in the example above configures this scenario.
 - **0.0.0.0/0:::0**; to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.
- The optional **persistent-keepalive** parameter defines an interval in seconds in which WireGuard sends a keepalive packet to the server. Set this parameter if you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity.

- b. Reload the **client-wg0** connection profile:

```
# nmcli connection load /etc/NetworkManager/system-connections/client-
wg0.nmconnection
```

8. Reactivate the **client-wg0** connection:

```
# nmcli connection up client-wg0
```

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

Note that the output has only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Additional resources

- **wg(8)** man page on your system
- The **WireGuard setting** section in the **nm-settings(5)** man page on your system

7.11. CONFIGURING A WIREGUARD CLIENT BY USING **NMTUI**

You can configure a WireGuard client by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Client:
 - Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**

- Tunnel IPv4 address: **192.0.2.2/24**
- Tunnel IPv6 address: **2001:db8:1::2/32**
- Server:
 - Public key: **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
 - Tunnel IPv4 address: **192.0.2.1/24**
 - Tunnel IPv6 address: **2001:db8:1::1/32**

Prerequisites

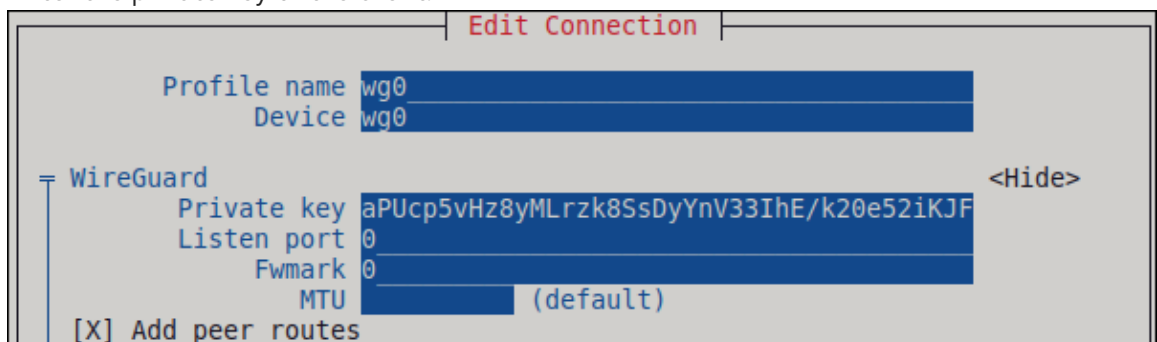
- You have generated the public and private key for both the server and client.
- You know the following information:
 - The private key of the client
 - The static tunnel IP addresses and subnet masks of the client
 - The public key of the server
 - The static tunnel IP addresses and subnet masks of the server
- You installed the **NetworkManager-tui** package

Procedure

1. Start the **nmtui** application:

```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.
3. Select **Add**, and press **Enter**.
4. Select the **WireGuard** connection type in the list, and press **Enter**.
5. In the **Edit connection** window:
 - a. Enter the name of the connection and the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
 - b. Enter the private key of the client.



- c. Click **Add** next to the **Peers** pane:
 - i. Enter the public key of the server.
 - ii. Set the **Allowed IPs** field. For example, set it to:
 - The tunnel IP addresses of the server to allow only the server to communicate with this client.
 - **0.0.0.0/0,::/0** to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.
 - iii. Enter the host name or IP address and port of the WireGuard server into the **Endpoint** field. Use the following format: **<hostname_or_IP>:<port_number>**
 - iv. Optional: If you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity, set a persistent keepalive interval in seconds. In this interval, the client sends a keepalive packet to the server.
 - v. Select **OK**, and press **Enter**.

Peer <Hide>

Public key	J57DeAscYKRfp7cFGi0qd0NRn69u249Fa406BE=
Allowed IPs	192.0.2.1,2001:db8:1::1
Endpoint	server.example.com:51820
Preshared key	
Persistent keepalive	20 seconds

<Cancel> <OK>

- d. Select **Show** next to **IPv4 Configuration**, and press **Enter**.
 - i. Select the IPv4 configuration method **Manual**.
 - ii. Enter the tunnel IPv4 address and the subnet mask. Leave the **Gateway** field empty.
- e. Select **Show** next to **IPv6 Configuration**, and press **Enter**.
 - i. Select the IPv6 configuration method **Manual**.
 - ii. Enter the tunnel IPv6 address and the subnet mask. Leave the **Gateway** field empty.
- f. Optional: Select **Automatically connect**.
- g. Select **OK**, and press **Enter**

```

= IPv4 CONFIGURATION <Manual> <Hide>
  Addresses 192.0.2.2/24 <Remove>
             <Add...>
  Gateway   <Add...>
  DNS servers <Add...>
  Search domains <Add...>

  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv4 addressing for this connection

= IPv6 CONFIGURATION <Manual> <Hide>
  Addresses 2001:db8:1::2/32 <Remove>
             <Add...>
  Gateway   <Add...>
  DNS servers <Add...>
  Search domains <Add...>
  Routing (No custom routes) <Edit...>
  [ ] Never use this network for default route
  [ ] Ignore automatically obtained routes
  [ ] Ignore automatically obtained DNS parameters
  [ ] Require IPv6 addressing for this connection

[X] Automatically connect
[X] Available to all users

<Cancel> <OK>

```

6. In the window with the list of connections, select **Back**, and press **Enter**.
7. In the **NetworkManager TUI** main window, select **Quit**, and press **Enter**.

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820_
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```


To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

Note that the output contains only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Additional resources

- **wg(8)** man page on your system

7.12. CONFIGURING A WIREGUARD CLIENT BY USING THE RHEL WEB CONSOLE

You can configure a WireGuard client by using the browser-based RHEL web console. Use this method to let NetworkManager manage the WireGuard connection.

Prerequisites

- You are logged in to the RHEL web console.
- You know the following information:
 - The static tunnel IP addresses and subnet masks of both the server and client
 - The public key of the server

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add VPN** in the **Interfaces** section.
3. If the **wireguard-tools** and **systemd-resolved** packages are not already installed, the web console displays a corresponding notification. Click **Install** to install these packages.
4. Enter the name of the WireGuard device that you want to create.
5. Configure the key pair of this host:
 - If you want use the keys that the web console has created:
 - i. Keep the pre-selected **Generated** option in the **Private key** area.

- ii. Note the **Public key** value. You require this information when you configure the client.
- If you want to use an existing private key:
 - i. Select **Paste existing key** in the **Private key** area.
 - ii. Paste the private key into the text field. The web console automatically calculates the corresponding public key.
6. Preserve the **0** value in the **Listen port** field.
7. Set the tunnel IPv4 address and subnet mask of the client.
To also set an IPv6 address, you must edit the connection after you have created it.
8. Add a peer configuration for the server that you want to allow to communicate with this client:
 - a. Click **Add peer**.
 - b. Enter the public key of the server.
 - c. Set the **Endpoint** field to the hostname or IP address and the port of the server, for example **server.example.com:51820**. The client uses this information to establish the connection.
 - d. Set the **Allowed IPs** field to the tunnel IP addresses of the clients that are allowed to send data to this server. For example, set the field to one of the following:
 - The tunnel IP addresses of the server to allow only the server to communicate with this client. The value in the screen capture below configures this scenario.
 - **0.0.0.0/0** to allow any remote IPv4 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.

Add WireGuard VPN

Name

Private key
☒ Generated
☐ Paste existing key

aPUcp5vHz8yMLrzk8SsDyYnV33IhE/k20e52iKJFV0A=

Public key

bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=

Listen port
Will be set to "Automatic"

IPv4 addresses

Multiple addresses can be specified using commas or spaces as delimiters.

Peers ⓘ

Public key
Endpoint
Allowed IPs

UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u24 ...
server.example.com ...
192.0.2.1/24

Add peer

Add Cancel

9. Click **Add** to create the WireGuard connection.

10. If you want to also set a tunnel IPv6 address:

- Click on the WireGuard connection's name in the **Interfaces** section.
- Click **Edit** next to **IPv6**.
- Set the **Addresses** field to **Manual**, and enter the tunnel IPv6 address and prefix of the client.
- Click **Save**.

Verification

- Ping the IP addresses of the server:

```
# ping 192.0.2.1
```

WireGuard establishes the connection when you try to send traffic through the tunnel.

- Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
private key: (hidden)
listening port: 45513
```

```

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
endpoint: server.example.com:51820
allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
latest handshake: 1 minute, 41 seconds ago
transfer: 824 B received, 1.01 KiB sent
persistent keepalive: every 20 seconds

```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show wg0** command.

Note that the output has only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```

# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWERUP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

CHAPTER 8. CONFIGURING IP TUNNELS

Similar to a VPN, an IP tunnel directly connects two networks over a third network, such as the internet. However, not all tunnel protocols support encryption.

The routers in both networks that establish the tunnel requires at least two interfaces:

- One interface that is connected to the local network
- One interface that is connected to the network through which the tunnel is established.

To establish the tunnel, you create a virtual interface on both routers with an IP address from the remote subnet.

NetworkManager supports the following IP tunnels:

- Generic Routing Encapsulation (GRE)
- Generic Routing Encapsulation over IPv6 (IP6GRE)
- Generic Routing Encapsulation Terminal Access Point (GRETAP)
- Generic Routing Encapsulation Terminal Access Point over IPv6 (IP6GRETAP)
- IPv4 over IPv4 (IPIP)
- IPv4 over IPv6 (IPIP6)
- IPv6 over IPv6 (IP6IP6)
- Simple Internet Transition (SIT)

Depending on the type, these tunnels act either on layer 2 or 3 of the Open Systems Interconnection (OSI) model.

8.1. CONFIGURING AN IPIP TUNNEL TO ENCAPSULATE IPV4 TRAFFIC IN IPV4 PACKETS

An IP over IP (IPIP) tunnel operates on OSI layer 3 and encapsulates IPv4 traffic in IPv4 packets as described in [RFC 2003](#).

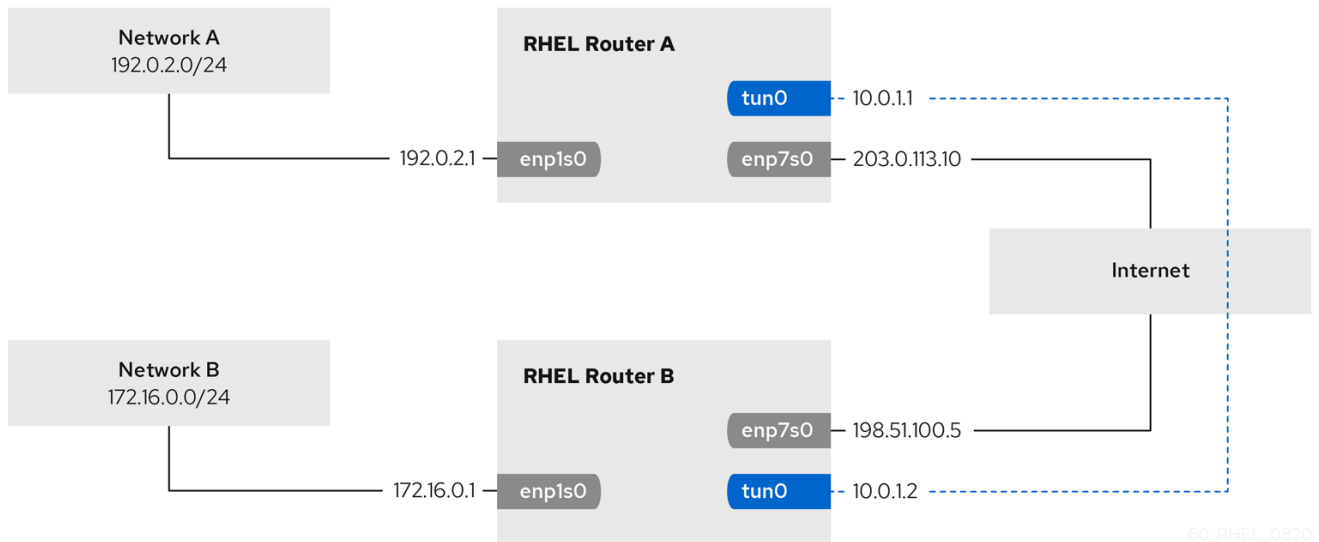


IMPORTANT

Data sent through an IPIP tunnel is not encrypted. For security reasons, use the tunnel only for data that is already encrypted, for example, by other protocols, such as HTTPS.

Note that IPIP tunnels support only unicast packets. If you require an IPv4 tunnel that supports multicast, see [Configuring a GRE tunnel using nmcli to encapsulate layer-3 traffic in IPv4 packets](#) .

For example, you can create an IPIP tunnel between two RHEL routers to connect two internal subnets over the internet as shown in the following diagram:



Prerequisites

- Each RHEL router has a network interface that is connected to its local subnet.
- Each RHEL router has a network interface that is connected to the internet.
- The traffic you want to send through the tunnel is IPv4 unicast.

Procedure

1. On the RHEL router in network A:

- a. Create an IPIP tunnel interface named **tun0**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode ipip con-name tun0 ifname tun0 remote 198.51.100.5 local 203.0.113.10
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

- b. Set the IPv4 address to the **tun0** device:

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.1/30'
```

Note that a **/30** subnet with two usable IP addresses is sufficient for the tunnel.

- c. Configure the **tun0** connection to use a manual IPv4 configuration:

```
# nmcli connection modify tun0 ipv4.method manual
```

- d. Add a static route that routes traffic to the **172.16.0.0/24** network to the tunnel IP on router B:

```
# nmcli connection modify tun0 +ipv4.routes "172.16.0.0/24 10.0.1.2"
```

- e. Enable the **tun0** connection.

```
# nmcli connection up tun0
```

- f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. On the RHEL router in network B:

- a. Create an IPIP tunnel interface named **tun0**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode ipip con-name tun0 ifname
tun0 remote 203.0.113.10 local 198.51.100.5
```

The **remote** and **local** parameters set the public IP addresses of the remote and local routers.

- b. Set the IPv4 address to the **tun0** device:

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.2/30'
```

- c. Configure the **tun0** connection to use a manual IPv4 configuration:

```
# nmcli connection modify tun0 ipv4.method manual
```

- d. Add a static route that routes traffic to the **192.0.2.0/24** network to the tunnel IP on router A:

```
# nmcli connection modify tun0 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

- e. Enable the **tun0** connection.

```
# nmcli connection up tun0
```

- f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

Verification

- From each RHEL router, ping the IP address of the internal interface of the other router:

- a. On Router A, ping **172.16.0.1**:

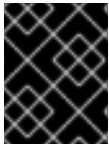
```
# ping 172.16.0.1
```

- b. On Router B, ping **192.0.2.1**:

```
# ping 192.0.2.1
```

8.2. CONFIGURING A GRE TUNNEL TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS

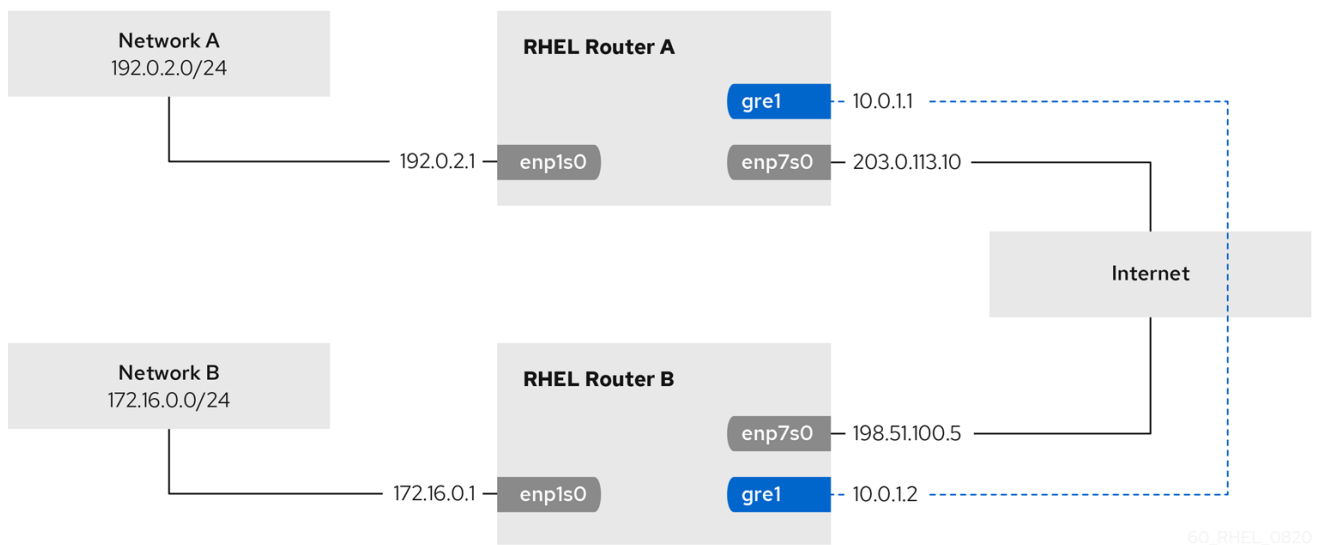
A Generic Routing Encapsulation (GRE) tunnel encapsulates layer-3 traffic in IPv4 packets as described in [RFC 2784](#). A GRE tunnel can encapsulate any layer 3 protocol with a valid Ethernet type.



IMPORTANT

Data sent through a GRE tunnel is not encrypted. For security reasons, use the tunnel only for data that is already encrypted, for example, by other protocols, such as HTTPS.

For example, you can create a GRE tunnel between two RHEL routers to connect two internal subnets over the internet as shown in the following diagram:



Prerequisites

- Each RHEL router has a network interface that is connected to its local subnet.
- Each RHEL router has a network interface that is connected to the internet.

Procedure

1. On the RHEL router in network A:
 - a. Create a GRE tunnel interface named **gre1**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gre con-name gre1 ifname gre1 remote 198.51.100.5 local 203.0.113.10
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.



IMPORTANT

The **gre0** device name is reserved. Use **gre1** or a different name for the device.

- b. Set the IPv4 address to the **gre1** device:

```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.1/30'
```

Note that a **/30** subnet with two usable IP addresses is sufficient for the tunnel.

- c. Configure the **gre1** connection to use a manual IPv4 configuration:

```
# nmcli connection modify gre1 ipv4.method manual
```

- d. Add a static route that routes traffic to the **172.16.0.0/24** network to the tunnel IP on router B:

```
# nmcli connection modify gre1 +ipv4.routes "172.16.0.0/24 10.0.1.2"
```

- e. Enable the **gre1** connection.

```
# nmcli connection up gre1
```

- f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. On the RHEL router in network B:

- a. Create a GRE tunnel interface named **gre1**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gre con-name gre1 ifname
gre1 remote 203.0.113.10 local 198.51.100.5
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

- b. Set the IPv4 address to the **gre1** device:

```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.2/30'
```

- c. Configure the **gre1** connection to use a manual IPv4 configuration:

```
# nmcli connection modify gre1 ipv4.method manual
```

- d. Add a static route that routes traffic to the **192.0.2.0/24** network to the tunnel IP on router A:

```
# nmcli connection modify gre1 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

- e. Enable the **gre1** connection.

```
# nmcli connection up gre1
```

- f. Enable packet forwarding:

■

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

Verification

- From each RHEL router, ping the IP address of the internal interface of the other router:

- On Router A, ping **172.16.0.1**:

```
# ping 172.16.0.1
```

- On Router B, ping **192.0.2.1**:

```
# ping 192.0.2.1
```

8.3. CONFIGURING A GRE TAP TUNNEL TO TRANSFER ETHERNET FRAMES OVER IPV4

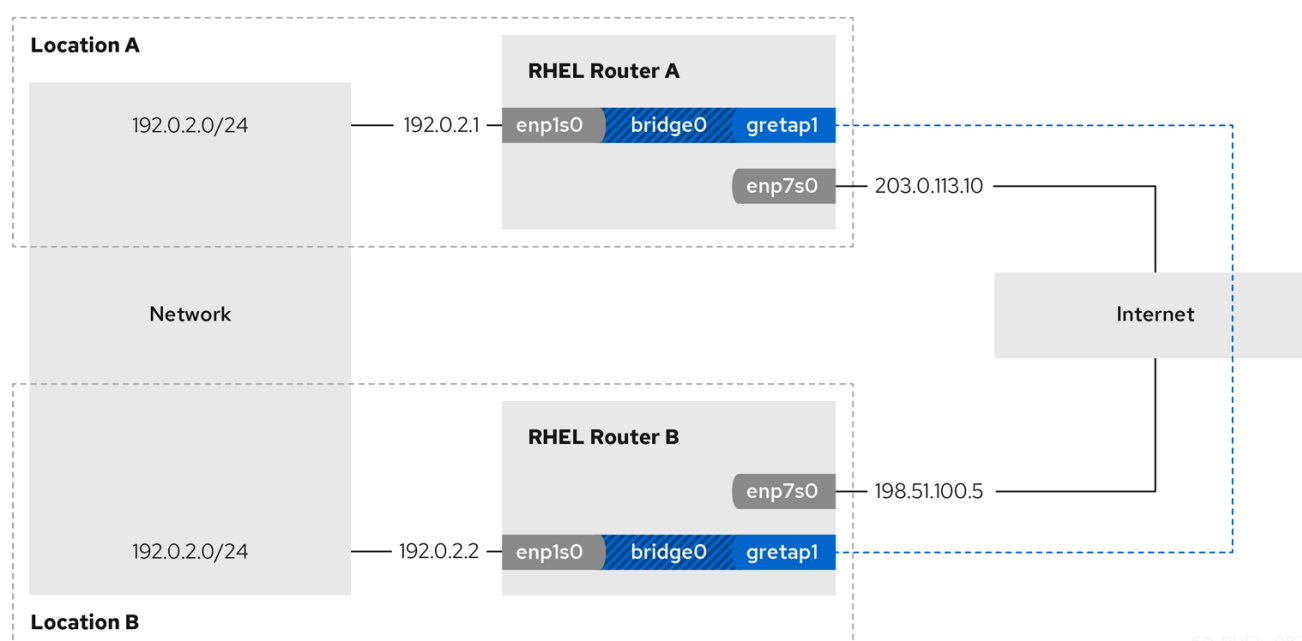
A Generic Routing Encapsulation Terminal Access Point (GRE TAP) tunnel operates on OSI level 2 and encapsulates Ethernet traffic in IPv4 packets as described in [RFC 2784](#).



IMPORTANT

Data sent through a GRE TAP tunnel is not encrypted. For security reasons, establish the tunnel over a VPN or a different encrypted connection.

For example, you can create a GRE TAP tunnel between two RHEL routers to connect two networks using a bridge as shown in the following diagram:



60_RHEL_0820

Prerequisites

- Each RHEL router has a network interface that is connected to its local network, and the interface has no IP configuration assigned.
- Each RHEL router has a network interface that is connected to the internet.

Procedure

1. On the RHEL router in network A:

- a. Create a bridge interface named **bridge0**:

```
# nmcli connection add type bridge con-name bridge0 ifname bridge0
```

- b. Configure the IP settings of the bridge:

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify bridge0 ipv4.method manual
```

- c. Add a new connection profile for the interface that is connected to local network to the bridge:

```
# nmcli connection add type ethernet port-type bridge con-name bridge0-port1
ifname enp1s0 controller bridge0
```

- d. Add a new connection profile for the GRE-TAP tunnel interface to the bridge:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap port-type bridge con-
name bridge0-port2 ifname gretap1 remote 198.51.100.5 local 203.0.113.10
controller bridge0
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.



IMPORTANT

The **gretap0** device name is reserved. Use **gretap1** or a different name for the device.

- e. Optional: Disable the Spanning Tree Protocol (STP) if you do not need it:

```
# nmcli connection modify bridge0 bridge.stp no
```

By default, STP is enabled and causes a delay before you can use the connection.

- f. Configure that activating the **bridge0** connection automatically activates the ports of the bridge:

```
# nmcli connection modify bridge0 connection.autoconnect-ports 1
```

- g. Active the **bridge0** connection:

```
# nmcli connection up bridge0
```

2. On the RHEL router in network B:

- a. Create a bridge interface named
- bridge0**
- :

```
# nmcli connection add type bridge con-name bridge0 ifname bridge0
```

- b. Configure the IP settings of the bridge:

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.2/24'
# nmcli connection modify bridge0 ipv4.method manual
```

- c. Add a new connection profile for the interface that is connected to local network to the bridge:

```
# nmcli connection add type ethernet port-type bridge con-name bridge0-port1
ifname enp1s0 controller bridge0
```

- d. Add a new connection profile for the GRE-TAP tunnel interface to the bridge:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap port-type bridge con-
name bridge0-port2 ifname gretap1 remote 203.0.113.10 local 198.51.100.5
controller bridge0
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

- e. Optional: Disable the Spanning Tree Protocol (STP) if you do not need it:

```
# nmcli connection modify bridge0 bridge.stp no
```

- f. Configure that activating the
- bridge0**
- connection automatically activates the ports of the bridge:

```
# nmcli connection modify bridge0 connection.autoconnect-ports 1
```

- g. Active the
- bridge0**
- connection:

```
# nmcli connection up bridge0
```

Verification

1. On both routers, verify that the
- enp1s0**
- and
- gretap1**
- connections are connected and that the
- CONNECTION**
- column displays the connection name of the port:

```
# nmcli device
nmcli device
DEVICE  TYPE    STATE    CONNECTION
...
bridge0 bridge  connected bridge0
enp1s0  ethernet connected bridge0-port1
gretap1 iptunnel connected bridge0-port2
```

2. From each RHEL router, ping the IP address of the internal interface of the other router:

- a. On Router A, ping **192.0.2.2**:

```
# ping 192.0.2.2
```

- b. On Router B, ping **192.0.2.1**:

```
# ping 192.0.2.1
```

CHAPTER 9. MANAGING WIFI CONNECTIONS

RHEL provides multiple utilities and applications to configure and connect to wifi networks, for example:

- Use the **nmcli** utility to configure connections by using the command line.
- Use the **nmtui** application to configure connections in a text-based user interface.
- Use the **GNOME Settings** application to configure connections by using the GNOME application.
- Use the **network** RHEL system role to automate the configuration of connections on one or multiple hosts.

9.1. SUPPORTED WIFI SECURITY TYPES

Depending on the security type a wifi network supports, you can transmit data more or less securely.



WARNING

Do not connect to wifi networks that do not use encryption or which support only the insecure WPA standard.

Red Hat Enterprise Linux supports the following wifi security types:

- **None:** Encryption is disabled, and data is transferred in plain text over the network.
- **Enhanced Open:** With opportunistic wireless encryption (OWE), devices negotiate unique pairwise master keys (PMK) to encrypt connections in wireless networks without authentication.
- **LEAP:** The Lightweight Extensible Authentication Protocol, which was developed by Cisco, is a proprietary version of the extensible authentication protocol (EAP).
- **WPA & WPA2 Personal:** In personal mode, the Wi-Fi Protected Access (WPA) and Wi-Fi Protected Access 2 (WPA2) authentication methods use a pre-shared key.
- **WPA & WPA2 Enterprise:** In enterprise mode, WPA and WPA2 use the EAP framework and authenticate users to a remote authentication dial-in user service (RADIUS) server.
- **WPA3 Personal:** Wi-Fi Protected Access 3 (WPA3) Personal uses simultaneous authentication of equals (SAE) instead of pre-shared keys (PSK) to prevent dictionary attacks. WPA3 uses perfect forward secrecy (PFS).

9.2. CONNECTING TO A WIFI NETWORK BY USING **nmcli**

You can use the **nmcli** utility to connect to a wifi network. When you attempt to connect to a network for the first time, the utility automatically creates a NetworkManager connection profile for it. If the network requires additional settings, such as static IP addresses, you can then modify the profile after it has been automatically created.

Prerequisites

- A wifi device is installed on the host.
- The wifi device is enabled. To verify, use the **nmcli radio** command.



Procedure

1. If the wifi radio has been disabled in NetworkManager, enable this feature:

```
# nmcli radio wifi on
```

2. Optional: Display the available wifi networks:

```
# nmcli device wifi list
```

IN-USE	BSSID	SSID	MODE	CHAN	RATE	SIGNAL	BARS	SECURITY
	00:53:00:2F:3B:08	Office	Infra	44	270 Mbit/s	57		WPA2 WPA3
	00:53:00:15:03:BF	--	Infra	1	130 Mbit/s	48		WPA2 WPA3

The service set identifier (**SSID**) column contains the names of the networks. If the column shows **--**, the access point of this network does not broadcast an SSID.

3. Connect to the wifi network:

```
# nmcli device wifi connect Office --ask
Password: wifi-password
```

If you prefer to set the password in the command instead of entering it interactively, use the **password <wifi_password>** option in the command instead of **--ask**:

```
# nmcli device wifi connect Office password <wifi_password>
```

Note that, if the network requires static IP addresses, NetworkManager fails to activate the connection at this point. You can configure the IP addresses in later steps.

4. If the network requires static IP addresses:
 - a. Configure the IPv4 address settings, for example:

```
# nmcli connection modify Office ipv4.method manual ipv4.addresses 192.0.2.1/24
ipv4.gateway 192.0.2.254 ipv4.dns 192.0.2.200 ipv4.dns-search example.com
```

- b. Configure the IPv6 address settings, for example:

```
# nmcli connection modify Office ipv6.method manual ipv6.addresses
2001:db8:1::1/64 ipv6.gateway 2001:db8:1::fffe ipv6.dns 2001:db8:1::ffbb ipv6.dns-
search example.com
```

5. Re-activate the connection:

```
# nmcli connection up Office
```

Verification

1. Display the active connections:

```
# nmcli connection show --active
NAME    ID                                     TYPE DEVICE
Office  2501eb7e-7b16-4dc6-97ef-7cc460139a58 wifi wlp0s20f3
```

If the output lists the wifi connection you have created, the connection is active.

2. Ping a hostname or IP address:

```
# *ping -c 3 example.com
```

Additional resources

- **nm-settings-nmcli(5)** man page on your system

9.3. CONNECTING TO A WIFI NETWORK BY USING THE GNOME SETTINGS APPLICATION

You can use the **GNOME settings** application, also named **gnome-control-center**, to connect to a wifi network and configure the connection. When you connect to the network for the first time, GNOME creates a NetworkManager connection profile for it.

In **GNOME settings**, you can configure wifi connections for all wifi network security types that RHEL supports.

Prerequisites

- A wifi device is installed on the host.
- The wifi device is enabled. To verify, use the **nmcli radio** command.

Procedure

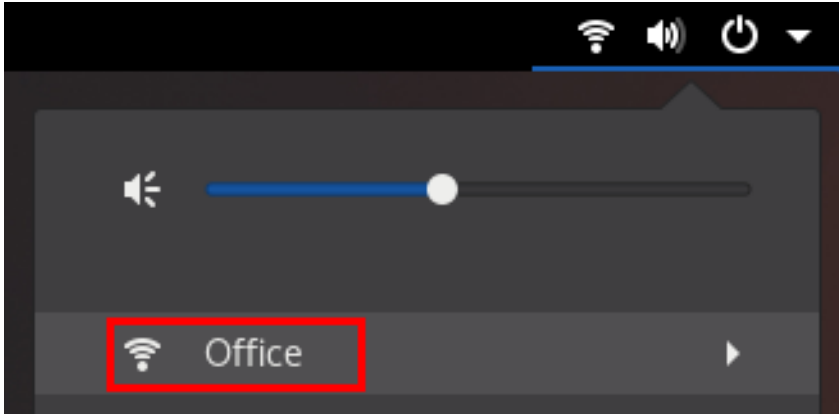
1. Press the **Super** key, type **Wi-Fi**, and press **Enter**.
2. Click on the name of the wifi network you want to connect to.
3. Enter the password for the network, and click **Connect**.
4. If the network requires additional settings, such as static IP addresses or a security type other than WPA2 Personal:
 - a. Click the gear icon next to the network's name.
 - b. Optional: Configure the network profile on the **Details** tab to not automatically connect. If you deactivate this feature, you must always manually connect to the network, for example, by using **GNOME settings** or the GNOME system menu.
 - c. Configure IPv4 settings on the **IPv4** tab, and IPv6 settings on the **IPv6** tab.
 - d. On the **Security** tab, select the authentication of the network, such as **WPA3 Personal**, and enter the password.

Depending on the selected security, the application shows additional fields. Fill them accordingly. For details, ask the administrator of the wifi network.

- e. Click **Apply**.

Verification

1. Open the system menu on the right side of the top bar, and verify that the wifi network is connected:



If the network appears in the list, it is connected.

2. Ping a hostname or IP address:

```
# ping -c 3 example.com
```

9.4. CONFIGURING A WIFI CONNECTION BY USING NMTUI

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to connect to a wifi network.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Procedure

1. Start **nmtui**:

```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.
3. Press the **Add** button.

4. Select **Wi-Fi** from the list of network types, and press **Enter**.
5. Optional: Enter a name for the NetworkManager profile to be created.
On hosts with multiple profiles, a meaningful name makes it easier to identify the purpose of a profile.
6. Enter the name of the Wi-Fi network, the Service Set Identifier (SSID), into the **SSID** field.
7. Leave the **Mode** field set to its default, **Client**.
8. Select the **Security** field, press **Enter**, and set the authentication type of the network from the list.
Depending on the authentication type you have selected, **nmtui** displays different fields.
9. Fill the authentication type-related fields.
10. If the Wi-Fi network requires static IP addresses:
 - a. Press the **Automatic** button next to the protocol, and select **Manual** from the displayed list.
 - b. Press the **Show** button next to the protocol you want to configure to display additional fields, and fill them.
11. Press the **OK** button to create and automatically activate the new connection.

The screenshot shows the 'Edit Connection' window in the nmtui application. The window has a title bar with 'Edit Connection' in red. The main content area is divided into sections. At the top, there are fields for 'Profile name' (Example-Connection) and 'Device' (wlp2s0). Below this is a section for 'WI-FI' with a '<Hide>' button. Inside the 'WI-FI' section, there are fields for 'SSID' (Example-Wi-Fi), 'Mode' (<Client>), 'Security' (<WPA3 Personal>), 'Password' (masked with asterisks), and a checkbox for 'Show password'. Below the 'WI-FI' section are fields for 'BSSID', 'Cloned MAC address', and 'MTU' (default). At the bottom, there are sections for 'IPv4 CONFIGURATION' and 'IPv6 CONFIGURATION', both set to '<Automatic>' with '<Show>' buttons. There are also checkboxes for 'Automatically connect' and 'Available to all users', both of which are checked. At the bottom right, there are '<Cancel>' and '<OK>' buttons.

12. Press the **Back** button to return to the main menu.
13. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

1. Open nmtui:

```
# nmcli connection show --active
NAME    ID                                     TYPE DEVICE
Office  2501eb7e-7b16-4dc6-97ef-7cc460139a58 wifi wlp0s20f3
```

If the output lists the wifi connection you have created, the connection is active.

2. Ping a hostname or IP address:

```
# ping -c 3 example.com
```

9.5. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION IN AN EXISTING PROFILE BY USING NMCLI

By using the **nmcli** utility, you can configure the client to authenticate itself to the network. For example, you can configure Protected Extensible Authentication Protocol (PEAP) authentication with the Microsoft Challenge-Handshake Authentication Protocol version 2 (MSCHAPv2) in an existing NetworkManager wifi connection profile named **wlp1s0**.

Prerequisites

- The network must have 802.1X network authentication.
- The wifi connection profile exists in NetworkManager and has a valid IP configuration.
- If the client is required to verify the certificate of the authenticator, the Certificate Authority (CA) certificate must be stored in the **/etc/pki/ca-trust/source/anchors/** directory.
- The **wpa_supplicant** package is installed.

Procedure

1. Set the wifi security mode to **wpa-eap**, the Extensible Authentication Protocol (EAP) to **peap**, the inner authentication protocol to **mschapv2**, and the user name:

```
# nmcli connection modify wlp1s0 wireless-security.key-mgmt wpa-eap 802-1x.eap
peap 802-1x.phase2-auth mschapv2 802-1x.identity <user_name>
```

Note that you must set the **wireless-security.key-mgmt**, **802-1x.eap**, **802-1x.phase2-auth**, and **802-1x.identity** parameters in a single command.

2. Optional: Store the password in the configuration:

```
# nmcli connection modify wlp1s0 802-1x.password <password>
```



IMPORTANT

By default, NetworkManager stores the password in plain text in the `/etc/sysconfig/network-scripts/keys-connection_name` file, which is readable only by the **root** user. However, plain text passwords in a configuration file can be a security risk.

To increase the security, set the **802-1x.password-flags** parameter to **agent-owned**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services, after you unlock the keyring first. In other cases, NetworkManager prompts for the password.

3. If the client needs to verify the certificate of the authenticator, set the **802-1x.ca-cert** parameter in the connection profile to the path of the CA certificate:

```
# nmcli connection modify wlp1s0 802-1x.ca-cert /etc/pki/ca-trust/source/anchors/ca.crt
```



NOTE

For security reasons, clients should validate the certificate of the authenticator.

4. Activate the connection profile:

```
# nmcli connection up wlp1s0
```

Verification

- Access resources on the network that require network authentication.

Additional resources

- [Managing wifi connections](#)
- **nm-settings(5)** and **nmcli(1)** man pages on your system

9.6. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE

Network Access Control (NAC) protects a network from unauthorized clients. You can specify the details that are required for the authentication in NetworkManager connection profiles to enable clients to access the network. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use an Ansible playbook to copy a private key, a certificate, and the CA certificate to the client, and then use the **network** RHEL system role to configure a connection profile with 802.1X network authentication.

Prerequisites

- [You have prepared the control node and the managed nodes](#)

- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The network supports 802.1X network authentication.
- You installed the **wpa_supplicant** package on the managed node.
- DHCP is available in the network of the managed node.
- The following files required for TLS authentication exist on the control node:
 - The client key is stored in the **/srv/data/client.key** file.
 - The client certificate is stored in the **/srv/data/client.crt** file.
 - The CA certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. Store your sensitive variables in an encrypted file:

- a. Create the vault:

```
$ ansible-vault create vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configure a wifi connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400

    - name: Copy client certificate for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"
```

```

- name: Wifi connection profile with dynamic IP address settings and 802.1X
  ansible.builtin.import_role:
    name: rhel-system-roles.network
  vars:
    network_connections:
      - name: Wifi connection profile with dynamic IP address settings and 802.1X
        interface_name: wlp1s0
        state: up
        type: wireless
        autoconnect: yes
        ip:
          dhcp4: true
          auto6: true
        wireless:
          ssid: "Example-wifi"
          key_mgmt: "wpa-eap"
        ieee802_1x:
          identity: <user_name>
          eap: tls
          private_key: "/etc/pki/tls/client.key"
          private_key_password: "{{ pwd }}"
          private_key_password_flags: none
          client_cert: "/etc/pki/tls/client.pem"
          ca_cert: "/etc/pki/tls/cacert.pem"
          domain_suffix_match: "example.com"

```

The settings specified in the example playbook include the following:

ieee802_1x

This variable contains the 802.1X-related settings.

eap: tls

Configures the profile to use the certificate-based **TLS** authentication method for the Extensible Authentication Protocol (EAP).

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

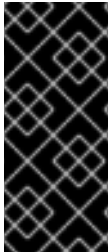
Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory

9.7. MANUALLY SETTING THE WIRELESS REGULATORY DOMAIN

On RHEL, a **udev** rule executes the **setregdomain** utility to set the wireless regulatory domain. The utility then provides this information to the kernel.

By default, **setregdomain** attempts to determine the country code automatically. If this fails, the wireless regulatory domain setting might be wrong. To work around this problem, you can manually set the country code.



IMPORTANT

Manually setting the regulatory domain disables the automatic detection. Therefore, if you later use the computer in a different country, the previously configured setting might no longer be correct. In this case, remove the **/etc/sysconfig/regdomain** file to switch back to automatic detection or use this procedure to manually update the regulatory domain setting again.

Procedure

1. Optional: Display the current regulatory domain settings:

```
# iw reg get
global
country US: DFS-FCC
...
```

2. Create the **/etc/sysconfig/regdomain** file with the following content:

```
COUNTRY=<country_code>
```

Set the **COUNTRY** variable to an ISO 3166-1 alpha2 country code, such as **DE** for Germany or **US** for the United States of America.

3. Set the regulatory domain:

```
# setregdomain
```

Verification

- Display the regulatory domain settings:

```
# iw reg get
global
country DE: DFS-ETSI
...
```

Additional resources

- [ISO 3166 Country Codes](#)

CHAPTER 10. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

You can use MACsec to secure the communication between two devices (point-to-point). For example, your branch office is connected over a Metro-Ethernet connection with the central office, you can configure MACsec on the two hosts that connect the offices to increase the security.

10.1. HOW MACSEC INCREASES SECURITY

Media Access Control security (MACsec) is a layer-2 protocol that secures different traffic types over the Ethernet links, including:

- Dynamic host configuration protocol (DHCP)
- address resolution protocol (ARP)
- IPv4 and IPv6 traffic
- Any traffic over IP such as TCP or UDP

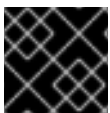
MACsec encrypts and authenticates all traffic in LANs, by default with the GCM-AES-128 algorithm, and uses a pre-shared key to establish the connection between the participant hosts. To change the pre-shared key, you must update the NM configuration on all network hosts that use MACsec.

A MACsec connection uses an Ethernet device, such as an Ethernet network card, VLAN, or tunnel device, as a parent. You can either set an IP configuration only on the MACsec device to communicate with other hosts only by using the encrypted connection, or you can also set an IP configuration on the parent device. In the latter case, you can use the parent device to communicate with other hosts using an unencrypted connection and the MACsec device for encrypted connections.

MACsec does not require any special hardware. For example, you can use any switch, except if you want to encrypt traffic only between a host and a switch. In this scenario, the switch must also support MACsec.

In other words, you can configure MACsec for two common scenarios:

- Host-to-host
- Host-to-switch and switch-to-other-hosts



IMPORTANT

You can use MACsec only between hosts being in the same physical or virtual LAN.

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

10.2. CONFIGURING A MACSEC CONNECTION BY USING **NMCLI**

You can use the **nmcli** utility to configure Ethernet interfaces to use MACsec. For example, you can create a MACsec connection between two hosts that are connected over Ethernet.

Procedure

1. On the first host on which you configure MACsec:

- Create the connectivity association key (CAK) and connectivity-association key name (CKN) for the pre-shared key:
 - a. Create a 16-byte hexadecimal CAK:

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

- b. Create a 32-byte hexadecimal CKN:

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. On both hosts you want to connect over a MACsec connection:

3. Create the MACsec connection:

```
# nmcli connection add type macsec con-name macsec0 ifname macsec0
connection.autoconnect yes macsec.parent enp1s0 macsec.mode psk macsec.mka-
cak 50b71a8ef0bd5751ea76de6d6c98c03a macsec.mka-ckn
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

Use the CAK and CKN generated in the previous step in the **macsec.mka-cak** and **macsec.mka-ckn** parameters. The values must be the same on every host in the MACsec-protected network.

4. Configure the IP settings on the MACsec connection.

- a. Configure the **IPv4** settings. For example, to set a static **IPv4** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

```
# nmcli connection modify macsec0 ipv4.method manual ipv4.addresses
'192.0.2.1/24' ipv4.gateway '192.0.2.254' ipv4.dns '192.0.2.253'
```

- b. Configure the **IPv6** settings. For example, to set a static **IPv6** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

```
# nmcli connection modify macsec0 ipv6.method manual ipv6.addresses
'2001:db8:1::1/32' ipv6.gateway '2001:db8:1::fffe' ipv6.dns '2001:db8:1::fffd'
```

5. Activate the connection:

```
# nmcli connection up macsec0
```

Verification

1. Verify that the traffic is encrypted:

```
# tcpdump -nn -i enp1s0
```

2. Optional: Display the unencrypted traffic:

```
# tcpdump -nn -i macsec0
```

3. Display MACsec statistics:

```
# ip macsec show
```

4. Display individual counters for each type of protection: integrity-only (encrypt off) and encryption (encrypt on)

```
# ip -s macsec show
```

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

10.3. CONFIGURING A MACSEC CONNECTION BY USING NMSTATECTL

You can configure Ethernet interfaces to use MACsec through the **nmstatectl** utility in a declarative way. For example, in a YAML file, you describe the desired state of your network, which is supposed to have a MACsec connection between two hosts connected over Ethernet. The **nmstatectl** utility interprets the YAML file and deploys persistent and consistent network configuration across the hosts.

Using the MACsec security standard for securing communication at the link layer, also known as layer 2 of the Open Systems Interconnection (OSI) model provides the following notable benefits:

- Encryption at layer 2 eliminates the need for encrypting individual services at layer 7. This reduces the overhead associated with managing a large number of certificates for each endpoint on each host.
- Point-to-point security between directly connected network devices such as routers and switches.
- No changes needed for applications and higher-layer protocols.

Prerequisites

- A physical or virtual Ethernet Network Interface Controller (NIC) exists in the server configuration.
- The **nmstate** package is installed.

Procedure

1. On the first host on which you configure MACsec, create the connectivity association key (CAK) and connectivity-association key name (CKN) for the pre-shared key:
 - a. Create a 16-byte hexadecimal CAK:

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

- b. Create a 32-byte hexadecimal CKN:

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. On both hosts that you want to connect over a MACsec connection, complete the following steps:

- a. Create a YAML file, for example **create-macsec-connection.yml**, with the following settings:

```
---
routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-interface: macsec0
      next-hop-address: 192.0.2.2
      table-id: 254
    - destination: 192.0.2.2/32
      next-hop-interface: macsec0
      next-hop-address: 0.0.0.0
      table-id: 254
  dns-resolver:
    config:
      search:
        - example.com
      server:
        - 192.0.2.200
        - 2001:db8:1::ffbb
  interfaces:
    - name: macsec0
      type: macsec
      state: up
      ipv4:
        enabled: true
        address:
          - ip: 192.0.2.1
            prefix-length: 32
      ipv6:
        enabled: true
        address:
          - ip: 2001:db8:1::1
            prefix-length: 64
      macsec:
        encrypt: true
        base-iface: enp0s1
        mka-cak: 50b71a8ef0bd5751ea76de6d6c98c03a
        mka-ckn: f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
        port: 0
        validation: strict
        send-sci: true
```

- b. Use the CAK and CKN generated in the previous step in the **mka-cak** and **mka-ckn** parameters. The values must be the same on every host in the MACsec-protected network.
- c. Optional: In the same YAML configuration file, you can also configure the following settings:
 - A static IPv4 address - **192.0.2.1** with the **/32** subnet mask

- A static IPv6 address - **2001:db8:1::1** with the **/64** subnet mask
- An IPv4 default gateway - **192.0.2.2**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

3. Apply the settings to the system:

```
# nmstatectl apply create-macsec-connection.yml
```

Verification

1. Display the current state in YAML format:

```
# nmstatectl show macsec0
```

2. Verify that the traffic is encrypted:

```
# tcpdump -nn -i enp0s1
```

3. Optional: Display the unencrypted traffic:

```
# tcpdump -nn -i macsec0
```

4. Display MACsec statistics:

```
# ip macsec show
```

5. Display individual counters for each type of protection: integrity-only (encrypt off) and encryption (encrypt on)

```
# ip -s macsec show
```

Additional resources

- [MACsec: a different solution to encrypt network traffic](#)

CHAPTER 11. CONFIGURING NETWORKMANAGER TO IGNORE CERTAIN DEVICES

By default, NetworkManager manages all devices. To ignore certain devices, you can configure them in NetworkManager as **unmanaged**.

11.1. PERMANENTLY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER

You can permanently configure devices as **unmanaged** based on several criteria, such as the interface name, MAC address, or device type.

Procedure

1. Optional: Display the list of devices to identify the device or MAC address you want to set as **unmanaged**:

```
# ip link show
...
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. Create the **/etc/NetworkManager/conf.d/99-unmanaged-devices.conf** file with the following content:

- To configure a specific interface as unmanaged, add:

```
[keyfile]
unmanaged-devices=interface-name:enp1s0
```

- To configure a device with a specific MAC address as unmanaged, add:

```
[keyfile]
unmanaged-devices=mac:52:54:00:74:79:56
```

- To configure all devices of a specific type as unmanaged, add:

```
[keyfile]
unmanaged-devices=type:ethernet
```

- To set multiple devices as unmanaged, separate the entries in the **unmanaged-devices** parameter with a semicolon, for example:

```
[keyfile]
unmanaged-devices=interface-name:enp1s0;interface-name:enp7s0
```

3. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

Verification

- Display the list of devices:

```
# nmcli device status
DEVICE TYPE   STATE   CONNECTION
enp1s0 ethernet unmanaged --
...
```

The **unmanaged** state next to the **enp1s0** device indicates that NetworkManager does not manage this device.

Troubleshooting

- If the device is not shown as **unmanaged**, display the NetworkManager configuration:

```
# NetworkManager --print-config
...
[keyfile]
unmanaged-devices=interface-name:enp1s0
...
```

If the output does not match the settings that you configured, ensure that no configuration file with a higher priority overrides your settings. For details about how NetworkManager merges multiple configuration files, see the **NetworkManager.conf(5)** man page.

11.2. TEMPORARILY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER

You can temporarily configure devices as **unmanaged**, for example, for testing purposes.

Procedure

1. Optional: Display the list of devices to identify the device you want to set as **unmanaged**:

```
# nmcli device status
DEVICE TYPE   STATE   CONNECTION
enp1s0 ethernet disconnected --
...
```

2. Set the **enp1s0** device to the **unmanaged** state:

```
# nmcli device set enp1s0 managed no
```

Verification

- Display the list of devices:

```
# nmcli device status
DEVICE TYPE   STATE   CONNECTION
enp1s0 ethernet unmanaged --
...
```

The **unmanaged** state next to the **enp1s0** device indicates that NetworkManager does not manage this device.

CHAPTER 12. MANAGING THE DEFAULT GATEWAY

The default gateway is a router that forwards network packets when no other route matches the destination of a packet. In a local network, the default gateway is typically the host that is one hop closer to the internet.

12.1. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING `nmcli`

In most situations, administrators set the default gateway when they create a connection. However, you can also set or update the default gateway setting on a previously created connection by using the `nmcli` utility.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, the user must have **root** permissions.

Procedure

1. Set the IP addresses of the default gateway:
To set the IPv4 default gateway, enter:

```
# nmcli connection modify <connection_name> ipv4.gateway  
"<IPv4_gateway_address>"
```

To set the IPv6 default gateway, enter:

```
# nmcli connection modify <connection_name> ipv6.gateway  
"<IPv6_gateway_address>"
```

2. Restart the network connection for changes to take effect:

```
# nmcli connection up <connection_name>
```



WARNING

All connections currently using this network connection are temporarily interrupted during the restart.

Verification

- Verify that the route is active:
 - a. To display the IPv4 default gateway, enter:

ip -4 route

```
default via 192.0.2.1 dev example proto static metric 100
```

- b. To display the IPv6 default gateway, enter:

ip -6 route

```
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

12.2. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING **NMSTATECTL**

In most situations, administrators set the default gateway when they create a connection. However, you can also set or update the default gateway setting on a previously created connection by using the **nmstatectl** utility.

Use the **nmstatectl** utility to set the default gateway through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- The **enp1s0** interface is configured, and the IP address of the default gateway is within the subnet of the IP configuration of this interface.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/set-default-gateway.yml**, with the following content:

```
---
routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-address: 192.0.2.1
      next-hop-interface: enp1s0
```

These settings define **192.0.2.1** as the default gateway, and the default gateway is reachable through the **enp1s0** interface.

2. Apply the settings to the system:

```
# nmstatectl apply ~/set-default-gateway.yml
```

Additional resources

- **nmstatectl(8)** man page on your system
- **/usr/share/doc/nmstate/examples/** directory

12.3. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

A host forwards a network packet to its default gateway if the packet's destination can neither be reached through the directly-connected networks nor through any of the routes configured on the host. To configure the default gateway of a host, set it in the NetworkManager connection profile of the interface that is connected to the same network as the default gateway. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

In most situations, administrators set the default gateway when they create a connection. However, you can also set or update the default gateway setting on a previously-created connection.



WARNING

You cannot use the **network** RHEL system role to update only specific values in an existing connection profile. The role ensures that a connection profile exactly matches the settings in a playbook. If a connection profile with the same name already exists, the role applies the settings from the playbook and resets all other settings in the profile to their defaults. To prevent resetting values, always specify the whole configuration of the network connection profile in the playbook, including the settings that you do not want to change.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Ethernet connection profile with static IP address settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 198.51.100.20/24
```

```

- 2001:db8:1::1/64
gateway4: 198.51.100.254
gateway6: 2001:db8:1::fffe
dns:
- 198.51.100.200
- 2001:db8:1::ffbb
dns_search:
- example.com
state: up

```

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Query the Ansible facts of the managed node and verify the active network settings:

```

# ansible managed-node-01.example.com -m ansible.builtin.setup
...
  "ansible_default_ipv4": {
    ...
    "gateway": "198.51.100.254",
    "interface": "enp1s0",
    ...
  },
  "ansible_default_ipv6": {
    ...
    "gateway": "2001:db8:1::fffe",
    "interface": "enp1s0",
    ...
  }
...

```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

12.4. HOW NETWORKMANAGER MANAGES MULTIPLE DEFAULT GATEWAYS

In certain situations, for example for fallback reasons, you set multiple default gateways on a host.

However, to avoid asynchronous routing issues, each default gateway of the same protocol requires a separate metric value. Note that RHEL only uses the connection to the default gateway that has the lowest metric set.

You can set the metric for both the IPv4 and IPv6 gateway of a connection using the following command:

```
# nmcli connection modify <connection_name> ipv4.route-metric <value> ipv6.route-metric <value>
```



IMPORTANT

Do not set the same metric value for the same protocol in multiple connection profiles to avoid routing issues.

If you set a default gateway without a metric value, NetworkManager automatically sets the metric value based on the interface type. For that, NetworkManager assigns the default value of this network type to the first connection that is activated, and sets an incremented value to each other connection of the same type in the order they are activated. For example, if two Ethernet connections with a default gateway exist, NetworkManager sets a metric of **100** on the route to the default gateway of the connection that you activate first. For the second connection, NetworkManager sets **101**.

The following is an overview of frequently-used network types and their default metrics:

Connection type	Default metric value
VPN	50
Ethernet	100
MACsec	125
InfiniBand	150
Bond	300
VLAN	400
Bridge	425
TUN	450
Wi-Fi	600
IP tunnel	675

Additional resources

- [Configuring policy-based routing to define alternative routes](#)

12.5. CONFIGURING NETWORKMANAGER TO AVOID USING A SPECIFIC PROFILE TO PROVIDE A DEFAULT GATEWAY

You can configure that NetworkManager never uses a specific profile to provide the default gateway. Follow this procedure for connection profiles that are not connected to the default gateway.

Prerequisites

- The NetworkManager connection profile for the connection that is not connected to the default gateway exists.

Procedure

1. If the connection uses a dynamic IP configuration, configure that NetworkManager does not use this connection as the default connection for the corresponding IP type, meaning that NetworkManager will never assign the default route to it:

```
# nmcli connection modify <connection_name> ipv4.never-default yes ipv6.never-default yes
```

Note that setting **ipv4.never-default** and **ipv6.never-default** to **yes**, automatically removes the default gateway's IP address for the corresponding protocol from the connection profile.

2. Activate the connection:

```
# nmcli connection up <connection_name>
```

Verification

- Use the **ip -4 route** and **ip -6 route** commands to verify that RHEL does not use the network interface for the default route for the IPv4 and IPv6 protocol.

12.6. FIXING UNEXPECTED ROUTING BEHAVIOR DUE TO MULTIPLE DEFAULT GATEWAYS

There are only a few scenarios, such as when using Multipath TCP, in which you require multiple default gateways on a host. In most cases, you configure only a single default gateway to avoid unexpected routing behavior or asynchronous routing issues.



NOTE

To route traffic to different internet providers, use policy-based routing instead of multiple default gateways.

Prerequisites

- The host uses NetworkManager to manage network connections, which is the default.
- The host has multiple network interfaces.
- The host has multiple default gateways configured.

Procedure

1. Display the routing table:

- For IPv4, enter:

```
# ip -4 route
default via 192.0.2.1 dev enp1s0 proto static metric 101
default via 198.51.100.1 dev enp7s0 proto static metric 102
...
```

- For IPv6, enter:

```
# ip -6 route
default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
default via 2001:db8:2::1 dev enp7s0 proto static metric 102 pref medium
...
```

Entries starting with **default** indicate a default route. Note the interface names of these entries displayed next to **dev**.

2. Use the following commands to display the NetworkManager connections that use the interfaces you identified in the previous step:

```
# nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp1s0
GENERAL.CONNECTION: Corporate-LAN
IP4.GATEWAY: 192.0.2.1
IP6.GATEWAY: 2001:db8:1::1

# nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp7s0
GENERAL.CONNECTION: Internet-Provider
IP4.GATEWAY: 198.51.100.1
IP6.GATEWAY: 2001:db8:2::1
```

In these examples, the profiles named **Corporate-LAN** and **Internet-Provider** have the default gateways set. Because, in a local network, the default gateway is typically the host that is one hop closer to the internet, the rest of this procedure assumes that the default gateways in the **Corporate-LAN** are incorrect.

3. Configure that NetworkManager does not use the **Corporate-LAN** connection as the default route for IPv4 and IPv6 connections:

```
# nmcli connection modify Corporate-LAN ipv4.never-default yes ipv6.never-default yes
```

Note that setting **ipv4.never-default** and **ipv6.never-default** to **yes**, automatically removes the default gateway's IP address for the corresponding protocol from the connection profile.

4. Activate the **Corporate-LAN** connection:

```
# nmcli connection up Corporate-LAN
```

Verification

- Display the IPv4 and IPv6 routing tables and verify that only one default gateway is available for each protocol:
 - For IPv4, enter:

```
# ip -4 route
default via 198.51.100.1 dev enp7s0 proto static metric 102
...
```

- For IPv6, enter:

```
# ip -6 route
default via 2001:db8:2::1 dev enp7s0 proto static metric 102 pref medium
...
```

Additional resources

- [Configuring policy-based routing to define alternative routes](#)

CHAPTER 13. CONFIGURING A STATIC ROUTE

Routing ensures that you can send and receive traffic between mutually-connected networks. In larger environments, administrators typically configure services so that routers can dynamically learn about other routers. In smaller environments, administrators often configure static routes to ensure that traffic can reach from one network to the next.

You need static routes to achieve a functioning communication among multiple networks if all of these conditions apply:

- The traffic has to pass multiple networks.
- The exclusive traffic flow through the default gateways is not sufficient.

The [Example of a network that requires static routes](#) section describes scenarios and how the traffic flows between different networks when you do not configure static routes.

13.1. EXAMPLE OF A NETWORK THAT REQUIRES STATIC ROUTES

You require static routes in this example because not all IP networks are directly connected through one router. Without the static routes, some networks cannot communicate with each other. Additionally, traffic from some networks flows only in one direction.



NOTE

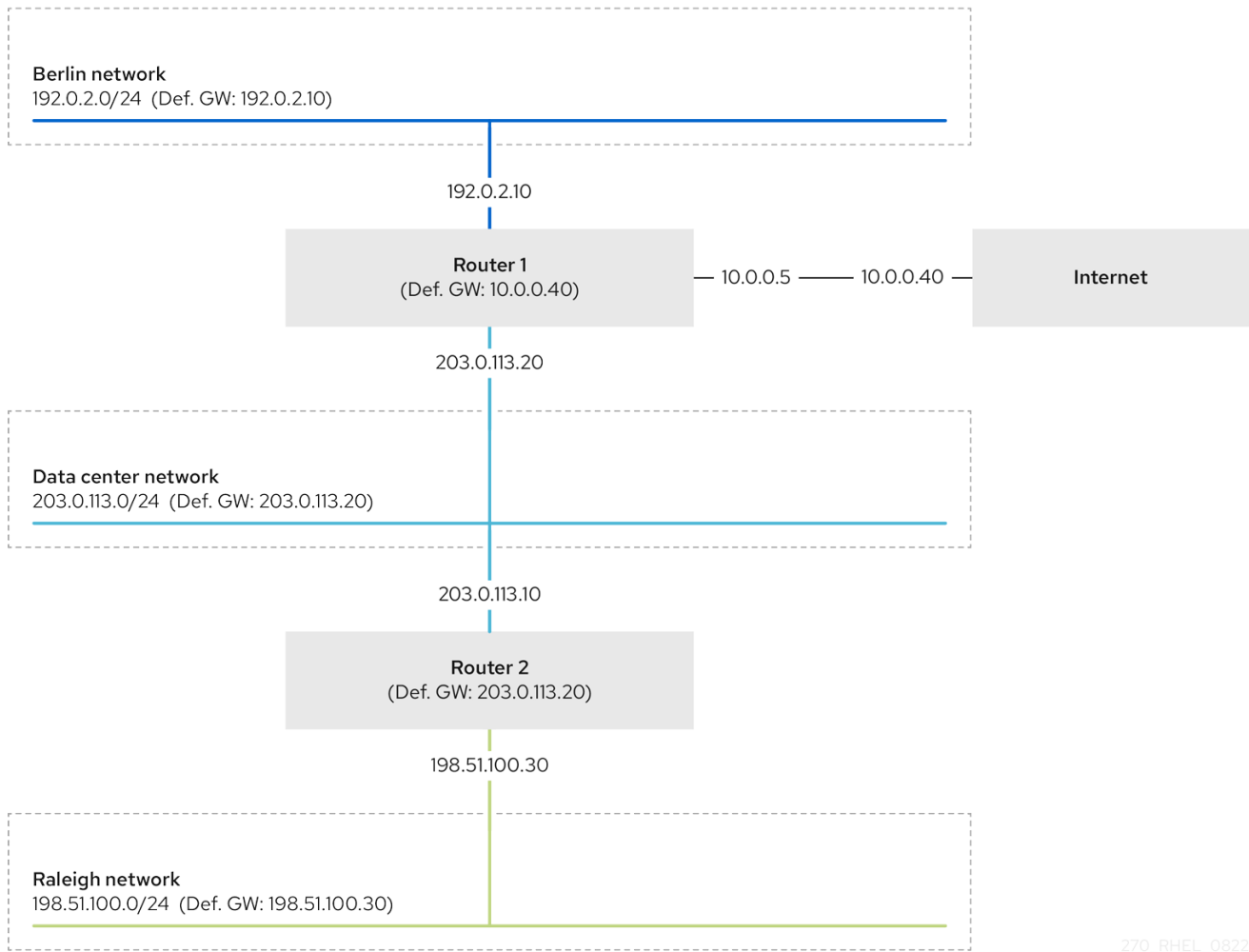
The network topology in this example is artificial and only used to explain the concept of static routing. It is not a recommended topology in production environments.

For a functioning communication among all networks in this example, configure a static route to Raleigh (**198.51.100.0/24**) with next the hop Router 2 (**203.0.113.10**). The IP address of the next hop is the one of Router 2 in the data center network (**203.0.113.0/24**).

You can configure the static route as follows:

- For a simplified configuration, set this static route only on Router 1. However, this increases the traffic on Router 1 because hosts from the data center (**203.0.113.0/24**) send traffic to Raleigh (**198.51.100.0/24**) always through Router 1 to Router 2.
- For a more complex configuration, configure this static route on all hosts in the data center (**203.0.113.0/24**). All hosts in this subnet then send traffic directly to Router 2 (**203.0.113.10**) that is closer to Raleigh (**198.51.100.0/24**).

For more details between which networks traffic flows or not, see the explanations below the diagram.



270_RHEL_0822

In case that the required static routes are not configured the following are the situations in which the communication works and when it does not:

- Hosts in the Berlin network (**192.0.2.0/24**):
 - Can communicate with other hosts in the same subnet because they are directly connected.
 - Can communicate with the internet because Router 1 is in the Berlin network (**192.0.2.0/24**) and has a default gateway, which leads to the internet.
 - Can communicate with the data center network (**203.0.113.0/24**) because Router 1 has interfaces in both the Berlin (**192.0.2.0/24**) and the data center (**203.0.113.0/24**) networks.
 - Cannot communicate with the Raleigh network (**198.51.100.0/24**) because Router 1 has no interface in this network. Therefore, Router 1 sends the traffic to its own default gateway (internet).
- Hosts in the data center network (**203.0.113.0/24**):
 - Can communicate with other hosts in the same subnet because they are directly connected.
 - Can communicate with the internet because they have their default gateway set to Router 1, and Router 1 has interfaces in both networks, the data center (**203.0.113.0/24**) and to the internet.

- Can communicate with the Berlin network (**192.0.2.0/24**) because they have their default gateway set to Router 1, and Router 1 has interfaces in both the data center (**203.0.113.0/24**) and the Berlin (**192.0.2.0/24**) networks.
- Cannot communicate with the Raleigh network (**198.51.100.0/24**) because the data center network has no interface in this network. Therefore, hosts in the data center (**203.0.113.0/24**) send traffic to their default gateway (Router 1). Router 1 also has no interface in the Raleigh network (**198.51.100.0/24**) and, as a result, Router 1 sends this traffic to its own default gateway (internet).
- Hosts in the Raleigh network (**198.51.100.0/24**):
 - Can communicate with other hosts in the same subnet because they are directly connected.
 - Cannot communicate with hosts on the internet. Router 2 sends the traffic to Router 1 because of the default gateway settings. The actual behavior of Router 1 depends on the reverse path filter (**rp_filter**) system control (**sysctl**) setting. By default on RHEL, Router 1 drops the outgoing traffic instead of routing it to the internet. However, regardless of the configured behavior, communication is not possible without the static route.
 - Cannot communicate with the data center network (**203.0.113.0/24**). The outgoing traffic reaches the destination through Router 2 because of the default gateway setting. However, replies to packets do not reach the sender because hosts in the data center network (**203.0.113.0/24**) send replies to their default gateway (Router 1). Router 1 then sends the traffic to the internet.
 - Cannot communicate with the Berlin network (**192.0.2.0/24**). Router 2 sends the traffic to Router 1 because of the default gateway settings. The actual behavior of Router 1 depends on the **sysctl** setting **rp_filter**. By default on RHEL, Router 1 drops the outgoing traffic instead of sending it to the Berlin network (**192.0.2.0/24**). However, regardless of the configured behavior, communication is not possible without the static route.



NOTE

In addition to configuring the static routes, you must enable IP forwarding on both routers.

Additional resources

- [Why cannot a server be pinged if net.ipv4.conf.all.rp_filter is set on the server?](#) (Red Hat Knowledgebase)
- [Enabling IP forwarding](#) (Red Hat Knowledgebase)

13.2. HOW TO USE THE **nmcli** UTILITY TO CONFIGURE A STATIC ROUTE

To configure a static route, use the **nmcli** utility with the following syntax:

```
$ nmcli connection modify connection_name ipv4.routes "ip[/prefix] [next_hop] [metric] [attribute=value] [attribute=value] ..."
```

The command supports the following route attributes:

- **cwnd=*n***: Sets the congestion window (CWND) size, defined in the number of packets.

- **lock-cwnd=true|false**: Defines whether or not the kernel can update the Cwnd value.
- **lock-mtu=true|false**: Defines whether or not the kernel can update the MTU to path MTU discovery.
- **lock-window=true|false**: Defines whether or not the kernel can update the maximum window size for TCP packets.
- **mtu=<mtu_value>**: Sets the maximum transfer unit (MTU) to use along the path to the destination.
- **onlink=true|false**: Defines whether the next hop is directly attached to this link even if it does not match any interface prefix.
- **scope=<scope>**: For an IPv4 route, this attribute sets the scope of the destinations covered by the route prefix. Set the value as an integer (0-255).
- **src=<source_address>**: Sets the source address to prefer when sending traffic to the destinations covered by the route prefix.
- **table=<table_id>**: Sets the ID of the table the route should be added to. If you omit this parameter, NetworkManager uses the **main** table.
- **tos=<type_of_service_key>**: Sets the type of service (TOS) key. Set the value as an integer (0-255).
- **type=<route_type>**: Sets the route type. NetworkManager supports the **unicast**, **local**, **blackhole**, **unreachable**, **prohibit**, and **throw** route types. The default is **unicast**.
- **window=<window_size>**: Sets the maximal window size for TCP to advertise to these destinations, measured in bytes.

IMPORTANT

If you use the **ipv4.routes** option without a preceding **+** sign, **nmcli** overrides all current settings of this parameter.

- To create an additional route, enter:

```
$ nmcli connection modify connection_name +ipv4.routes "<route>"
```

- To remove a specific route, enter:

```
$ nmcli connection modify connection_name -ipv4.routes "<route>"
```

13.3. CONFIGURING A STATIC ROUTE BY USING NMCLI

You can add a static route to an existing NetworkManager connection profile using the **nmcli connection modify** command.

The procedure below configures the following routes:

- An IPv4 route to the remote **198.51.100.0/24** network. The corresponding gateway with the IP address **192.0.2.10** is reachable through the **LAN** connection profile.

- An IPv6 route to the remote **2001:db8:2::/64** network. The corresponding gateway with the IP address **2001:db8:1::10** is reachable through the **LAN** connection profile.

Prerequisites

- The **LAN** connection profile exists and it configures this host to be in the same IP subnet as the gateways.

Procedure

1. Add the static IPv4 route to the **LAN** connection profile:

```
# nmcli connection modify LAN +ipv4.routes "198.51.100.0/24 192.0.2.10"
```

To set multiple routes in one step, pass the individual routes comma-separated to the command:

```
# nmcli connection modify <connection_profile> +ipv4.routes
"<remote_network_1>/<subnet_mask_1> <gateway_1>,
<remote_network_n>/<subnet_mask_n> <gateway_n>, ..."
```

2. Add the static IPv6 route to the **LAN** connection profile:

```
# nmcli connection modify LAN +ipv6.routes "2001:db8:2::/64 2001:db8:1::10"
```

3. Re-activate the connection:

```
# nmcli connection up LAN
```

Verification

1. Display the IPv4 routes:

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. Display the IPv6 routes:

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

13.4. CONFIGURING A STATIC ROUTE BY USING **NMTUI**

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to configure static routes on a host without a graphical interface.

For example, the procedure below adds a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**, which is reachable through an existing connection profile.



NOTE

In **nmtui**:

- Navigate by using the cursor keys.
- Press a button by selecting it and hitting **Enter**.
- Select and clear checkboxes by using **Space**.
- To return to the previous screen, use **ESC**.

Prerequisites

- The network is configured.
- The gateway for the static route must be directly reachable on the interface.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, the command requires root permissions.

Procedure

1. Start **nmtui**:

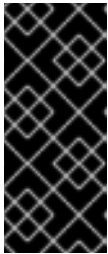
```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.
3. Select the connection profile through which you can reach the next hop to the destination network, and press **Enter**.
4. Depending on whether it is an IPv4 or IPv6 route, press the **Show** button next to the protocol's configuration area.
5. Press the **Edit** button next to **Routing**. This opens a new window where you configure static routes:
 - a. Press the **Add** button and fill in:
 - The destination network, including the prefix in Classless Inter-Domain Routing (CIDR) format
 - The IP address of the next hop
 - A metric value, if you add multiple routes to the same network and want to prioritize the routes by efficiency
 - b. Repeat the previous step for every route you want to add and that is reachable through this connection profile.
 - c. Press the **OK** button to return to the window with the connection settings.

Figure 13.1. Example of a static route without metric

Destination/Prefix	Next Hop	Metric	
192.0.2.0/24	198.51.100.1		<Remove>
<Add...>			
			<Cancel> <OK>

6. Press the OK button to return to the **nmtui** main menu.
7. Select **Activate a connection** and press **Enter**.
8. Select the connection profile that you edited, and press **Enter** twice to deactivate and activate it again.



IMPORTANT

Skip this step if you run **nmtui** over a remote connection, such as SSH, that uses the connection profile you want to reactivate. In this case, if you deactivate it in **nmtui**, the connection is terminated and, consequently, you cannot activate it again. To avoid this problem, use the **nmcli connection <connection_profile> up** command to reactivate the connection in the mentioned scenario.

9. Press the **Back** button to return to the main menu.
10. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

- Verify that the route is active:

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

13.5. CONFIGURING A STATIC ROUTE BY USING **NMSTATECTL**

Use the **nmstatectl** utility to configure a static route through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Prerequisites

- The **enp1s0** network interface is configured and is in the same IP subnet as the gateways.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/add-static-route-to-enp1s0.yml**, with the following content:

```
---
routes:
  config:
    - destination: 198.51.100.0/24
      next-hop-address: 192.0.2.10
      next-hop-interface: enp1s0
    - destination: 2001:db8:2::/64
      next-hop-address: 2001:db8:1::10
      next-hop-interface: enp1s0
```

These settings define the following static routes:

- An IPv4 route to the remote **198.51.100.0/24** network. The corresponding gateway with the IP address **192.0.2.10** is reachable through the **enp1s0** interface.
- An IPv6 route to the remote **2001:db8:2::/64** network. The corresponding gateway with the IP address **2001:db8:1::10** is reachable through the **enp1s0** interface.

2. Apply the settings to the system:

```
# nmstatectl apply ~/add-static-route-to-enp1s0.yml
```

Verification

1. Display the IPv4 routes:

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. Display the IPv6 routes:

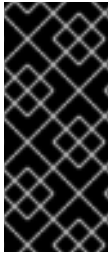
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

Additional resources

- **nmstatectl(8)** man page on your system
- **/usr/share/doc/nmstate/examples/** directory

13.6. CONFIGURING A STATIC ROUTE BY USING THE `network` RHEL SYSTEM ROLE

You can use the **network** RHEL system role to configure static routes.



IMPORTANT

When you run a play that uses the **network** RHEL system role and if the setting values do not match the values specified in the play, the role overrides the existing connection profile with the same name. To prevent resetting these values to their defaults, always specify the whole configuration of the network connection profile in the play, even if the configuration, for example the IP configuration, already exists.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP and additional routes
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp7s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            route:
              - network: 198.51.100.0
                prefix: 24
                gateway: 192.0.2.10
              - network: 2001:db8:2::
                prefix: 64
                gateway: 2001:db8:1::10
            state: up
```

Depending on whether it already exists, the procedure creates or updates the **enp7s0** connection profile with the following settings:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask

- A static IPv6 address – **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway – **192.0.2.254**
- An IPv6 default gateway – **2001:db8:1::ffe**
- An IPv4 DNS server – **192.0.2.200**
- An IPv6 DNS server – **2001:db8:1::ffbb**
- A DNS search domain – **example.com**
- Static routes:
 - **198.51.100.0/24** with gateway **192.0.2.10**
 - **2001:db8:2::/64** with gateway **2001:db8:1::10**

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. On the managed nodes:

a. Display the IPv4 routes:

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp7s0
```

b. Display the IPv6 routes:

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory

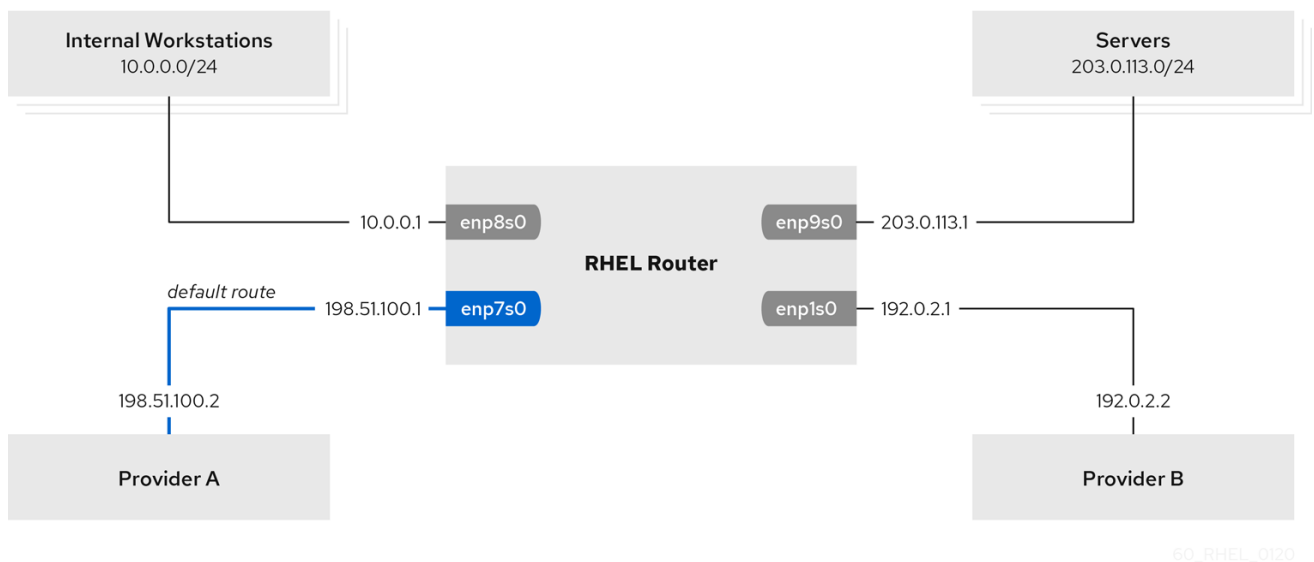
CHAPTER 14. CONFIGURING POLICY-BASED ROUTING TO DEFINE ALTERNATIVE ROUTES

By default, the kernel in RHEL decides where to forward network packets based on the destination address using a routing table. Policy-based routing enables you to configure complex routing scenarios. For example, you can route packets based on various criteria, such as the source address, packet metadata, or protocol.

14.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY BY USING NMCLI

You can use policy-based routing to configure a different default gateway for traffic from certain subnets. For example, you can configure RHEL as a router that, by default, routes all traffic to internet provider A using the default route. However, traffic received from the internal workstations subnet is routed to provider B.

The procedure assumes the following network topology:



Prerequisites

- The system uses **NetworkManager** to configure the network, which is the default.
- The RHEL router you want to set up in the procedure has four network interfaces:
 - The **enp7s0** interface is connected to the network of provider A. The gateway IP in the provider's network is **198.51.100.2**, and the network uses a **/30** network mask.
 - The **enp1s0** interface is connected to the network of provider B. The gateway IP in the provider's network is **192.0.2.2**, and the network uses a **/30** network mask.
 - The **enp8s0** interface is connected to the **10.0.0.0/24** subnet with internal workstations.
 - The **enp9s0** interface is connected to the **203.0.113.0/24** subnet with the company's servers.

- Hosts in the internal workstations subnet use **10.0.0.1** as the default gateway. In the procedure, you assign this IP address to the **enp8s0** network interface of the router.
- Hosts in the server subnet use **203.0.113.1** as the default gateway. In the procedure, you assign this IP address to the **enp9s0** network interface of the router.
- The **firewalld** service is enabled and active.

Procedure

1. Configure the network interface to provider A:

```
# nmcli connection add type ethernet con-name Provider-A ifname enp7s0
ipv4.method manual ipv4.addresses 198.51.100.1/30 ipv4.gateway 198.51.100.2
ipv4.dns 198.51.100.200 connection.zone external
```

The **nmcli connection add** command creates a NetworkManager connection profile. The command uses the following options:

- **type ethernet**: Defines that the connection type is Ethernet.
 - **con-name <connection_name>**: Sets the name of the profile. Use a meaningful name to avoid confusion.
 - **ifname <network_device>**: Sets the network interface.
 - **ipv4.method manual**: Enables to configure a static IP address.
 - **ipv4.addresses <IP_address>/<subnet_mask>**: Sets the IPv4 addresses and subnet mask.
 - **ipv4.gateway <IP_address>**: Sets the default gateway address.
 - **ipv4.dns <IP_of_DNS_server>**: Sets the IPv4 address of the DNS server.
 - **connection.zone <firewalld_zone>**: Assigns the network interface to the defined **firewalld** zone. Note that **firewalld** automatically enables masquerading for interfaces assigned to the **external** zone.
2. Configure the network interface to provider B:

```
# nmcli connection add type ethernet con-name Provider-B ifname enp1s0
ipv4.method manual ipv4.addresses 192.0.2.1/30 ipv4.routes "0.0.0.0/0 192.0.2.2
table=5000" connection.zone external
```

This command uses the **ipv4.routes** parameter instead of **ipv4.gateway** to set the default gateway. This is required to assign the default gateway for this connection to a different routing table (**5000**) than the default. NetworkManager automatically creates this new routing table when the connection is activated.

3. Configure the network interface to the internal workstations subnet:

```
# nmcli connection add type ethernet con-name Internal-Workstations ifname enp8s0
ipv4.method manual ipv4.addresses 10.0.0.1/24 ipv4.routes "10.0.0.0/24 table=5000"
ipv4.routing-rules "priority 5 from 10.0.0.0/24 table 5000" connection.zone trusted
```

This command uses the **ipv4.routes** parameter to add a static route to the routing table with ID **5000**. This static route for the **10.0.0.0/24** subnet uses the IP of the local network interface to provider B (**192.0.2.1**) as next hop.

Additionally, the command uses the **ipv4.routing-rules** parameter to add a routing rule with priority **5** that routes traffic from the **10.0.0.0/24** subnet to table **5000**. Low values have a high priority.

Note that the syntax in the **ipv4.routing-rules** parameter is the same as in an **ip rule add** command, except that **ipv4.routing-rules** always requires specifying a priority.

4. Configure the network interface to the server subnet:

```
# nmcli connection add type ethernet con-name Servers ifname enp9s0 ipv4.method
manual ipv4.addresses 203.0.113.1/24 connection.zone trusted
```

Verification

1. On a RHEL host in the internal workstation subnet:

- a. Install the **traceroute** package:

```
# dnf install traceroute
```

- b. Use the **traceroute** utility to display the route to a host on the internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1)  0.337 ms 0.260 ms 0.223 ms
 2 192.0.2.1 (192.0.2.1) 0.884 ms 1.066 ms 1.248 ms
 ...
```

The output of the command displays that the router sends packets over **192.0.2.1**, which is the network of provider B.

2. On a RHEL host in the server subnet:

- a. Install the **traceroute** package:

```
# dnf install traceroute
```

- b. Use the **traceroute** utility to display the route to a host on the internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1 203.0.113.1 (203.0.113.1) 2.179 ms 2.073 ms 1.944 ms
 2 198.51.100.2 (198.51.100.2) 1.868 ms 1.798 ms 1.549 ms
 ...
```

The output of the command displays that the router sends packets over **198.51.100.2**, which is the network of provider A.

Troubleshooting steps

On the RHEL router:

1. Display the rule list:

```
# ip rule list
0: from all lookup local
5: from 10.0.0.0/24 lookup 5000
32766: from all lookup main
32767: from all lookup default
```

By default, RHEL contains rules for the tables **local**, **main**, and **default**.

2. Display the routes in table **5000**:

```
# ip route list table 5000
0.0.0.0/0 via 192.0.2.2 dev enp1s0 proto static metric 100
10.0.0.0/24 dev enp8s0 proto static scope link src 192.0.2.1 metric 102
```

3. Display the interfaces and firewall zones:

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

4. Verify that the **external** zone has masquerading enabled:

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

Additional resources

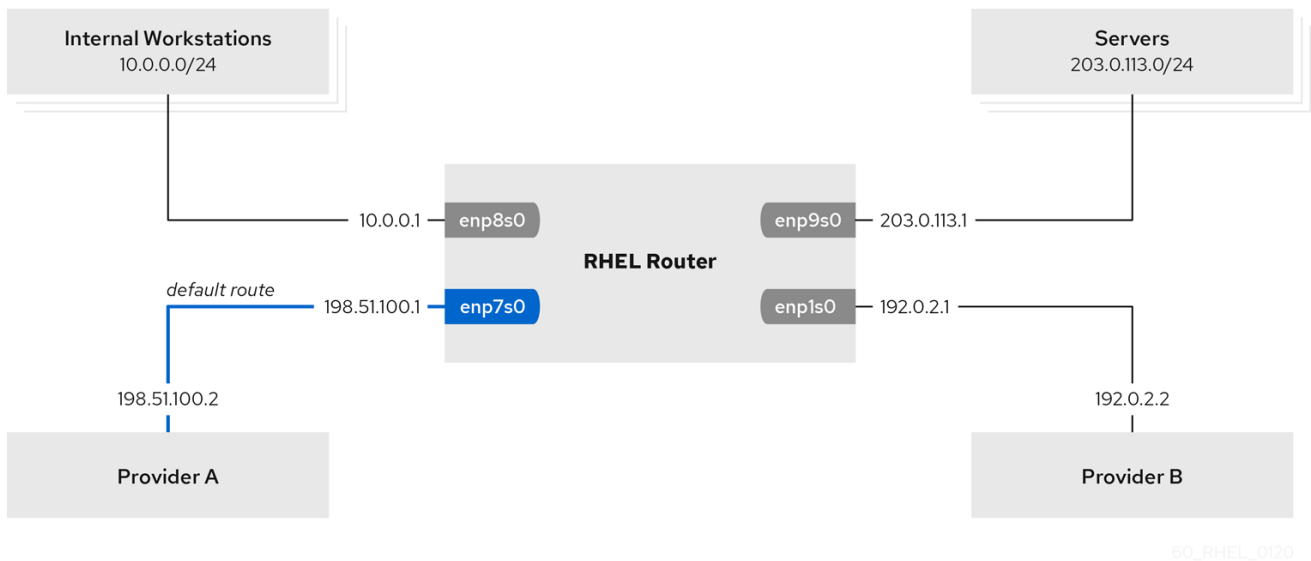
- **nm-settings(5)** and **nmcli(1)** man pages on your system

14.2. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY BY USING THE **NETWORK** RHEL SYSTEM ROLE

You can use policy-based routing to configure a different default gateway for traffic from certain subnets. For example, you can configure RHEL as a router that, by default, routes all traffic to internet provider A using the default route. However, traffic received from the internal workstations subnet is routed to provider B.

To configure policy-based routing remotely and on multiple nodes, you can use the **network** RHEL system role.

This procedure assumes the following network topology:



Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The managed nodes use NetworkManager and the **firewalld** service.
- The managed nodes you want to configure has four network interfaces:
 - The **enp7s0** interface is connected to the network of provider A. The gateway IP in the provider's network is **198.51.100.2**, and the network uses a **/30** network mask.
 - The **enp1s0** interface is connected to the network of provider B. The gateway IP in the provider's network is **192.0.2.2**, and the network uses a **/30** network mask.
 - The **enp8s0** interface is connected to the **10.0.0.0/24** subnet with internal workstations.
 - The **enp9s0** interface is connected to the **203.0.113.0/24** subnet with the company's servers.
- Hosts in the internal workstations subnet use **10.0.0.1** as the default gateway. In the procedure, you assign this IP address to the **enp8s0** network interface of the router.
- Hosts in the server subnet use **203.0.113.1** as the default gateway. In the procedure, you assign this IP address to the **enp9s0** network interface of the router.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configuring policy-based routing
  hosts: managed-node-01.example.com
```

tasks:

- name: Routing traffic from a specific subnet to a different default gateway

ansible.builtin.include_role:

name: rhel-system-roles.network

vars:

network_connections:

- name: Provider-A

interface_name: enp7s0

type: ethernet

autoconnect: True

ip:

address:

- 198.51.100.1/30

gateway4: 198.51.100.2

dns:

- 198.51.100.200

state: up

zone: external

- name: Provider-B

interface_name: enp1s0

type: ethernet

autoconnect: True

ip:

address:

- 192.0.2.1/30

route:

- network: 0.0.0.0

prefix: 0

gateway: 192.0.2.2

table: 5000

state: up

zone: external

- name: Internal-Workstations

interface_name: enp8s0

type: ethernet

autoconnect: True

ip:

address:

- 10.0.0.1/24

route:

- network: 10.0.0.0

prefix: 24

table: 5000

routing_rule:

- priority: 5

from: 10.0.0.0/24

table: 5000

state: up

zone: trusted

- name: Servers

interface_name: enp9s0

type: ethernet

autoconnect: True

```
ip:
  address:
    - 203.0.113.1/24
  state: up
  zone: trusted
```

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. On a RHEL host in the internal workstation subnet:

- a. Install the **traceroute** package:

```
# dnf install traceroute
```

- b. Use the **traceroute** utility to display the route to a host on the internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.337 ms  0.260 ms  0.223 ms
 2  192.0.2.1 (192.0.2.1)  0.884 ms  1.066 ms  1.248 ms
 ...
```

The output of the command displays that the router sends packets over **192.0.2.1**, which is the network of provider B.

2. On a RHEL host in the server subnet:

- a. Install the **traceroute** package:

```
# dnf install traceroute
```

- b. Use the **traceroute** utility to display the route to a host on the internet:

```
# traceroute redhat.com
traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
 1  203.0.113.1 (203.0.113.1)  2.179 ms  2.073 ms  1.944 ms
 2  198.51.100.2 (198.51.100.2)  1.868 ms  1.798 ms  1.549 ms
 ...
```

The output of the command displays that the router sends packets over **198.51.100.2**, which is the network of provider A.

3. On the RHEL router that you configured using the RHEL system role:

- a. Display the rule list:

```
# ip rule list
0:    from all lookup local
5:    from 10.0.0.0/24 lookup 5000
32766: from all lookup main
32767: from all lookup default
```

By default, RHEL contains rules for the tables **local**, **main**, and **default**.

- b. Display the routes in table **5000**:

```
# ip route list table 5000
0.0.0.0/0 via 192.0.2.2 dev enp1s0 proto static metric 100
10.0.0.0/24 dev enp8s0 proto static scope link src 192.0.2.1 metric 102
```

- c. Display the interfaces and firewall zones:

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

- d. Verify that the **external** zone has masquerading enabled:

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
masquerade: yes
...
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

CHAPTER 15. CHANGING A HOSTNAME BY USING HOSTNAMECTL

You can use the **hostnamectl** utility to update the hostname. By default, this utility sets the following hostname types:

- Static hostname: Stored in the **/etc/hostname** file. Typically, services use this name as the hostname.
- Pretty hostname: A descriptive name, such as **Proxy server in data center A**.
- Transient hostname: A fall-back value that is typically received from the network configuration.

Procedure

1. Optional: Display the current hostname setting:

```
# hostnamectl status --static  
old-hostname.example.com
```

2. Set the new hostname:

```
# hostnamectl set-hostname new-hostname.example.com
```

This command sets the static, pretty, and transient hostname to the new value. To set only a specific type, pass the **--static**, **--pretty**, or **--transient** option to the command.

3. The **hostnamectl** utility automatically restarts the **systemd-hostnamed** to activate the new name. For the changes to take effect, reboot the host:

```
# reboot
```

Alternatively, if you know which services use the hostname:

- a. Restart all services that only read the hostname when the service starts:

```
# systemctl restart <service_name>
```

- b. Active shell users must re-login for the changes to take effect.

Verification

- Display the hostname:

```
# hostnamectl status --static  
new-hostname.example.com
```

Additional resources

- **hostnamectl(1)**
- **systemd-hostnamed.service(8)**

CHAPTER 16. CONFIGURING THE DHCP TIMEOUT BEHAVIOR OF A NETWORKMANAGER CONNECTION

A Dynamic Host Configuration Protocol (DHCP) client requests the dynamic IP address and corresponding configuration information from a DHCP server each time a client connects to the network.

When you enable DHCP in a connection profile, NetworkManager waits, by default, 45 seconds for this request to be completed.



NOTE

Red Hat supports only the NetworkManager-internal DHCP client.

Prerequisites

- A connection that uses DHCP is configured on the host.

Procedure

1. Optional: Set the **ipv4.dhcp-timeout** and **ipv6.dhcp-timeout** properties. For example, to set both options to **30** seconds, enter:

```
# nmcli connection modify <connection_name> ipv4.dhcp-timeout 30 ipv6.dhcp-timeout 30
```

Alternatively, set the parameters to **infinity** to configure that NetworkManager does not stop trying to request and renew an IP address until it is successful.

2. Optional: Configure the behavior if NetworkManager does not receive an IPv4 address before the timeout:

```
# nmcli connection modify <connection_name> ipv4.may-fail <value>
```

If you set the **ipv4.may-fail** option to:

- **yes**, the status of the connection depends on the IPv6 configuration:
 - If the IPv6 configuration is enabled and successful, NetworkManager activates the IPv6 connection and no longer tries to activate the IPv4 connection.
 - If the IPv6 configuration is disabled or not configured, the connection fails.
- **no**, the connection is deactivated. In this case:
 - If the **autoconnect** property of the connection is enabled, NetworkManager retries to activate the connection as many times as set in the **autoconnect-retries** property. The default is **4**.
 - If the connection still cannot acquire a DHCP address, auto-activation fails. Note that after 5 minutes, the auto-connection process starts again to acquire an IP address from the DHCP server.
- 3. Optional: Configure the behavior if NetworkManager does not receive an IPv6 address before the timeout:

```
| # nmcli connection modify <connection_name> ipv6.may-fail <value>
```

Additional resources

- **nm-settings(5)** man page on your system

CHAPTER 17. CONFIGURING THE ORDER OF DNS SERVERS

Most applications use the **getaddrinfo()** function of the **glibc** library to resolve DNS requests. By default, **glibc** sends all DNS requests to the first DNS server specified in the **/etc/resolv.conf** file. If this server does not reply, RHEL uses the next server in this file. NetworkManager enables you to influence the order of DNS servers in **etc/resolv.conf**.

17.1. HOW NETWORKMANAGER ORDERS DNS SERVERS IN /ETC/RESOLV.CONF

NetworkManager orders DNS servers in the **/etc/resolv.conf** file based on the following rules:

- If only one connection profile exists, NetworkManager uses the order of IPv4 and IPv6 DNS server specified in that connection.
- If multiple connection profiles are activated, NetworkManager orders DNS servers based on a DNS priority value. If you set DNS priorities, the behavior of NetworkManager depends on the value set in the **dns** parameter. You can set this parameter in the **[main]** section in the **/etc/NetworkManager/NetworkManager.conf** file:
 - **dns=default** or if the **dns** parameter is not set:
NetworkManager orders the DNS servers from different connections based on the **ipv4.dns-priority** and **ipv6.dns-priority** parameters in each connection.

If you set no value or you set **ipv4.dns-priority** and **ipv6.dns-priority** to **0**, NetworkManager uses the global default value. See [Default values of DNS priority parameters](#).
 - **dns=dnsmasq** or **dns=systemd-resolved**:
When you use one of these settings, NetworkManager sets either **127.0.0.1** for **dnsmasq** or **127.0.0.53** as **nameserver** entry in the **/etc/resolv.conf** file.

Both the **dnsmasq** and **systemd-resolved** services forward queries for the search domain set in a NetworkManager connection to the DNS server specified in that connection, and forwards queries to other domains to the connection with the default route. When multiple connections have the same search domain set, **dnsmasq** and **systemd-resolved** forward queries for this domain to the DNS server set in the connection with the lowest priority value.

Default values of DNS priority parameters

NetworkManager uses the following default values for connections:

- **50** for VPN connections
- **100** for other connections

Valid DNS priority values:

You can set both the global default and connection-specific **ipv4.dns-priority** and **ipv6.dns-priority** parameters to a value between **-2147483647** and **2147483647**.

- A lower value has a higher priority.
- Negative values have the special effect of excluding other configurations with a greater value. For example, if at least one connection with a negative priority value exists, NetworkManager uses only the DNS servers specified in the connection profile with the lowest priority.

- If multiple connections have the same DNS priority, NetworkManager prioritizes the DNS in the following order:
 - a. VPN connections
 - b. Connection with an active default route. The active default route is the default route with the lowest metric.

Additional resources

- **nm-settings(5)** man page on your system
- [Using dnsmasq in NetworkManager to send DNS requests for a specific domain to a selected DNS server](#)

17.2. SETTING A NETWORKMANAGER-WIDE DEFAULT DNS SERVER PRIORITY VALUE

NetworkManager uses the following DNS priority default values for connections:

- **50** for VPN connections
- **100** for other connections

You can override these system-wide defaults with a custom default value for IPv4 and IPv6 connections.

Procedure

1. Edit the **/etc/NetworkManager/NetworkManager.conf** file:

- a. Add the **[connection]** section, if it does not exist:

```
[connection]
```

- b. Add the custom default values to the **[connection]** section. For example, to set the new default for both IPv4 and IPv6 to **200**, add:

```
ipv4.dns-priority=200
ipv6.dns-priority=200
```

You can set the parameters to a value between **-2147483647** and **2147483647**. Note that setting the parameters to **0** enables the built-in defaults (**50** for VPN connections and **100** for other connections).

2. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

Additional resources

- **NetworkManager.conf(5)** man page on your system

17.3. SETTING THE DNS PRIORITY OF A NETWORKMANAGER CONNECTION

If you require a specific order of DNS servers you can set priority values in connection profiles. NetworkManager uses these values to order the servers when the service creates or updates the **/etc/resolv.conf** file.

Note that setting DNS priorities makes only sense if you have multiple connections with different DNS servers configured. If you have only one connection with multiple DNS servers configured, manually set the DNS servers in the preferred order in the connection profile.

Prerequisites

- The system has multiple NetworkManager connections configured.
- The system either has no **dns** parameter set in the **/etc/NetworkManager/NetworkManager.conf** file or the parameter is set to **default**.

Procedure

1. Optionally, display the available connections:

```
# nmcli connection show
NAME          UUID                                TYPE    DEVICE
Example_con_1 d17ee488-4665-4de2-b28a-48befab0cd43 ethernet enp1s0
Example_con_2 916e4f67-7145-3ffa-9f7b-e7cada8f6bf7 ethernet enp7s0
...
```

2. Set the **ipv4.dns-priority** and **ipv6.dns-priority** parameters. For example, to set both parameters to **10**, enter:

```
# nmcli connection modify <connection_name> ipv4.dns-priority 10 ipv6.dns-priority 10
```

3. Optionally, repeat the previous step for other connections.
4. Re-activate the connection you updated:

```
# nmcli connection up <connection_name>
```

Verification

- Display the contents of the **/etc/resolv.conf** file to verify that the DNS server order is correct:

```
# cat /etc/resolv.conf
```

CHAPTER 18. USING `dnsmasq` IN NETWORKMANAGER TO SEND DNS REQUESTS FOR A SPECIFIC DOMAIN TO A SELECTED DNS SERVER

By default, Red Hat Enterprise Linux (RHEL) sends all DNS requests to the first DNS server specified in the `/etc/resolv.conf` file. If this server does not reply, RHEL uses the next server in this file. In environments where one DNS server cannot resolve all domains, administrators can configure RHEL to send DNS requests for a specific domain to a selected DNS server.

For example, you connect a server to a Virtual Private Network (VPN), and hosts in the VPN use the **example.com** domain. In this case, you can configure RHEL to process DNS queries in the following way:

- Send only DNS requests for **example.com** to the DNS server in the VPN network.
- Send all other requests to the DNS server that is configured in the connection profile with the default gateway.

You can configure NetworkManager to start an instance of **dnsmasq**. This DNS caching server then listens on port **53** on the **loopback** device. Consequently, this service is only reachable from the local system and not from the network.

With this configuration, NetworkManager adds the **nameserver 127.0.0.1** entry to the `/etc/resolv.conf` file, and **dnsmasq** dynamically routes DNS requests to the corresponding DNS servers specified in the NetworkManager connection profiles.

Prerequisites

- The system has multiple NetworkManager connections configured.
- A DNS server and search domain are configured in the NetworkManager connection profile that is responsible for resolving a specific domain.
For example, to ensure that the DNS server specified in a VPN connection resolves queries for the **example.com** domain, the VPN connection profile must contain the following settings:
 - A DNS server that can resolve **example.com**
 - A search domain set to **example.com** in the **ipv4.dns-search** and **ipv6.dns-search** parameters
- The **dnsmasq** service is not running or configured to listen on a different interface than **localhost**.

Procedure

1. Install the **dnsmasq** package:

```
# dnf install dnsmasq
```

2. Edit the `/etc/NetworkManager/NetworkManager.conf` file, and set the following entry in the **[main]** section:

```
dns=dnsmasq
```


3. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

Verification

1. Search in the **systemd** journal of the **NetworkManager** unit for which domains the service uses a different DNS server:

```
# journalctl -xeu NetworkManager
...
Jun 02 13:30:17 <client_hostname>_ dnsmasq[5298]: using nameserver 198.51.100.7#53
for domain example.com
...
```

2. Use the **tcpdump** packet sniffer to verify the correct route of DNS requests:

- a. Install the **tcpdump** package:

```
# dnf install tcpdump
```

- b. On one terminal, start **tcpdump** to capture DNS traffic on all interfaces:

```
# tcpdump -i any port 53
```

- c. On a different terminal, resolve host names for a domain for which an exception exists and another domain, for example:

```
# host -t A www.example.com
# host -t A www.redhat.com
```

- d. Verify in the **tcpdump** output that Red Hat Enterprise Linux sends only DNS queries for the **example.com** domain to the designated DNS server and through the corresponding interface:

```
...
13:52:42.234533 tun0 Out IP server.43534 > 198.51.100.7.domain: 50121+ A?
www.example.com. (33)
...
13:52:57.753235 enp1s0 Out IP server.40864 > 192.0.2.1.domain: 6906+ A?
www.redhat.com. (33)
...
```

Red Hat Enterprise Linux sends the DNS query for **www.example.com** to the DNS server on **198.51.100.7** and the query for **www.redhat.com** to **192.0.2.1**.

Troubleshooting

1. Verify that the **nameserver** entry in the **/etc/resolv.conf** file refers to **127.0.0.1**:

```
# cat /etc/resolv.conf
nameserver 127.0.0.1
```

If the entry is missing, check the **dns** parameter in the **/etc/NetworkManager/NetworkManager.conf** file.

2. Verify that the **dnsmasq** service listens on port **53** on the **loopback** device:

```
# ss -tulpn | grep "127.0.0.1:53"
udp UNCONN 0 0 127.0.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=7340,fd=18))
tcp LISTEN 0 32 127.0.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=7340,fd=19))
```

If the service does not listen on **127.0.0.1:53**, check the journal entries of the **NetworkManager** unit:

```
# journalctl -u NetworkManager
```

CHAPTER 19. AUTHENTICATING A RHEL CLIENT TO THE NETWORK BY USING THE 802.1X STANDARD WITH A CERTIFICATE STORED ON THE FILE SYSTEM

Administrators frequently use port-based Network Access Control (NAC) based on the IEEE 802.1X standard to protect a network from unauthorized LAN and Wi-Fi clients. If the network uses the Extensible Authentication Protocol Transport Layer Security (EAP-TLS) mechanism, you require a certificate to authenticate the client to this network.

19.1. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING ETHERNET CONNECTION BY USING **NMCLI**

You can use the **nmcli** utility to configure an Ethernet connection with 802.1X network authentication on the command line.

Prerequisites

- The network supports 802.1X network authentication.
- The Ethernet connection profile exists in NetworkManager and has a valid IP configuration.
- The following files required for TLS authentication exist on the client:
 - The client key stored is in the **/etc/pki/tls/private/client.key** file, and the file is owned and only readable by the **root** user.
 - The client certificate is stored in the **/etc/pki/tls/certs/client.crt** file.
 - The Certificate Authority (CA) certificate is stored in the **/etc/pki/tls/certs/ca.crt** file.
- The **wpa_supplicant** package is installed.

Procedure

1. Set the Extensible Authentication Protocol (EAP) to **tls** and the paths to the client certificate and key file:

```
# nmcli connection modify enp1s0 802-1x.eap tls 802-1x.client-cert
/etc/pki/tls/certs/client.crt 802-1x.private-key /etc/pki/tls/certs/certs/client.key
```

Note that you must set the **802-1x.eap**, **802-1x.client-cert**, and **802-1x.private-key** parameters in a single command.

2. Set the path to the CA certificate:

```
# nmcli connection modify enp1s0 802-1x.ca-cert /etc/pki/tls/certs/ca.crt
```

3. Set the identity of the user used in the certificate:

```
# nmcli connection modify enp1s0 802-1x.identity user@example.com
```

4. Optional: Store the password in the configuration:

■

```
# nmcli connection modify enp1s0 802-1x.private-key-password password
```



IMPORTANT

By default, NetworkManager stores the password in clear text in the connection profile on the disk, but the file is readable only by the **root** user. However, clear text passwords in a configuration file can be a security risk.

To increase the security, set the **802-1x.password-flags** parameter to **agent-owned**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services, after you unlock the keyring. In other cases, NetworkManager prompts for the password.

5. Activate the connection profile:

```
# nmcli connection up enp1s0
```

Verification

- Access resources on the network that require network authentication.

Additional resources

- [Configuring an Ethernet connection](#)

19.2. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING NMSTATECTL

Use the **nmstatectl** utility to configure an Ethernet connection with 802.1X network authentication through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.



NOTE

The **nmstate** library only supports the **TLS** Extensible Authentication Protocol (EAP) method.

Prerequisites

- The network supports 802.1X network authentication.
- The managed node uses NetworkManager.
- The following files required for TLS authentication exist on the client:
 - The client key stored is in the **/etc/pki/tls/private/client.key** file, and the file is owned and only readable by the **root** user.
 - The client certificate is stored in the **/etc/pki/tls/certs/client.crt** file.
 - The Certificate Authority (CA) certificate is stored in the **/etc/pki/tls/certs/ca.crt** file.

Procedure

1. Create a YAML file, for example `~/create-ethernet-profile.yml`, with the following content:

```
---
interfaces:
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
      - ip: 2001:db8:1::1
        prefix-length: 64
    autoconf: false
    dhcp: false
  802.1x:
    ca-cert: /etc/pki/tls/certs/ca.crt
    client-cert: /etc/pki/tls/certs/client.crt
    eap-methods:
      - tls
    identity: client.example.org
    private-key: /etc/pki/tls/private/client.key
    private-key-password: password
  routes:
    config:
      - destination: 0.0.0.0/0
        next-hop-address: 192.0.2.254
        next-hop-interface: enp1s0
      - destination: ::0
        next-hop-address: 2001:db8:1::fffe
        next-hop-interface: enp1s0
  dns-resolver:
    config:
      search:
        - example.com
      server:
        - 192.0.2.200
        - 2001:db8:1::ffbb
```

These settings define an Ethernet connection profile for the **enp1s0** device with the following settings:

- A static IPv4 address – **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address – **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway – **192.0.2.254**
- An IPv6 default gateway – **2001:db8:1::fffe**

- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- 802.1X network authentication using the **TLS** EAP protocol

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification

- Access resources on the network that require network authentication.

19.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE **network** RHEL SYSTEM ROLE

Network Access Control (NAC) protects a network from unauthorized clients. You can specify the details that are required for the authentication in NetworkManager connection profiles to enable clients to access the network. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use an Ansible playbook to copy a private key, a certificate, and the CA certificate to the client, and then use the **network** RHEL system role to configure a connection profile with 802.1X network authentication.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The network supports 802.1X network authentication.
- The managed nodes use NetworkManager.
- The following files required for the TLS authentication exist on the control node:
 - The client key is stored in the **/srv/data/client.key** file.
 - The client certificate is stored in the **/srv/data/client.crt** file.
 - The Certificate Authority (CA) certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. Store your sensitive variables in an encrypted file:
 - a. Create the vault:

```
$ ansible-vault create vault.yml
```

```
New Vault password: <vault_password>
```

```
Confirm New Vault password: <vault_password>
```

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: managed-node-01.example.com
  vars_files:
    - vault.yml
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

    - name: Copy client certificate for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"

    - name: Ethernet connection profile with static IP address settings and 802.1X
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
              dns:
                - 192.0.2.200
                - 2001:db8:1::ffbb
              dns_search:
                - example.com
            ieee802_1x:
```

```

identity: <user_name>
eap: tls
private_key: "/etc/pki/tls/private/client.key"
private_key_password: "{{ pwd }}"
client_cert: "/etc/pki/tls/certs/client.crt"
ca_cert: "/etc/pki/ca-trust/source/anchors/ca.crt"
domain_suffix_match: example.com
state: up

```

The settings specified in the example playbook include the following:

ieee802_1x

This variable contains the 802.1X-related settings.

eap: tls

Configures the profile to use the certificate-based **TLS** authentication method for the Extensible Authentication Protocol (EAP).

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Verification

- Access resources on the network that require network authentication.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory
- [Ansible vault](#)

19.4. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION BY USING THE NETWORK RHEL SYSTEM ROLE

Network Access Control (NAC) protects a network from unauthorized clients. You can specify the details that are required for the authentication in NetworkManager connection profiles to enable clients to access the network. By using Ansible and the **network** RHEL system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use an Ansible playbook to copy a private key, a certificate, and the CA certificate to the client, and then use the **network** RHEL system role to configure a connection profile with 802.1X network authentication.

Prerequisites

- You have prepared the control node and the managed nodes
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The network supports 802.1X network authentication.
- You installed the **wpa_supplicant** package on the managed node.
- DHCP is available in the network of the managed node.
- The following files required for TLS authentication exist on the control node:
 - The client key is stored in the **/srv/data/client.key** file.
 - The client certificate is stored in the **/srv/data/client.crt** file.
 - The CA certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. Store your sensitive variables in an encrypted file:

- a. Create the vault:

```
$ ansible-vault create vault.yml
New Vault password: <vault_password>
Confirm New Vault password: <vault_password>
```

- b. After the **ansible-vault create** command opens an editor, enter the sensitive data in the **<key>: <value>** format:

```
pwd: <password>
```

- c. Save the changes, and close the editor. Ansible encrypts the data in the vault.

2. Create a playbook file, for example **~/playbook.yml**, with the following content:

```
---
- name: Configure a wifi connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400

    - name: Copy client certificate for 802.1X authentication
```

```

ansible.builtin.copy:
  src: "/srv/data/client.crt"
  dest: "/etc/pki/tls/certs/client.crt"

- name: Copy CA certificate for 802.1X authentication
  ansible.builtin.copy:
    src: "/srv/data/ca.crt"
    dest: "/etc/pki/ca-trust/source/anchors/ca.crt"

- name: Wifi connection profile with dynamic IP address settings and 802.1X
  ansible.builtin.import_role:
    name: rhel-system-roles.network
  vars:
    network_connections:
      - name: Wifi connection profile with dynamic IP address settings and 802.1X
        interface_name: wlp1s0
        state: up
        type: wireless
        autoconnect: yes
        ip:
          dhcp4: true
          auto6: true
        wireless:
          ssid: "Example-wifi"
          key_mgmt: "wpa-eap"
        ieee802_1x:
          identity: <user_name>
          eap: tls
          private_key: "/etc/pki/tls/client.key"
          private_key_password: "{{ pwd }}"
          private_key_password_flags: none
          client_cert: "/etc/pki/tls/client.pem"
          ca_cert: "/etc/pki/tls/cacert.pem"
          domain_suffix_match: "example.com"

```

The settings specified in the example playbook include the following:

ieee802_1x

This variable contains the 802.1X-related settings.

eap: tls

Configures the profile to use the certificate-based **TLS** authentication method for the Extensible Authentication Protocol (EAP).

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

3. Validate the playbook syntax:

```
$ ansible-playbook --ask-vault-pass --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

4. Run the playbook:

```
$ ansible-playbook --ask-vault-pass ~/playbook.yml
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/doc/rhel-system-roles/network/** directory

CHAPTER 20. SETTING UP AN 802.1X NETWORK AUTHENTICATION SERVICE FOR LAN CLIENTS BY USING HOSTAPD WITH FREERADIUS BACKEND

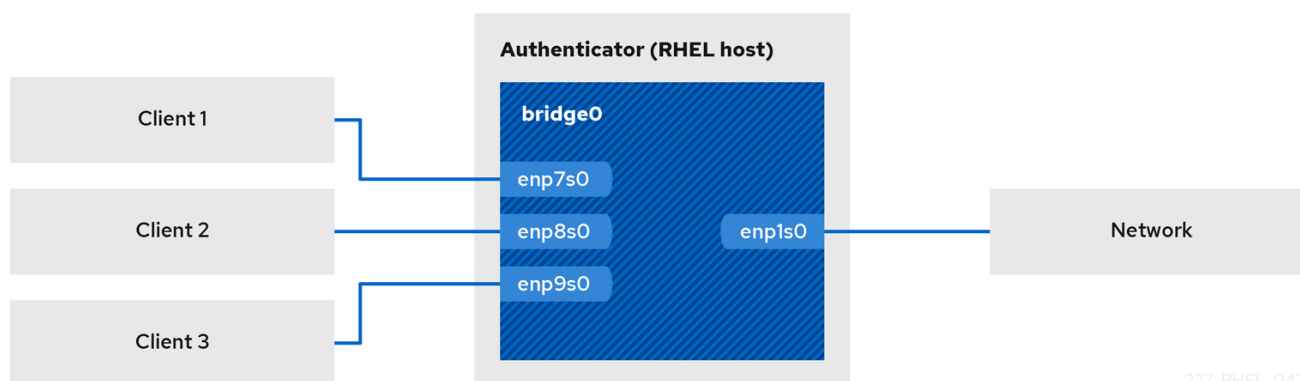
The IEEE 802.1X standard defines secure authentication and authorization methods to protect networks from unauthorized clients. By using the **hostapd** service and FreeRADIUS, you can provide network access control (NAC) in your network.



NOTE

Red Hat supports only FreeRADIUS with Red Hat Identity Management (IdM) as the backend source of authentication.

In this documentation, the RHEL host acts as a bridge to connect different clients with an existing network. However, the RHEL host grants only authenticated clients access to the network.



20.1. PREREQUISITES

- A clean installation of the **freeradius** and **freeradius-ldap** packages.
If the packages are already installed, remove the **/etc/raddb/** directory, uninstall and then install the packages again. Do not reinstall the packages by using the **dnf reinstall** command, because the permissions and symbolic links in the **/etc/raddb/** directory are then different.
- The host on which you want to configure FreeRADIUS is a [client in an IdM domain](#).

20.2. SETTING UP THE BRIDGE ON THE AUTHENTICATOR

A network bridge is a link-layer device which forwards traffic between hosts and networks based on a table of MAC addresses. If you set up RHEL as an 802.1X authenticator, add both the interfaces on which to perform authentication and the LAN interface to the bridge.

Prerequisites

- The server has multiple Ethernet interfaces.

Procedure

1. If the bridge interface does not exist, create it:

```
# nmcli connection add type bridge con-name br0 ifname br0
```

2. Assign the Ethernet interfaces to the bridge:

```
# nmcli connection add type ethernet port-type bridge con-name br0-port1 ifname enp1s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port2 ifname enp7s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port3 ifname enp8s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port4 ifname enp9s0 controller br0
```

3. Enable the bridge to forward extensible authentication protocol over LAN (EAPOL) packets:

```
# nmcli connection modify br0 group-forward-mask 8
```

4. Configure the connection to automatically activate the ports:

```
# nmcli connection modify br0 connection.autoconnect-ports 1
```

5. Activate the connection:

```
# nmcli connection up br0
```

Verification

1. Display the link status of Ethernet devices that are ports of a specific bridge:

```
# ip link show master br0
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
...
```

2. Verify if forwarding of EAPOL packets is enabled on the **br0** device:

```
# cat /sys/class/net/br0/bridge/group_fwd_mask
0x8
```

If the command returns **0x8**, forwarding is enabled.

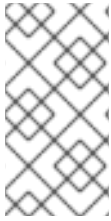
Additional resources

- **nm-settings(5)** man page on your system

20.3. CONFIGURING FREERADIUS TO AUTHENTICATE NETWORK CLIENTS SECURELY BY USING EAP

FreeRADIUS supports different methods of the Extensible authentication protocol (EAP). However, for a supported and secure scenario, use EAP-TTLS (tunneled transport layer security).

With EAP-TTLS, the clients use a secure TLS connection as the outer authentication protocol to set up the tunnel. The inner authentication then uses LDAP to authenticate to Identity Management. To use EAP-TTLS, you need a TLS server certificate.



NOTE

The default FreeRADIUS configuration files serve as documentation and describe all parameters and directives. If you want to disable certain features, comment them out instead of removing the corresponding parts in the configuration files. This enables you to preserve the structure of the configuration files and the included documentation.

Prerequisites

- You installed the **freeradius** and **freeradius-ldap** packages.
- The configuration files in the **/etc/raddb/** directory are unchanged and as provided by the **freeradius** packages.
- The host is enrolled in a Red Hat Identity Management (IdM) domain.

Procedure

1. Create a private key and request a certificate from IdM:

```
# ipa-getcert request -w -k /etc/pki/tls/private/radius.key -f /etc/pki/tls/certs/radius.pem
-o "root:radiusd" -m 640 -O "root:radiusd" -M 640 -T calPAserviceCert -C 'systemctl
restart radiusd.service' -N freeradius.idm.example.com -D freeradius.idm.example.com
-K radius/freeradius.idm.example.com
```

The **certmonger** service stores the private key in the **/etc/pki/tls/private/radius.key** file and the certificate in the **/etc/pki/tls/certs/radius.pem** file, and it sets secure permissions. Additionally, **certmonger** will monitor the certificate, renew it before it expires, and restart the **radiusd** service after the certificate was renewed.

2. Verify that the CA successfully issued the certificate:

```
# ipa-getcert list -f /etc/pki/tls/certs/radius.pem
...
Number of certificates and requests being tracked: 1.
Request ID '20240918142211':
  status: MONITORING
  stuck: no
  key pair storage: type=FILE,location='/etc/pki/tls/private/radius.key'
  certificate: type=FILE,location='/etc/pki/tls/certs/radius.crt'
...
```

3. Create the **/etc/raddb/certs/dh** file with Diffie-Hellman (DH) parameters. For example, to create a DH file with a 2048 bits prime, enter:

```
# openssl dhparam -out /etc/raddb/certs/dh 2048
```

For security reasons, do not use a DH file with less than a 2048 bits prime. Depending on the number of bits, the creation of the file can take several minutes.

4. Edit the **/etc/raddb/mods-available/eap** file:

- a. Configure the TLS-related settings in the **tls-config tls-common** directive:

```
eap {
    ...
    tls-config tls-common {
        ...
        private_key_file = /etc/pki/tls/private/radius.key
        certificate_file = /etc/pki/tls/certs/radius.pem
        ca_file = /etc/ipa/ca.crt
        ...
    }
}
```

- b. Set the **default_eap_type** parameter in the **eap** directive to **ttls**:

```
eap {
    ...
    default_eap_type = ttls
    ...
}
```

- c. Comment out the **md5** directives to disable the insecure EAP-MD5 authentication method:

```
eap {
    ...
    # md5 {
    # }
    ...
}
```

Note that, in the default configuration file, other insecure EAP authentication methods are commented out by default.

5. Edit the **/etc/raddb/sites-available/default** file, and comment out all authentication methods other than **eap**:

```
authenticate {
    ...
    # Auth-Type PAP {
    #   pap
    # }

    # Auth-Type CHAP {
    #   chap
    # }

    # Auth-Type MS-CHAP {
    #   mschap
    # }

    # mschap

    # digest
    ...
}
```

This leaves only EAP enabled for the outer authentication and disables plain-text authentication methods.

6. Edit the **/etc/raddb/sites-available/inner-tunnel** file, and make the following changes:

- a. Comment out the **-ldap** entry and add the **ldap** module configuration to the **authorize** directive:

```
authorize {
    ...

    #-ldap
    ldap
    if ((ok || updated) && User-Password) {
        update {
            control:Auth-Type := ldap
        }
    }

    ...
}
```

- b. Uncomment the LDAP authentication type in the **authenticate** directive:

```
authenticate {
    Auth-Type LDAP {
        ldap
    }
}
```

7. Enable the **ldap** module:

```
# ln -s /etc/raddb/mods-available/ldap /etc/raddb/mods-enabled/ldap
```

8. Edit the **/etc/raddb/mods-available/ldap** file, and make the following changes:

- a. In the **ldap** directive, set the IdM LDAP server URL and the base distinguished name (DN):

```
ldap {
    ...
    server = 'ldaps://idm_server.idm.example.com'
    base_dn = 'cn=users,cn=accounts,dc=idm,dc=example,dc=com'
    ...
}
```

Specify the **ldaps** protocol in the server URL to use TLS-encrypted connections between the FreeRADIUS host and the IdM server.

- b. In the **ldap** directive, enable TLS certificate validation of the IdM LDAP server:

```
tls {
    ...
    require_cert = 'demand'
    ...
}
```


9. Edit the **/etc/raddb/clients.conf** file:

a. Set a secure password in the **localhost** and **localhost_ipv6** client directives:

```
client localhost {
    ipaddr = 127.0.0.1
    ...
    secret = localhost_client_password
    ...
}

client localhost_ipv6 {
    ipv6addr = ::1
    secret = localhost_client_password
}
```

b. Add a client directive for the network authenticator:

```
client hostapd.example.org {
    ipaddr = 192.0.2.2/32
    secret = hostapd_client_password
}
```

c. Optional: If other hosts should also be able to access the FreeRADIUS service, add client directives for them as well, for example:

```
client <hostname_or_description> {
    ipaddr = <IP_address_or_range>
    secret = <client_password>
}
```

The **ipaddr** parameter accepts IPv4 and IPv6 addresses, and you can use the optional classless inter-domain routing (CIDR) notation to specify ranges. However, you can set only one value in this parameter. For example, to grant access to both an IPv4 and IPv6 address, you must add two client directives.

Use a descriptive name for the client directive, such as a hostname or a word that describes where the IP range is used.

10. Verify the configuration files:

```
# radiusd -XC
...
Configuration appears to be OK
```

11. Open the RADIUS ports in the **firewalld** service:

```
# firewall-cmd --permanent --add-service=radius
# firewall-cmd --reload
```

12. Enable and start the **radiusd** service:

```
# systemctl enable --now radiusd
```

Verification

- [Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator](#)

Troubleshooting

- If the **radiusd** service fails to start, verify that you can resolve the IdM server host name:

```
# host -v idm_server.idm.example.com
```

- For other problems, run **radiusd** in debug mode:

- a. Stop the **radiusd** service:

```
# systemctl stop radiusd
```

- b. Start the service in debug mode:

```
# radiusd -X
...
Ready to process requests
```

- c. Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

Next steps

- Disable no longer required authentication methods and other features you do not use.

20.4. CONFIGURING HOSTAPD AS AN AUTHENTICATOR IN A WIRED NETWORK

The host access point daemon (**hostapd**) service can act as an authenticator in a wired network to provide 802.1X authentication. For this, the **hostapd** service requires a RADIUS server that authenticates the clients.

The **hostapd** service provides an integrated RADIUS server. However, use the integrated RADIUS server only for testing purposes. For production environments, use FreeRADIUS server, which supports additional features, such as different authentication methods and access control.



IMPORTANT

The **hostapd** service does not interact with the traffic plane. The service acts only as an authenticator. For example, use a script or service that uses the **hostapd** control interface to allow or deny traffic based on the result of authentication events.

Prerequisites

- You installed the **hostapd** package.
- The FreeRADIUS server has been configured, and it is ready to authenticate clients.

Procedure

1. Create the **/etc/hostapd/hostapd.conf** file with the following content:

```
# General settings of hostapd
# =====

# Control interface settings
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel

# Enable logging for all modules
logger_syslog=-1
logger_stdout=-1

# Log level
logger_syslog_level=2
logger_stdout_level=2

# Wired 802.1X authentication
# =====

# Driver interface type
driver=wired

# Enable IEEE 802.1X authorization
ieee8021x=1

# Use port access entry (PAE) group address
# (01:80:c2:00:00:03) when sending EAPOL frames
use_pae_group_addr=1

# Network interface for authentication requests
interface=br0

# RADIUS client configuration
# =====

# Local IP address used as NAS-IP-Address
own_ip_addr=192.0.2.2

# Unique NAS-Identifier within scope of RADIUS server
nas_identifier=hostapd.example.org

# RADIUS authentication server
auth_server_addr=192.0.2.1
auth_server_port=1812
auth_server_shared_secret=hostapd_client_password

# RADIUS accounting server
acct_server_addr=192.0.2.1
acct_server_port=1813
acct_server_shared_secret=hostapd_client_password
```

For further details about the parameters used in this configuration, see their descriptions in the `/usr/share/doc/hostapd/hostapd.conf` example configuration file.

2. Enable and start the **hostapd** service:

```
# systemctl enable --now hostapd
```

Verification

- [Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator](#)

Troubleshooting

- If the **hostapd** service fails to start, verify that the bridge interface you use in the `/etc/hostapd/hostapd.conf` file is present on the system:

```
# ip link show br0
```

- For other problems, run **hostapd** in debug mode:

- a. Stop the **hostapd** service:

```
# systemctl stop hostapd
```

- b. Start the service in debug mode:

```
# hostapd -d /etc/hostapd/hostapd.conf
```

- c. Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

Additional resources

- **hostapd.conf(5)** man page on your system
- `/usr/share/doc/hostapd/hostapd.conf` file

20.5. TESTING EAP-TTLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR

To test if authentication by using extensible authentication protocol (EAP) over tunneled transport layer security (EAP-TTLS) works as expected, run this procedure:

- After you set up the FreeRADIUS server
- After you set up the **hostapd** service as an authenticator for 802.1X network authentication.

The output of the test utilities used in this procedure provide additional information about the EAP communication and help you to debug problems.

Prerequisites

- When you want to authenticate to:

- A FreeRADIUS server:
 - The **eapol_test** utility, provided by the **hostapd** package, is installed.
 - The client, on which you run this procedure, has been authorized in the FreeRADIUS server's client databases.
- An authenticator, the **wpa_supplicant** utility, provided by the same-named package, is installed.
- You stored the certificate authority (CA) certificate in the **/etc/ipa/ca.cert** file.

Procedure

1. Optional: Create a user in Identity Management (IdM):

```
# ipa user-add --first "Test" --last "User" idm_user --password
```

2. Create the **/etc/wpa_supplicant/wpa_supplicant-TTLS.conf** file with the following content:

```
ap_scan=0

network={
    eap=TTLS
    eapol_flags=0
    key_mgmt=IEEE8021X

    # Anonymous identity (sent in unencrypted phase 1)
    # Can be any string
    anonymous_identity="anonymous"

    # Inner authentication (sent in TLS-encrypted phase 2)
    phase2="auth=PAP"
    identity="idm_user"
    password="idm_user_password"

    # CA certificate to validate the RADIUS server's identity
    ca_cert="/etc/ipa/ca.crt"
}
```

3. To authenticate to:

- A FreeRADIUS server, enter:

```
# eapol_test -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -a 192.0.2.1 -s
<client_password>
...
EAP: Status notification: remote certificate verification (param=success)
...
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
SUCCESS
```

The **-a** option defines the IP address of the FreeRADIUS server, and the **-s** option specifies the password for the host on which you run the command in the FreeRADIUS server's client configuration.

- An authenticator, enter:

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -D wired -i
enp0s31f6
...
enp0s31f6: CTRL-Event-EAP-SUCCESS EAP authentication completed successfully
...
```

The **-i** option specifies the network interface name on which **wpa_supplicant** sends out extended authentication protocol over LAN (EAPOL) packets.

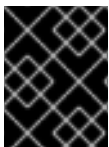
For more debugging information, pass the **-d** option to the command.

Additional resources

- `/usr/share/doc/wpa_supplicant/wpa_supplicant.conf` file

20.6. BLOCKING AND ALLOWING TRAFFIC BASED ON `HOSTAPD` AUTHENTICATION EVENTS

The **hostapd** service does not interact with the traffic plane. The service acts only as an authenticator. However, you can write a script to allow and deny traffic based on the result of authentication events.



IMPORTANT

This procedure is not supported and is no enterprise-ready solution. It only demonstrates how to block or allow traffic by evaluating events retrieved by **hostapd_cli**.

When the **802-1x-tr-mgmt** systemd service starts, RHEL blocks all traffic on the listen port of **hostapd** except extensible authentication protocol over LAN (EAPOL) packets and uses the **hostapd_cli** utility to connect to the **hostapd** control interface. The `/usr/local/bin/802-1x-tr-mgmt` script then evaluates events. Depending on the different events received by **hostapd_cli**, the script allows or blocks traffic for MAC addresses. Note that, when the **802-1x-tr-mgmt** service stops, all traffic is automatically allowed again.

Perform this procedure on the **hostapd** server.

Prerequisites

- The **hostapd** service has been configured, and the service is ready to authenticate clients.

Procedure

1. Create the `/usr/local/bin/802-1x-tr-mgmt` file with the following content:

```
#!/bin/sh

TABLE="tr-mgmt-${1}"
read -r -d " TABLE_DEF << EOF
table bridge ${TABLE} {
```

```

set allowed_macs {
    type ether_addr
}

chain accesscontrol {
    ether saddr @allowed_macs accept
    ether daddr @allowed_macs accept
    drop
}

chain forward {
    type filter hook forward priority 0; policy accept;
    meta ibpname "br0" jump accesscontrol
}
}
EOF

case ${2:-NOTANEVENT} in
    block_all)
        nft destroy table bridge "$TABLE"
        printf "$TABLE_DEF" | nft -f -
        echo "$1: All the bridge traffic blocked. Traffic for a client with a given MAC will be
allowed after 802.1x authentication"
        ;;

    AP-STA-CONNECTED | CTRL-EVENT-EAP-SUCCESS | CTRL-EVENT-EAP-
SUCCESS2)
        nft add element bridge tr-mgmt-br0 allowed_macs { $3 }
        echo "$1: Allowed traffic from $3"
        ;;

    AP-STA-DISCONNECTED | CTRL-EVENT-EAP-FAILURE)
        nft delete element bridge tr-mgmt-br0 allowed_macs { $3 }
        echo "$1: Denied traffic from $3"
        ;;

    allow_all)
        nft destroy table bridge "$TABLE"
        echo "$1: Allowed all bridge traffic again"
        ;;

    NOTANEVENT)
        echo "$0 was called incorrectly, usage: $0 interface event [mac_address]"
        ;;
esac

```

2. Create the **/etc/systemd/system/802-1x-tr-mgmt@.service** systemd service file with the following content:

```

[Unit]
Description=Example 802.1x traffic management for hostapd
After=hostapd.service
After=sys-devices-virtual-net-%i.device

[Service]
Type=simple

```

```
ExecStartPre=bash -c '/usr/sbin/hostapd_cli ping | grep PONG'
ExecStartPre=/usr/local/bin/802-1x-tr-mgmt %i block_all
ExecStart=/usr/sbin/hostapd_cli -i %i -a /usr/local/bin/802-1x-tr-mgmt
ExecStopPost=/usr/local/bin/802-1x-tr-mgmt %i allow_all
```

```
[Install]
WantedBy=multi-user.target
```

3. Reload systemd:

```
# systemctl daemon-reload
```

4. Enable and start the **802-1x-tr-mgmt** service with the interface name **hostapd** is listening on:

```
# systemctl enable --now 802-1x-tr-mgmt@br0.service
```

Verification

- Authenticate with a client to the network. See [Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator](#).

Additional resources

- **systemd.service(5)** man page on your system

CHAPTER 21. NETWORKMANAGER CONNECTION PROFILES IN KEYFILE FORMAT

NetworkManager in Red Hat Enterprise Linux (RHEL) stores connection profiles in keyfile format. This format supports all connection settings that NetworkManager provides.



NOTE

Connection profiles in **ifcfg** format are not supported on RHEL 10. NetworkManager ignores files in this format and it is also not possible to convert them to the keyfile format on RHEL 10.

21.1. THE KEYFILE FORMAT OF NETWORKMANAGER PROFILES

The keyfile format is similar to the INI format. For example, the following is an Ethernet connection profile in keyfile format:

```
[connection]
id=example_connection
uuid=82c6272d-1ff7-4d56-9c7c-0eb27c300029
type=ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[ethernet]
mac-address=00:53:00:8f:fa:66
```



WARNING

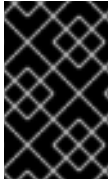
Typos or incorrect placements of parameters can lead to unexpected behavior. Therefore, do not manually edit or create NetworkManager profiles.

Use the **nmcli** utility, the **network** RHEL system role, or the **nmstate** API to manage NetworkManager connections. For example, you can use the **nmcli** utility in **offline mode** to create connection profiles.

Each section corresponds to a NetworkManager setting name as described in the **nm-settings(5)** man page. Each key-value-pair in a section is one of the properties listed in the settings specification of the man page.

Most variables in NetworkManager keyfiles have a one-to-one mapping. This means that a NetworkManager property is stored in the keyfile as a variable of the same name and in the same format. However, there are exceptions, mainly to make the keyfile syntax easier to read. For a list of

these exceptions, see the **nm-settings-keyfile(5)** man page on your system.



IMPORTANT

For security reasons, because connection profiles can contain sensitive information, such as private keys and passphrases, NetworkManager uses only configuration files owned by the **root** user and that are only readable and writable by **root**.

Save the connection profile with a **.nmconnection** suffix in the **/etc/NetworkManager/system-connections/** directory. This directory contains persistent profiles. If you modify a persistent profile by using the NetworkManager API, NetworkManager writes and overwrites files in this directory.

NetworkManager does not automatically reload profiles from disk. When you create or update a connection profile in keyfile format, use the **nmcli connection reload** command to inform NetworkManager about the changes.

21.2. USING nmcli TO CREATE KEYFILE CONNECTION PROFILES IN OFFLINE MODE

Use NetworkManager utilities, such as **nmcli**, the **network** RHEL system role, or the **nmstate** API to manage NetworkManager connections, to create and update configuration files. However, you can also create various connection profiles in the keyfile format in offline mode by using the **nmcli --offline connection add** command.

The offline mode ensures that **nmcli** operates without the **NetworkManager** service to produce keyfile connection profiles through standard output. This feature can be useful in the following scenarios:

- You want to create your connection profiles that need to be pre-deployed somewhere. For example in a container image, or as an RPM package.
- You want to create your connection profiles in an environment where the **NetworkManager** service is not available, for example, when you want to use the **chroot** utility. Alternatively, when you want to create or modify the network configuration of the RHEL system to be installed through the Kickstart **%post** script.

Procedure

1. Create a new connection profile in the keyfile format. For example, for a connection profile of an Ethernet device that does not use DHCP, run a similar **nmcli** command:

```
# nmcli --offline connection add type ethernet con-name Example-Connection
ipv4.addresses 192.0.2.1/24 ipv4.dns 192.0.2.200 ipv4.method manual >
/etc/NetworkManager/system-connections/example.nmconnection
```



NOTE

The connection name you specified with the **con-name** key is saved into the **id** variable of the generated profile. When you use the **nmcli** command to manage this connection later, specify the connection as follows:

- When the **id** variable is not omitted, use the connection name, for example **Example-Connection**.
- When the **id** variable is omitted, use the file name without the **.nmconnection** suffix, for example **output**.

2. Set permissions to the configuration file so that only the **root** user can read and update it:

```
# chmod 600 /etc/NetworkManager/system-connections/example.nmconnection
# chown root:root /etc/NetworkManager/system-connections/example.nmconnection
```

3. Reload the NetworkManager connections:

```
# nmcli connection reload
```

4. If you set the **autoconnect** variable in the profile to **false**, activate the connection:

```
# nmcli connection up Example-Connection
```

Verification

1. Verify that the **NetworkManager** service is running:

```
# systemctl status NetworkManager.service
• NetworkManager.service - Network Manager
  Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; vendor preset:
  enabled)
  Active: active (running) since Wed 2022-08-03 13:08:32 CEST; 1min 40s ago
  ...
```

2. Verify that NetworkManager can read the profile from the configuration file:

```
# nmcli -f TYPE,FILENAME,NAME connection
TYPE      FILENAME                                     NAME
ethernet  /etc/NetworkManager/system-connections/example.nmconnection  Example-
Connection
...
```

If the output does not show the newly created connection, verify that the keyfile permissions and the syntax you used are correct.

3. Display the connection profile:

```
# nmcli connection show Example-Connection
connection.id:           Example-Connection
connection.uuid:         232290ce-5225-422a-9228-cb83b22056b4
connection.stable-id:    --
connection.type:         802-3-ethernet
```

```
connection.interface-name:  --
connection.autoconnect:    yes
...
```

Additional resources

- **nmcli(1)**, **nm-settings(5)**, and **nm-settings-keyfile(5)** man pages on your system

21.3. MANUALLY CREATING A NETWORKMANAGER PROFILE IN KEYFILE FORMAT

You can manually create a NetworkManager connection profile in keyfile format.



WARNING

Manually creating or updating the configuration files can result in an unexpected or non-functional network configuration. As an alternative, you can use **nmcli** in offline mode. See [Using nmcli to create keyfile connection profiles in offline mode](#)

Procedure

1. Create a connection profile. For example, for a connection profile for the **enp1s0** Ethernet device that uses DHCP, create the **/etc/NetworkManager/system-connections/example.nmconnection** file with the following content:

```
[connection]
id=Example-Connection
type=ethernet
autoconnect=true
interface-name=enp1s0

[ipv4]
method=auto

[ipv6]
method=auto
```



NOTE

You can use any file name with a **.nmconnection** suffix. However, when you later use **nmcli** commands to manage the connection, you must use the connection name set in the **id** variable when you refer to this connection. When you omit the **id** variable, use the file name without the **.nmconnection** to refer to this connection.

2. Set permissions on the configuration file so that only the **root** user can read and update it:

```
# chown root:root /etc/NetworkManager/system-connections/example.nmconnection
# chmod 600 /etc/NetworkManager/system-connections/example.nmconnection
```

3. Reload the connection profiles:

```
# nmcli connection reload
```

4. Verify that NetworkManager read the profile from the configuration file:

```
# nmcli -f NAME,UUID,FILENAME connection
NAME          UUID          FILENAME
Example-Connection 86da2486-068d-4d05-9ac7-957ec118afba
/etc/NetworkManager/system-connections/example.nmconnection
...
```

If the command does not show the newly added connection, verify that the file permissions and the syntax you used in the file are correct.

5. If you set the **autoconnect** variable in the profile to **false**, activate the connection:

```
# nmcli connection up example_connection
```

Verification

- Display the connection profile:

```
# nmcli connection show example_connection
```

Additional resources

- **nm-settings(5)** and **nm-settings-keyfile(5)** man pages on your system

CHAPTER 22. SYSTEMD NETWORK TARGETS AND SERVICES

RHEL uses the **network** and **network-online** targets and the **NetworkManager-wait-online** service while applying network settings. Also, you can configure **systemd** services to start after the network is fully available if those services expect the network to be up and they cannot react dynamically to a change in the network state.

22.1. DIFFERENCES BETWEEN THE NETWORK AND NETWORK-ONLINE SYSTEMD TARGET

Systemd maintains the **network** and **network-online** target units. The special units such as **NetworkManager-wait-online.service**, have **WantedBy=network-online.target** and **Before=network-online.target** parameters. If enabled, these units get started with **network-online.target** and delay the target to be reached until some form of network connectivity is established. They delay the **network-online** target until the network is connected.

The **network-online** target starts a service, which adds substantial delays to further execution. Systemd automatically adds dependencies with **Wants** and **After** parameters for this target unit to all the System V (SysV) **init** script service units with a Linux Standard Base (LSB) header referring to the **\$network** facility. The LSB header is metadata for **init** scripts. You can use it to specify dependencies. This is similar to the **systemd** target.

The **network** target does not significantly delay the execution of the boot process. Reaching the **network** target means that the service that is responsible for setting up the network has started. However, it does not mean that a network device was configured. This target is important during the shutdown of the system. For example, if you have a service that was ordered after the **network** target during bootup, then this dependency is reversed during the shutdown. The network does not get disconnected until your service has been stopped. All mount units for remote network file systems automatically start the **network-online** target unit and order themselves after it.



NOTE

The **network-online** target unit is only useful during the system startup. After the system has completed booting up, this target does not track the online state of the network. Therefore, you cannot use **network-online** to monitor the network connection. This target provides a one-time system startup concept.

22.2. OVERVIEW OF NETWORKMANAGER-WAIT-ONLINE

The **NetworkManager-wait-online** service delays reaching the **network-online** target until NetworkManager reports that the startup is complete. During boot, NetworkManager activates all profiles with the **connection.autoconnect** parameter set to **yes**. However, the activation of profiles is not complete as long as NetworkManager profiles are in an activating state. If activation fails, NetworkManager retries the activation depending on the value of the **connection.autoconnect-retries**.

When a device reaches the activated state depends on its configuration. For example, if a profiles contains both IPv4 and IPv6 configuration, by default, NetworkManager considers the device as fully activated when only one of the Address families is ready. The **ipv4.may-fail** and **ipv6.may-fail** parameters in a connection profile control this behavior.

For Ethernet devices, NetworkManager waits for the carrier with a timeout. Consequently, if the Ethernet cable is not connected, this can further delay **NetworkManager-wait-online.service**.

When the startup completes, either all profiles are in a disconnected state or are successfully activated. You can configure profiles to auto-connect. The following are a few examples of parameters that set timeouts or define when the connection is considered active:

- **connection.wait-device-timeout**: Sets the timeout for the driver to detect the device.
- **ipv4.may-fail** and **ipv6.may-fail**: Sets activation with one IP address family ready, or whether a particular address family must have completed configuration.
- **ipv4.gateway-ping-timeout**: Delays activation.

Additional resources

- **nm-settings(5)**, **systemd.special(7)**, **NetworkManager-wait-online.service(8)** man pages on your system

22.3. CONFIGURING A SYSTEMD SERVICE TO START AFTER THE NETWORK HAS BEEN STARTED

Red Hat Enterprise Linux installs **systemd** service files in the **/usr/lib/systemd/system/** directory. This procedure creates a drop-in snippet for a service file in **/etc/systemd/system/<service_name>.service.d/** that is used together with the service file in **/usr/lib/systemd/system/** to start a particular service after the network is online. It has a higher priority if settings in the drop-in snippet overlap with the ones in the service file in **/usr/lib/systemd/system/**.

Procedure

1. Open a service file in the editor:

```
# systemctl edit <service_name>
```

2. Enter the following, and save the changes:

```
[Unit]
After=network-online.target
```

3. Reload the **systemd** service.

```
# systemctl daemon-reload
```

CHAPTER 23. INTRODUCTION TO NMSTATE

Nmstate is a declarative network manager API. When you use Nmstate, you describe the expected networking state by using YAML or JSON-formatted instructions.

Nmstate has many benefits. For example, it:

- Provides a stable and extensible interface to manage RHEL network capabilities
- Supports atomic and transactional operations at the host and cluster level
- Supports partial editing of most properties and preserves existing settings that are not specified in the instructions

Nmstate consists of the following packages:

Packages	Contents
nmstate	The nmstatectl command-line utility
python3-libnmstate	The libnmstate Python library
nmstate-libs	The Nmstate C library
nmstate-devel	The Nmstate C library headers

23.1. USING THE LIBNMSTATE LIBRARY IN A PYTHON APPLICATION

The **libnmstate** Python library enables developers to use Nmstate in their own application

To use the library, import it in your source code:

```
import libnmstate
```

Note that you must install the **nmstate** and **python3-libnmstate** packages to use this library.

Example 23.1. Querying the network state by using the libnmstate library

The following Python code imports the **libnmstate** library and displays the available network interfaces and their state:

```
import libnmstate
from libnmstate.schema import Interface

net_state = libnmstate.show()
for iface_state in net_state[Interface.KEY]:
    print(iface_state[Interface.NAME] + ": "
          + iface_state[Interface.STATE])
```


23.2. UPDATING THE CURRENT NETWORK CONFIGURATION BY USING NMSTATECTL

You can use the **nmstatectl** utility to store the current network configuration of one or all interfaces in a file. You can then use this file to:

- Modify the configuration and apply it to the same system.
- Copy the file to a different host and configure the host with the same or modified settings.

For example, you can export the settings of the **enp1s0** interface to a file, modify the configuration, and apply the settings to the host.

Prerequisites

- The **nmstate** package is installed.

Procedure

1. Export the settings of the **enp1s0** interface to the **~/network-config.yml** file:

```
# nmstatectl show enp1s0 > ~/network-config.yml
```

This command stores the configuration of **enp1s0** in YAML format. To store the output in JSON format, pass the **--json** option to the command.

If you do not specify an interface name, **nmstatectl** exports the configuration of all interfaces.

2. Modify the **~/network-config.yml** file using a text editor to update the configuration.
3. Apply the settings from the **~/network-config.yml** file:

```
# nmstatectl apply ~/network-config.yml
```

If you exported the settings in JSON format, pass the **--json** option to the command.

23.3. THE NMSTATE SYSTEMD SERVICE

With the **nmstate** package installed, you can automatically apply new network settings when the Red Hat Enterprise Linux system boots by configuring the **nmstate** systemd service. This service is a **oneshot** systemd service. Consequently, systemd executes it only when the system boots and when you manually restart the service.



NOTE

By default, the **nmstate** service is disabled. Use the **systemctl enable nmstate** command to enable it. Afterwards, **systemd** executes this service each time when the system starts.

To use this service, store ***.yml** files with Nmstate instructions in the **/etc/nmstate/** directory. The **nmstate** service then automatically applies the files on the next reboot or when you manually restart the service. By default, after Nmstate successfully applies a file, it renames the file's **.yml** suffix to **.applied** to prevent the service from processing the same file again.

You can configure the **nmstate** service in the `/etc/nmstate/nmstate.conf` file. For example, to preserve the original `*.yml` file after it was applied and to create only a copy with the `.applied` suffix, add the following lines to `/etc/nmstate/nmstate.conf`:

```
[service]
keep_state_file_after_apply = false
```

For further details and other configuration options, see the **nmstate.service(8)** man page on your system.

23.4. NETWORK STATES FOR THE NETWORK RHEL SYSTEM ROLE

The **network** RHEL system role supports state configurations in playbooks to configure the devices. For this, use the **network_state** variable followed by the state configurations.

Benefits of using the **network_state** variable in a playbook:

- Using the declarative method with the state configurations, you can configure interfaces, and the NetworkManager creates a profile for these interfaces in the background.
- With the **network_state** variable, you can specify the options that you require to change, and all the other options will remain the same as they are. However, with the **network_connections** variable, you must specify all settings to change the network connection profile.



IMPORTANT

You can set only Nmstate YAML instructions in **network_state**. These instructions differ from the variables you can set in **network_connections**.

For example, to create an Ethernet connection with dynamic IP address settings, use the following **vars** block in your playbook:

Playbook with state configurations	Regular playbook
------------------------------------	------------------

```
vars:
  network_state:
    interfaces:
      - name: enp7s0
        type: ethernet
        state: up
    ipv4:
      enabled: true
      auto-dns: true
      auto-gateway: true
      auto-routes: true
      dhcp: true
    ipv6:
      enabled: true
      auto-dns: true
      auto-gateway: true
      auto-routes: true
      autoconf: true
      dhcp: true
```

```
vars:
  network_connections:
    - name: enp7s0
      interface_name: enp7s0
      type: ethernet
      autoconnect: yes
    ip:
      dhcp4: yes
      auto6: yes
      state: up
```

For example, to only change the connection status of dynamic IP address settings that you created as above, use the following **vars** block in your playbook:

Playbook with state configurations	Regular playbook
<pre>vars: network_state: interfaces: - name: enp7s0 type: ethernet state: down</pre>	<pre>vars: network_connections: - name: enp7s0 interface_name: enp7s0 type: ethernet autoconnect: yes ip: dhcp4: yes auto6: yes state: down</pre>

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file
- `/usr/share/doc/rhel-system-roles/network/` directory

CHAPTER 24. GETTING STARTED WITH MULTIPATH TCP

Transmission Control Protocol (TCP) ensures reliable delivery of the data through the internet and automatically adjusts its bandwidth in response to network load. Multipath TCP (MPTCP) is an extension to the original TCP protocol (single-path). MPTCP enables a transport connection to operate across multiple paths simultaneously, and brings network connection redundancy to user endpoint devices.

24.1. UNDERSTANDING MPTCP

The Multipath TCP (MPTCP) protocol allows for simultaneous usage of multiple paths between connection endpoints. The protocol design improves connection stability and also brings other benefits compared to the single-path TCP.



NOTE

In MPTCP terminology, links are considered as paths.

The following are some of the advantages of using MPTCP:

- It allows a connection to simultaneously use multiple network interfaces.
- In case a connection is bound to a link speed, the usage of multiple links can increase the connection throughput. Note, that in case of the connection is bound to a CPU, the usage of multiple links causes the connection slowdown.
- It increases the resilience to link failures.

For more details about MPTCP, review the *Additional resources*.

Additional resources

- [Understanding Multipath TCP: High availability for endpoints and the networking highway of the future](#)

24.2. PERMANENTLY CONFIGURING MULTIPLE PATHS FOR MPTCP APPLICATIONS

You can configure MultiPath TCP (MPTCP) by using the **nmcli** command to permanently establish multiple subflows between a source and destination system. The subflows can use different resources, different routes to the destination, and even different networks. Such as Ethernet, cellular, wifi, and so on. As a result, you achieve combined connections, which increase network resilience and throughput.

The server uses the following network interfaces in our example:

- enp4s0: **192.0.2.1/24**
- enp1s0: **198.51.100.1/24**
- enp7s0: **192.0.2.3/24**

The client uses the following network interfaces in our example:

- enp4s0f0: **192.0.2.2/24**

- enp4sOf1: **198.51.100.2/24**
- enp6s0: **192.0.2.5/24**

Prerequisites

- You configured the default gateway on the relevant interfaces.

Procedure

1. Enable MPTCP sockets in the kernel:

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. Optional: The RHEL kernel default for subflow limit is 2. If you require more:

- a. Create the **/etc/systemd/system/set_mptcp_limit.service** file with the following content:

```
[Unit]
Description=Set MPTCP subflow limit to 3
After=network.target

[Service]
ExecStart=ip mptcp limits set subflows 3
Type=oneshot

[Install]
WantedBy=multi-user.target
```

The **oneshot** unit executes the **ip mptcp limits set subflows 3** command after your network (**network.target**) is operational during every boot process.

The **ip mptcp limits set subflows 3** command sets the maximum number of *additional* subflows for each connection, so 4 in total. It is possible to add maximally 3 additional subflows.

- b. Enable the **set_mptcp_limit** service:

```
# systemctl enable --now set_mptcp_limit
```

3. Enable MPTCP on all connection profiles that you want to use for connection aggregation:

```
# nmcli connection modify <profile_name> connection.mptcp-flags
signal,subflow,also-without-default-route
```

The **connection.mptcp-flags** parameter configures MPTCP endpoints and the IP address flags. If MPTCP is enabled in a NetworkManager connection profile, the setting will configure the IP addresses of the relevant network interface as MPTCP endpoints.

By default, NetworkManager does not add MPTCP flags to IP addresses if there is no default gateway. If you want to bypass that check, you need to use the **also-without-default-route** flag.

Verification

1. Verify that you enabled the MPTCP kernel parameter:

```
# sysctl net.mptcp.enabled
net.mptcp.enabled = 1
```

2. Verify that you set the subflow limit correctly, in case the default was not enough:

```
# ip mptcp limit show
add_addr_accepted 2 subflows 3
```

3. Verify that you configured the per-address MPTCP setting correctly:

```
# ip mptcp endpoint show
192.0.2.1 id 1 subflow dev enp4s0
198.51.100.1 id 2 subflow dev enp1s0
192.0.2.3 id 3 subflow dev enp7s0
192.0.2.4 id 4 subflow dev enp3s0
...
```

Additional resources

- **nm-settings-nmcli(5)**
- **ip-mptcp(8)**
- [Section 24.1, “Understanding MPTCP”](#)
- [Understanding Multipath TCP: High availability for endpoints and the networking highway of the future](#)
- [Using Multipath TCP to better survive outages and increase bandwidth](#)

24.3. CONFIGURING MPTCPD

The **mptcpd** service is a component of the **mptcp** protocol which provides an instrument to configure **mptcp** endpoints. The **mptcpd** service creates a subflow endpoint for each address by default. The endpoint list is updated dynamically according to IP addresses modification on the running host. The **mptcpd** service creates the list of endpoints automatically. It enables multiple paths as an alternative to using the **ip** utility.

Prerequisites

- The **mptcpd** package installed

Procedure

1. Enable **mptcp.enabled** option in the kernel with the following command:

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. Enable and start the **mptcpd** service:

```
# systemctl enable --now mptcp.service
```

3. To configure **mptcpd** service manually, modify the **/etc/mptcpd/mptcpd.conf** configuration file.
Note, that the endpoint, which mptcpd service creates, lasts till the host shutdown.

Verification

- Verify endpoint creation:

```
# ip mptcp endpoint
```

Additional resources

- **mptcpd(8)** man page on your system.

CHAPTER 25. LINUX TRAFFIC CONTROL

Linux offers tools for managing and manipulating the transmission of packets. The Linux Traffic Control (TC) subsystem helps in policing, classifying, shaping, and scheduling network traffic. TC also mangles the packet content during classification by using filters and actions. The TC subsystem achieves this by using queuing disciplines (**qdisc**), a fundamental element of the TC architecture.

The scheduling mechanism arranges or rearranges the packets before they enter or exit different queues. The most common scheduler is the First-In-First-Out (FIFO) scheduler. You can do the **qdiscs** operations temporarily using the **tc** utility or permanently using NetworkManager.

In Red Hat Enterprise Linux, you can configure default queueing disciplines in various ways to manage the traffic on a network interface.

25.1. OVERVIEW OF QUEUING DISCIPLINES

Queueing disciplines (**qdiscs**) help with queuing up and, later, scheduling of traffic transmission by a network interface. A **qdisc** has two operations;

- enqueue requests so that a packet can be queued up for later transmission and
- dequeue requests so that one of the queued-up packets can be chosen for immediate transmission.

Every **qdisc** has a 16-bit hexadecimal identification number called a **handle**, with an attached colon, such as **1:** or **abcd:**. This number is called the **qdisc** major number. If a **qdisc** has classes, then the identifiers are formed as a pair of two numbers with the major number before the minor, **<major>:<minor>**, for example **abcd:1**. The numbering scheme for the minor numbers depends on the **qdisc** type. Sometimes the numbering is systematic, where the first-class has the ID **<major>:1**, the second one **<major>:2**, and so on. Some **qdiscs** allow the user to set class minor numbers arbitrarily when creating the class.

Classful qdiscs

Different types of **qdiscs** exist and help in the transfer of packets to and from a networking interface. You can configure **qdiscs** with root, parent, or child classes. The point where children can be attached are called classes. Classes in **qdisc** are flexible and can always contain either multiple children classes or a single child, **qdisc**. There is no prohibition against a class containing a classful **qdisc** itself, this facilitates complex traffic control scenarios.

Classful **qdiscs** do not store any packets themselves. Instead, they enqueue and dequeue requests down to one of their children according to criteria specific to the **qdisc**. Eventually, this recursive packet passing ends up where the packets are stored (or picked up from in the case of dequeuing).

Classless qdiscs

Some **qdiscs** contain no child classes and they are called classless **qdiscs**. Classless **qdiscs** require less customization compared to classful **qdiscs**. It is usually enough to attach them to an interface.

Additional resources

- **tc(8)** and **tc-actions(8)** man pages on your system

25.2. INSPECTING QDISCS OF A NETWORK INTERFACE BY USING THE **tc** UTILITY

By default, Red Hat Enterprise Linux systems use **fq_codel qdisc**. You can inspect the **qdisc** counters using the **tc** utility.

Procedure

1. Optional: View your current **qdisc**:

```
# tc qdisc show dev enp0s1
```

2. Inspect the current **qdisc** counters:

```
# tc -s qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval
100.0ms memory_limit 32Mb ecn
Sent 1008193 bytes 5559 pkt (dropped 233, overlimits 55 requeues 77)
backlog 0b 0p requeues 0
```

- **dropped** - the number of times a packet is dropped because all queues are full
- **overlimits** - the number of times the configured link capacity is filled
- **sent** - the number of dequeues

25.3. UPDATING THE DEFAULT QDISC

If you observe networking packet losses with the current **qdisc**, you can change the **qdisc** based on your network-requirements.

Procedure

1. View the current default **qdisc**:

```
# sysctl -a | grep qdisc
net.core.default_qdisc = fq_codel
```

2. View the **qdisc** of current Ethernet connection:

```
# tc -s qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval
100.0ms memory_limit 32Mb ecn
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
maxpacket 0 drop_overlimit 0 new_flow_count 0 ecn_mark 0
new_flows_len 0 old_flows_len 0
```

3. Update the existing **qdisc**:

```
# sysctl -w net.core.default_qdisc=pfifo_fast
```

4. To apply the changes, reload the network driver:

```
# modprobe -r NETWORKDRIVERNAME
# modprobe NETWORKDRIVERNAME
```

5. Start the network interface:

```
# ip link set enp0s1 up
```

Verification

- View the **qdisc** of the Ethernet connection:

```
# tc -s qdisc show dev enp0s1
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
Sent 373186 bytes 5333 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
...
```

Additional resources

- [How to set **sysctl** variables on Red Hat Enterprise Linux](#) (Red Hat Knowledgebase)

25.4. TEMPORARILY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE BY USING THE TC UTILITY

You can update the current **qdisc** without changing the default one.

Procedure

1. Optional: View the current **qdisc**:

```
# tc -s qdisc show dev enp0s1
```

2. Update the current **qdisc**:

```
# tc qdisc replace dev enp0s1 root htb
```

Verification

- View the updated current **qdisc**:

```
# tc -s qdisc show dev enp0s1
qdisc htb 8001: root refcnt 2 r2q 10 default 0 direct_packets_stat 0 direct_qlen 1000
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
```

25.5. PERMANENTLY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE BY USING NETWORKMANAGER

You can update the current **qdisc** value of a NetworkManager connection.

Procedure

1. Optional: View the current **qdisc**:

```
# tc qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2
```

2. Update the current **qdisc**:

```
# nmcli connection modify enp0s1 tc.qdiscs 'root pfifo_fast'
```

3. Optional: To add another **qdisc** over the existing **qdisc**, use the **+tc.qdisc** option:

```
# nmcli connection modify enp0s1 +tc.qdisc 'ingress handle ffff:'
```

4. Activate the changes:

```
# nmcli connection up enp0s1
```

Verification

- View current **qdisc** the network interface:

```
# tc qdisc show dev enp0s1
qdisc pfifo_fast 8001: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc ingress ffff: parent ffff:fff1 .....

```

Additional resources

- **nm-settings(5)** man page on your system

25.6. CONFIGURING THE RATE LIMITING OF PACKETS BY USING THE TC-CTINFO UTILITY

You can limit network traffic and prevent the exhaustion of resources in the network by using rate limiting. With rate limiting, you can also reduce the load on servers by limiting repetitive packet requests in a specific time frame. In addition, you can manage bandwidth rate by configuring traffic control in the kernel with the **tc-ctinfo** utility.

The connection tracking entry stores the **Netfilter** mark and connection information. When a router forwards a packet from the firewall, the router either removes or modifies the connection tracking entry from the packet. The connection tracking information (**ctinfo**) module retrieves data from connection tracking marks into various fields. This kernel module preserves the **Netfilter** mark by copying it into a socket buffer (**skb**) mark metadata field.

Prerequisites

- The **iperf3** utility is installed on a server and a client.

Procedure

1. Perform the following steps on the server:
 - a. Add a virtual link to the network interface:

```
# ip link add name ifb4eth0 numtxqueues 48 numrxqueues 48 type ifb
```

This command has the following parameters:

name ifb4eth0

Sets a new virtual device interface.

numtxqueues 48

Sets the number of transmit queues.

numrxqueues 48

Sets the number of receive queues.

type ifb

Sets the type of the new device.

- b. Change the state of the interface:

```
# ip link set dev ifb4eth0 up
```

- c. Add the **qdisc** attribute on the physical network interface and apply it to the incoming traffic:

```
# tc qdisc add dev enp1s0 handle ffff: ingress
```

In the **handle ffff:** option, the **handle** parameter assigns the major number **ffff:** as a default value to a classful **qdisc** on the **enp1s0** physical network interface, where **qdisc** is a queueing discipline parameter to analyze traffic control.

- d. Add a filter on the physical interface of the **ip** protocol to classify packets:

```
# tc filter add dev enp1s0 parent ffff: protocol ip u32 match u32 0 0 action ctinfo  
cpmark 100 action mirrored egress redirect dev ifb4eth0
```

This command has the following attributes:

parent ffff:

Sets major number **ffff:** for the parent **qdisc**.

u32 match u32 0 0

Sets the **u32** filter to **match** the IP headers of the **u32** pattern. The first **0** represents the second byte of IP header while the other **0** is for the mask match telling the filter which bits to match.

action ctinfo

Sets action to retrieve data from the connection tracking mark into various fields.

cpmark 100

Copies the connection tracking mark (connmark) **100** into the packet IP header field.

action mirrored egress redirect dev ifb4eth0

Sets the **action** to **mirrored** to redirect the received packets to the **ifb4eth0** destination interface.

- e. Add a classful **qdisc** to the interface:

```
# tc qdisc add dev ifb4eth0 root handle 1: htb default 1000
```

This command sets the major number **1** to root **qdisc** and uses the **htb** hierarchy token bucket with classful **qdisc** of minor-id **1000**.

- f. Limit the traffic on the interface to 1 Mbit/s with an upper limit of 2 Mbit/s:

```
# tc class add dev ifb4eth0 parent 1:1 classid 1:100 htb ceil 2mbit rate 1mbit prio 100
```

This command has the following parameters:

parent 1:1

Sets **parent** with **classid** as **1** and **root** as **1**.

classid 1:100

Sets **classid** as **1:100** where **1** is the number of parent **qdisc** and **100** is the number of classes of the parent **qdisc**.

htb ceil 2mbit

The **htb** classful **qdisc** allows upper limit bandwidth of **2 Mbit/s** as the **ceil** rate limit.

- g. Apply the Stochastic Fairness Queuing (**sfq**) of classless **qdisc** to interface with a time interval of **60** seconds to reduce queue algorithm perturbation:

```
# tc qdisc add dev ifb4eth0 parent 1:100 sfq perturb 60
```

- h. Add the firewall mark (**fw**) filter to the interface:

```
# tc filter add dev ifb4eth0 parent 1:0 protocol ip prio 100 handle 100 fw classid 1:100
```

- i. Restore the packet meta mark from the connection mark (**CONNMARK**):

```
# nft add rule ip mangle PREROUTING counter meta mark set ct mark
```

In this command, the **nft** utility has a **mangle** table with the **PREROUTING** chain rule specification that alters incoming packets before routing to replace the packet mark with **CONNMARK**.

- j. If no **nft** table and chain exist, create a table and add a chain rule:

```
# nft add table ip mangle
# nft add chain ip mangle PREROUTING {type filter hook prerouting priority mangle \;}
```

- k. Set the meta mark on **tcp** packets that are received on the specified destination address **192.0.2.3**:

```
# nft add rule ip mangle PREROUTING ip daddr 192.0.2.3 counter meta mark set 0x64
```

- l. Save the packet mark into the connection mark:

```
# nft add rule ip mangle PREROUTING counter ct mark set mark
```

- m. Run the **iperf3** utility as the server on a system by using the **-s** parameter and the server then waits for the response of the client connection:

```
# iperf3 -s
```

2. On the client, run **iperf3** as a client and connect to the server that listens on IP address **192.0.2.3** for periodic HTTP request-response timestamp:

```
# iperf3 -c 192.0.2.3 -t TCP_STREAM | tee rate
```

192.0.2.3 is the IP address of the server while **192.0.2.4** is the IP address of the client.

3. Terminate the **iperf3** utility on the server by pressing **Ctrl+C**:

```
Accepted connection from 192.0.2.4, port 52128
[5] local 192.0.2.3 port 5201 connected to 192.0.2.4 port 52130
[ID] Interval      Transfer Bitrate
[5] 0.00-1.00    sec  119 KBytes  973 Kbits/sec
[5] 1.00-2.00    sec  116 KBytes  950 Kbits/sec
...
[ID] Interval      Transfer Bitrate
[5] 0.00-14.81   sec  1.51 MBytes  853 Kbits/sec receiver

iperf3: interrupt - the server has terminated
```

4. Terminate the **iperf3** utility on the client by pressing **Ctrl+C**:

```
Connecting to host 192.0.2.3, port 5201
[5] local 192.0.2.4 port 52130 connected to 192.0.2.3 port 5201
[ID] Interval      Transfer Bitrate   Retr Cwnd
[5] 0.00-1.00    sec  481 KBytes  3.94 Mb/s 0  76.4 KBytes
[5] 1.00-2.00    sec  223 KBytes  1.83 Mb/s 0  82.0 KBytes
...
[ID] Interval      Transfer Bitrate   Retr
[5] 0.00-14.00    sec  3.92 MBytes  2.35 Mb/s 32  sender
[5] 0.00-14.00    sec  0.00 Bytes  0.00 b/s   receiver

iperf3: error - the server has terminated
```

Verification

1. Display the statistics about packet counts of the **htb** and **sfq** classes on the interface:

```
# tc -s qdisc show dev ifb4eth0

qdisc htb 1: root
...
Sent 26611455 bytes 3054 pkt (dropped 76, overlimits 4887 requeues 0)
...
qdisc sfq 8001: parent
...
Sent 26535030 bytes 2296 pkt (dropped 76, overlimits 0 requeues 0)
...
```

2. Display the statistics of packet counts for the **mirred** and **ctinfo** actions:

```
# tc -s filter show dev enp1s0 ingress
filter parent ffff: protocol ip pref 49152 u32 chain 0
filter parent ffff: protocol ip pref 49152 u32 chain 0 fh 800: ht divisor 1
filter parent ffff: protocol ip pref 49152 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0
terminal flowid not_in_hw (rule hit 8075 success 8075)
  match 00000000/00000000 at 0 (success 8075 )
  action order 1: ctinfo zone 0 pipe
    index 1 ref 1 bind 1 cpmark 0x00000064 installed 3105 sec firstused 3105 sec DSCP set
0 error 0
  CPMARK set 7712
  Action statistics:
  Sent 25891504 bytes 3137 pkt (dropped 0, overlimits 0 requeues 0)
  backlog 0b 0p requeues 0

  action order 2: mirred (Egress Redirect to device ifb4eth0) stolen
    index 1 ref 1 bind 1 installed 3105 sec firstused 3105 sec
  Action statistics:
  Sent 25891504 bytes 3137 pkt (dropped 0, overlimits 61 requeues 0)
  backlog 0b 0p requeues 0
```

3. Display the statistics of the **htb** rate-limiter and its configuration:

```
# tc -s class show dev ifb4eth0
class htb 1:100 root leaf 8001: prio 7 rate 1Mbit ceil 2Mbit burst 1600b cburst 1600b
Sent 26541716 bytes 2373 pkt (dropped 61, overlimits 4887 requeues 0)
backlog 0b 0p requeues 0
lended: 7248 borrowed: 0 giants: 0
tokens: 187250 ctokens: 93625
```

Additional resources

- **tc(8)**, **tc-ctinfo(8)**, **nft(8)** man pages on your system

25.7. AVAILABLE QDISCS IN RHEL

Each **qdisc** addresses unique networking-related issues. The following is the list of **qdiscs** available in RHEL. You can use any of the following **qdisc** to shape network traffic based on your networking requirements.

Table 25.1. Available schedulers in RHEL

qdisc name	Included in	Offload support
Credit-Based Shaper	kernel-modules-extra	Yes
CHOOSE and Keep for responsive flows, CHOOSE and Kill for unresponsive flows (CHOKe)	kernel-modules-extra	
Controlled Delay (CoDel)	kernel-core	

qdisc name	Included in	Offload support
Enhanced Transmission Selection (ETS)	kernel-modules-extra	Yes
Earliest TxTime First (ETF)	kernel-modules-extra	
Fair Queue (FQ)	kernel-core	
Fair Queuing Controlled Delay (FQ_CODEL)	kernel-core	
Generalized Random Early Detection (GRED)	kernel-modules-extra	
Hierarchical Fair Service Curve (HSFC)	kernel-core	
Hierarchy Token Bucket (HTB)	kernel-core	Yes
INGRESS	kernel-core	Yes
Multi Queue Priority (MQPRIO)	kernel-modules-extra	Yes
Multiqueue (MULTIQ)	kernel-modules-extra	Yes
Network Emulator (NETEM)	kernel-modules-extra	
Random Early Detection (RED)	kernel-modules-extra	Yes
Stochastic Fairness Queueing (SFQ)	kernel-core	
Time-aware Priority Shaper (TAPRIO)	kernel-modules-extra	
Token Bucket Filter (TBF)	kernel-core	Yes



IMPORTANT

The **qdisc** offload requires hardware and driver support on NIC.

Additional resources

- **tc(8)** man page on your system

CHAPTER 26. CONFIGURING INFINIBAND AND RDMA NETWORKS

You can configure and manage Remote Directory Memory Access (RDMA) networks and InfiniBand hardware at an enterprise level by using various protocols. These include RDMA over Converged Ethernet (RoCE), the software implementation of RoCE (Soft-RoCE), the IP networks protocol such as iWARP, and the Network File System over RDMA (NFS over RDMA) protocol as a native support on RDMA-supported hardware. For low-latency and high-throughput connections, you can configure IP over InfiniBand (IPoIB).

26.1. INTRODUCTION TO INFINIBAND AND RDMA

InfiniBand refers to two distinct things:

- The physical link-layer protocol for InfiniBand networks
- The InfiniBand Verbs API, an implementation of the remote direct memory access (RDMA) technology

RDMA provides access between the main memory of two computers without involving an operating system, cache, or storage. By using RDMA, data transfers with high-throughput, low-latency, and low CPU utilization.

In a typical IP data transfer, when an application on one machine sends data to an application on another machine, the following actions happen on the receiving end:

1. The kernel must receive the data.
2. The kernel must determine that the data belongs to the application.
3. The kernel wakes up the application.
4. The kernel waits for the application to perform a system call into the kernel.
5. The application copies the data from the internal memory space of the kernel into the buffer provided by the application.

This process means that most network traffic is copied across the main memory of the system if the host adapter uses direct memory access (DMA) or otherwise at least twice. Additionally, the computer executes some context switches to switch between the kernel and application. These context switches can cause a higher CPU load with high traffic rates while slowing down the other tasks.

Unlike traditional IP communication, RDMA communication bypasses the kernel intervention in the communication process. This reduces the CPU overhead. After a packet enters a network, the RDMA protocol enables the host adapter to decide which application should receive it and where to store it in the memory space of that application. Instead of sending the packet for processing to the kernel and copying it into the memory of the user application, the host adapter directly places the packet contents in the application buffer. This process requires a separate API, the InfiniBand Verbs API, and applications need to implement the InfiniBand Verbs API to use RDMA.

Red Hat Enterprise Linux supports both the InfiniBand hardware and the InfiniBand Verbs API. Additionally, it supports the following technologies to use the InfiniBand Verbs API on non-InfiniBand hardware:

- iWARP: A network protocol that implements RDMA over IP networks

- RDMA over Converged Ethernet (RoCE), which is also known as InfiniBand over Ethernet (IBoE): A network protocol that implements RDMA over Ethernet networks

26.2. CONFIGURING THE CORE RDMA SUBSYSTEM

The **rdma** service configuration manages the network protocols and communication standards such as InfiniBand, iWARP, and RoCE.

Procedure

- Install the **rdma-core** package:

```
# dnf install rdma-core
```

Verification

1. Install the **libibverbs-utils** and **infiniband-diags** packages:

```
# dnf install libibverbs-utils infiniband-diags
```

2. List the available InfiniBand devices:

```
# ibv_devices
```

device	node GUID
-----	-----
mlx4_0	0002c903003178f0
mlx4_1	f4521403007bcba0

3. Display the information of the **mlx4_1** device:

```
# ibv_devinfo -d mlx4_1
```

```
hca_id: mlx4_1
transport:      InfiniBand (0)
fw_ver:         2.30.8000
node_guid:      f452:1403:007b:cba0
sys_image_guid: f452:1403:007b:cba3
vendor_id:      0x02c9
vendor_part_id: 4099
hw_ver:         0x0
board_id:       MT_1090120019
phys_port_cnt:  2
  port: 1
    state:       PORT_ACTIVE (4)
    max_mtu:     4096 (5)
    active_mtu:  2048 (4)
    sm_lid:      2
    port_lid:    2
    port_lmc:    0x01
    link_layer:  InfiniBand

  port: 2
    state:       PORT_ACTIVE (4)
```

```

max_mtu:      4096 (5)
active_mtu:    4096 (5)
sm_lid:        0
port_lid:      0
port_lmc:      0x00
link_layer:    Ethernet

```

4. Display the status of the **mlx4_1** device:

```

# ibstat mlx4_1

CA 'mlx4_1'
CA type: MT4099
Number of ports: 2
Firmware version: 2.30.8000
Hardware version: 0
Node GUID: 0xf4521403007bcba0
System image GUID: 0xf4521403007bcba3
Port 1:
  State: Active
  Physical state: LinkUp
  Rate: 56
  Base lid: 2
  LMC: 1
  SM lid: 2
  Capability mask: 0x0251486a
  Port GUID: 0xf4521403007bcba1
  Link layer: InfiniBand
Port 2:
  State: Active
  Physical state: LinkUp
  Rate: 40
  Base lid: 0
  LMC: 0
  SM lid: 0
  Capability mask: 0x04010000
  Port GUID: 0xf65214ffe7bcba2
  Link layer: Ethernet

```

5. The **ibping** utility pings an InfiniBand address and runs as a client/server by configuring the parameters.
 - a. Start server mode **-S** on port number **-P** with **-C** InfiniBand channel adapter (CA) name on the host:

```
# ibping -S -C mlx4_1 -P 1
```

- b. Start client mode, send some packets **-c** on port number **-P** by using **-C** InfiniBand channel adapter (CA) name with **-L** Local Identifier (LID) on the host:

```
# ibping -c 50 -C mlx4_0 -P 1 -L 2
```

26.3. CONFIGURING IPOIB

By default, InfiniBand does not use the internet protocol (IP) for communication. However, IP over InfiniBand (IPoIB) provides an IP network emulation layer on top of InfiniBand remote direct memory access (RDMA) networks. This allows existing unmodified applications to transmit data over InfiniBand networks, but the performance is lower than if the application would use RDMA natively.



NOTE

The Mellanox devices, starting from ConnectX-4 and above, on RHEL 8 and later use Enhanced IPoIB mode by default (datagram only). Connected mode is not supported on these devices.

26.3.1. The IPoIB communication modes

An IPoIB device is configurable in either **Datagram** or **Connected** mode. The difference is the type of queue pair the IPoIB layer attempts to open with the machine at the other end of the communication:

- In the **Datagram** mode, the system opens an unreliable, disconnected queue pair. This mode does not support packages larger than Maximum Transmission Unit (MTU) of the InfiniBand link layer. During transmission of data, the IPoIB layer adds a 4-byte IPoIB header on top of the IP packet. As a result, the IPoIB MTU is 4 bytes less than the InfiniBand link-layer MTU. As **2048** is a common InfiniBand link-layer MTU, the common IPoIB device MTU in **Datagram** mode is **2044**.
- In the **Connected** mode, the system opens a reliable, connected queue pair. This mode allows messages larger than the InfiniBand link-layer MTU. The host adapter handles packet segmentation and reassembly. As a result, in the **Connected** mode, the messages sent from InfiniBand adapters have no size limits. However, there are limited IP packets due to the **data** field and TCP/IP **header** field. For this reason, the IPoIB MTU in the **Connected** mode is **65520** bytes.

The **Connected** mode has a higher performance but consumes more kernel memory.

Though a system is configured to use the **Connected** mode, a system still sends multicast traffic by using the **Datagram** mode because InfiniBand switches and fabric cannot pass multicast traffic in the **Connected** mode. Also, when the host is not configured to use the **Connected** mode, the system falls back to the **Datagram** mode.

While running an application that sends multicast data up to the MTU on the interface, configure the interface in **Datagram** mode or configure the application to cap the send size of a packet that will fit in datagram-sized packets.

26.3.2. Understanding IPoIB hardware addresses

IPoIB devices have a **20** byte hardware address that consists of the following parts:

- The first 4 bytes are flags and queue pair numbers
- The next 8 bytes are the subnet prefix
The default subnet prefix is **0xfe:80:00:00:00:00:00:00**. After the device connects to the subnet manager, the device changes this prefix to match with the configured subnet manager.
- The last 8 bytes are the Globally Unique Identifier (GUID) of the InfiniBand port that attaches to the IPoIB device

**NOTE**

As the first 12 bytes can change, do not use them in the **udev** device manager rules.

26.3.3. Renaming IPoIB devices by using systemd link file

By default, the kernel names Internet Protocol over InfiniBand (IPoIB) devices, for example, **ib0**, **ib1**, and so on. To avoid conflicts, create a **systemd** link file to create persistent and meaningful names such as **mlx4_ib0**.

Prerequisites

- You have installed an InfiniBand device.

Procedure

1. Display the hardware address of the device **ib0**:

```
# ip addr show ib0
```

```
7: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state UP
group default qlen 256
    link/infiniband 80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1 brd
    00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    altname ibp7s0
    altname ibs2
    inet 172.31.0.181/24 brd 172.31.0.255 scope global dynamic noprefixroute ib0
        valid_lft 2899sec preferred_lft 2899sec
    inet6 fe80::f652:1403:7b:e1b1/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

2. For naming the interface with MAC address **80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1** to **mlx4_ib0**, create the **/etc/systemd/network/70-custom-ifnames.link** file with following contents:

```
[Match]
MACAddress=80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1

[Link]
Name=_mlx4_ib0
```

This link file matches a MAC address and renames the network interface to the name set in the **Name** parameter.

Verification

1. Reboot the host:

```
# reboot
```

2. Verify that the device with the MAC address you specified in the link file has been assigned to **mlx4_ib0**:

```
# ip addr show mlx4_ib0
```

```

7: mlx4_ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state
UP group default qlen 256
    link/infiniband 80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1 brd
00:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    altname ibp7s0
    altname ibs2
    inet 172.31.0.181/24 brd 172.31.0.255 scope global dynamic noprefixroute mlx4_ib0
        valid_lft 2899sec preferred_lft 2899sec
    inet6 fe80::f652:1403:7b:e1b1/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

Additional resources

- **systemd.link(5)** man page on your system

26.3.4. Configuring an IPoIB connection by using nmcli

You can use the **nmcli** utility to create an IP over InfiniBand connection on the command line.

Prerequisites

- An InfiniBand device is installed on the server
- The corresponding kernel module is loaded

Procedure

1. Create the InfiniBand connection to use the **mlx4_ib0** interface in the **Connected** transport mode and the maximum MTU of **65520** bytes:

```
# nmcli connection add type infiniband con-name mlx4_ib0 ifname mlx4_ib0 transport-
mode Connected mtu 65520
```

2. Set a **P_Key**, for example:

```
# nmcli connection modify mlx4_ib0 infiniband.p-key 0x8002
```

3. Configure the IPv4 settings:

- To use DHCP, enter:

```
# nmcli connection modify mlx4_ib0 ipv4.method auto
```

Skip this step if **ipv4.method** is already set to **auto** (default).

- To set a static IPv4 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify mlx4_ib0 ipv4.method manual ipv4.addresses
192.0.2.1/24 ipv4.gateway 192.0.2.254 ipv4.dns 192.0.2.200 ipv4.dns-search
example.com
```

4. Configure the IPv6 settings:

- To use stateless address autoconfiguration (SLAAC), enter:

```
# nmcli connection modify mlx4_ib0 ipv6.method auto
```

Skip this step if **ipv6.method** is already set to **auto** (default).

- To set a static IPv6 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify mlx4_ib0 ipv6.method manual ipv6.addresses
2001:db8:1::fffe/64 ipv6.gateway 2001:db8:1::fffe ipv6.dns 2001:db8:1::ffbb
ipv6.dns-search example.com
```

5. To customize other settings in the profile, use the following command:

```
# nmcli connection modify mlx4_ib0 <setting> <value>
```

Enclose values with spaces or semicolons in quotes.

6. Activate the profile:

```
# nmcli connection up mlx4_ib0
```

Verification

- Use the **ping** utility to send ICMP packets to the remote host's InfiniBand adapter, for example:

```
# ping -c5 192.0.2.2
```

26.3.5. Configuring an IPoIB connection by using the **network** RHEL system role

You can use IP over InfiniBand (IPoIB) to send IP packets over an InfiniBand interface. To configure IPoIB, create a NetworkManager connection profile. By using Ansible and the **network** system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure IPoIB and, if a connection profile for the InfiniBand's parent device does not exist, the role can create it as well.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- An InfiniBand device named **mlx4_ib0** is installed in the managed nodes.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: IPoIB connection profile with static IP address settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # InfiniBand connection mlx4_ib0
          - name: mlx4_ib0
            interface_name: mlx4_ib0
            type: infiniband

            # IPoIB device mlx4_ib0.8002 on top of mlx4_ib0
            - name: mlx4_ib0.8002
              type: infiniband
              autoconnect: yes
              infiniband:
                p_key: 0x8002
                transport_mode: datagram
              parent: mlx4_ib0
              ip:
                address:
                  - 192.0.2.1/24
                  - 2001:db8:1::1/64
              state: up
```

The settings specified in the example playbook include the following:

type: `<profile_type>`

Sets the type of the profile to create. The example playbook creates two connection profiles: One for the InfiniBand connection and one for the IPoIB device.

parent: `<parent_device>`

Sets the parent device of the IPoIB connection profile.

p_key: `<value>`

Sets the InfiniBand partition key. If you set this variable, do not set **interface_name** on the IPoIB device.

transport_mode: `<mode>`

Sets the IPoIB connection operation mode. You can set this variable to **datagram** (default) or **connected**.

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.network/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```


Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the IP settings of the **mlx4_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'ip address show
mlx4_ib0.8002'
managed-node-01.example.com | CHANGED | rc=0 >>
...
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute ib0.8002
    valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/64 scope link tentative noprefixroute
    valid_lft forever preferred_lft forever
```

2. Display the partition key (P_Key) of the **mlx4_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/pkey'
managed-node-01.example.com | CHANGED | rc=0 >>
0x8002
```

3. Display the mode of the **mlx4_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/mode'
managed-node-01.example.com | CHANGED | rc=0 >>
datagram
```

Additional resources

- [/usr/share/ansible/roles/rhel-system-roles.network/README.md](#) file
- [/usr/share/doc/rhel-system-roles/network/](#) directory

26.3.6. Configuring an IPoIB connection by using **nmstatectl**

You can use the **nmstatectl** utility to configure an IP over InfiniBand (IPoIB) connection through the Nmstate API. The Nmstate API ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

Prerequisites

- An InfiniBand device is installed on the server.
- The kernel module for the InfiniBand device is loaded.

Procedure

Procedure

1. Create a YAML file, for example `~/create-IPoIB-profile.yml`, with the following content:

```

interfaces:
- name: mlx4_ib0.8002
  type: infiniband
  state: up
  ipv4:
    enabled: true
    address:
      - ip: 192.0.2.1
        prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
      - ip: 2001:db8:1::1
        prefix-length: 64
    autoconf: false
    dhcp: false
  infiniband:
    base-iface: "mlx4_ib0"
    mode: datagram
    pkey: "0x8002"

routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-address: 192.0.2.254
      next-hop-interface: mlx4_ib0.8002
    - destination: ::0
      next-hop-address: 2001:db8:1::fffe
      next-hop-interface: mlx4_ib0.8002

```

An IPoIB connection has now the following settings:

- IPOIB device name: **mlx4_ib0.8002**
- Base interface (parent): **mlx4_ib0**
- InfiniBand partition key: **0x8002**
- Transport mode: **datagram**
- Static IPv4 address: **192.0.2.1** with the **/24** subnet mask
- Static IPv6 address: **2001:db8:1::1** with the **/64** subnet mask
- IPv4 default gateway: **192.0.2.254**
- IPv6 default gateway: **2001:db8:1::fffe**

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-IPoIB-profile.yml
```

Verification

1. Display the IP settings of the **mlx4_ib0.8002** device:

```
# ip address show mlx4_ib0.8002
...
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute ib0.8002
    valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/64 scope link tentative noprefixroute
    valid_lft forever preferred_lft forever
```

2. Display the partition key (P_Key) of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/pkey
0x8002
```

3. Display the mode of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/mode
datagram
```

Additional resources

- **nmstatectl(8)** man page on your system
- **/usr/share/doc/nmstate/examples/** directory

26.3.7. Testing an RDMA network by using iperf3 after IPoIB is configured

In the following example, the large buffer size is used to perform a 60 seconds test to measure maximum throughput and fully use the bandwidth and latency between two hosts by using the **iperf3** utility.

Prerequisites

- You have configured IPoIB on both hosts.

Procedure

1. To run **iperf3** as a server on a system, define a time interval to provide periodic bandwidth updates **-i** to listen as a server **-s** that waits for the response of the client connection:

```
# iperf3 -i 5 -s
```

2. To run **iperf3** as a client on another system, define a time interval to provide periodic bandwidth updates **-i** to connect to the listening server **-c** of IP address **192.168.2.2** with **-t** time in seconds:

```
# iperf3 -i 5 -t 60 -c 192.168.2.2
```

3. Use the following commands:

- a. Display test results on the system that acts as a server:

```
-
```

iperf3 -i 10 -s

```
-----
Server listening on 5201
-----
```

```
Accepted connection from 192.168.2.3, port 22216
[5] local 192.168.2.2 port 5201 connected to 192.168.2.3 port 22218
[ID] Interval      Transfer    Bandwidth
[5]  0.00-10.00 sec  17.5 GBytes 15.0 Gbits/sec
[5] 10.00-20.00 sec  17.6 GBytes 15.2 Gbits/sec
[5] 20.00-30.00 sec  18.4 GBytes 15.8 Gbits/sec
[5] 30.00-40.00 sec  18.0 GBytes 15.5 Gbits/sec
[5] 40.00-50.00 sec  17.5 GBytes 15.1 Gbits/sec
[5] 50.00-60.00 sec  18.1 GBytes 15.5 Gbits/sec
[5] 60.00-60.04 sec  82.2 MBytes 17.3 Gbits/sec
-----
[ID] Interval      Transfer    Bandwidth
[5]  0.00-60.04 sec   0.00 Bytes  0.00 bits/sec sender
[5]  0.00-60.04 sec  107 GBytes 15.3 Gbits/sec receiver
```

- b. Display test results on the system that acts as a client:

iperf3 -i 1 -t 60 -c 192.168.2.2

```
Connecting to host 192.168.2.2, port 5201
[4] local 192.168.2.3 port 22218 connected to 192.168.2.2 port 5201
[ID] Interval      Transfer    Bandwidth    Retr Cwnd
[4]  0.00-10.00 sec  17.6 GBytes 15.1 Gbits/sec  0  6.01 MBytes
[4] 10.00-20.00 sec  17.6 GBytes 15.1 Gbits/sec  0  6.01 MBytes
[4] 20.00-30.00 sec  18.4 GBytes 15.8 Gbits/sec  0  6.01 MBytes
[4] 30.00-40.00 sec  18.0 GBytes 15.5 Gbits/sec  0  6.01 MBytes
[4] 40.00-50.00 sec  17.5 GBytes 15.1 Gbits/sec  0  6.01 MBytes
[4] 50.00-60.00 sec  18.1 GBytes 15.5 Gbits/sec  0  6.01 MBytes
-----
[ID] Interval      Transfer    Bandwidth    Retr
[4]  0.00-60.00 sec  107 GBytes 15.4 Gbits/sec  0 sender
[4]  0.00-60.00 sec  107 GBytes 15.4 Gbits/sec  receiver
```

Additional resources

- **iperf3(1)** man page on your system

26.4. CONFIGURING ROCE

Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) is a network protocol that utilizes RDMA over an Ethernet network. For configuration, RoCE requires specific hardware and some of the hardware vendors are Mellanox, Broadcom, and QLogic.

26.4.1. Overview of RoCE protocol versions

The following are the different RoCE versions:

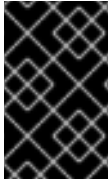
RoCE v1

The RoCE version 1 protocol is an Ethernet link layer protocol with Ethertype **0x8915** that enables the communication between any two hosts in the same Ethernet broadcast domain.

RoCE v2

The RoCE version 2 protocol exists on the top of either the UDP over IPv4 or the UDP over IPv6 protocol. For RoCE v2, the UDP destination port number is **4791**.

The RDMA_CM sets up a reliable connection between a client and a server for transferring data. RDMA_CM provides an RDMA transport-neutral interface for establishing connections. The communication uses a specific RDMA device and message-based data transfers.



IMPORTANT

Using different versions, such as RoCE v2 on the client and RoCE v1 on the server is not supported. In such a case, configure both the server and client to communicate over RoCE v1.

RoCE v1 works at the Data Link layer (Layer 2) and only supports the communication of two machines in the same network. By default, RoCE v2 is available. It works at the Network Layer (Layer 3). RoCE v2 supports packet routing that provides a connection with multiple Ethernet.

26.4.2. Temporarily changing the default RoCE version

Using the RoCE v2 protocol on the client and RoCE v1 on the server is not supported. If the hardware in your server supports RoCE v1 only, configure your clients for RoCE v1 to communicate with the server. For example, you can configure a client that uses the **mlx5_0** driver for the Mellanox ConnectX-5 InfiniBand device that only supports RoCE v1.



NOTE

The changes described here will remain effective until you reboot the host.

Prerequisites

- The client uses an InfiniBand device with RoCE v2 protocol.
- The server uses an InfiniBand device that only supports RoCE v1.

Procedure

1. Create the `/sys/kernel/config/rdma_cm/mlx5_0/` directory:

```
# mkdir /sys/kernel/config/rdma_cm/mlx5_0/
```

2. Display the default RoCE mode:

```
# cat /sys/kernel/config/rdma_cm/mlx5_0/ports/1/default_roce_mode
RoCE v2
```

3. Change the default RoCE mode to version 1:

```
# echo "IB/RoCE v1" > /sys/kernel/config/rdma_cm/mlx5_0/ports/1/default_roce_mode
```

26.4.3. Configuring Soft-RoCE

Soft-RoCE is a software implementation of remote direct memory access (RDMA) over Ethernet, which is also called RXE. Use Soft-RoCE on hosts without RoCE host channel adapters (HCA).



IMPORTANT

The Soft-RoCE feature is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See [Technology Preview Features Support Scope](#) on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

Prerequisites

- An Ethernet adapter is installed

Procedure

1. Install the **iproute**, **libibverbs**, **libibverbs-utils**, and **infiniband-diags** packages:

```
# dnf install iproute libibverbs libibverbs-utils infiniband-diags
```

2. Display the RDMA links:

```
# rdma link show
```

3. Add a new **rx**e device named **rx**e0 that uses the **enp0s1** interface:

```
# rdma link add rx0 type rx netdev enp1s0
```

Verification

1. View the state of all RDMA links:

```
# rdma link show
```

```
link rx0/1 state ACTIVE physical_state LINK_UP netdev enp1s0
```

2. List the available RDMA devices:

```
# ibv_devices
```

device	node GUID
-----	-----
rx0	505400ffed5e0fb

3. You can use the **ibstat** utility to display a detailed status:

```
# ibstat rx0
```

```
CA 'rx0'
```

```

CA type:
Number of ports: 1
Firmware version:
Hardware version:
Node GUID: 0x505400ffed5e0fb
System image GUID: 0x0000000000000000
Port 1:
State: Active
Physical state: LinkUp
Rate: 100
Base lid: 0
LMC: 0
SM lid: 0
Capability mask: 0x00890000
Port GUID: 0x505400ffed5e0fb
Link layer: Ethernet

```

26.5. INCREASING THE AMOUNT OF MEMORY THAT USERS ARE ALLOWED TO PIN IN THE SYSTEM

Remote direct memory access (RDMA) operations require the pinning of physical memory. As a consequence, the kernel is not allowed to write memory into the swap space. If a user pins too much memory, the system can run out of memory, and the kernel terminates processes to free up more memory. Therefore, memory pinning is a privileged operation.

If non-root users need to run large RDMA applications, it is necessary to increase the amount of memory to maintain pages in primary memory pinned all the time.

Procedure

- As the **root** user, create the file **/etc/security/limits.conf** with the following contents:

```

@rdma soft memlock unlimited
@rdma hard memlock unlimited

```

Verification

- Log in as a member of the **rdma** group after editing the **/etc/security/limits.conf** file. Note that Red Hat Enterprise Linux applies updated **ulimit** settings when the user logs in.
- Use the **ulimit -l** command to display the limit:

```

$ ulimit -l
unlimited

```

If the command returns **unlimited**, the user can pin an unlimited amount of memory.

Additional resources

- limits.conf(5)** man page on your system

26.6. ENABLING NFS OVER RDMA ON AN NFS SERVER

Remote Direct Memory Access (RDMA) is a protocol that enables a client system to directly transfer data from the memory of a storage server into its own memory. This enhances storage throughput, decreases latency in data transfer between the server and client, and reduces CPU load on both ends. If both the NFS server and clients are connected over RDMA, clients can use NFSoRDMA to mount an exported directory.

Prerequisites

- The NFS service is running and configured
- An InfiniBand or RDMA over Converged Ethernet (RoCE) device is installed on the server.
- IP over InfiniBand (IPoIB) is configured on the server, and the InfiniBand device has an IP address assigned.

Procedure

1. Install the **rdma-core** package:

```
# dnf install rdma-core
```

2. If the package was already installed, verify that the **xprtrdma** and **svcrdma** modules in the **/etc/rdma/modules/rdma.conf** file are uncommented:

```
# NFS over RDMA client support
xprtrdma
# NFS over RDMA server support
svcrdma
```

3. Optional: By default, NFS over RDMA uses port 20049. If you want to use a different port, set the **rdma-port** setting in the **[nfsd]** section of the **/etc/nfs.conf** file:

```
rdma-port=<port>
```

4. Open the NFSoRDMA port in **firewalld**:

```
# firewall-cmd --permanent --add-port={20049/tcp,20049/udp}
# firewall-cmd --reload
```

Adjust the port numbers if you set a different port than 20049.

5. Restart the **nfs-server** service:

```
# systemctl restart nfs-server
```

Verification

1. On a client with InfiniBand hardware, perform the following steps:
 - a. Install the following packages:

```
# dnf install nfs-utils rdma-core
```


- b. Mount an exported NFS share over RDMA:

```
# mount -o rdma server.example.com:/nfs/projects/ /mnt/
```

If you set a port number other than the default (20049), pass **port=<port_number>** to the command:

```
# mount -o rdma,port=<port_number> server.example.com:/nfs/projects/ /mnt/
```

- c. Verify that the share was mounted with the **rdma** option:

```
# mount | grep "/mnt"
server.example.com:/nfs/projects/ on /mnt type nfs (... ,proto=rdma,...)
```

Additional resources

- [Configuring InfiniBand and RDMA networks](#)

26.7. INFINIBAND SUBNET MANAGER

All InfiniBand networks must have a subnet manager running for the network to function. This is true even if two machines are connected directly with no switch involved.

It is possible to have more than one subnet manager. In that case, one acts as a master and another subnet manager acts as a slave that will take over in case the master subnet manager fails.

Red Hat Enterprise Linux provides **OpenSM**, an implementation of an InfiniBand subnet manager. However, the features of **OpenSM** are limited and there is no active upstream development. Typically, embedded subnet managers in InfiniBand switches provide more features and support up-to-date InfiniBand hardware. For further details, see [Installing and configuring the OpenSM InfiniBand subnet manager](#).