



Red Hat Enterprise Linux 8

Configuring and managing networking

Managing network interfaces, firewalls, and advanced networking features

Red Hat Enterprise Linux 8 Configuring and managing networking

Managing network interfaces, firewalls, and advanced networking features

Legal Notice

Copyright © 2022 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

The networking capabilities of Red Hat Enterprise Linux (RHEL) enable you to configure your host to meet your organization's network and security requirements. You can configure Bonds, VLANs, bridges, tunnels and other network types to connect the host to the network. You can build complex and performance-critical firewalls for the local host and the entire network. Packet filtering software, such as the firewalld service, the nftables framework, and Express Data Path (XDP). RHEL also supports advanced networking features. For example, with policy-based routing, you can set up complex routing scenarios, and MultiPath TCP (MPTCP) enables clients to roam among different networks without interrupting the TCP connection to the application running on your RHEL server.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	12
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	13
CHAPTER 1. CONSISTENT NETWORK INTERFACE DEVICE NAMING	14
1.1. NETWORK INTERFACE DEVICE NAMING HIERARCHY	14
1.2. HOW THE NETWORK DEVICE RENAMING WORKS	15
1.3. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE X86_64 PLATFORM EXPLAINED	16
1.4. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE SYSTEM Z PLATFORM EXPLAINED	16
1.5. CUSTOMIZING THE PREFIX OF ETHERNET INTERFACES DURING THE INSTALLATION	17
1.6. ASSIGNING USER-DEFINED NETWORK INTERFACE NAMES USING UDEV RULES	18
1.7. ASSIGNING USER-DEFINED NETWORK INTERFACE NAMES USING SYSTEMD LINK FILES	19
CHAPTER 2. CONFIGURING AN ETHERNET CONNECTION	21
2.1. CONFIGURING A STATIC ETHERNET CONNECTION USING NMCLI	21
2.2. CONFIGURING A STATIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR	23
2.3. CONFIGURING A STATIC ETHERNET CONNECTION USING NMTUI	25
2.4. CONFIGURING A STATIC ETHERNET CONNECTION USING NMSTATECTL	27
2.5. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME	29
2.6. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH	30
2.7. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMCLI	32
2.8. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR	33
2.9. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMTUI	35
2.10. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMSTATECTL	36
2.11. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME	38
2.12. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH	39
2.13. CONFIGURING AN ETHERNET CONNECTION USING CONTROL-CENTER	40
2.14. CONFIGURING AN ETHERNET CONNECTION USING NM-CONNECTION-EDITOR	43
2.15. CHANGING THE DHCP CLIENT OF NETWORKMANAGER	45
2.16. CONFIGURING THE DHCP BEHAVIOR OF A NETWORKMANAGER CONNECTION	46
2.17. CONFIGURING MULTIPLE ETHERNET INTERFACES USING A SINGLE CONNECTION PROFILE BY INTERFACE NAME	47
2.18. CONFIGURING A SINGLE CONNECTION PROFILE FOR MULTIPLE ETHERNET INTERFACES USING PCI IDS	48
CHAPTER 3. MANAGING WIFI CONNECTIONS	50
3.1. SUPPORTED WIFI SECURITY TYPES	50
3.2. CONNECTING TO A WPA2 OR WPA3 PERSONAL-PROTECTED WIFI NETWORK USING NMCLI COMMANDS	51
3.3. CONFIGURING A WIFI CONNECTION USING NMTUI	52
3.4. CONNECTING TO A WIFI NETWORK USING THE GNOME SYSTEM MENU	54
3.5. CONNECTING TO A WIFI NETWORK USING THE GNOME SETTINGS APPLICATION	55
3.6. CONFIGURING A WIFI CONNECTION USING NM-CONNECTION-EDITOR	57
3.7. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING THE RHEL SYSTEM ROLES	58
3.8. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING WIFI CONNECTION USING NMCLI	60
3.9. MANUALLY SETTING THE WIRELESS REGULATORY DOMAIN	61
CHAPTER 4. CONFIGURING RHEL AS A WIFI ACCESS POINT	63

4.1. IDENTIFYING WHETHER A WIFI DEVICE SUPPORTS THE ACCESS POINT MODE	63
4.2. CONFIGURING RHEL AS A WPA2 OR WPA3 PERSONAL ACCESS POINT	63
CHAPTER 5. CONFIGURING VLAN TAGGING	66
5.1. CONFIGURING VLAN TAGGING USING NMCLI COMMANDS	66
5.2. CONFIGURING VLAN TAGGING USING THE RHEL WEB CONSOLE	68
5.3. CONFIGURING VLAN TAGGING USING NMTUI	70
5.4. CONFIGURING VLAN TAGGING USING NM-CONNECTION-EDITOR	74
5.5. CONFIGURING VLAN TAGGING USING NMSTATECTL	76
5.6. CONFIGURING VLAN TAGGING USING RHEL SYSTEM ROLES	78
5.7. ADDITIONAL RESOURCES	80
CHAPTER 6. USING A VXLAN TO CREATE A VIRTUAL LAYER-2 DOMAIN FOR VMS	81
6.1. BENEFITS OF VXLANs	81
6.2. CONFIGURING THE ETHERNET INTERFACE ON THE HOSTS	82
6.3. CREATING A NETWORK BRIDGE WITH A VXLAN ATTACHED	83
6.4. CREATING A VIRTUAL NETWORK IN LIBVIRT WITH AN EXISTING BRIDGE	84
6.5. CONFIGURING VIRTUAL MACHINES TO USE VXLAN	85
CHAPTER 7. CONFIGURING A NETWORK BRIDGE	87
7.1. CONFIGURING A NETWORK BRIDGE USING NMCLI COMMANDS	87
7.2. CONFIGURING A NETWORK BRIDGE USING THE RHEL WEB CONSOLE	90
7.3. CONFIGURING A NETWORK BRIDGE USING NMTUI	92
7.4. CONFIGURING A NETWORK BRIDGE USING NM-CONNECTION-EDITOR	96
7.5. CONFIGURING A NETWORK BRIDGE USING NMSTATECTL	99
7.6. CONFIGURING A NETWORK BRIDGE USING RHEL SYSTEM ROLES	101
CHAPTER 8. CONFIGURING NETWORK TEAMING	103
8.1. UNDERSTANDING NETWORK TEAMING	103
8.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES	103
8.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES	104
8.4. UNDERSTANDING THE TEAMD SERVICE, RUNNERS, AND LINK-WATCHERS	105
8.5. CONFIGURING A NETWORK TEAM USING NMCLI COMMANDS	106
8.6. CONFIGURING A NETWORK TEAM USING THE RHEL WEB CONSOLE	109
8.7. CONFIGURING A NETWORK TEAM USING NM-CONNECTION-EDITOR	112
CHAPTER 9. CONFIGURING NETWORK BONDING	116
9.1. UNDERSTANDING NETWORK BONDING	116
9.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES	116
9.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES	117
9.4. UPSTREAM SWITCH CONFIGURATION DEPENDING ON THE BONDING MODES	118
9.5. CONFIGURING A NETWORK BOND USING NMCLI COMMANDS	118
9.6. CONFIGURING A NETWORK BOND USING THE RHEL WEB CONSOLE	121
9.7. CONFIGURING A NETWORK BOND USING NMTUI	124
9.8. CONFIGURING A NETWORK BOND USING NM-CONNECTION-EDITOR	128
9.9. CONFIGURING A NETWORK BOND USING NMSTATECTL	130
9.10. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES	133
9.11. CREATING A NETWORK BOND TO ENABLE SWITCHING BETWEEN AN ETHERNET AND WIRELESS CONNECTION WITHOUT INTERRUPTING THE VPN	134
9.12. THE DIFFERENT NETWORK BONDING MODES	137
9.13. THE XMIT_HASH_POLICY BONDING PARAMETER	139
CHAPTER 10. CONFIGURING A VPN CONNECTION	141
10.1. CONFIGURING A VPN CONNECTION WITH CONTROL-CENTER	141
10.2. CONFIGURING A VPN CONNECTION USING NM-CONNECTION-EDITOR	145

10.3. CONFIGURING AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION	148
10.4. CONFIGURING ESP HARDWARE OFFLOAD ON A BOND TO ACCELERATE AN IPSEC CONNECTION	149
CHAPTER 11. CONFIGURING IP TUNNELS	151
11.1. CONFIGURING AN IPIP TUNNEL USING NMCLI TO ENCAPSULATE IPV4 TRAFFIC IN IPV4 PACKETS	151
11.2. CONFIGURING A GRE TUNNEL USING NMCLI TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS	154
11.3. CONFIGURING A GRE TAP TUNNEL TO TRANSFER ETHERNET FRAMES OVER IPV4	156
11.4. ADDITIONAL RESOURCES	159
CHAPTER 12. CHANGING A HOSTNAME	160
12.1. CHANGING A HOSTNAME USING NMCLI	160
12.2. CHANGING A HOSTNAME USING HOSTNAMECTL	160
CHAPTER 13. LEGACY NETWORK SCRIPTS SUPPORT IN RHEL	162
13.1. INSTALLING THE LEGACY NETWORK SCRIPTS	162
CHAPTER 14. PORT MIRRORING	163
14.1. MIRRORING A NETWORK INTERFACE USING NMCLI	163
CHAPTER 15. CONFIGURING NETWORKMANAGER TO IGNORE CERTAIN DEVICES	165
15.1. PERMANENTLY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER	165
15.2. TEMPORARILY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER	166
CHAPTER 16. CONFIGURING NETWORK DEVICES TO ACCEPT TRAFFIC FROM ALL MAC ADDRESSES	167
16.1. TEMPORARILY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING IPROUTE2	167
16.2. PERMANENTLY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMCLI	168
16.3. PERMANENTLY CONFIGURING A NETWORK NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMSTATECTL	169
CHAPTER 17. SETTING UP AN 802.1X NETWORK AUTHENTICATION SERVICE FOR LAN CLIENTS USING HOSTAPD WITH FREERADIUS BACKEND	170
17.1. PREREQUISITES	170
17.2. SETTING UP THE BRIDGE ON THE AUTHENTICATOR	170
17.3. CERTIFICATE REQUIREMENTS BY FREERADIUS	171
17.4. CREATING A SET OF CERTIFICATES ON A FREERADIUS SERVER FOR TESTING PURPOSES	172
17.5. CONFIGURING FREERADIUS TO AUTHENTICATE NETWORK CLIENTS SECURELY USING EAP	174
17.6. CONFIGURING HOSTAPD AS AN AUTHENTICATOR IN A WIRED NETWORK	177
17.7. TESTING EAP-TTLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR	180
17.8. TESTING EAP-TLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR	181
17.9. BLOCKING AND ALLOWING TRAFFIC BASED ON HOSTAPD AUTHENTICATION EVENTS	183
CHAPTER 18. AUTHENTICATING A RHEL CLIENT TO THE NETWORK USING THE 802.1X STANDARD WITH A CERTIFICATE STORED ON THE FILE SYSTEM	186
18.1. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING ETHERNET CONNECTION USING NMCLI	186
18.2. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING NMSTATECTL	187
18.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES	189
CHAPTER 19. MANAGING THE DEFAULT GATEWAY SETTING	192
19.1. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMCLI	192
19.2. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING THE NMCLI INTERACTIVE MODE	193

19.3. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NM-CONNECTION-EDITOR	194
19.4. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING CONTROL-CENTER	196
19.5. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMSTATECTL	197
19.6. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING RHEL SYSTEM ROLES	198
19.7. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION WHEN USING THE LEGACY NETWORK SCRIPTS	199
19.8. HOW NETWORKMANAGER MANAGES MULTIPLE DEFAULT GATEWAYS	200
19.9. CONFIGURING NETWORKMANAGER TO AVOID USING A SPECIFIC PROFILE TO PROVIDE A DEFAULT GATEWAY	201
19.10. FIXING UNEXPECTED ROUTING BEHAVIOR DUE TO MULTIPLE DEFAULT GATEWAYS	202
CHAPTER 20. CONFIGURING STATIC ROUTES	204
20.1. EXAMPLE OF A NETWORK THAT REQUIRES STATIC ROUTES	204
20.2. HOW TO USE THE NMCLI COMMAND TO CONFIGURE A STATIC ROUTE	206
20.3. CONFIGURING A STATIC ROUTE USING AN NMCLI COMMAND	207
20.4. CONFIGURING A STATIC ROUTE USING NMTUI	208
20.5. CONFIGURING A STATIC ROUTE USING CONTROL-CENTER	210
20.6. CONFIGURING A STATIC ROUTE USING NM-CONNECTION-EDITOR	212
20.7. CONFIGURING A STATIC ROUTE USING THE NMCLI INTERACTIVE MODE	213
20.8. CONFIGURING A STATIC ROUTE USING NMSTATECTL	215
20.9. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES	216
20.10. CREATING STATIC ROUTES CONFIGURATION FILES IN KEY-VALUE FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS	218
20.11. CREATING STATIC ROUTES CONFIGURATION FILES IN IP-COMMAND FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS	219
CHAPTER 21. CONFIGURING POLICY-BASED ROUTING TO DEFINE ALTERNATIVE ROUTES	222
21.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING NETWORKMANAGER	222
21.2. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING RHEL SYSTEM ROLES	226
21.3. OVERVIEW OF CONFIGURATION FILES INVOLVED IN POLICY-BASED ROUTING WHEN USING THE LEGACY NETWORK SCRIPTS	229
21.4. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING THE LEGACY NETWORK SCRIPTS	230
CHAPTER 22. CREATING A DUMMY INTERFACE	236
22.1. CREATING A DUMMY INTERFACE WITH BOTH AN IPV4 AND IPV6 ADDRESS USING NMCLI	236
CHAPTER 23. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE THE NETWORK STATE USING LLDP	237
23.1. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE NETWORK INTERFACES	237
CHAPTER 24. USING LLDP TO DEBUG NETWORK CONFIGURATION PROBLEMS	240
24.1. DEBUGGING AN INCORRECT VLAN CONFIGURATION USING LLDP INFORMATION	240
CHAPTER 25. MANUALLY CREATING NETWORKMANAGER PROFILES IN KEYFILE FORMAT	243
25.1. THE KEYFILE FORMAT OF NETWORKMANAGER PROFILES	243
25.2. CREATING A NETWORKMANAGER PROFILE IN KEYFILE FORMAT	244
25.3. MIGRATING NETWORKMANAGER PROFILES FROM IFCFG TO KEYFILE FORMAT	245
25.4. USING NMCLI TO CREATE KEYFILE CONNECTION PROFILES IN OFFLINE MODE	246
CHAPTER 26. SYSTEMD NETWORK TARGETS AND SERVICES	250
26.1. DIFFERENCES BETWEEN THE NETWORK AND NETWORK-ONLINE SYSTEMD TARGET	250
26.2. OVERVIEW OF NETWORKMANAGER-WAIT-ONLINE	250

26.3. CONFIGURING A SYSTEMD SERVICE TO START AFTER THE NETWORK HAS BEEN STARTED	251
CHAPTER 27. LINUX TRAFFIC CONTROL	252
27.1. OVERVIEW OF QUEUING DISCIPLINES	252
27.2. AVAILABLE QDISCS IN RHEL	252
27.3. INSPECTING QDISCS OF A NETWORK INTERFACE USING THE TC UTILITY	254
27.4. UPDATING THE DEFAULT QDISC	255
27.5. TEMPORARILY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE USING THE TC UTILITY	256
27.6. PERMANENTLY SETTING THE CURRENT QDISC OF A NETWORK INTERFACE USING NETWORKMANAGER	256
CHAPTER 28. GETTING STARTED WITH MULTIPATH TCP	258
28.1. UNDERSTANDING MPTCP	258
28.2. PREPARING RHEL TO ENABLE MPTCP SUPPORT	258
28.3. USING IPROUTE2 TO TEMPORARILY CONFIGURE AND ENABLE MULTIPLE PATHS FOR MPTCP APPLICATIONS	261
28.4. PERMANENTLY CONFIGURING MULTIPLE PATHS FOR MPTCP APPLICATIONS	263
28.5. MONITORING MPTCP SUB-FLOWS	265
28.6. DISABLING MULTIPATH TCP IN THE KERNEL	268
CHAPTER 29. CONFIGURING THE ORDER OF DNS SERVERS	269
29.1. HOW NETWORKMANAGER ORDERS DNS SERVERS IN /ETC/RESOLV.CONF	269
Default values of DNS priority parameters	269
Valid DNS priority values:	269
29.2. SETTING A NETWORKMANAGER-WIDE DEFAULT DNS SERVER PRIORITY VALUE	270
29.3. SETTING THE DNS PRIORITY OF A NETWORKMANAGER CONNECTION	271
CHAPTER 30. CONFIGURING IP NETWORKING WITH IFCFG FILES	272
30.1. CONFIGURING AN INTERFACE WITH STATIC NETWORK SETTINGS USING IFCFG FILES	272
30.2. CONFIGURING AN INTERFACE WITH DYNAMIC NETWORK SETTINGS USING IFCFG FILES	273
30.3. MANAGING SYSTEM-WIDE AND PRIVATE CONNECTION PROFILES WITH IFCFG FILES	273
CHAPTER 31. USING NETWORKMANAGER TO DISABLE IPV6 FOR A SPECIFIC CONNECTION	275
31.1. DISABLING IPV6 ON A CONNECTION USING NMCLI	275
CHAPTER 32. MANUALLY CONFIGURING THE /ETC/RESOLV.CONF FILE	277
32.1. DISABLING DNS PROCESSING IN THE NETWORKMANAGER CONFIGURATION	277
32.2. REPLACING /ETC/RESOLV.CONF WITH A SYMBOLIC LINK TO MANUALLY CONFIGURE DNS SETTINGS	278
CHAPTER 33. MONITORING AND TUNING NIC RING BUFFERS	279
33.1. DISPLAYING THE NUMBER OF DROPPED PACKETS	279
33.2. INCREASING THE RING BUFFERS TO REDUCE A HIGH PACKET DROP RATE	279
CHAPTER 34. CONFIGURING 802.3 LINK SETTINGS	281
34.1. UNDERSTANDING AUTO-NEGOTIATION	281
34.2. CONFIGURING 802.3 LINK SETTINGS USING THE NMCLI UTILITY	281
CHAPTER 35. CONFIGURING ETHTOOL OFFLOAD FEATURES	283
35.1. OFFLOAD FEATURES SUPPORTED BY NETWORKMANAGER	283
35.2. CONFIGURING AN ETHTOOL OFFLOAD FEATURE USING NETWORKMANAGER	285
35.3. USING RHEL SYSTEM ROLES TO SET ETHTOOL FEATURES	285
CHAPTER 36. CONFIGURING ETHTOOL COALESCE SETTINGS	288
36.1. COALESCE SETTINGS SUPPORTED BY NETWORKMANAGER	288

36.2. CONFIGURING ETHTOOL COALESCE SETTINGS USING NETWORKMANAGER	289
36.3. USING RHEL SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS	289
CHAPTER 37. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK	292
37.1. CONFIGURING A MACSEC CONNECTION USING NMCLI	292
37.2. ADDITIONAL RESOURCES	294
CHAPTER 38. USING DIFFERENT DNS SERVERS FOR DIFFERENT DOMAINS	295
38.1. SENDING DNS REQUESTS FOR A SPECIFIC DOMAIN TO A SELECTED DNS SERVER	295
CHAPTER 39. GETTING STARTED WITH IPVLAN	297
39.1. IPVLAN MODES	297
39.2. COMPARISON OF IPVLAN AND MACVLAN	297
39.3. CREATING AND CONFIGURING THE IPVLAN DEVICE USING IPROUTE2	298
CHAPTER 40. REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES	300
40.1. PERMANENTLY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES	300
40.2. TEMPORARILY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES	301
40.3. ADDITIONAL RESOURCES	303
CHAPTER 41. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK	304
41.1. CONFIGURING A VRF DEVICE	304
41.2. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK	305
CHAPTER 42. RUNNING DHCLIENT EXIT HOOKS USING NETWORKMANAGER A DISPATCHER SCRIPT	308
42.1. THE CONCEPT OF NETWORKMANAGER DISPATCHER SCRIPTS	308
42.2. CREATING A NETWORKMANAGER DISPATCHER SCRIPT THAT RUNS DHCLIENT EXIT HOOKS	308
CHAPTER 43. INTRODUCTION TO NETWORKMANAGER DEBUGGING	310
43.1. DEBUGGING LEVELS AND DOMAINS	310
43.2. SETTING THE NETWORKMANAGER LOG LEVEL	310
43.3. TEMPORARILY SETTING LOG LEVELS AT RUN TIME USING NMCLI	311
43.4. VIEWING NETWORKMANAGER LOGS	312
CHAPTER 44. INTRODUCTION TO NMSTATE	313
44.1. USING THE LIBNMSTATE LIBRARY IN A PYTHON APPLICATION	313
44.2. UPDATING THE CURRENT NETWORK CONFIGURATION USING NMSTATECTL	313
44.3. NETWORK STATES FOR THE NETWORK RHEL SYSTEM ROLE	314
44.4. ADDITIONAL RESOURCES	315
CHAPTER 45. CAPTURING NETWORK PACKETS	316
45.1. USING XDPDUMP TO CAPTURE NETWORK PACKETS INCLUDING PACKETS DROPPED BY XDP PROGRAMS	316
45.2. ADDITIONAL RESOURCES	317
CHAPTER 46. USING AND CONFIGURING FIREWALLD	318
46.1. GETTING STARTED WITH FIREWALLD	318
46.1.1. When to use firewalld, nftables, or iptables	318
46.1.2. Zones	318
46.1.3. Predefined services	320
46.1.4. Starting firewalld	320
46.1.5. Stopping firewalld	320
46.1.6. Verifying the permanent firewalld configuration	321
46.2. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD	321
46.2.1. Viewing the current status of firewalld	321
46.2.2. Viewing allowed services using GUI	322

46.2.3. Viewing firewalld settings using CLI	322
46.3. CONTROLLING NETWORK TRAFFIC USING FIREWALLD	323
46.3.1. Disabling all traffic in case of emergency using CLI	323
46.3.2. Controlling traffic with predefined services using CLI	324
46.3.3. Controlling traffic with predefined services using GUI	324
46.3.4. Adding new services	325
46.3.5. Opening ports using GUI	326
46.3.6. Controlling traffic with protocols using GUI	326
46.3.7. Opening source ports using GUI	327
46.4. CONTROLLING PORTS USING CLI	327
46.4.1. Opening a port	327
46.4.2. Closing a port	328
46.5. WORKING WITH FIREWALLD ZONES	328
46.5.1. Listing zones	328
46.5.2. Modifying firewalld settings for a certain zone	329
46.5.3. Changing the default zone	329
46.5.4. Assigning a network interface to a zone	329
46.5.5. Assigning a zone to a connection using nmcli	330
46.5.6. Manually assigning a zone to a network connection in an ifcfg file	330
46.5.7. Creating a new zone	330
46.5.8. Zone configuration files	331
46.5.9. Using zone targets to set default behavior for incoming traffic	331
46.6. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE	332
46.6.1. Adding a source	332
46.6.2. Removing a source	333
46.6.3. Adding a source port	333
46.6.4. Removing a source port	333
46.6.5. Using zones and sources to allow a service for only a specific domain	333
46.7. FILTERING FORWARDED TRAFFIC BETWEEN ZONES	334
46.7.1. The relationship between policy objects and zones	335
46.7.2. Using priorities to sort policies	335
46.7.3. Using policy objects to filter traffic between locally hosted Containers and a network physically connected to the host	335
46.7.4. Setting the default target of policy objects	336
46.8. CONFIGURING NAT USING FIREWALLD	336
46.8.1. NAT types	337
46.8.2. Configuring IP address masquerading	337
46.9. USING DNAT TO FORWARD HTTPS TRAFFIC TO A DIFFERENT HOST	338
46.10. MANAGING ICMP REQUESTS	340
46.10.1. Listing and blocking ICMP requests	340
46.10.2. Configuring the ICMP filter using GUI	342
46.11. SETTING AND CONTROLLING IP SETS USING FIREWALLD	342
46.11.1. Configuring IP set options using CLI	342
46.12. PRIORITIZING RICH RULES	344
46.12.1. How the priority parameter organizes rules into different chains	344
46.12.2. Setting the priority of a rich rule	344
46.13. CONFIGURING FIREWALL LOCKDOWN	345
46.13.1. Configuring lockdown using CLI	345
46.13.2. Configuring lockdown allowlist options using CLI	345
46.13.3. Configuring lockdown allowlist options using configuration files	347
46.14. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE	348
46.14.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT	348

46.14.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network	348
46.15. CONFIGURING FIREWALLD USING SYSTEM ROLES	349
46.15.1. Introduction to the firewall RHEL System Role	350
46.15.2. Resetting the firewalld settings using the firewall RHEL System Role	350
46.15.3. Forwarding incoming traffic from one local port to a different local port	351
46.15.4. Configuring ports using System Roles	352
46.15.5. Configuring a DMZ firewalld zone by using the firewalld RHEL System Role	353
46.16. ADDITIONAL RESOURCES	355
CHAPTER 47. GETTING STARTED WITH NFTABLES	356
47.1. MIGRATING FROM IPTABLES TO NFTABLES	356
47.1.1. When to use firewalld, nftables, or iptables	356
47.1.2. Converting iptables and ip6tables rule sets to nftables	356
47.1.3. Converting single iptables and ip6tables rules to nftables	357
47.1.4. Comparison of common iptables and nftables commands	358
47.1.5. Additional resources	359
47.2. WRITING AND EXECUTING NFTABLES SCRIPTS	359
47.2.1. Supported nftables script formats	359
47.2.2. Running nftables scripts	360
47.2.3. Using comments in nftables scripts	361
47.2.4. Using variables in nftables script	361
47.2.5. Including files in nftables scripts	362
47.2.6. Automatically loading nftables rules when the system boots	362
47.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES	363
47.3.1. Basics of nftables tables	363
47.3.2. Basics of nftables chains	364
Chain types	364
Chain priorities	364
Chain policies	365
47.3.3. Basics of nftables rules	365
47.3.4. Managing tables, chains, and rules using nft commands	366
47.4. CONFIGURING NAT USING NFTABLES	368
47.4.1. NAT types	368
47.4.2. Configuring masquerading using nftables	369
47.4.3. Configuring source NAT using nftables	370
47.4.4. Configuring destination NAT using nftables	370
47.4.5. Configuring a redirect using nftables	371
47.5. USING SETS IN NFTABLES COMMANDS	372
47.5.1. Using anonymous sets in nftables	372
47.5.2. Using named sets in nftables	372
47.5.3. Additional resources	374
47.6. USING VERDICT MAPS IN NFTABLES COMMANDS	374
47.6.1. Using anonymous maps in nftables	374
47.6.2. Using named maps in nftables	375
47.6.3. Additional resources	377
47.7. EXAMPLE: PROTECTING A LAN AND DMZ USING AN NFTABLES SCRIPT	377
47.7.1. Network conditions	377
47.7.2. Security requirements to the firewall script	378
47.7.3. Configuring logging of dropped packets to a file	378
47.7.4. Writing and activating the nftables script	379
47.8. CONFIGURING PORT FORWARDING USING NFTABLES	382
47.8.1. Forwarding incoming packets to a different local port	382
47.8.2. Forwarding incoming packets on a specific local port to a different host	383

47.9. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS	383
47.9.1. Limiting the number of connections using nftables	384
47.9.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute	384
47.10. DEBUGGING NFTABLES RULES	385
47.10.1. Creating a rule with a counter	385
47.10.2. Adding a counter to an existing rule	386
47.10.3. Monitoring packets that match an existing rule	386
47.11. BACKING UP AND RESTORING THE NFTABLES RULE SET	387
47.11.1. Backing up the nftables rule set to a file	387
47.11.2. Restoring the nftables rule set from a file	388
47.12. ADDITIONAL RESOURCES	388
CHAPTER 48. USING XDP-FILTER FOR HIGH-PERFORMANCE TRAFFIC FILTERING TO PREVENT DDOS ATTACKS	389
48.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE	389
48.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE	390
CHAPTER 49. GETTING STARTED WITH DPDK	393
49.1. INSTALLING THE DPDK PACKAGE	393
49.2. ADDITIONAL RESOURCES	393
CHAPTER 50. UNDERSTANDING THE EBPf NETWORKING FEATURES IN RHEL 8	394
50.1. OVERVIEW OF NETWORKING EBPf FEATURES IN RHEL 8	394
XDP	394
AF_XDP	395
Traffic Control	396
Socket filter	396
Control Groups	396
Stream Parser	397
SO_REUSEPORT socket selection	397
Flow dissector	397
TCP Congestion Control	397
Routes with encapsulation	397
Socket lookup	398
50.2. OVERVIEW OF XDP FEATURES IN RHEL 8 BY NETWORK CARDS	398
CHAPTER 51. NETWORK TRACING USING THE BPF COMPILER COLLECTION	400
51.1. AN INTRODUCTION TO BCC	400
51.2. INSTALLING THE BCC-TOOLS PACKAGE	400
51.3. DISPLAYING TCP CONNECTIONS ADDED TO THE KERNEL'S ACCEPT QUEUE	401
51.4. TRACING OUTGOING TCP CONNECTION ATTEMPTS	401
51.5. MEASURING THE LATENCY OF OUTGOING TCP CONNECTIONS	402
51.6. DISPLAYING DETAILS ABOUT TCP PACKETS AND SEGMENTS THAT WERE DROPPED BY THE KERNEL	402
51.7. TRACING TCP SESSIONS	403
51.8. TRACING TCP RETRANSMISSIONS	404
51.9. DISPLAYING TCP STATE CHANGE INFORMATION	404
51.10. SUMMARIZING AND AGGREGATING TCP TRAFFIC SENT TO SPECIFIC SUBNETS	405
51.11. DISPLAYING THE NETWORK THROUGHPUT BY IP ADDRESS AND PORT	406
51.12. TRACING ESTABLISHED TCP CONNECTIONS	406
51.13. TRACING IPV4 AND IPV6 LISTEN ATTEMPTS	407
51.14. SUMMARIZING THE SERVICE TIME OF SOFT INTERRUPTS	407
51.15. ADDITIONAL RESOURCES	408

CHAPTER 52. GETTING STARTED WITH TIPC	409
52.1. THE ARCHITECTURE OF TIPC	409
52.2. LOADING THE TIPC MODULE WHEN THE SYSTEM BOOTS	409
52.3. CREATING A TIPC NETWORK	410
52.4. ADDITIONAL RESOURCES	411
CHAPTER 53. AUTOMATICALLY CONFIGURING NETWORK INTERFACES IN PUBLIC CLOUDS USING NM-CLOUD-SETUP	412
53.1. CONFIGURING AND PRE-DEPLOYING NM-CLOUD-SETUP	412

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the [Bugzilla](#) website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. CONSISTENT NETWORK INTERFACE DEVICE NAMING

The Linux kernel assigns names to network interfaces by combining a fixed prefix and a number that increases as the kernel initializes the network devices. For instance, **eth0** represents the first device being probed on start-up. If you add another network interface card to the system, the assignment of the kernel device names is no longer fixed. Consequently, after a reboot, the kernel can name the device differently.

To solve this problem, the **udev** device manager supports a number of different naming schemes. By default, **udev** assigns fixed names based on firmware, topology, and location information. This has the following advantages:

- Device names are fully predictable.
- Device names stay fixed even if you add or remove hardware, because no re-enumeration takes place.
- Defective hardware can be seamlessly replaced.



WARNING

Red Hat does not support systems with consistent device naming disabled. For further details, see [ls it safe to set net.ifnames=0?](#)

1.1. NETWORK INTERFACE DEVICE NAMING HIERARCHY

If consistent device naming is enabled, which is the default in Red Hat Enterprise Linux, the **udev** device manager generates device names based on the following schemes:

Scheme	Description	Example
1	Device names incorporate firmware or BIOS-provided index numbers for onboard devices. If this information is not available or applicable, udev uses scheme 2.	eno1
2	Device names incorporate firmware or BIOS-provided PCI Express (PCIe) hot plug slot index numbers. If this information is not available or applicable, udev uses scheme 3.	ens1
3	Device names incorporate the physical location of the connector of the hardware. If this information is not available or applicable, udev uses scheme 5.	enp2s0
4	Device names incorporate the MAC address. Red Hat Enterprise Linux does not use this scheme by default, but administrators can optionally use it.	enx525400d5e0fb

Scheme	Description	Example
5	The traditional unpredictable kernel naming scheme. If udev cannot apply any of the other schemes, the device manager uses this scheme.	eth0

By default, Red Hat Enterprise Linux selects the device name based on the **NamePolicy** setting in the **/usr/lib/systemd/network/99-default.link** file. The order of the values in **NamePolicy** is important. Red Hat Enterprise Linux uses the first device name that is both specified in the file and that **udev** generated.

If you manually configured **udev** rules to change the name of kernel devices, those rules take precedence.

1.2. HOW THE NETWORK DEVICE RENAMING WORKS

By default, consistent device naming is enabled in Red Hat Enterprise Linux. The **udev** device manager processes different rules to rename the devices. The **udev** service processes these rules in the following order:

1. The **/usr/lib/udev/rules.d/60-net.rules** file defines that the **/lib/udev/rename_device** helper utility searches for the **HWADDR** parameter in **/etc/sysconfig/network-scripts/ifcfg-*** files. If the value set in the variable matches the MAC address of an interface, the helper utility renames the interface to the name set in the **DEVICE** parameter of the file.
2. The **/usr/lib/udev/rules.d/71-biosdevname.rules** file defines that the **biosdevname** utility renames the interface according to its naming policy, provided that it was not renamed in the previous step.
3. The **/usr/lib/udev/rules.d/75-net-description.rules** file defines that **udev** examines the network interface device and sets the properties in **udev**-internal variables that will be processed in the next step. Note that some of these properties might be undefined.
4. The **/usr/lib/udev/rules.d/80-net-setup-link.rules** file calls the **net_setup_link udev** built-in which then applies the policy. The following is the default policy that is stored in the **/usr/lib/systemd/network/99-default.link** file:

```
[Link]
NamePolicy=kernel database onboard slot path
MACAddressPolicy=persistent
```

With this policy, if the kernel uses a persistent name, **udev** does not rename the interface. If the kernel does not use a persistent name, **udev** renames the interface to the name provided by the hardware database of **udev**. If this database is not available, Red Hat Enterprise Linux falls back to the mechanisms described above.

Alternatively, set the **NamePolicy** parameter in this file to **mac** for media access control (MAC) address-based interface names.

5. The **/usr/lib/udev/rules.d/80-net-setup-link.rules** file defines that **udev** renames the interface based on the **udev**-internal parameters in the following order:
 - a. **ID_NET_NAME_ONBOARD**
 - b. **ID_NET_NAME_SLOT**

c. `ID_NET_NAME_PATH`

If one parameter is not set, **udev** uses the next one. If none of the parameters are set, the interface is not renamed.

Steps 3 and 4 implement the naming schemes 1 to 4 described in [Network interface device naming hierarchy](#).

Additional resources

- [Customizing the prefix of Ethernet interfaces during the installation](#)
- **systemd.link(5)** man page

1.3. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE X86_64 PLATFORM EXPLAINED

When the consistent network device name feature is enabled, the **udev** device manager creates the names of devices based on different criteria. The interface name starts with a two-character prefix based on the type of interface:

- **en** for Ethernet
- **wl** for wireless LAN (WLAN)
- **ww** for wireless wide area network (WWAN)

Additionally, one of the following is appended to one of the above-mentioned prefix based on the schema the **udev** device manager applies:

- **o<on-board_index_number>**
- **s<hot_plug_slot_index_number>[f<function>][d<device_id>]**
Note that all multi-function PCI devices have the **[f<function>]** number in the device name, including the function **0** device.
- **x<MAC_address>**
- **[P<domain_number>]p<bus>s<slot>[f<function>][d<device_id>]**
The **[P<domain_number>]** part defines the PCI geographical location. This part is only set if the domain number is not **0**.
- **[P<domain_number>]p<bus>s<slot>[f<function>][u<usb_port>][...][c<config>][i<interface>]**
For USB devices, the full chain of port numbers of hubs is composed. If the name is longer than the maximum (15 characters), the name is not exported. If there are multiple USB devices in the chain, **udev** suppresses the default values for USB configuration descriptors (**c1**) and USB interface descriptors (**i0**).

1.4. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE SYSTEM Z PLATFORM EXPLAINED

When the consistent network device name feature is enabled, the **udev** device manager on the System z platform creates the names of devices based on the bus ID. The bus ID identifies a device in the s390 channel subsystem.

For a channel command word (CCW) device, the bus ID is the device number with a leading **0.n** prefix where **n** is the subchannel set ID.

Ethernet interfaces are named, for example, **enccw0.0.1234**. Serial Line Internet Protocol (SLIP) channel-to-channel (CTC) network devices are named, for example, **slccw0.0.1234**.

Use the **znetconf -c** or the **lscss -a** commands to display available network devices and their bus IDs.

Red Hat Enterprise Linux also supports predictable and persistent interface names for RDMA over Converged Ethernet (RoCE) Express PCI functions. Two identifiers provide predictable interface names: user identifier (UID) and function identifier (FID). On a system to get UID-based predictable interface names, enforce UID uniqueness, which is the preferred naming scheme. If no unique UIDs are available, then RHEL uses FIDs to set predictable interface names.

1.5. CUSTOMIZING THE PREFIX OF ETHERNET INTERFACES DURING THE INSTALLATION

You can customize the prefix of Ethernet interface names during the Red Hat Enterprise Linux installation.



IMPORTANT

Red Hat does not support customizing the prefix using the **prefixdevname** utility on already deployed systems.

After the RHEL installation, the **udev** service names Ethernet devices **<prefix>.<index>**. For example, if you select the prefix **net**, RHEL names Ethernet interfaces **net0**, **net1**, and so on.

Prerequisites

- The prefix you want to set meets the following requirements:
 - It consists of ASCII characters.
 - It is an alpha-numeric string.
 - It is shorter than 16 characters.
 - It does not conflict with any other well-known prefix used for network interface naming, such as **eth**, **eno**, **ens**, and **em**.

Procedure

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

Verification

- After the installation, display the Ethernet 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 00:53:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
3: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:c2:39:9e brd ff:ff:ff:ff:ff:ff
...
```

1.6. ASSIGNING USER-DEFINED NETWORK INTERFACE NAMES USING UDEV RULES

The **udev** device manager supports a set of rules to customize the interface names.

Procedure

- Display all network interfaces and their MAC addresses:

```
# ip link list

enp6s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
enp6s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
enp4s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff
```

- Create the file **/etc/udev/rules.d/70-custom-ifnames.rules** with the following contents:

```
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="b4:96:91:14:ae:58",ATTR{type}
=="1",NAME="provider0"
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="b4:96:91:14:ae:5a",ATTR{type}
=="1",NAME="provider1"
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="00:90:fa:6a:7d:90",ATTR{type}
=="1",NAME="dmz"
```

These rules match the MAC address of the network interfaces and rename them to the name given in the **NAME** property. In these examples, **ATTR{type}** parameter value **1** defines that the interface is of type Ethernet.

Verification

- Reboot the system.

```
# reboot
```

2. Verify that interface names for each MAC address match the value you set in the **NAME** parameter of the rule file:

ip link show

```

provider0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
    altname enp6s0f0
provider1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
    altname enp6s0f1
dmz: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff
    altname enp4s0f0

```

Additional resources

- **udev(7)** man page
- **udevadm(8)** man page
- **/usr/src/kernels/<kernel_version>/include/uapi/linux/if_arp.h** provided by the **kernel-doc** package

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

Create a naming scheme by renaming network interfaces to **provider0**.

Procedure

1. Display all interfaces names and their MAC addresses:

ip link show

```

enp6s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
enp6s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
enp4s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff

```

2. For naming the interface with MAC address **b4:96:91:14:ae:58** to **provider0**, create the **/etc/systemd/network/70-custom-ifnames.link** file with following contents:

[Match]

```
MACAddress=b4:96:91:14:ae:58
```

```
[Link]  
Name=provider0
```

This link file matches a MAC address and renames the network interface to the name set in the **Name** parameter.

Verification

1. Reboot the system:

```
# reboot
```

2. Verify that the device with the MAC address you specified in the link file has been assigned 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 b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
```

Additional resources

- **systemd.link(5)** man page

CHAPTER 2. CONFIGURING AN ETHERNET CONNECTION

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 RHEL System Roles to automate the configuration of connections on one or multiple hosts.
- Use the GNOME Settings menu or **nm-connection-editor** application to configure connections in a graphical interface.
- Use **nmstatectl** to configure connections through the Nmstate API.

2.1. CONFIGURING A STATIC ETHERNET CONNECTION USING NMCLI

This procedure describes adding an Ethernet connection with the following settings using the **nmcli** utility:

- 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**

Procedure

1. Add a new NetworkManager connection profile for the Ethernet connection:

```
# nmcli connection add con-name Example-Connection ifname enp7s0 type ethernet
```

The further steps modify the **Example-Connection** connection profile you created.

2. Set the IPv4 address:

```
# nmcli connection modify Example-Connection ipv4.addresses 192.0.2.1/24
```

3. Set the IPv6 address:

```
# nmcli connection modify Example-Connection ipv6.addresses 2001:db8:1::1/64
```

4. Set the IPv4 and IPv6 connection method to **manual**:

```
# nmcli connection modify Example-Connection ipv4.method manual
# nmcli connection modify Example-Connection ipv6.method manual
```

5. Set the IPv4 and IPv6 default gateways:

```
# nmcli connection modify Example-Connection ipv4.gateway 192.0.2.254
# nmcli connection modify Example-Connection ipv6.gateway 2001:db8:1::fffe
```

6. Set the IPv4 and IPv6 DNS server addresses:

```
# nmcli connection modify Example-Connection ipv4.dns "192.0.2.200"
# nmcli connection modify Example-Connection ipv6.dns "2001:db8:1::ffbb"
```

To set multiple DNS servers, specify them space-separated and enclosed in quotes.

7. Set the DNS search domain for the IPv4 and IPv6 connection:

```
# nmcli connection modify Example-Connection ipv4.dns-search example.com
# nmcli connection modify Example-Connection ipv6.dns-search example.com
```

8. Activate the connection profile:

```
# nmcli connection up Example-Connection
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/13)
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- **nm-settings(5)** man page
- **nmcli(1)** man page
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.2. CONFIGURING A STATIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR

This procedure describes adding an Ethernet connection with the following settings using the **nmcli** interactive mode:

- 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**

Procedure

1. To add a new NetworkManager connection profile for the Ethernet connection, and starting the interactive mode, enter:

```
# nmcli connection edit type ethernet con-name Example-Connection
```

2. Set the network interface:

```
nmcli> set connection.interface-name enp7s0
```

3. Set the IPv4 address:

```
nmcli> set ipv4.addresses 192.0.2.1/24
```

4. Set the IPv6 address:

```
nmcli> set ipv6.addresses 2001:db8:1::1/64
```

5. Set the IPv4 and IPv6 connection method to **manual**:

```
nmcli> set ipv4.method manual
nmcli> set ipv6.method manual
```

6. Set the IPv4 and IPv6 default gateways:

```
nmcli> set ipv4.gateway 192.0.2.254
nmcli> set ipv6.gateway 2001:db8:1::fffe
```

7. Set the IPv4 and IPv6 DNS server addresses:

```
nmcli> set ipv4.dns 192.0.2.200
nmcli> set ipv6.dns 2001:db8:1::ffbb
```

To set multiple DNS servers, specify them space-separated and enclosed in quotation marks.

8. Set the DNS search domain for the IPv4 and IPv6 connection:

```
nmcli> set ipv4.dns-search example.com
nmcli> set ipv6.dns-search example.com
```

9. Save and activate the connection:

```
nmcli> save persistent
Saving the connection with 'autoconnect=yes'. That might result in an immediate activation of
the connection.
Do you still want to save? (yes/no) [yes] yes
```

10. Leave the interactive mode:

```
nmcli> quit
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- **nm-settings(5)** man page
- **nmcli(1)** man page
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.3. CONFIGURING A STATIC ETHERNET CONNECTION USING NMTUI

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to configure an Ethernet connection with a static IP address 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 deselect checkboxes by using **Space**.

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
enp7s0  ethernet unavailable --
...
```

2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press the **Add** button.
5. Select **Ethernet** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
7. Enter the network device name into the **Device** field.
8. Configure the IPv4 and IPv6 address settings in the **IPv4 configuration** and **IPv6 configuration** areas:
 - a. Press the **Automatic** button, and select **Manual** from the displayed list.
 - b. Press the **Show** button next to the protocol you want to configure to display additional fields.

- c. Press the **Add** button 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.
- d. Enter the address of the default gateway.
- e. Press the **Add** button next to **DNS servers**, and enter the DNS server address.
- f. Press the **Add** button next to **Search domains**, and enter the DNS search domain.

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

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

☒ Automatically connect
☒ Available to all users

<Cancel> <OK>

9. Press the **OK** button to create and automatically activate the new connection.
10. Press the **Back** button to return to the main menu.

11. Select **Quit**, and press **Enter** to close the **nmtui** application.

Verification

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.4. CONFIGURING A STATIC ETHERNET CONNECTION USING NMSTATECTL

This procedure describes how to configure an Ethernet connection for the **enp7s0** device with the following settings using the **nmstatectl** utility:

- 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**

The **nmstatectl** utility 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.

The procedure defines the interface configuration in YAML format. Alternatively, you can also specify the configuration in JSON format.

Prerequisites

- The **nmstate** package is installed.

Procedure

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

```
---
interfaces:
- name: enp7s0
  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: enp7s0
      - destination: ::0
        next-hop-address: 2001:db8:1::fffe
        next-hop-interface: enp7s0
  dns-resolver:
    config:
      search:
        - example.com
      server:
        - 192.0.2.200
        - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

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

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected enp7s0
```

2. Display all settings of the connection profile:

```
# nmcli connection show enp7s0
connection.id:      enp7s0
connection.uuid:    b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id: --
connection.type:    802-3-ethernet
connection.interface-name: enp7s0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show enp7s0
```

Additional resources

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

2.5. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use the **network** RHEL System Role to remotely add an Ethernet connection for the **enp7s0** interface with the following settings by running an Ansible playbook:

- 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**

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- The managed nodes use NetworkManager to configure the network.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP
      include_role:
        name: rhel-system-roles.network

  vars:
    network_connections:
      - name: enp7s0
        interface_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
        state: up
```

2. Run the playbook:

```
# ansible-playbook ~/ethernet-static-IP.yml
```

Additional resources

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

2.6. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add an Ethernet connection with static IP address for devices that match a specific device path by running an Ansible playbook.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

This procedure sets the following settings to the device that matches the PCI ID **0000:00:0[1-3].0** expression, but not **0000:00:02.0**:

- 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**

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- The managed nodes use NetworkManager to configure the network.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with dynamic IP
      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 **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the [/usr/share/ansible/roles/rhel-system-roles.network/README.md](#) file.

2. Run the playbook:

```
# ansible-playbook ~/ethernet-static-IP.yml
```

Additional resources

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

2.7. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMCLI

This procedure describes adding an dynamic Ethernet connection using the **nmcli** utility. With this setting, NetworkManager requests the IP settings for this connection from a DHCP server.

Prerequisites

- A DHCP server is available in the network.

Procedure

1. Add a new NetworkManager connection profile for the Ethernet connection:

```
# nmcli connection add con-name Example-Connection ifname enp7s0 type ethernet
```

2. Optionally, change the host name NetworkManager sends to the DHCP server when using the **Example-Connection** profile:

```
# nmcli connection modify Example-Connection ipv4.dhcp-hostname Example
ipv6.dhcp-hostname Example
```

3. Optionally, change the client ID NetworkManager sends to an IPv4 DHCP server when using the **Example-Connection** profile:

```
# nmcli connection modify Example-Connection ipv4.dhcp-client-id client-ID
```

Note that there is no **dhcp-client-id** parameter for IPv6. To create an identifier for IPv6, configure the **dhclient** service.

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- **dhclient(8)** man page
- **nm-settings(5)**
- **nmcli(1)** man page
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.8. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR

This procedure describes adding an dynamic Ethernet connection using the interactive editor of the **nmcli** utility. With this setting, NetworkManager requests the IP settings for this connection from a DHCP server.

Prerequisites

- A DHCP server is available in the network.

Procedure

1. To add a new NetworkManager connection profile for the Ethernet connection, and starting the interactive mode, enter:

```
# nmcli connection edit type ethernet con-name Example-Connection
```

2. Set the network interface:

```
nmcli> set connection.interface-name enp7s0
```

3. Optionally, change the host name NetworkManager sends to the DHCP server when using the **Example-Connection** profile:

```
nmcli> set ipv4.dhcp-hostname Example
nmcli> set ipv6.dhcp-hostname Example
```

4. Optionally, change the client ID NetworkManager sends to an IPv4 DHCP server when using the **Example-Connection** profile:

```
nmcli> set ipv4.dhcp-client-id client-ID
```

Note that there is no **dhcp-client-id** parameter for IPv6. To create an identifier for IPv6, configure the **dhclient** service.

5. Save and activate the connection:

```
nmcli> save persistent
Saving the connection with 'autoconnect=yes'. That might result in an immediate activation of
the connection.
Do you still want to save? (yes/no) [yes] yes
```

6. Leave the interactive mode:

```
nmcli> quit
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE   TYPE   STATE   CONNECTION
enp7s0   ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- **nm-settings(5)** man page
- **nmcli(1)** man page
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.9. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMTUI

The **nmtui** application provides a text-based user interface for NetworkManager. You can use **nmtui** to configure an Ethernet connection with a dynamic IP address 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 deselect checkboxes by using **Space**.

Procedure

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

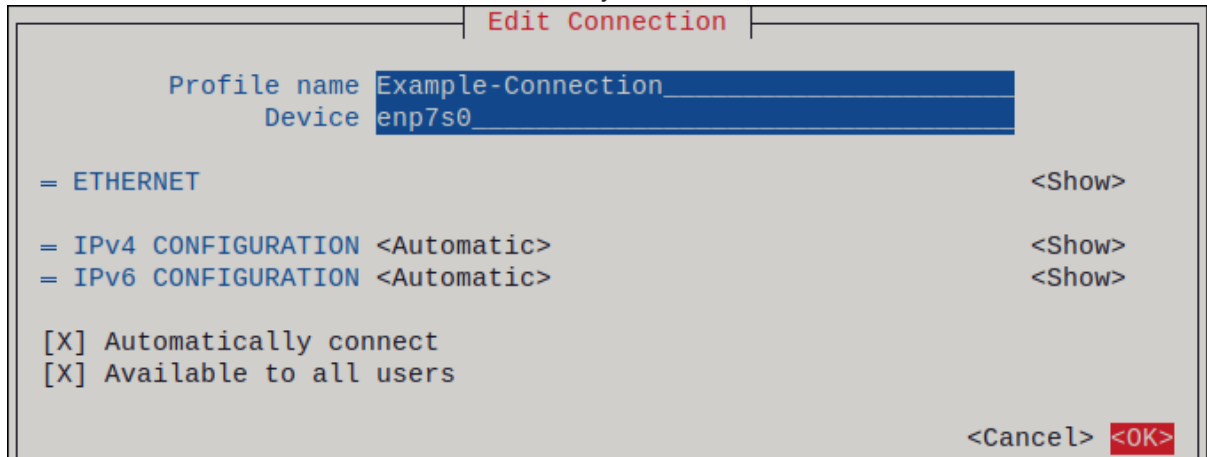
```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
...
```

2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press the **Add** button.
5. Select **Ethernet** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.

- Enter the network device name into the **Device** field.
- Press the **OK** button to create and automatically activate the new connection.



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

Verification

- Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting

- Make sure 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 defect 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 [NetworkManager duplicates a connection after restart of NetworkManager service](#).

Additional resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.10. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMSTATECTL

This procedure describes how to add a dynamic Ethernet for the **enp7s0** device using the **nmstatectl** utility. With the settings in this procedure, NetworkManager requests the IP settings for this connection from a DHCP server.

The **nmstatectl** utility 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.

The procedure defines the interface configuration in YAML format. Alternatively, you can also specify the configuration in JSON format.

Prerequisites

- The **nmstate** package is installed.

Procedure

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

```
---
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
```

2. Apply the settings to the system:

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

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE   TYPE   STATE   CONNECTION
enp7s0   ethernet connected enp7s0
```

2. Display all settings of the connection profile:

```
# nmcli connection show enp7s0
connection.id:      enp7s0_
```

```

connection.uuid:      b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id: --
connection.type:      802-3-ethernet
connection.interface-name: enp7s0
...

```

3. Display the connection settings in YAML format:

```
# nmstatectl show enp7s0
```

Additional resources

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

2.11. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for the **enp7s0** interface by running an Ansible playbook. With this setting, the network connection requests the IP settings for this connection from a DHCP server.

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- A DHCP server is available in the network
- The managed nodes use NetworkManager to configure the network.

Procedure

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

```

---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with dynamic IP
      include_role:
        name: rhel-system-roles.network

  vars:
    network_connections:

```

```
- name: enp7s0
  interface_name: enp7s0
  type: ethernet
  autoconnect: yes
  ip:
    dhcp4: yes
    auto6: yes
  state: up
```

2. Run the playbook:

```
# ansible-playbook ~/ethernet-dynamic-IP.yml
```

Additional resources

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

2.12. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for devices that match a specific device path by running an Ansible playbook. With dynamic IP settings, the network connection requests the IP settings for this connection from a DHCP server.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- A DHCP server is available in the network.
- The managed hosts use NetworkManager to configure the network.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with dynamic IP
```

```
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 **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the </usr/share/ansible/roles/rhel-system-roles.network/README.md> file.

2. Run the playbook:

```
# ansible-playbook ~/ethernet-dynamic-IP.yml
```

Additional resources

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

2.13. CONFIGURING AN ETHERNET CONNECTION USING CONTROL-CENTER

Ethernet connections are the most frequently used connections types in physical or virtual machines. If you use Red Hat Enterprise Linux with a graphical interface, you can configure this connection type in the GNOME **control-center**.

Note that **control-center** does not support as many configuration options as the **nm-connection-editor** application or the **nmcli** utility.

Prerequisites

- A physical or virtual Ethernet device 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. Click the **+** button next to the **Wired** entry to create a new profile.

- Optional: Set a name for the connection on the **Identity** tab.
- On the **IPv4** tab, configure the IPv4 settings. For example, select method **Manual**, set a static IPv4 address, network mask, default gateway, and DNS server:

The screenshot shows the 'New Profile' dialog box with the 'IPv4' tab selected. The 'IPv4 Method' section has three radio buttons: 'Automatic (DHCP)', 'Manual' (which is selected), and 'Link-Local Only'. There is also a 'Disable' option. Below this is the 'Addresses' section with a table for configuring static IP addresses. The first row is filled with '192.0.2.1' for the Address, '24' for the Netmask, and '192.0.2.254' for the Gateway. The second row is empty. To the right of the table is a button with a minus sign. Below the table is the 'DNS' section with a toggle switch set to 'ON' and a text field containing '192.0.2.1'. At the bottom, there is a note: 'Separate IP addresses with commas'.

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

- On the **IPv6** tab, configure the IPv6 settings. For example, select method **Manual**, set a static IPv6 address, network mask, default gateway, and DNS server:

The screenshot shows the 'New Profile' dialog box with the 'IPv6' tab selected. The 'IPv6 Method' section has four radio buttons: 'Automatic', 'Automatic, DHCP only', 'Link-Local Only', and 'Manual' (which is selected). There is also a 'Disable' option. Below this is the 'Addresses' section with a table:

Address	Prefix	Gateway	
2001:db8:1::1	64	2001:db8:1::ff3	✕
			✕

Below the table is the 'DNS' section with a text field containing '2001:db8:1::fffd' and a toggle switch for 'Automatic' which is currently 'ON'.

- Click the **Add** button to save the connection. The GNOME **control-center** automatically activates the connection.

Verification steps

- Display the status of the devices and connections:

```
# nmcli device status
DEVICE  TYPE  STATE  CONNECTION
enp7s0  ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Troubleshooting steps

- Make sure 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 defect 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 [NetworkManager duplicates a](#)

connection after restart of NetworkManager service.

Additional Resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.14. CONFIGURING AN ETHERNET CONNECTION USING NM-CONNECTION-EDITOR

Ethernet connections are the most frequently used connection types in physical or virtual servers. If you use Red Hat Enterprise Linux with a graphical interface, you can configure this connection type using the **nm-connection-editor** application.

Prerequisites

- A physical or virtual Ethernet device exists in the server's configuration.
- GNOME is installed.

Procedure

1. Open a terminal, and enter:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Ethernet** connection type, and click **Create**.
4. On the **General** tab:
 - a. To automatically enable this connection when the system boots or when you restart the **NetworkManager** service:
 - i. Select **Connect automatically with priority**.
 - ii. Optional: Change the priority value next to **Connect automatically with priority**.
If multiple connection profiles exist for the same device, NetworkManager enables only one profile. By default, NetworkManager activates the last-used profile that has auto-connect enabled. However, if you set priority values in the profiles, NetworkManager activates the profile with the highest priority.
 - b. Clear the **All users may connect to this network** check box if the profile should be available only to the user that created the connection profile.

Editing Example connection 1

Connection name: Example connection 1

General Ethernet 802.1X Security DCB Proxy IPv4 Settings IPv6 Settings

☒ Connect automatically with priority 100 - +

☒ All users may connect to this network

☐ Automatically connect to VPN

Metered connection: Automatic

5. On the **Ethernet** tab, select a device and, optionally, further Ethernet-related settings.

Editing Ethernet connection 1

Connection name: Ethernet connection 1

General **Ethernet** 802.1X Security DCB Proxy IPv4 Settings IPv6 Settings

Device: enp1s0 (52:54:00:6B:74:BE)

Cloned MAC address:

MTU: automatic - + bytes

Wake on LAN: ☒ Default ☐ Phy ☐ Unicast ☐ Multicast
☐ Ignore ☐ Broadcast ☐ Arp ☐ Magic

Wake on LAN password:

Link negotiation: Ignore

Speed: 100 Mb/s

Duplex: Full

6. On the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, and DNS server:

Method: Manual

Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

Add Delete

DNS servers: 192.0.2.1

- On the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, and DNS server:

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

Buttons: Add, Delete

DNS servers: 2001:db8:1::fffd

- Save the connection.
- Close **nm-connection-editor**.

Verification steps

- Display the status of the devices and connections:

```
# nmcli device status
DEVICE    TYPE    STATE    CONNECTION
enp7s0    ethernet connected Example-Connection
```

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

```
# ping host_name_or_IP_address
```

Additional Resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

2.15. CHANGING THE DHCP CLIENT OF NETWORKMANAGER

By default, NetworkManager uses its internal DHCP client. However, if you require a DHCP client with features that the built-in client does not provide, you can alternatively configure NetworkManager to use **dhclient**.

Note that RHEL does not provide **dhcpcd** and, therefore, NetworkManager can not use this client.

Procedure

- Create the **/etc/NetworkManager/conf.d/dhcp-client.conf** file with the following content:

```
[main]
dhcp=dhclient
```

You can set the **dhcp** parameter to **internal** (default) or **dhclient**.

- If you set the **dhcp** parameter to **dhclient**, install the **dhcp-client** package:

```
# yum install dhcp-client
```

- Restart NetworkManager:

```
# systemctl restart NetworkManager
```

Note that the restart temporarily interrupts all network connections.

Verification

- Search in the `/var/log/messages` log file for an entry similar to the following:

```
Apr 26 09:54:19 server NetworkManager[27748]: <info> [1650959659.8483] dhcp-init: Using
DHCP client 'dhclient'
```

This log entry confirms that NetworkManager uses **dhclient** as DHCP client.

Additional resources

- NetworkManager.conf(5)** man page

2.16. CONFIGURING THE DHCP 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 configured a connection to retrieve an IP address from a DHCP server, the NetworkManager requests an IP address from a DHCP server. By default, the client waits 45 seconds for this request to be completed. When a **DHCP** connection is started, a dhcp client requests an IP address from a **DHCP** server.

Prerequisites

- A connection that uses DHCP is configured on the host.

Procedure

- 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.

- 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

2.17. CONFIGURING MULTIPLE ETHERNET INTERFACES 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

- DHCP is available in the network
- The host has multiple Ethernet adapters
- 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 Example connection.multi-connect multiple  
match.interface-name enp* type ethernet
```

Verification steps

1. Display all settings of the single connection profile:

#nmcli connection show Example

```

connection.id:          Example
...
connection.multi-connect: 3 (multiple)
match.interface-name:   `enp*`
...

```

3 indicates the number of interfaces active on the connection profile at the same time, and not the number of network interfaces in the connection profile. 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
...
Example 6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp7s0
Example 6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp8s0
Example 6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1 ethernet enp9s0

```

Additional resources

- **nmcli(1)** man page
- **nm-settings(5)** man page

2.18. 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

- DHCP server is available in the network
- The host has multiple Ethernet adapters
- No connection profile exists on system

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 Example
```

Verification steps

1. Display the status of the connection:

```
#nmcli connection show

NAME    UUID                                  TYPE    DEVICE
...
Example  9cee0958-512f-4203-9d3d-b57af1d88466 ethernet enp7s0
Example  9cee0958-512f-4203-9d3d-b57af1d88466 ethernet enp8s0
...
```

2. To display all settings of the connection profile:

```
#nmcli connection show Example

connection.id:      Example
...
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)** man page
- **nm-settings(5)** man page

CHAPTER 3. MANAGING WIFI CONNECTIONS

RHEL provides multiple utilities and applications to configure and connect to wifi networks, for example:

- The **nmcli** utility
- The GNOME system menu
- The **GNOME Settings** application
- The **nm-connection-editor** application

3.1. 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 WEP or WPA standards.

RHEL 8 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.
- **WEP 40/128-bit Key (Hex or ASCII):** The Wired Equivalent Privacy (WEP) protocol in this mode uses pre-shared keys only in hex or ASCII format. WEP is deprecated and will be removed in RHEL 9.1.
- **WEP 128-bit Passphrase.** The WEP protocol in this mode uses an MD5 hash of the passphrase to derive a WEP key. WEP is deprecated and will be removed in RHEL 9.1.
- **Dynamic WEP (802.1x):** A combination of 802.1X and EAP that uses the WEP protocol with dynamic keys. WEP is deprecated and will be removed in RHEL 9.1.
- **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).

3.2. CONNECTING TO A WPA2 OR WPA3 PERSONAL-PROTECTED WIFI NETWORK USING NMCLI COMMANDS

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, if it has a hardware switch.



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 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

3.3. CONFIGURING A WIFI CONNECTION 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 deselect checkboxes by using **Space**.

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
wlp2s0 wifi unavailable --
...
```

2. Start **nmtui**:

■

nmtui

3. Select **Edit a connection**, and press **Enter**.
4. Press the **Add** button.
5. Select **Wi-Fi** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
7. Enter the network device name into the **Device** field.
8. Enter the name of the Wi-Fi network, the Service Set Identifier (SSID), into the **SSID** field.
9. Leave the **Mode** field set to its default, **Client**.
10. 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.
11. Fill the authentication type-related fields.
12. 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.
13. 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>), and 'Password' (masked with asterisks). There is a '[] Show password' button. Below the password field are fields for 'BSSID' and 'Cloned MAC address'. At the bottom of the 'WI-FI' section is an 'MTU' field with '(default)' next to it. Below the 'WI-FI' section are two sections for IP configuration: 'IPv4 CONFIGURATION' with '<Automatic>' and '<Show>' buttons, and 'IPv6 CONFIGURATION' with '<Automatic>' and '<Show>' buttons. At the very bottom, there are two checked options: '[X] Automatically connect' and '[X] Available to all users'. In the bottom right corner, there are '<Cancel>' and '<OK>' buttons.

14. Press the **Back** button to return to the main menu.
15. Select **Quit**, and press **Enter** to close the **nmtui** application.

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
```

3.4. CONNECTING TO A WIFI NETWORK USING THE GNOME SYSTEM MENU

You can use the GNOME system menu to connect to a wifi network. When you connect to a network for the first time, GNOME creates a NetworkManager connection profile for it. If you configure the connection profile to not automatically connect, you can also use the GNOME system menu to manually connect to a wifi network with an existing NetworkManager connection profile.



NOTE

Using the GNOME system menu to establish a connection to a wifi network for the first time has certain limitations. For example, you can not configure IP address settings. In this case first configure the connections:

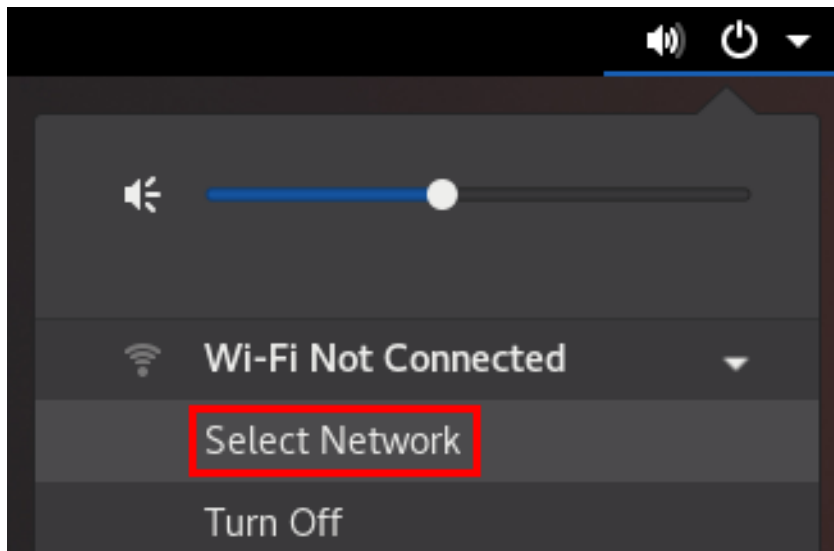
- In the [GNOME settings](#) application
- In the [nm-connection-editor](#) application
- Using [nmcli](#) commands

Prerequisites

- A wifi device is installed on the host.
- The wifi device is enabled, if it has a hardware switch.

Procedure

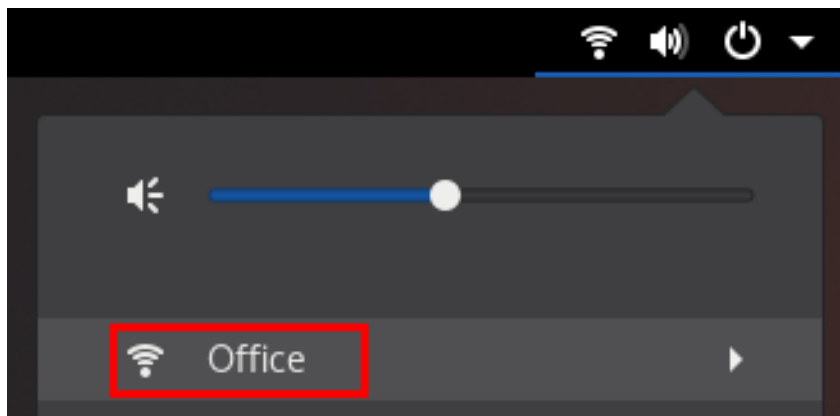
1. Open the system menu on the right side of the top bar.
2. Expand the **Wi-Fi Not Connected** entry.
3. Click **Select Network**:



4. Select the wifi network you want to connect to.
5. Click **Connect**.
6. If this is the first time you connect to this network, enter the password for the network, and click **Connect**.

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
```

3.5. CONNECTING TO A WIFI NETWORK 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

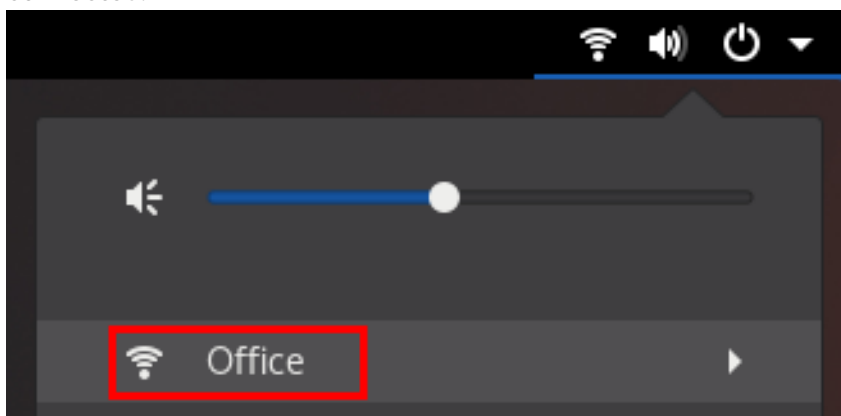
- A wifi device is installed on the host.
- The wifi device is enabled, if it has a hardware switch.

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, 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
```

3.6. CONFIGURING A WIFI CONNECTION USING NM-CONNECTION-EDITOR

You can use the **nm-connection-editor** application to create a connection profile for a wireless network. In this application you can configure all wifi network authentication types that RHEL supports.

By default, NetworkManager enables the auto-connect feature for connection profiles and automatically connects to a saved network if it is available.

Prerequisites

- A wifi device is installed on the host.
- The wifi device is enabled, if it has a hardware switch.

Procedure

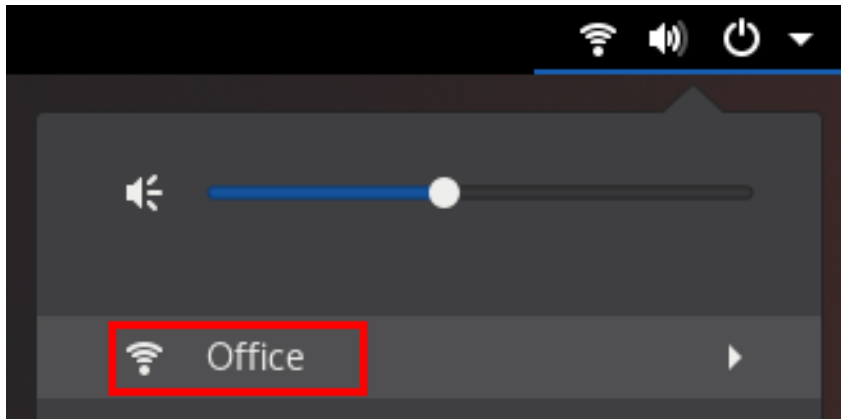
1. Open a terminal and enter:

```
# nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Wi-Fi** connection type, and click **Create**.
4. Optional: Set a name for the connection profile.
5. Optional: Configure the network profile on the **General** tab to not automatically connect. If you deactivate this feature, you must always manually connect to the network, for example, using **GNOME settings** or the GNOME system menu.
6. On the **Wi-Fi** tab, enter the service set identifier (SSID) in the **SSID** field.
7. On the **Wi-Fi Security** tab, select the authentication type for 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.
8. Configure IPv4 settings on the **IPv4** tab, and IPv6 settings on the **IPv6** tab.
9. Click **Save**.
10. Close the **Network Connections** window.

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
```

3.7. CONFIGURING A WIFI CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING THE RHEL SYSTEM ROLES

Using RHEL System Roles, you can automate the creation of a wifi connection. This procedure describes how to remotely add a wireless connection profile for the **wlp1s0** interface using an Ansible playbook. The created profile uses the 802.1X standard to authenticate the client to a wifi network. The playbook configures the connection profile to use DHCP. To configure static IP settings, adapt the parameters in the **ip** dictionary accordingly.

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- 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. Create a playbook file, for example `~/enable-802.1x.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
      copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400

    - name: Copy client certificate for 802.1X authentication
      copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"

    - block:
      - import_role:
          name: linux-system-roles.network
        vars:
          network_connections:
            - name: Configure the Example-wifi profile
              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: "password"
                private_key_password_flags: none
                client_cert: "/etc/pki/tls/client.pem"
                ca_cert: "/etc/pki/tls/cacert.pem"
                domain_suffix_match: "example.com"
```

2. Run the playbook:

```
# ansible-playbook ~/enable-802.1x.yml
```

Additional resources

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

3.8. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING WIFI CONNECTION USING NMCLI

Using the **nmcli** utility, you can configure the client to authenticate itself to the network. This procedure describes how to 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

1. The network must have 802.1X network authentication.
2. The wifi connection profile exists in NetworkManager and has a valid IP configuration.
3. 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.
4. 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. Optionally, store the password in the configuration:

```
# nmcli connection modify wlp1s0 802-1x.password password
```

IMPORTANT

By default, NetworkManager stores the password in clear text in the `/etc/sysconfig/network-scripts/keys-connection_name` file, that 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 **0x1**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services. In other cases, NetworkManager prompts for the password.

3. If the client is required 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, Red Hat recommends using the certificate of the authenticator to enable clients to validate the identity of the authenticator.

4. Activate the connection profile:

```
# nmcli connection up wlp1s0
```

Verification steps

- Access resources on the network that require network authentication.

Additional resources

- [Managing wifi connections](#)
- **nm-settings(5)** man page
- **nmcli(1)** man page

3.9. 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

- **setregdomain(1)** man page
- **iw(8)** man page
- **regulatory.bin(5)** man page
- [ISO 3166 Country Codes](#)

CHAPTER 4. CONFIGURING RHEL AS A WIFI ACCESS POINT

On a host with a wifi device, you can use NetworkManager to configure this host as an access point. Wireless clients can then use the access point to connect to services on the RHEL host or in the network.

When you configure an access point, NetworkManager automatically:

- Configures the **dnsmasq** service to provide DHCP and DNS services for clients
- Enables IP forwarding
- Adds **nftables** firewall rules to masquerade traffic from the wifi device and configures IP forwarding

4.1. IDENTIFYING WHETHER A WIFI DEVICE SUPPORTS THE ACCESS POINT MODE

To use a wifi device as an access point, the device must support this feature. You can use the **nmcli** utility to identify if the hardware supports access point mode.

Prerequisites

- A wifi device is installed on the host.

Procedure

1. List the wifi devices to identify the one that should provide the access point:

```
# nmcli device status | grep wifi
wlp0s20f3 wifi disconnected --
```

2. Verify that the device supports the access point mode:

```
# nmcli -f WIFI-PROPERTIES.AP device show wlp0s20f3
WIFI-PROPERTIES.AP: yes
```

4.2. CONFIGURING RHEL AS A WPA2 OR WPA3 PERSONAL ACCESS POINT

Wi-Fi Protected Access 2 (WPA2) and Wi-Fi Protected Access 3 (WPA3) Personal provide secure authentication methods in wireless networks. Users can connect to the access point using a pre-shared key (PSK).

Prerequisites

- The wifi device supports running in access point mode.
- The wifi device is not in use.
- The host has internet access.

Procedure

1. Install the **dnsmasq** and **NetworkManager-wifi** packages:

```
# yum install dnsmasq NetworkManager-wifi
```

NetworkManager uses the **dnsmasq** service to provide DHCP and DNS services to clients of the access point.

2. Create the initial access point configuration:

```
# nmcli device wifi hotspot ifname wlp0s20f3 con-name Example-Hotspot ssid
Example-Hotspot password "password"
```

This command creates a connection profile for an access point on the **wlp0s20f3** device that provides WPA2 and WPA3 Personal authentication. The name of the wireless network, the Service Set Identifier (SSID), is **Example-Hotspot** and uses the pre-shared key **password**.

3. Optional: Configure the access point to support only WPA3:

```
# nmcli connection modify Example-Hotspot 802-11-wireless-security.key-mgmt sae
```

4. By default, NetworkManager uses the IP address **10.42.0.1** for the wifi device and assigns IP addresses from the remaining **10.42.0.0/24** subnet to clients. To configure a different subnet and IP address, enter:

```
# nmcli connection modify Example-Hotspot ipv4.addresses 192.0.2.254/24
```

The IP address you set, in this case **192.0.2.254**, is the one that NetworkManager assigns to the wifi device. Clients will use this IP address as default gateway and DNS server.

5. Activate the connection profile:

```
# nmcli connection up Example-Hotspot
```

Verification

1. On the server:
 - a. Verify that NetworkManager started the **dnsmasq** service and that the service listens on port 67 (DHCP) and 53 (DNS):

```
# ss -tulpn | egrep ":53|:67"
udp UNCONN 0 0 10.42.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=55905,fd=6))
udp UNCONN 0 0 0.0.0.0:67 0.0.0.0:* users:(("dnsmasq",pid=55905,fd=4))
tcp LISTEN 0 32 10.42.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=55905,fd=7))
```

- b. Display the **nftables** rule set to ensure that NetworkManager enabled forwarding and masquerading for traffic from the **10.42.0.0/24** subnet:

```
# nft list ruleset
table ip nm-shared-wlp0s20f3 {
    chain nat_postrouting {
        type nat hook postrouting priority srcnat; policy accept;
```

```

    ip saddr 10.42.0.0/24 ip daddr != 10.42.0.0/24 masquerade
}


chain filter_forward {
    type filter hook forward priority filter; policy accept;
    ip daddr 10.42.0.0/24 oifname "wlp0s20f3" ct state { established, related } accept
    ip saddr 10.42.0.0/24 iifname "wlp0s20f3" accept
    iifname "wlp0s20f3" oifname "wlp0s20f3" accept
    iifname "wlp0s20f3" reject
    oifname "wlp0s20f3" reject
}
}

```

2. On a client with a wifi adapter:

a. Display the list of available networks:

```

# nmcli device wifi
IN-USE BSSID          SSID          MODE  CHAN  RATE   SIGNAL  BARS
SECURITY
    00:53:00:88:29:04 Example-Hotspot Infra 11   130 Mbit/s 62   WPA3
...

```

b. Connect to the **Example-Hotspot** wireless network. See [Managing Wi-Fi connections](#).

c. Ping a host on the remote network or the internet to verify that the connection works:

```

# ping -c 3 www.redhat.com

```

Additional resources

- [Identifying whether a wifi device supports the access point mode](#)
- **nm-settings(5)** man page

CHAPTER 5. 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 the **nm-connection-editor** application to configure connections in a graphical 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 VLAN TAGGING USING NMCLI COMMANDS

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 IP settings of the VLAN device. Skip this step if you want to use this VLAN device as a port of other devices.
 - a. Configure the IPv4 settings. For example, 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'
# nmcli connection modify vlan10 ipv4.gateway '192.0.2.254'
# nmcli connection modify vlan10 ipv4.dns '192.0.2.253'
# nmcli connection modify vlan10 ipv4.method manual
```

- b. Configure the IPv6 settings. For example, 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'
# nmcli connection modify vlan10 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify vlan10 ipv6.dns '2001:db8:1::fffd'
# nmcli connection modify vlan10 ipv6.method manual
```

5. Activate the connection:

```
# nmcli connection up vlan10
```

Verification steps

- 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

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- **nm-settings(5)** man page

5.2. CONFIGURING VLAN TAGGING USING THE RHEL WEB CONSOLE

Use the RHEL web console to configure VLAN tagging if you prefer to configure 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**

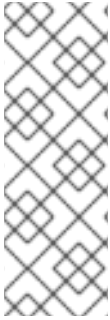
Verification

- 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

5.3. CONFIGURING VLAN TAGGING 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 deselect checkboxes by using **Space**.

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 the **Add** button.
5. Select **VLAN** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
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 **Automatic** button, 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 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 the **Show** button next to the protocol you want to configure to display additional fields.
 - ii. Press the **Add** button 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 the **Add** button next to **DNS servers**, and enter the DNS server address.
 - v. Press the **Add** button next to **Search domains**, and enter the DNS search domain.

Figure 5.1. Example of a VLAN connection with static IP address settings

Edit Connection

Profile name vlan10
Device vlan10

VLAN
Parent enp1s0
VLAN id 10
Cloned MAC address
MTU (default)

IPv4 CONFIGURATION <Manual>
Addresses 192.0.2.1/24 <Remove>
<Add...>
Gateway 192.0.2.254
DNS servers 192.0.2.253 <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>
Addresses 2001:db8:1::1/32 <Remove>
<Add...>
Gateway 2001:db8:1::fffe
DNS servers 2001:db8:1::fffd <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 the **OK** button to create and automatically activate the new connection.
12. Press the **Back** button 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
```

5.4. CONFIGURING VLAN TAGGING USING NM-CONNECTION-EDITOR

You can configure Virtual Local Area Network (VLAN) tagging in a graphical interface using the **nm-connection-editor** application.

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 switch, the host is connected, to is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **VLAN** connection type, and click **Create**.
4. On the **VLAN** tab:
 - a. Select the parent interface.
 - b. Select the VLAN id. Note that the VLAN must be within the range from **0** to **4094**.
 - c. By default, the VLAN connection inherits the maximum transmission unit (MTU) from the parent interface. Optionally, set a different MTU value.
 - d. Optionally, set the name of the VLAN interface and further VLAN-specific options.

Editing VLAN connection 1

Connection name: VLAN connection 1

General | **VLAN** | Proxy | IPv4 Settings | IPv6 Settings

Parent interface: enp1s0 (52:54:00:72:2F:6E)

VLAN id: 10

VLAN interface name: vlan10

Cloned MAC address:

MTU: automatic bytes

Flags: ☒ Reorder headers ☐ GVRP ☐ Loose binding ☐ MVRP

5. Configure the IP settings of the VLAN device. Skip this step if you want to use this VLAN device as a port of other devices.
 - a. On the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, and DNS server:

Editing VLAN connection 1

Connection name: VLAN connection 1

General | VLAN | Proxy | **IPv4 Settings** | IPv6 Settings

Method: Manual

Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.253

- b. On the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, and DNS server:

Editing VLAN connection 1

Connection name:

General **VLAN** Proxy IPv4 Settings **IPv6 Settings**

Method:

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers:

6. Click **Save** to save the VLAN connection.
7. Close **nm-connection-editor**.

Verification steps

1. 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:d5:e0:fb 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

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)

5.5. CONFIGURING VLAN TAGGING USING NMSTATECTL

You can use the **nmstatectl** utility to configure Virtual Local Area Network (VLAN) tagging. This example configures a VLAN with ID 10 that uses an Ethernet connection. As the child device, the VLAN connection contains the IP, default gateway, and DNS configurations.

Depending on your environment, adjust the YAML file accordingly. For example, to use a bridge, or bond device 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 contents:

```
---
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
```

2. Apply the settings to the system:

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

■

Verification steps

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

5.6. CONFIGURING VLAN TAGGING USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure VLAN tagging. This example adds an Ethernet connection and a VLAN with ID **10** on top of this Ethernet connection. As the child device, the VLAN connection contains the IP, default gateway, and DNS configurations.

Depending on your environment, adjust the play accordingly. For example:

- To use the VLAN as a port in other connections, such as a bond, omit the **ip** attribute, and set the IP configuration in the child configuration.
- To use team, bridge, or bond devices in the VLAN, adapt the **interface_name** and **type** attributes of the ports you use in the VLAN.

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a VLAN that uses an Ethernet connection
      include_role:
        name: rhel-system-roles.network

  vars:
    network_connections:
      # Add an Ethernet profile for the underlying device of the VLAN
      - name: enp1s0
        type: ethernet
        interface_name: enp1s0
        autoconnect: yes
        state: up
        ip:
          dhcp4: no
          auto6: no

      # Define the VLAN profile
      - name: enp1s0.10
        type: vlan
        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
        vlan_id: 10
        parent: enp1s0
        state: up
```

The **parent** attribute in the VLAN profile configures the VLAN to operate on top of the **enp1s0** device.

2. Run the playbook:

```
# ansible-playbook ~/vlan-ethernet.yml
```

Additional resources

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

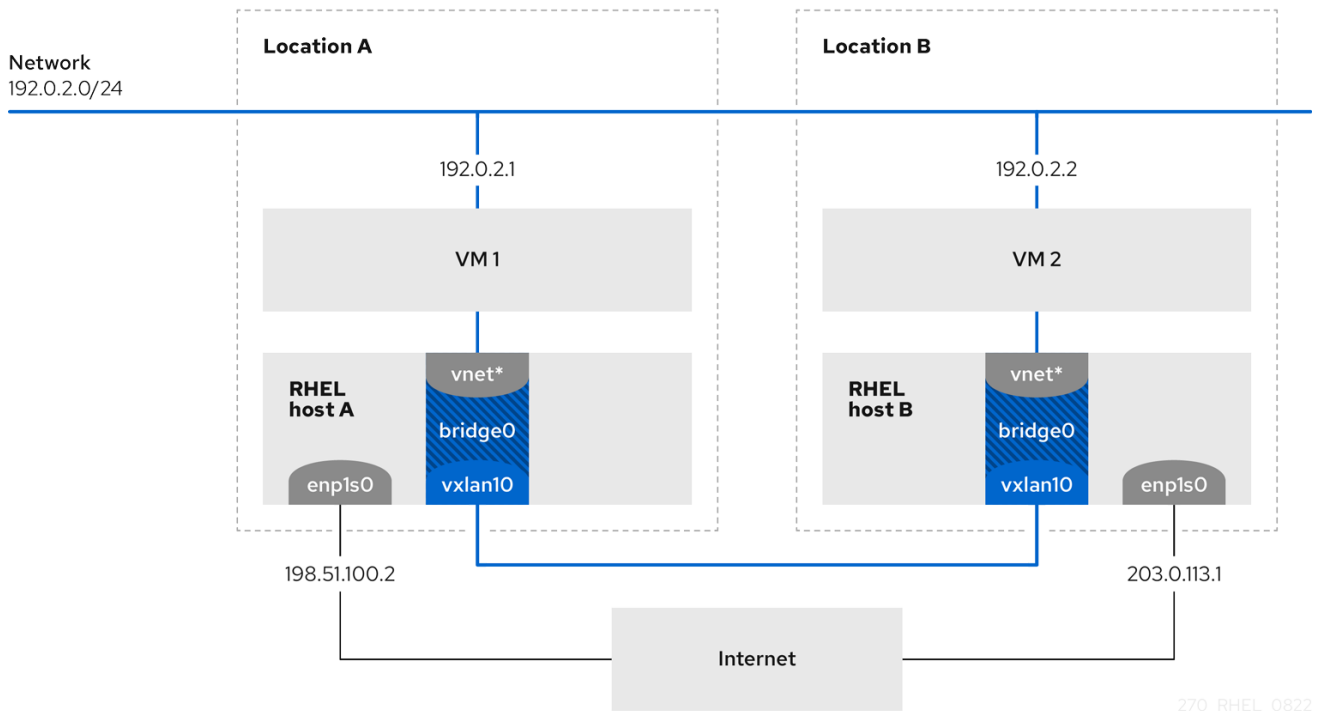
5.7. ADDITIONAL RESOURCES

- [VLANs for sysadmins: The basics](#)

CHAPTER 6. USING A VXLAN TO CREATE A VIRTUAL LAYER-2 DOMAIN FOR VMS

A virtual extensible LAN (VXLAN) is a networking protocol that tunnels layer-2 traffic over an IP network using the UDP protocol. For example, certain virtual machines (VMs), that are running on different hosts can communicate over a VXLAN tunnel. The hosts can be in different subnets or even in different data centers around the world. From the perspective of the VMs, other VMs in the same VXLAN are within the same layer-2 domain.

This documentation describes how to configure a VXLAN on RHEL hosts, which is invisible to the VMs:



In this example, RHEL-host-A and RHEL-host-B use a bridge, **br0**, to connect the virtual network of a VM on each host with a VXLAN named **vxlan10**. Due to this configuration, the VXLAN is invisible to the VMs, and the VMs do not require any special configuration. If you later connect more VMs to the same virtual network, the VMs are automatically members of the same virtual layer-2 domain.



IMPORTANT

Just as normal layer-2 traffic, data in a VXLAN is not encrypted. For security reasons, use a VXLAN over a VPN or other types of encrypted connections.

6.1. BENEFITS OF VXLANs

A virtual extensible LAN (VXLAN) provides the following major benefits:

- VXLANs use a 24-bit ID. Therefore, you can create up to 16,777,216 isolated networks. For example, a virtual LAN (VLAN), supports only 4,096 isolated networks.
- VXLANs use the IP protocol. This enables you to route the traffic and virtually run systems in different networks and locations within the same layer-2 domain.

- Unlike most tunnel protocols, a VXLAN is not only a point-to-point network. A VXLAN can learn the IP addresses of the other endpoints either dynamically or use statically-configured forwarding entries.
- Certain network cards support UDP tunnel-related offload features.

Additional resources

- `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vxlan.rst` provided by the **kernel-doc** package

6.2. CONFIGURING THE ETHERNET INTERFACE ON THE HOSTS

To connect a RHEL VM host to the Ethernet, create a network connection profile, configure the IP settings, and activate the profile.

Run this procedure on both RHEL hosts, and adjust the IP address configuration accordingly.

Prerequisites

- The host is connected to the Ethernet hosts.

Procedure

1. Add a new Ethernet connection profile to NetworkManager:

```
# nmcli connection add con-name Example ifname enp1s0 type ethernet
```

2. Configure the IPv4 settings:

```
# nmcli connection modify Example ipv4.addresses 198.51.100.2/24 ipv4.method manual ipv4.gateway 198.51.100.254 ipv4.dns 198.51.100.200 ipv4.dns-search example.com
```

Skip this step if the network uses DHCP.

3. Activate the **Example** connection:

```
# nmcli connection up Example
```

Verification

1. Display the status of the devices and connections:

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

2. Ping a host in a remote network to verify the IP settings:

```
# ping RHEL-host-B.example.com
```

Note that you cannot ping the other VM host before you have configured the network on that host as well.

Additional resources

- **nm-settings(5)**

6.3. CREATING A NETWORK BRIDGE WITH A VXLAN ATTACHED

To make a virtual extensible LAN (VXLAN) invisible to virtual machines (VMs), create a bridge on a host, and attach the VXLAN to the bridge. Use NetworkManager to create both the bridge and the VXLAN. You do not add any traffic access point (TAP) devices of the VMs, typically named **vnet*** on the host, to the bridge. The **libvirt** service adds them dynamically when the VMs start.

Run this procedure on both RHEL hosts, and adjust the IP addresses accordingly.

Procedure

1. Create the bridge **br0**:

```
# nmcli connection add type bridge con-name br0 ifname br0 ipv4.method disabled
ipv6.method disabled
```

This command sets no IPv4 and IPv6 addresses on the bridge device, because this bridge works on layer 2.

2. Create the VXLAN interface and attach it to **br0**:

```
# nmcli connection add type vxlan slave-type bridge con-name br0-vxlan10 ifname
vxlan10 id 10 local 198.51.100.2 remote 203.0.113.1 master br0
```

This command uses the following settings:

- **id 10**: Sets the VXLAN identifier.
- **local 198.51.100.2**: Sets the source IP address of outgoing packets.
- **remote 203.0.113.1**: Sets the unicast or multicast IP address to use in outgoing packets when the destination link layer address is not known in the VXLAN device forwarding database.
- **master br0**: Sets this VXLAN connection to be created as a port in the **br0** connection.
- **ipv4.method disabled** and **ipv6.method disabled**: Disables IPv4 and IPv6 on the bridge.

By default, NetworkManager uses **8472** as the destination port. If the destination port is different, additionally, pass the **destination-port <port_number>** option to the command.

3. Activate the **br0** connection profile:

```
# nmcli connection up br0
```

4. Open port **8472** for incoming UDP connections in the local firewall:

```
# firewall-cmd --permanent --add-port=8472/udp
# firewall-cmd --reload
```

Verification

- Display the forwarding table:

```
# bridge fdb show dev vxlan10
2a:53:bd:d5:b3:0a master br0 permanent
00:00:00:00:00:00 dst 203.0.113.1 self permanent
...
```

Additional resources

- [nm-settings\(5\)](#)

6.4. CREATING A VIRTUAL NETWORK IN LIBVIRT WITH AN EXISTING BRIDGE

To enable virtual machines (VM) to use the **br0** bridge with the attached virtual extensible LAN (VXLAN), first add a virtual network to the **libvirt** service that uses this bridge.

Prerequisites

- You installed the **libvirt** package.
- You started and enabled the **libvirtd** service.
- You configured the **br0** device with the VXLAN on RHEL.

Procedure

1. Create the **~/vxlan10-bridge.xml** file with the following content:

```
<network>
  <name>vxlan10-bridge</name>
  <forward mode="bridge" />
  <bridge name="br0" />
</network>
```

2. Use the **~/vxlan10-bridge.xml** file to create a new virtual network in **libvirt**:

```
# virsh net-define ~/vxlan10-bridge.xml
```

3. Remove the **~/vxlan10-bridge.xml** file:

```
# rm ~/vxlan10-bridge.xml
```

4. Start the **vxlan10-bridge** virtual network:

```
# virsh net-start vxlan10-bridge
```


- Configure the **vxlan10-bridge** virtual network to start automatically when the **libvirt** service starts:

```
# virsh net-autostart vxlan10-bridge
```

Verification

- Display the list of virtual networks:

```
# virsh net-list
Name          State   Autostart Persistent
-----
vxlan10-bridge active  yes      yes
...
```

Additional resources

- virsh(1)** man page

6.5. CONFIGURING VIRTUAL MACHINES TO USE VXLAN

To configure a VM to use a bridge device with an attached virtual extensible LAN (VXLAN) on the host, create a new VM that uses the **vxlan10-bridge** virtual network or update the settings of existing VMs to use this network.

Perform this procedure on the RHEL hosts.

Prerequisites

- You configured the **vxlan10-bridge** virtual network in **libvirt**.

Procedure

- To create a new VM and configure it to use the **vxlan10-bridge** network, pass the **--network network:vxlan10-bridge** option to the **virt-install** command when you create the VM:

```
# virt-install ... --network network:vxlan10-bridge
```

- To change the network settings of an existing VM:
 - Connect the VM's network interface to the **vxlan10-bridge** virtual network:

```
# virt-xml VM_name --edit --network network=vxlan10-bridge
```

- Shut down the VM, and start it again:

```
# virsh shutdown VM_name
# virsh start VM_name
```

Verification

- Display the virtual network interfaces of the VM on the host:

```
■
```

```
# virsh domiflist VM_name
Interface Type Source Model MAC
-----
vnet1 bridge vxlan10-bridge virtio 52:54:00:c5:98:1c
```

2. Display the interfaces attached to the **vxlan10-bridge** bridge:

```
# ip link show master vxlan10-bridge
18: vxlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 2a:53:bd:d5:b3:0a brd ff:ff:ff:ff:ff:ff
19: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
```

Note that the **libvirtd** service dynamically updates the bridge's configuration. When you start a VM which uses the **vxlan10-bridge** network, the corresponding **vnet*** device on the host appears as a port of the bridge.

3. Use address resolution protocol (ARP) requests to verify whether VMs are in the same VXLAN:
 - a. Start two or more VMs in the same VXLAN.
 - b. Send an ARP request from one VM to the other one:

```
# arping -c 1 192.0.2.2
ARPING 192.0.2.2 from 192.0.2.1 enp1s0
Unicast reply from 192.0.2.2 [52:54:00:c5:98:1c] 1.450ms
Sent 1 probe(s) (0 broadcast(s))
Received 1 response(s) (0 request(s), 0 broadcast(s))
```

If the command shows a reply, the VM is in the same layer-2 domain and, in this case in the same VXLAN.

Install the **iputils** package to use the **arping** utility.

Additional resources

- **virt-install(1)** man page
- **virt-xml(1)** man page
- **virsh(1)** man page
- **arping(8)** man page

CHAPTER 7. CONFIGURING A NETWORK BRIDGE

A network bridge is a link-layer device which forwards traffic between networks based on a table of MAC addresses. The bridge builds the MAC addresses table by listening to network traffic and thereby learning what hosts are connected to each network. For example, you can use a software bridge on a Red Hat Enterprise Linux host to emulate a hardware bridge or in virtualization environments, to integrate virtual machines (VM) to the same network as the host.

A bridge requires a network device in each network the bridge should connect. When you configure a bridge, the bridge is called **controller** and the devices it uses **ports**.

You can create bridges on different types of devices, such as:

- Physical and virtual Ethernet devices
- Network bonds
- Network teams
- VLAN devices

Due to the IEEE 802.11 standard which specifies the use of 3-address frames in Wi-Fi for the efficient use of airtime, you cannot configure a bridge over Wi-Fi networks operating in Ad-Hoc or Infrastructure modes.

7.1. CONFIGURING A NETWORK BRIDGE USING NMCLI COMMANDS

This section explains how to configure a network bridge using 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 team, 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 team using nmcli commands](#)
 - [Configuring a network bond using nmcli commands](#)
 - [Configuring VLAN tagging using nmcli commands](#)

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.

- a. 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.

- b. 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.

4. Configure the IP settings of the bridge. Skip this step if you want to use this bridge as a ports of other devices.

- a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain of the **bridge0** connection, enter:

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

- b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain of the **bridge0** connection, enter:

```
# nmcli connection modify bridge0 ipv6.addresses '2001:db8:1::1/64'
# nmcli connection modify bridge0 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify bridge0 ipv6.dns '2001:db8:1::fffd'
```

```
# nmcli connection modify bridge0 ipv6.dns-search 'example.com'
# nmcli connection modify bridge0 ipv6.method manual
```

5. 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.

6. Activate the connection:

```
# nmcli connection up bridge0
```

7. 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:

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

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

- b. Reactivate the bridge:

```
# nmcli connection up bridge0
```

Verification steps

- 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

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- **nm-settings(5)** man page
- **bridge(8)** man page
- [NetworkManager duplicates a connection after restart of NetworkManager service](#)
- [How to configure bridge with vlan information?](#)

7.2. CONFIGURING A NETWORK BRIDGE USING THE RHEL WEB CONSOLE

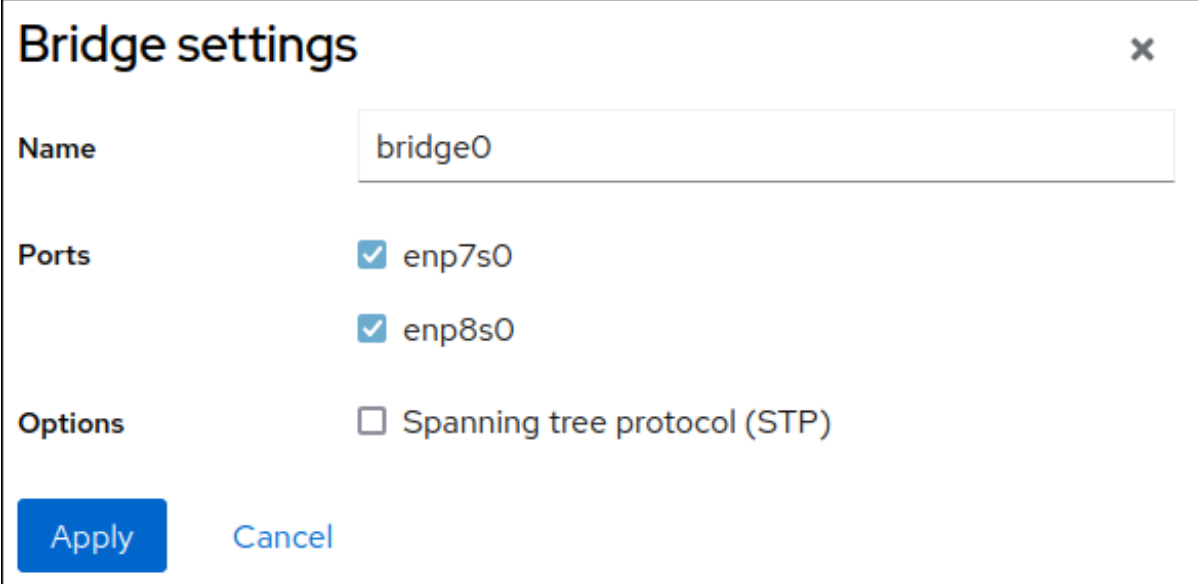
This section explains how to configure a network bridge using the RHEL web console.

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 team, 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 team using the RHEL web console](#)
 - [Configuring a network bond using the RHEL web console](#)
 - [Configuring VLAN tagging 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 a 'Bridge settings' dialog box. The dialog has a title bar with the text 'Bridge settings' and a close button (an 'x' icon) in the top right corner. Inside the dialog, there are three sections: 'Name', 'Ports', and 'Options'. The 'Name' section has a text input field containing 'bridge0'. The 'Ports' section has two checkboxes, both of which are checked; the first is labeled 'enp7s0' and the second is labeled 'enp8s0'. The 'Options' section has one checkbox labeled 'Spanning tree protocol (STP)', which is currently unchecked. At the bottom left of the dialog is a blue button labeled 'Apply', and at the bottom right is a text label '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

7.3. CONFIGURING A NETWORK BRIDGE 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 deselect checkboxes by using **Space**.

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:

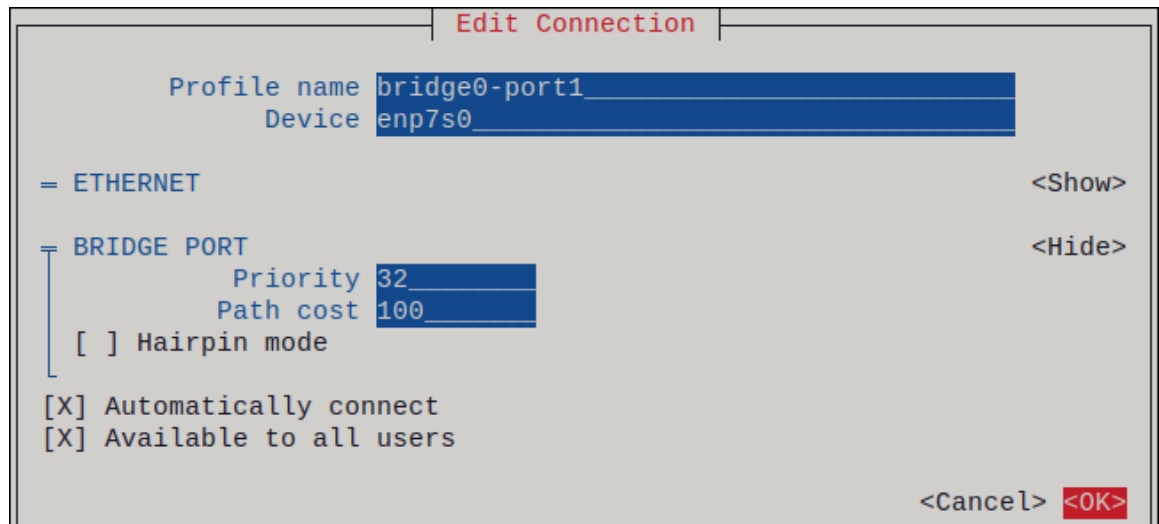
```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

2. Start **nmtui**:

```
# nmtui
```

3. Select **Edit a connection**, and press **Enter**.
4. Press the **Add** button.
5. Select **Bridge** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
7. Enter the bridge device name to be created into the **Device** field.
8. Add ports to the bridge to be created:
 - a. Press the **Add** button 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 the **OK** button to return to the window with the bridge settings.

Figure 7.1. 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 **Automatic** button, and select:
 - **Disabled**, if the bridge does not require an IP address.
 - **Automatic**, if a DHCP server 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 the **Show** button next to the protocol you want to configure to display additional fields.
 - ii. Press the **Add** button 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 the **Add** button next to **DNS servers**, and enter the DNS server address.
 - v. Press the **Add** button next to **Search domains**, and enter the DNS search domain.

Figure 7.2. 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)' options are checked. The 'Priority' is 32768, 'Forward delay' is 15 seconds, 'Hello time' is 2 seconds, and 'Max age' is 20 seconds. The 'Group forward mask' is 0. Both 'IPv4 CONFIGURATION' and 'IPv6 CONFIGURATION' are disabled. The 'Automatically connect' and 'Available to all users' options are checked. The window has '<Cancel>' and '<OK>' buttons at the bottom right.

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

7.4. CONFIGURING A NETWORK BRIDGE USING NM-CONNECTION-EDITOR

This section explains how to configure a network bridge using the **nm-connection-editor** application.

Note that **nm-connection-editor** can add only new ports to a bridge. To use an existing connection profile as a port, create the bridge using the **nmcli** utility as described in [Configuring a network bridge using nmcli commands](#).

Prerequisites

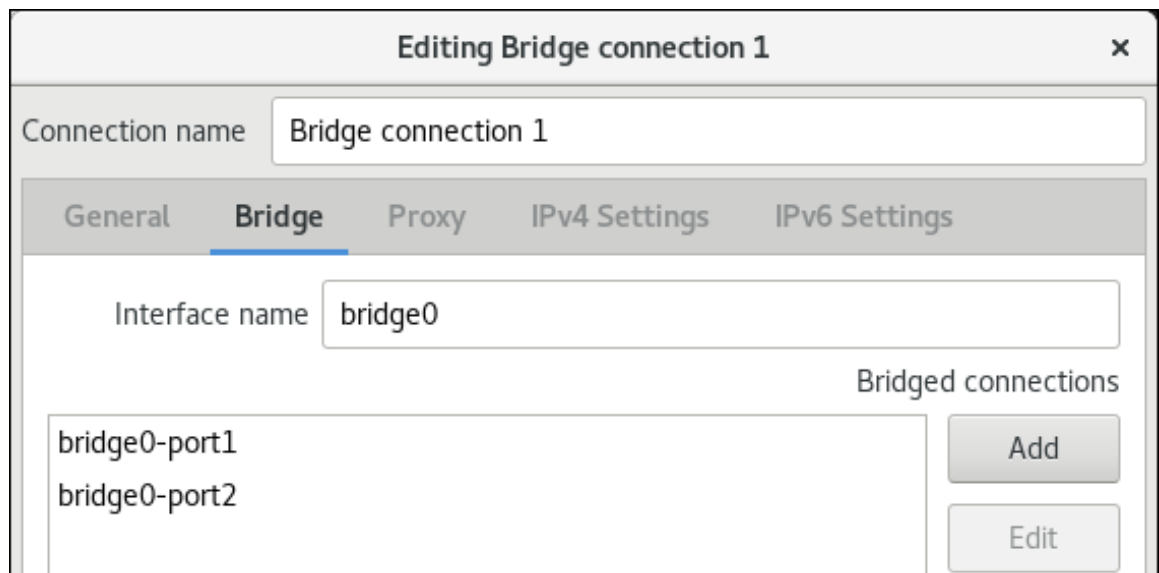
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bridge, ensure that these devices are not already configured.

Procedure

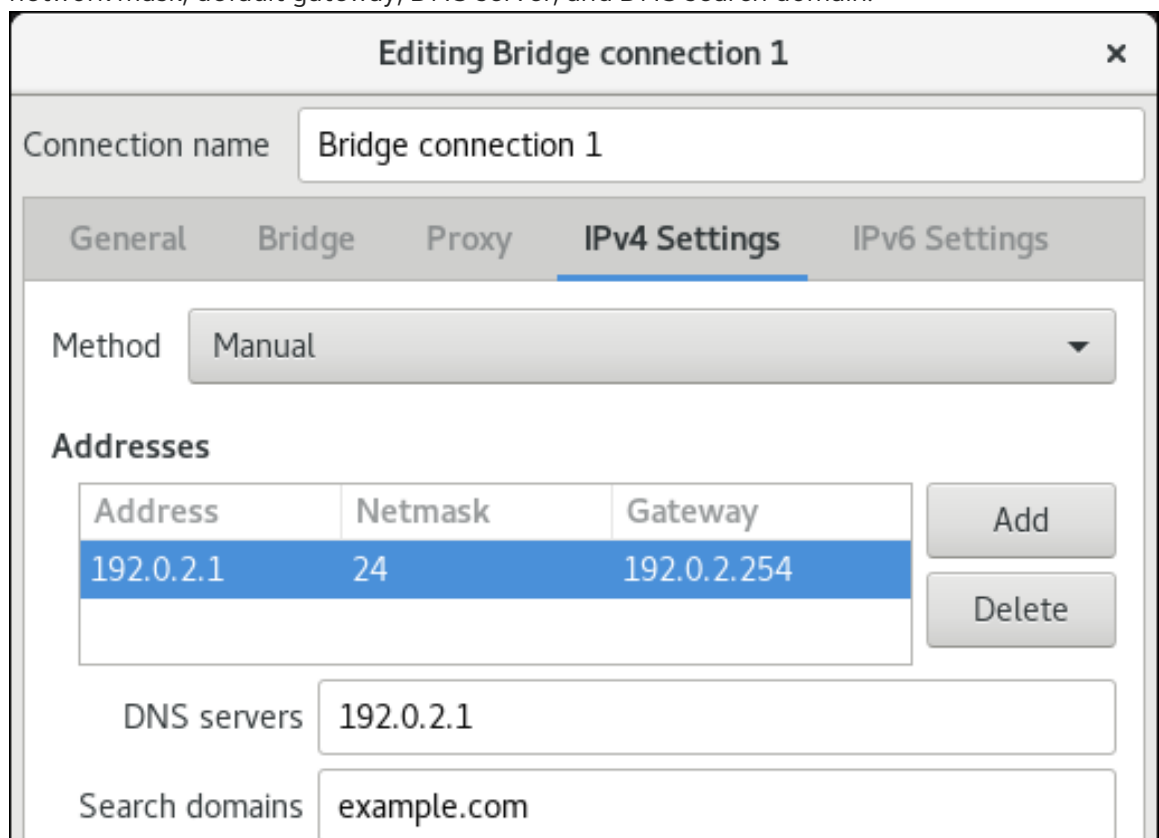
1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Bridge** connection type, and click **Create**.
4. In the **Bridge** tab:
 - a. Optional: Set the name of the bridge interface in the **Interface name** field.
 - b. Click the **Add** button to create a new connection profile for a network interface and adding the profile as a port to the bridge.
 - i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.
 - ii. Optionally, set a connection name for the port device.
 - iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the bridge. If you selected a different device type, configure it accordingly.
 - iv. Click **Save**.
 - c. Repeat the previous step for each interface you want to add to the bridge.



5. Optional: Configure further bridge settings, such as Spanning Tree Protocol (STP) options.
6. Configure the IP settings of the bridge. Skip this step if you want to use this bridge as a port of other devices.
 - a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:



- b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:

Editing Bridge connection 1

Connection name: Bridge connection 1

General Bridge Proxy IPv4 Settings **IPv6 Settings**

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

Search domains: example.com

7. Save the bridge connection.
8. Close **nm-connection-editor**.

Verification steps

- 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 in 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

- [Configuring a network bond using nm-connection-editor](#)
- [Configuring a network team using nm-connection-editor](#)
- [Configuring VLAN tagging using nm-connection-editor](#)
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- [How to configure bridge with vlan information?](#)

7.5. CONFIGURING A NETWORK BRIDGE USING NMSTATECTL

This section describes how to use the **nmstatectl** utility to configure a Linux network bridge **bridge0** with 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**

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 contents:

```
---
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
```

2. Apply the settings to the system:

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

Verification steps

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
- `/usr/share/doc/nmstate/examples/` directory
- [How to configure bridge with vlan information?](#)

7.6. CONFIGURING A NETWORK BRIDGE USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure a Linux bridge. This procedure describes how to configure a network bridge that uses two Ethernet devices, and sets IPv4 and IPv6 addresses, default gateways, and DNS configuration.



NOTE

Set the IP configuration on the bridge and not on the ports of the Linux bridge.

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bridge that uses two Ethernet ports
      include_role:
        name: rhel-system-roles.network

  vars:
    network_connections:
      # Define the bridge profile
      - name: bridge0
        type: bridge
        interface_name: bridge0
        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

      # Add an Ethernet profile to the bridge
      - name: bridge0-port1
        interface_name: enp7s0
        type: ethernet
        controller: bridge0
        port_type: bridge
        state: up

      # Add a second Ethernet profile to the bridge
      - name: bridge0-port2
        interface_name: enp8s0
        type: ethernet
        controller: bridge0
        port_type: bridge
        state: up
```

2. Run the playbook:

```
# ansible-playbook ~/bridge-ethernet.yml
```

Additional resources

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

CHAPTER 8. CONFIGURING NETWORK TEAMING

A network team is a method to combine or aggregate physical and virtual network interfaces to provide a logical interface with higher throughput or redundancy. In a network team, both a small kernel module and a user-space service process the operations. You can create network teams 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 teams connections using the command line.
- Use the RHEL web console to configure team connections using a web browser.
- Use the **nm-connection-editor** application to configure team connections in a graphical interface.



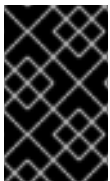
IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see [Configuring network bonding](#).

8.1. UNDERSTANDING NETWORK TEAMING

Network teaming is a feature that combines or aggregates network interfaces to provide a logical interface with higher throughput or redundancy.

Network teaming uses a kernel driver to implement fast handling of packet flows, as well as user-space libraries and services for other tasks. This way, network teaming is an easily extensible and scalable solution for load-balancing and redundancy requirements.



IMPORTANT

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

8.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES

Consider the following default behavior of, when managing or troubleshooting team or 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 carrier.

8.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES

Learn about the features supported in network teams and network bonds:

Feature	Network bond	Network team
Broadcast Tx policy	Yes	Yes
Round-robin Tx policy	Yes	Yes
Active-backup Tx policy	Yes	Yes
LACP (802.3ad) support	Yes (active only)	Yes
Hash-based Tx policy	Yes	Yes
User can set hash function	No	Yes
Tx load-balancing support (TLB)	Yes	Yes
LACP hash port select	Yes	Yes
Load-balancing for LACP support	No	Yes
Ethtool link monitoring	Yes	Yes
ARP link monitoring	Yes	Yes
NS/NA (IPv6) link monitoring	No	Yes
Ports up/down delays	Yes	Yes
Port priorities and stickiness ("primary" option enhancement)	No	Yes
Separate per-port link monitoring setup	No	Yes
Multiple link monitoring setup	Limited	Yes
Lockless Tx/Rx path	No (rwlock)	Yes (RCU)
VLAN support	Yes	Yes

Feature	Network bond	Network team
User-space runtime control	Limited	Yes
Logic in user-space	No	Yes
Extensibility	Hard	Easy
Modular design	No	Yes
Performance overhead	Low	Very low
D-Bus interface	No	Yes
Multiple device stacking	Yes	Yes
Zero config using LLDP	No	(in planning)
NetworkManager support	Yes	Yes

8.4. UNDERSTANDING THE TEAMD SERVICE, RUNNERS, AND LINK-WATCHERS

The team service, **teamd**, controls one instance of the team driver. This instance of the driver adds instances of a hardware device driver to form a team of network interfaces. The team driver presents a network interface, for example **team0**, to the kernel.

The **teamd** service implements the common logic to all methods of teaming. Those functions are unique to the different load sharing and backup methods, such as round-robin, and implemented by separate units of code referred to as **runners**. Administrators specify runners in JavaScript Object Notation (JSON) format, and the JSON code is compiled into an instance of **teamd** when the instance is created. Alternatively, when using **NetworkManager**, you can set the runner in the **team.runner** parameter, and **NetworkManager** auto-creates the corresponding JSON code.

The following runners are available:

- **broadcast**: Transmits data over all ports.
- **roundrobin**: Transmits data over all ports in turn.
- **activebackup**: Transmits data over one port while the others are kept as a backup.
- **loadbalance**: Transmits data over all ports with active Tx load balancing and Berkeley Packet Filter (BPF)-based Tx port selectors.
- **random**: Transmits data on a randomly selected port.
- **lacp**: Implements the 802.3ad Link Aggregation Control Protocol (LACP).

The **teamd** services uses a link watcher to monitor the state of subordinate devices. The following link-watchers are available:

- **ethtool**: The **libteam** library uses the **ethtool** utility to watch for link state changes. This is the default link-watcher.
- **arp_ping**: The **libteam** library uses the **arp_ping** utility to monitor the presence of a far-end hardware address using Address Resolution Protocol (ARP).
- **nsna_ping**: On IPv6 connections, the **libteam** library uses the Neighbor Advertisement and Neighbor Solicitation features from the IPv6 Neighbor Discovery protocol to monitor the presence of a neighbor's interface.

Each runner can use any link watcher, with the exception of **lACP**. This runner can only use the **ethtool** link watcher.

8.5. CONFIGURING A NETWORK TEAM USING NMCLI COMMANDS

You can use the **nmcli** utility to configure a network team on the command line.



IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see [Configuring network bonding](#).

Prerequisites

- The **teamd** and **NetworkManager-team** packages are installed.
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server and connected to a switch.
- To use bond, bridge, or VLAN devices as ports of the team, you can either create these devices while you create the team or you can create them in advance as described in:
 - [Configuring a network bond using nmcli commands](#)
 - [Configuring a network bridge using nmcli commands](#)
 - [Configuring VLAN tagging using nmcli commands](#)

Procedure

1. Create a team interface:

```
# nmcli connection add type team con-name team0 ifname team0 team.runner
activebackup
```

This command creates a network team named **team0** that uses the **activebackup** runner.

2. Optionally, set a link watcher. For example, to set the **ethtool** link watcher in the **team0** connection profile:

```
# nmcli connection modify team0 team.link-watchers "name=ethtool"
```

Link watchers support different parameters. To set parameters for a link watcher, specify them space-separated in the **name** property. Note that the name property must be surrounded by quotation marks. For example, to use the **ethtool** link watcher and set its **delay-up** parameter to **2500** milliseconds (2.5 seconds):

```
# nmcli connection modify team0 team.link-watchers "name=ethtool delay-up=2500"
```

To set multiple link watchers and each of them with specific parameters, the link watchers must be separated by a comma. The following example sets the **ethtool** link watcher with the **delay-up** parameter and the **arp_ping** link watcher with the **source-host** and **target-host** parameter:

```
# nmcli connection modify team0 team.link-watchers "name=ethtool delay-up=2,
name=arp_ping source-host=192.0.2.1 target-host=192.0.2.2"
```

3. Display the network interfaces, and note the names of the interfaces you want to add to the team:

```
# 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. Note that you can only use Ethernet interfaces in a team that are not assigned to any connection.
- **bond0** and **bond1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

4. Assign the port interfaces to the team:

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

```
# nmcli connection add type ethernet slave-type team con-name team0-port1
ifname enp7s0 master team0
# nmcli connection add type ethernet slave-type team con-name team0-port2
ifname enp8s0 master team0
```

. These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **team0** connection.

- b. To assign an existing connection profile to the team, set the **master** parameter of these connections to **team0**:

```
# nmcli connection modify bond0 master team0
# nmcli connection modify bond1 master team0
```

These commands assign the existing connection profiles named **bond0** and **bond1** to the **team0** connection.

5. Configure the IP settings of the team. Skip this step if you want to use this team as a ports of other devices.
 - a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain the **team0** connection, enter:

```
# nmcli connection modify team0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify team0 ipv4.gateway '192.0.2.254'
# nmcli connection modify team0 ipv4.dns '192.0.2.253'
# nmcli connection modify team0 ipv4.dns-search 'example.com'
# nmcli connection modify team0 ipv4.method manual
```

- b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain of the **team0** connection, enter:

```
# nmcli connection modify team0 ipv6.addresses '2001:db8:1::1/64'
# nmcli connection modify team0 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify team0 ipv6.dns '2001:db8:1::fffd'
# nmcli connection modify team0 ipv6.dns-search 'example.com'
# nmcli connection modify team0 ipv6.method manual
```

6. Activate the connection:

```
# nmcli connection up team0
```

Verification steps

- Display the status of the team:

```
# teamdctl team0 state
setup:
  runner: activebackup
ports:
  enp7s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  enp8s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
runner:
  active port: enp7s0
```

In this example, both ports are up.

Additional resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- [Understanding the teamd service, runners, and link-watchers](#)
- **nm-settings(5)** man page
- **teamd.conf(5)** man page

8.6. CONFIGURING A NETWORK TEAM USING THE RHEL WEB CONSOLE

You can use the RHEL web console to configure a network team using a web browser.



IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see [Configuring network bonding](#).

Prerequisites

- The **teamd** and **NetworkManager-team** packages are installed.
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server and connected to a switch.
- To use bond, bridge, or VLAN devices as ports of the team, create them in advance as described in:
 - [Configuring a network bond using the RHEL web console](#)
 - [Configuring a network bridge using the RHEL web console](#)
 - [Configuring VLAN tagging using the RHEL web console](#)

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add team** in the **Interfaces** section.
3. Enter the name of the team device you want to create.
4. Select the interfaces that should be ports of the team.
5. Select the runner of the team.
If you select **Load balancing** or **802.3ad LACP**, the web console shows the additional field **Balancer**.
6. Set the link watcher:
 - If you select **Ethtool**, additionally, set a link up and link down delay.
 - If you set **ARP ping** or **NSNA ping**, additionally, set a ping interval and ping target.

Team settings ×

Name	<input type="text" value="team0"/>
Ports	<input checked="" type="checkbox"/> enp7s0 <input checked="" type="checkbox"/> enp8s0
Runner	<input type="text" value="Active backup"/>
Link watch	<input type="text" value="Ethtool"/>
Link up delay	<input type="text" value="0"/>
Link down delay	<input type="text" value="0"/>
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

7. Click **Apply**.
8. By default, the team uses a dynamic IP address. If you want to set a static IP address:
 - a. Click the name of the team 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
team0	192.0.2.1/24	1.11 Mbps	61.2 Mbps

2. Display the status of the team:

```
# teamdctl team0 state
setup:
runner: activebackup
ports:
enp7s0
link watches:
link summary: up
instance[link_watch_0]:
name: ethtool
link: up
```

```

    down count: 0
    enp8s0
    link watches:
    link summary: up
    instance[link_watch_0]:
    name: ethtool
    link: up
    down count: 0
runner:
    active port: enp7s0

```

In this example, both ports are up.

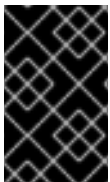
Additional resources

- [Network team runners](#)

8.7. CONFIGURING A NETWORK TEAM USING NM-CONNECTION-EDITOR

You can use the **nm-connection-editor** application to configure a network team in a graphical environment.

Note that **nm-connection-editor** can add only new ports to a team. To use an existing connection profile as a port, create the team using the **nmcli** utility as described in [Configuring a network team using nmcli commands](#).



IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see [Configuring network bonding](#).

Prerequisites

- The **teamd** and **NetworkManager-team** packages are installed.
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the team, ensure that these devices are not already configured.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Team** connection type, and click **Create**.

4. In the **Team** tab:

- a. Optional: Set the name of the team interface in the **Interface name** field.
- b. Click the **Add** button to add a new connection profile for a network interface and adding the profile as a port to the team.
 - i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.
 - ii. Optional: Set a connection name for the port.
 - iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the team. If you selected a different device type, configure it accordingly. Note that you can only use Ethernet interfaces in a team that are not assigned to any connection.
- iv. Click **Save**.
- c. Repeat the previous step for each interface you want to add to the team.

The screenshot shows a window titled "Editing Team connection 1". At the top, there's a tab bar with "General", "Team" (selected), "Proxy", "IPv4 Settings", and "IPv6 Settings". Under the "Team" tab, there's a "Connection name" field with the value "Team connection 1". Below that is the "Interface name" field with the value "team0". Then, there's an "MTU" field with a dropdown set to "automatic", minus and plus buttons, and the text "bytes". At the bottom, there's a section titled "Teamed connections" containing a list with two items: "team0-port1" and "team0-port2". To the right of this list are two buttons: "Add" and "Edit".

- d. Click the **Advanced** button to set advanced options to the team connection.
 - i. In the **Runner** tab, select the runner.
 - ii. In the **Link Watcher** tab, set the link watcher and its optional settings.
 - iii. Click **OK**.
5. Configure the IP settings of the team. Skip this step if you want to use this team as a port of other devices.

- a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:

The screenshot shows the 'Editing Team connection 1' dialog box with the 'IPv4 Settings' tab selected. The 'Connection name' is 'Team connection 1'. The 'Method' is set to 'Manual'. Under 'Addresses', there is a table with one row: Address '192.0.2.1', Netmask '24', and Gateway '192.0.2.254'. To the right of the table are 'Add' and 'Delete' buttons. Below the table, 'DNS servers' is '192.0.2.253' and 'Search domains' is 'example.com'.

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.253
Search domains: example.com

- b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:

The screenshot shows the 'Editing Team connection 1' dialog box with the 'IPv6 Settings' tab selected. The 'Connection name' is 'Team connection 1'. The 'Method' is set to 'Manual'. Under 'Addresses', there is a table with one row: Address '2001:db8:1::1', Prefix '64', and Gateway '2001:db8:1::ff3'. To the right of the table are 'Add' and 'Delete' buttons. Below the table, 'DNS servers' is '2001:db8:1::fffd' and 'Search domains' is 'example.com'.

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::ff3

DNS servers: 2001:db8:1::fffd
Search domains: example.com

6. Save the team connection.
7. Close **nm-connection-editor**.

Verification steps

- Display the status of the team:

```
# teamdctl team0 state
setup:
  runner: activebackup
ports:
  enp7s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  enp8s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
runner:
  active port: enp7s0
```

Additional resources

- [Configuring a network bond using nm-connection-editor](#)
- [Configuring a network team using nm-connection-editor](#)
- [Configuring VLAN tagging using nm-connection-editor](#)
- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- [Understanding the teamd service, runners, and link-watchers](#)
- [NetworkManager duplicates a connection after restart of NetworkManager service](#)

CHAPTER 9. CONFIGURING NETWORK BONDING

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:

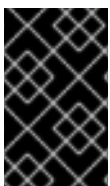
- Use **nmcli** to configure bond connections using the command line.
- Use the RHEL web console to configure bond connections using a web browser.
- Use **nmtui** to configure bond connections in a text-based user interface.
- Use the **nm-connection-editor** application to configure bond connections in a graphical 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.

9.1. UNDERSTANDING NETWORK BONDING

Network bonding is a method to combine or aggregate network interfaces to provide a logical interface with higher throughput or redundancy.

The **active-backup**, **balance-tlb**, and **balance-alb** modes do not require any specific configuration of the network switch. However, other bonding modes require configuring the switch to aggregate the links. For example, Cisco switches requires **EtherChannel** for modes 0, 2, and 3, but for mode 4, the Link Aggregation Control Protocol (LACP) and **EtherChannel** are required.

For further details, see the documentation of your switch and [Linux Ethernet Bonding Driver HOWTO](#).



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 [ls bonding supported with direct connection using crossover cables?](#) KCS solution.

9.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES

Consider the following default behavior of, when managing or troubleshooting team or 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 carrier.

9.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES

Learn about the features supported in network teams and network bonds:

Feature	Network bond	Network team
Broadcast Tx policy	Yes	Yes
Round-robin Tx policy	Yes	Yes
Active-backup Tx policy	Yes	Yes
LACP (802.3ad) support	Yes (active only)	Yes
Hash-based Tx policy	Yes	Yes
User can set hash function	No	Yes
Tx load-balancing support (TLB)	Yes	Yes
LACP hash port select	Yes	Yes
Load-balancing for LACP support	No	Yes
Ethtool link monitoring	Yes	Yes
ARP link monitoring	Yes	Yes
NS/NA (IPv6) link monitoring	No	Yes
Ports up/down delays	Yes	Yes
Port priorities and stickiness ("primary" option enhancement)	No	Yes
Separate per-port link monitoring setup	No	Yes
Multiple link monitoring setup	Limited	Yes
Lockless Tx/Rx path	No (rwlock)	Yes (RCU)
VLAN support	Yes	Yes

Feature	Network bond	Network team
User-space runtime control	Limited	Yes
Logic in user-space	No	Yes
Extensibility	Hard	Easy
Modular design	No	Yes
Performance overhead	Low	Very low
D-Bus interface	No	Yes
Multiple device stacking	Yes	Yes
Zero config using LLDP	No	(in planning)
NetworkManager support	Yes	Yes

9.4. UPSTREAM SWITCH CONFIGURATION DEPENDING ON THE BONDING MODES

The following table describes which settings you must apply to the upstream switch depending on the bonding mode:

Bonding mode	Configuration on the switch
0 - balance-rr	Requires static Etherchannel enabled (not LACP-negotiated)
1 - active-backup	Requires autonomous ports
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	Requires autonomous ports
6 - balance-alb	Requires autonomous ports

For configuring these settings on your switch, see the switch documentation.

9.5. CONFIGURING A NETWORK BOND USING NMCLI COMMANDS

This section describes how to configure a network bond using **nmcli** commands.

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 team, 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 team using nmcli commands](#)
 - [Configuring a network bridge using nmcli commands](#)
 - [Configuring VLAN tagging using nmcli commands](#)

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, to use the same command but, additionally, set the MII monitoring interval to **1000** milliseconds (1 second), enter:

```
# 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 slave-type bond con-name bond0-port1
ifname enp7s0 master bond0
# nmcli connection add type ethernet slave-type bond con-name bond0-port2
ifname enp8s0 master 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, set the **master** parameter of these connections to **bond0**:

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

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

4. Configure the IP settings of the bond. Skip this step if you want to use this bond as a port of other devices.
 - a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection, enter:

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

- b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection, enter:

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

5. Activate the connection:

```
# nmcli connection up bond0
```

6. 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:

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

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

- b. Reactivate the bridge:

```
# nmcli connection up bond0
```

Verification steps

1. Temporarily remove the network cable from the host.
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
```

Additional resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- [Network bonding documentation](#)

9.6. CONFIGURING A NETWORK BOND USING THE RHEL WEB CONSOLE

This section describes how to configure a network bond using the RHEL web console.

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 team, bridge, or VLAN devices as members of the bond, create them in advance as described in:
 - [Configuring a network team using the RHEL web console](#)
 - [Configuring a network bridge using the RHEL web console](#)
 - [Configuring VLAN tagging 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	<input type="text" value="bond0"/>
Interfaces	<div><input checked="" type="checkbox"/> enp7s0</div> <div><input checked="" type="checkbox"/> enp8s0</div>
MAC	<input type="text"/>
Mode	<input type="text" value="Active backup"/>
Primary	<input type="text" value="enp7s0"/>
Link monitoring	<input type="text" value="MII (recommended)"/>
Monitoring interval	<input type="text" value="100"/>
Link up delay	<input type="text" value="0"/>
Link down delay	<input type="text" value="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> Add bond Add team Add bridge Add VLAN </div>			
Name	IP address	Sending	Receiving
bond0	192.0.2.1/24	1.11 Mbps	61.2 Mbps

- Temporarily remove the network cable from the host.

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.

- Display the status of the bond:

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

9.7. CONFIGURING A NETWORK BOND 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 deselect checkboxes by using **Space**.

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

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

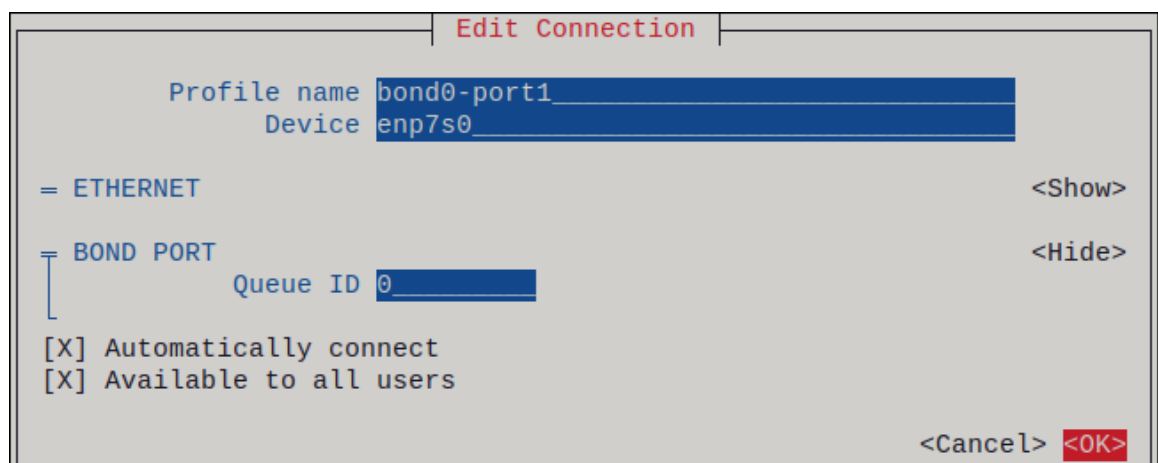
```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

- Start **nmtui**:

```
# nmtui
```


3. Select **Edit a connection**, and press **Enter**.
4. Press the **Add** button.
5. Select **Bond** from the list of network types, and press **Enter**.
6. Optional: Enter a name for the NetworkManager profile to be created.
7. Enter the bond device name to be created into the **Device** field.
8. Add ports to the bond to be created:
 - a. Press the **Add** button 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 the **OK** button to return to the window with the bond settings.

Figure 9.1. 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 **Automatic** button, and select:
 - **Disabled**, if the bond does not require an IP address.
 - **Automatic**, if a DHCP server 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 the **Show** button next to the protocol you want to configure to display additional fields.
 - ii. Press the **Add** button 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 the **Add** button next to **DNS servers**, and enter the DNS server address.
- v. Press the **Add** button next to **Search domains**, and enter the DNS search domain.

Figure 9.2. 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

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

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 the **OK** button to create and automatically activate the new connection.

12. Press the **Back** button 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 the host.
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
```

9.8. CONFIGURING A NETWORK BOND USING NM-CONNECTION-EDITOR

This section describes how to configure a network bond using the **nm-connection-editor** application.

Note that **nm-connection-editor** can add only new ports to a bond. To use an existing connection profile as a port, create the bond using the **nmcli** utility as described in [Configuring a network bond using nmcli commands](#).

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 team, bond, or VLAN devices as ports of the bond, ensure that these devices are not already configured.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Bond** connection type, and click **Create**.
4. In the **Bond** tab:
 - a. Optional: Set the name of the bond interface in the **Interface name** field.
 - b. Click the **Add** button to add a network interface as a port to the bond.
 - i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.
 - ii. Optional: Set a connection name for the port.

- iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the bond. If you selected a different device type, configure it accordingly. Note that you can only use Ethernet interfaces in a bond that are not configured.
- iv. Click **Save**.
- c. Repeat the previous step for each interface you want to add to the bond:

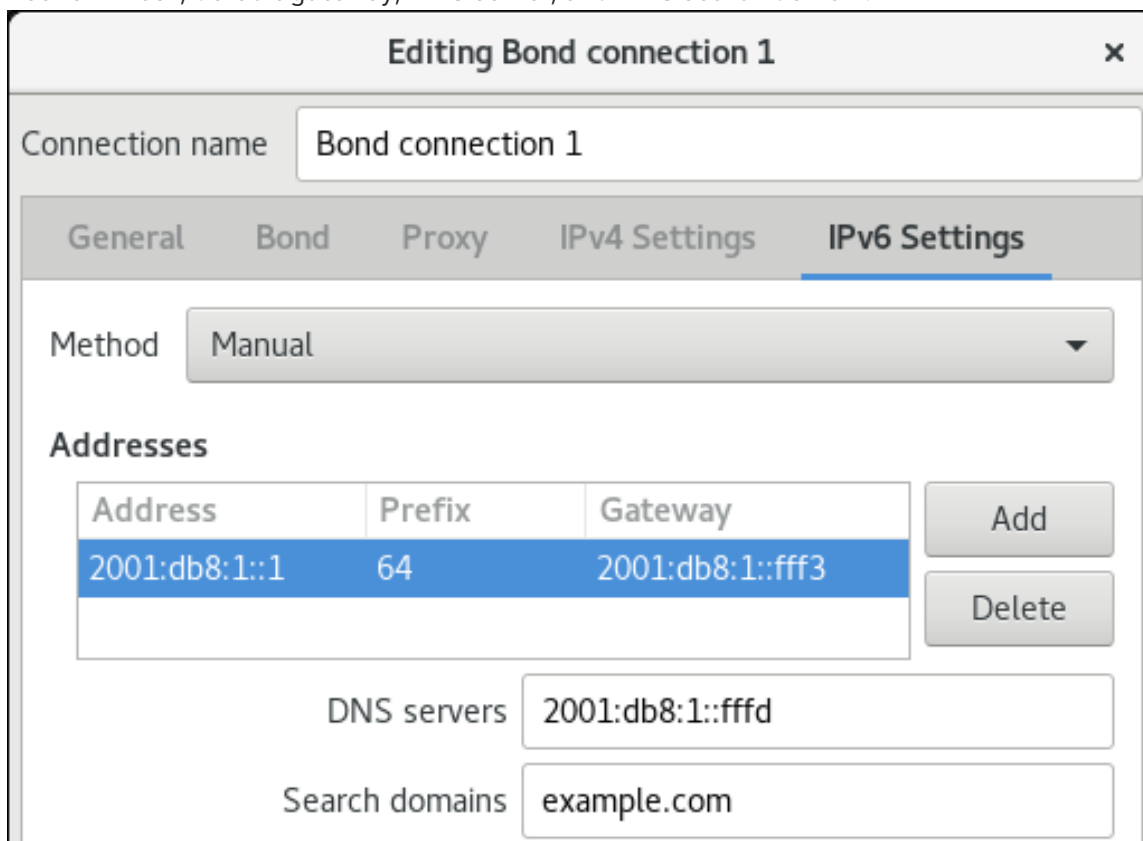
The screenshot shows the 'Editing Bond connection 1' dialog box with the 'Bond' tab selected. The 'Connection name' field is 'Bond connection 1'. The 'Interface name' field is 'bond0'. Under 'Bonded connections', there is a list with 'bond0-port1' and 'bond0-port2'. To the right of this list are 'Add' and 'Edit' buttons.

- d. Optional: Set other options, such as the Media Independent Interface (MII) monitoring interval.
5. Configure the IP settings of the bond. Skip this step if you want to use this bond as a port of other devices.
 - a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:

The screenshot shows the 'Editing Bond connection 1' dialog box with the 'IPv4 Settings' tab selected. The 'Connection name' field is 'Bond connection 1'. The 'Method' dropdown is set to 'Manual'. Under 'Addresses', there is a table with one row: Address '192.0.2.1', Netmask '24', and Gateway '192.0.2.254'. To the right of the table are 'Add' and 'Delete' buttons. Below the table, the 'DNS servers' field is '192.0.2.253' and the 'Search domains' field is 'example.com'.

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

- b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Bond connection 1

Connection name: Bond connection 1

General Bond Proxy IPv4 Settings **IPv6 Settings**

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

Add Delete

DNS servers: 2001:db8:1::fffd

Search domains: example.com

- Click **Save** to save the bond connection.
- Close **nm-connection-editor**.

Verification steps

- Temporarily remove the network cable from the host.
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.
- Display the status of the bond:

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

Additional resources

- [Configuring NetworkManager to avoid using a specific profile to provide a default gateway](#)
- [Configuring a network team using nm-connection-editor](#)
- [Configuring a network bridge using nm-connection-editor](#)
- [Configuring VLAN tagging using nm-connection-editor](#)

9.9. CONFIGURING A NETWORK BOND USING NMSTATECTL

This section describes how to use the **nmstatectl** utility to configure a network bond, **bond0**, 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**

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 contents:

```
---
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

```

2. Apply the settings to the system:

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

Verification steps

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:

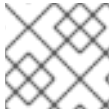
```
# nmstatectl show bond0
```


Additional resources

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

9.10. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure a network bond. This procedure describes how to configure a bond in active-backup mode that uses two Ethernet devices, and sets an IPv4 and IPv6 addresses, default gateways, and DNS configuration.



NOTE

Set the IP configuration on the bond and not on the ports of the Linux bond.

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- Two or more physical or virtual network devices are installed on the server.

Procedure

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

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    name: Configure a network bond that uses two Ethernet ports
    - include_role:
        name: rhel-system-roles.network

  vars:
    network_connections:
      # Define the bond profile
      - name: bond0
        type: bond
        interface_name: bond0
      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
bond:
  mode: active-backup
  state: up

# Add an Ethernet profile to the bond
- name: bond0-port1
  interface_name: enp7s0
  type: ethernet
  controller: bond0
  state: up

# Add a second Ethernet profile to the bond
- name: bond0-port2
  interface_name: enp8s0
  type: ethernet
  controller: bond0
  state: up

```

2. Run the playbook:

```
# ansible-playbook ~/bond-ethernet.yml
```

Additional resources

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

9.11. CREATING A NETWORK BOND TO ENABLE SWITCHING BETWEEN AN ETHERNET AND WIRELESS CONNECTION WITHOUT INTERRUPTING THE VPN

RHEL users who connect their workstation to their company's network typically use a VPN to access remote resources. However, if the workstation switches between an Ethernet and Wi-Fi connection, for example, if you release a laptop from a docking station with an Ethernet connection, the VPN connection is interrupted. To avoid this problem, you can create a network bond that uses the Ethernet and Wi-Fi connection in **active-backup** mode.

Prerequisites

- The host contains an Ethernet and a Wi-Fi device.
- An Ethernet and Wi-Fi NetworkManager connection profile has been created and both connections work independently.
This procedure uses the following connection profiles to create a network bond named **bond0**:
 - **Docking_station** associated with the **enp11s0u1** Ethernet device
 - **Wi-Fi** associated with the **wlp1s0** Wi-Fi device

Procedure

1. Create a bond interface in **active-backup** mode:

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

This command names both the interface and connection profile **bond0**.

2. Configure the IPv4 settings of the bond:

- If a DHCP server in your network assigns IPv4 addresses to hosts, no action is required.
- If your local network requires static IPv4 addresses, set the address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection:

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

3. Configure the IPv6 settings of the bond:

- If your router or a DHCP server in your network assigns IPv6 addresses to hosts, no action is required.
- If your local network requires static IPv6 addresses, set the address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection:

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

4. Display the connection profiles:

```
# nmcli connection show
NAME          UUID                                  TYPE    DEVICE
Docking_station 256dd073-fecc-339d-91ae-9834a00407f9 ethernet enp11s0u1
Wi-Fi          1f1531c7-8737-4c60-91af-2d21164417e8 wifi     wlp1s0
...
```

You require the names of the connection profiles and the Ethernet device name in the next steps.

5. Assign the connection profile of the Ethernet connection to the bond:

```
# nmcli connection modify Docking_station master bond0
```

6. Assign the connection profile of the Wi-Fi connection to the bond:

```
# nmcli connection modify Wi-Fi master bond0
```

7. If your Wi-Fi network uses MAC filtering to allow only MAC addresses on a allow list to access the network, configure that NetworkManager dynamically assigns the MAC address of the active port to the bond:

```
# nmcli connection modify bond0 +bond.options fail_over_mac=1
```

With this setting, you must set only the MAC address of the Wi-Fi device to the allow list instead of the MAC address of both the Ethernet and Wi-Fi device.

8. Set the device associated with the Ethernet connection as primary device of the bond:

```
# nmcli con modify bond0 +bond.options "primary=enp11s0u1"
```

With this setting, the bond always uses the Ethernet connection if it is available.

9. Configure that NetworkManager automatically activates ports when the **bond0** device is activated:

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

10. Activate the **bond0** connection:

```
# nmcli connection up bond0
```

Verification steps

- Display the currently active device, the status of the bond and its ports:

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

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
```

```
Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
```

```
Primary Slave: enp11s0u1 (primary_reselect always)
```

```
Currently Active Slave: enp11s0u1
```

```
MII Status: up
```

```
MII Polling Interval (ms): 1
```

```
Up Delay (ms): 0
```

```
Down Delay (ms): 0
```

```
Peer Notification Delay (ms): 0
```

```
Slave Interface: enp11s0u1
```

```
MII Status: up
```

```
Speed: 1000 Mbps
```

```
Duplex: full
```

```
Link Failure Count: 0
```

```
Permanent HW addr: 00:53:00:59:da:b7
```

```
Slave queue ID: 0
```

```
Slave Interface: wlp1s0
```

```
MII Status: up
```

```
Speed: Unknown
```

```
Duplex: Unknown
```

```
Link Failure Count: 2
```

```
Permanent HW addr: 00:53:00:b3:22:ba
```

```
Slave queue ID: 0
```

Additional resources

- [Configuring an Ethernet connection](#)
- [Managing Wi-Fi connections](#)
- [Configuring network bonding](#)

9.12. 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 following modes exist:

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](#).

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?](#)

- How are the values for different policies in "xmit_hash_policy" bonding parameter calculated?

9.13. 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 10. CONFIGURING A VPN CONNECTION

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.

10.1. 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.

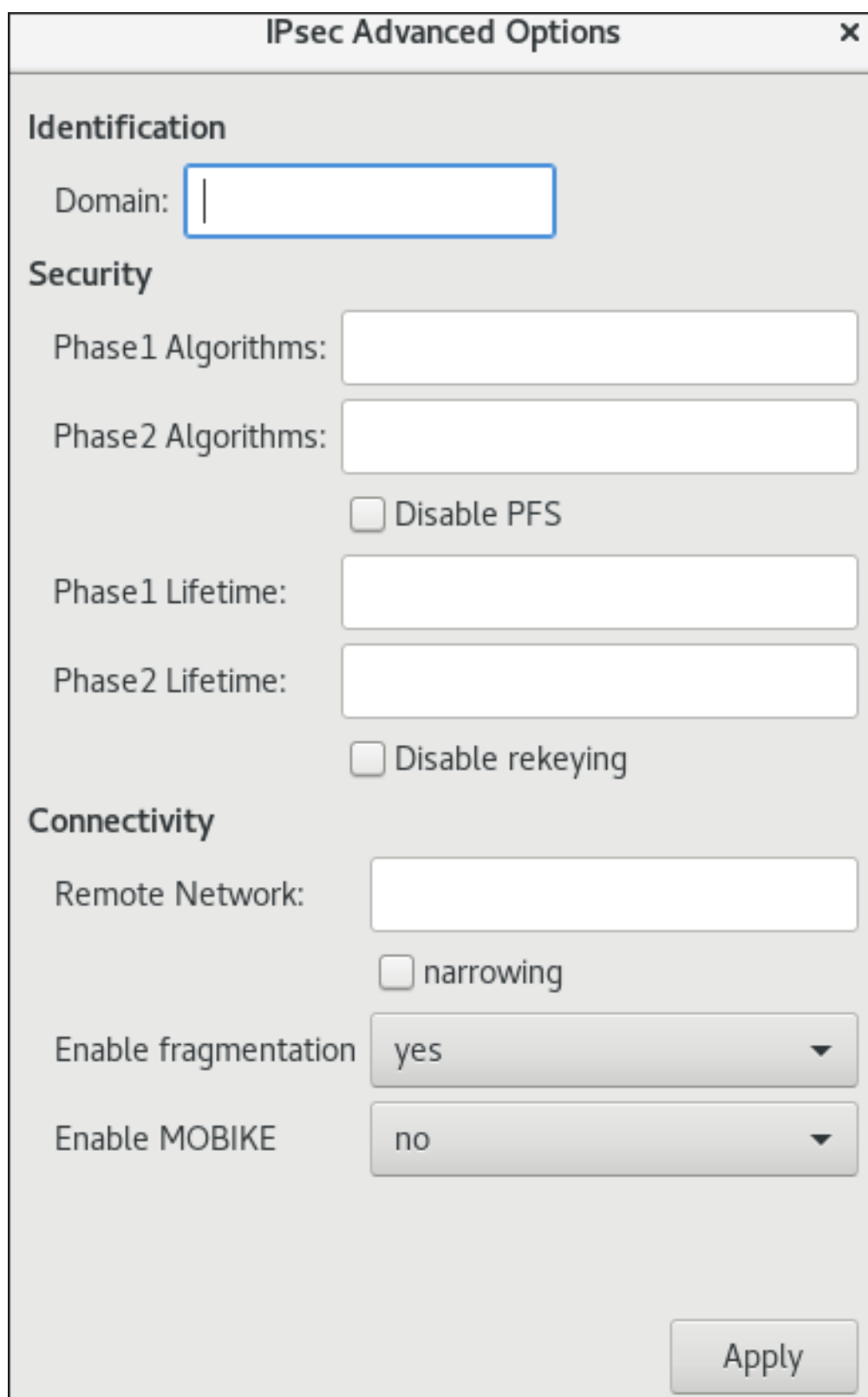
Authentication

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 10.1. Advanced options of a VPN connection



The image shows a window titled "IPsec Advanced Options" with a close button (X) in the top right corner. The window is divided into three sections: Identification, Security, and Connectivity. The Identification section has a "Domain:" label followed by an empty text input field. The Security section has "Phase1 Algorithms:" and "Phase2 Algorithms:" labels, each followed by an empty text input field. Below these are two checkboxes: "Disable PFS" and "Disable rekeying", both of which are unchecked. The Connectivity section has a "Remote Network:" label followed by an empty text input field. Below this is a checkbox labeled "narrowing", which is unchecked. There are two dropdown menus: "Enable fragmentation" with "yes" selected, and "Enable MOBIKE" with "no" selected. An "Apply" button is located at the bottom right of the window.

IPsec Advanced Options ✕

Identification

Domain:

Security

Phase1 Algorithms:

Phase2 Algorithms:

☐ Disable PFS

Phase1 Lifetime:

Phase2 Lifetime:

☐ Disable rekeying

Connectivity

Remote Network:

☐ narrowing

Enable fragmentation yes ▼

Enable MOBIKE no ▼

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 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 active the **VPN** connection.

Additional resources

- **nm-settings-libreswan(5)**

10.2. CONFIGURING A VPN CONNECTION USING NM-CONNECTION-EDITOR

If you use Red Hat Enterprise Linux with a graphical interface, you can configure a VPN connection in the **nm-connection-editor** application.

Prerequisites

- The **NetworkManager-libreswan-gnome** package is installed.
- If you configure an Internet Key Exchange version 2 (IKEv2) connection:
 - The certificate is imported into the IPsec network security services (NSS) database.
 - The nickname of the certificate in the NSS database is known.

Procedure

1. Open a terminal, and enter:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **IPsec based VPN** connection type, and click **Create**.
4. On the **VPN** tab:
 - a. Enter the host name or IP address of the VPN gateway into the **Gateway** field, and select an authentication type. Based on the authentication type, you must enter different additional information:
 - **IKEv2 (Certificate)** authenticates the client by using a certificate, which is more secure. This setting requires the nickname of the certificate in the IPsec NSS database
 - **IKEv1 (XAUTH)** authenticates the user by using a user name and password (pre-shared key). This setting requires that you enter the following values:
 - User name
 - Password
 - Group name
 - Secret
 - b. If the remote server specifies a local identifier for the IKE exchange, enter the exact string in the **Remote ID** field. In the remote server runs Libreswan, this value is set in the server's **leftid** parameter.

Editing VPN connection 1 [X]

Connection name:

General | **VPN** | Proxy | IPv4 Settings

General

Gateway:

Authentication

Type:

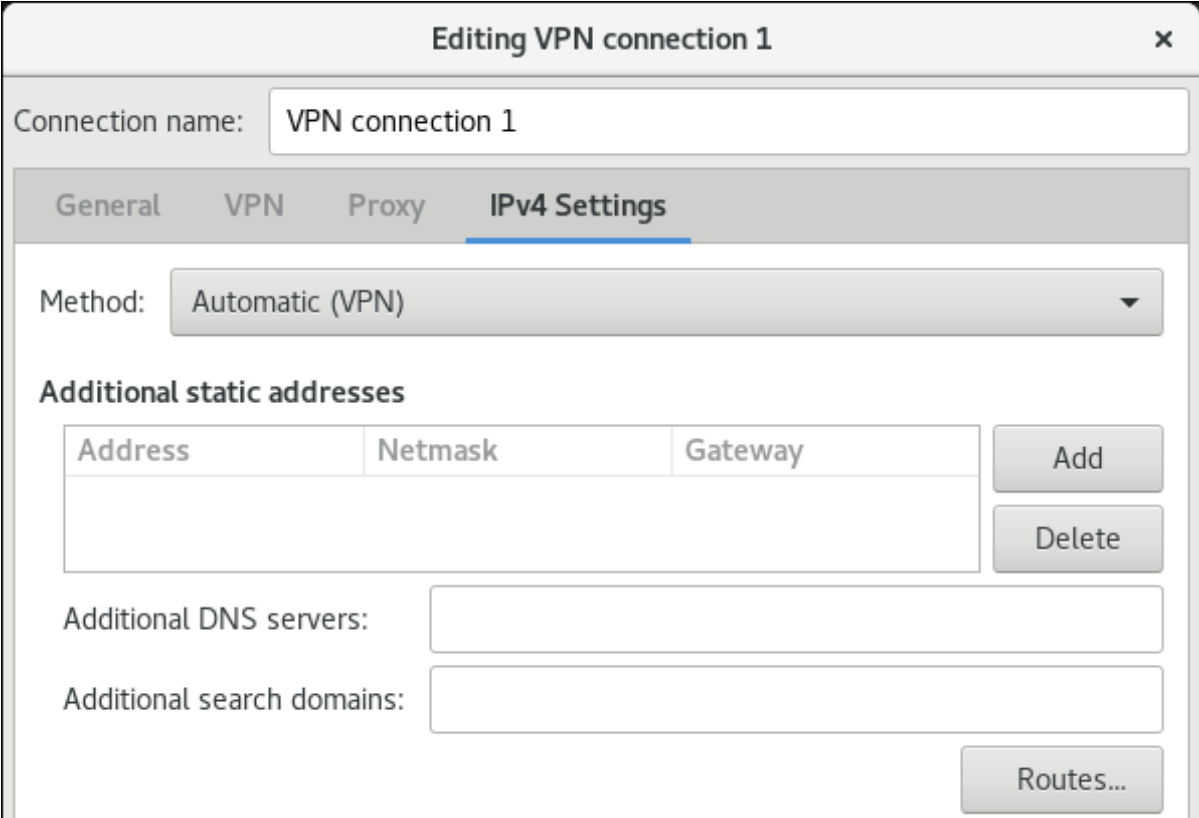
Certificate name:

Remote ID:

Advanced...

- c. Optionally, configure additional settings by clicking the **Advanced** button. You can configure the following settings:
- 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. This parameter defines how long the key used to encrypt the traffic is valid.
 - **Phase2 Lifetime** corresponds to the **salifetime** Libreswan parameter. This parameter defines how long a security association is valid.
 - Connectivity

- **Remote network** corresponds to the **rightsubnet** Libreswan parameter and defines 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 the IKEv2 negotiation.
 - **Enable fragmentation** corresponds to the **fragmentation** Libreswan parameter and defines whether or not to allow IKE fragmentation. Valid values are **yes** (default) or **no**.
 - **Enable Mobike** corresponds to the **mobike** Libreswan parameter. The parameter defines whether 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**.
5. On the **IPv4 Settings** tab, select the IP assignment method and, optionally, set additional static addresses, DNS servers, search domains, and routes.



Editing VPN connection 1 [X]

Connection name:

General VPN Proxy **IPv4 Settings**

Method:

Additional static addresses

Address	Netmask	Gateway
<input type="text"/>		

Additional DNS servers:

Additional search domains:

6. Save the connection.
7. Close **nm-connection-editor**.



NOTE

When you add a new connection by clicking the **+** button, **NetworkManager** creates a new configuration file for that connection and then opens the same dialog that is used for editing an existing connection. The difference between these dialogs is that an existing connection profile has a **Details** menu entry.

Additional resources

- **nm-settings-libreswan(5)** man page

10.3. CONFIGURING AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections over Ethernet. By default, Libreswan detects if hardware supports this feature and, as a result, enables ESP hardware offload. In case that the feature was disabled or explicitly enabled, you can switch back to automatic detection.

Prerequisites

- The network card supports ESP hardware offload.
- The network driver supports ESP hardware offload.
- The IPsec connection is configured and works.

Procedure

1. Edit the Libreswan configuration file in the `/etc/ipsec.d/` directory of the connection that should use automatic detection of ESP hardware offload support.
2. Ensure the **nic-offload** parameter is not set in the connection's settings.
3. If you removed **nic-offload**, restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

If the network card supports ESP hardware offload support, following these steps to verify the result:

1. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device the IPsec connection uses:

```
# ethtool -S enp1s0 | egrep "_ipsec"
tx_ipsec: 10
rx_ipsec: 10
```

2. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

3. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device again:

```
# ethtool -S enp1s0 | egrep "_ipsec"
tx_ipsec: 15
rx_ipsec: 15
```

If the counter values have increased, ESP hardware offload works.

Additional resources

- [Configuring a VPN with IPsec](#)

10.4. CONFIGURING ESP HARDWARE OFFLOAD ON A BOND TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections. If you use a network bond for fail-over reasons, the requirements and the procedure to configure ESP hardware offload are different from those using a regular Ethernet device. For example, in this scenario, you enable the offload support on the bond, and the kernel applies the settings to the ports of the bond.

Prerequisites

- All network cards in the bond support ESP hardware offload.
- The network driver supports ESP hardware offload on a bond device. In RHEL, only the **ixgbe** driver supports this feature.
- The bond is configured and works.
- The bond uses the **active-backup** mode. The bonding driver does not support any other modes for this feature.
- The IPsec connection is configured and works.

Procedure

1. Enable ESP hardware offload support on the network bond:

```
# nmcli connection modify bond0 ethtool.feature-esp-hw-offload on
```

This command enables ESP hardware offload support on the **bond0** connection.

2. Reactivate the **bond0** connection:

```
# nmcli connection up bond0
```

3. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use ESP hardware offload, and append the **nic-offload=yes** statement to the connection entry:

```
conn example
...
nic-offload=yes
```

4. Restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

1. Display the active port of the bond:

```
# grep "Currently Active Slave" /proc/net/bonding/bond0
Currently Active Slave: enp1s0
```

2. Display the **tx_ipsec** and **rx_ipsec** counters of the active port:

—

```
# ethtool -S enp1s0 | egrep "_ipsec"
tx_ipsec: 10
rx_ipsec: 10
```

3. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

4. Display the **tx_ipsec** and **rx_ipsec** counters of the active port again:

```
# ethtool -S enp1s0 | egrep "_ipsec"
tx_ipsec: 15
rx_ipsec: 15
```

If the counter values have increased, ESP hardware offload works.

Additional resources

- [Configuring network bonding](#)
- [Configuring a VPN with IPsec](#) section in the Securing networks document

CHAPTER 11. 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.

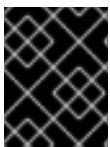
NetworkManager supports the following IP tunnels:

- 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.

11.1. CONFIGURING AN IPIP TUNNEL USING NMCLI 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](#).

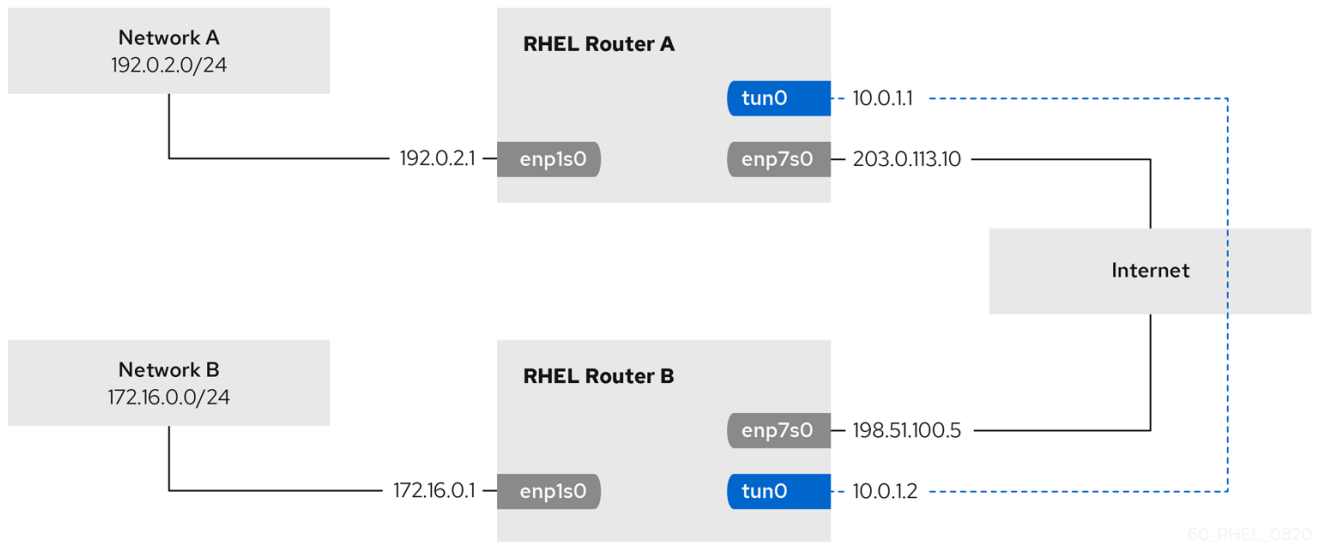


IMPORTANT

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 steps

- 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**:

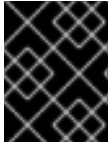
```
# ping 192.0.2.1
```

Additional resources

- **nmcli(1)** man page
- **nm-settings(5)** man page

11.2. CONFIGURING A GRE TUNNEL USING NMCLI TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS

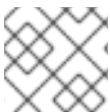
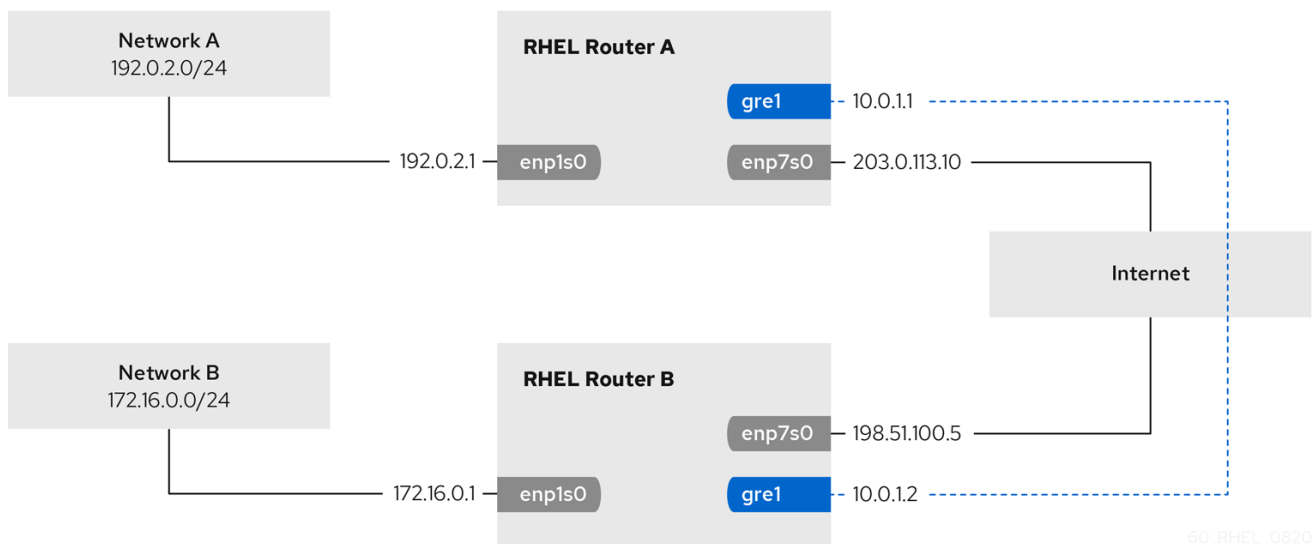
A Generic Routing Encapsulation (GRE) tunnel encapsulates layer-3 traffic in IPv4 packets as described in [RFC 2784](#). A GRE tunnel can encapsulate any layer 3 protocol with a valid Ethernet type.



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:



NOTE

The **gre0** device name is reserved. Use **gre1** or a different name for the device.

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.

- 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 steps

1. 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
```

Additional resources

- **nmcli(1)** man page
- **nm-settings(5)** man page

11.3. CONFIGURING A GRE TAP TUNNEL TO TRANSFER ETHERNET FRAMES OVER IPV4

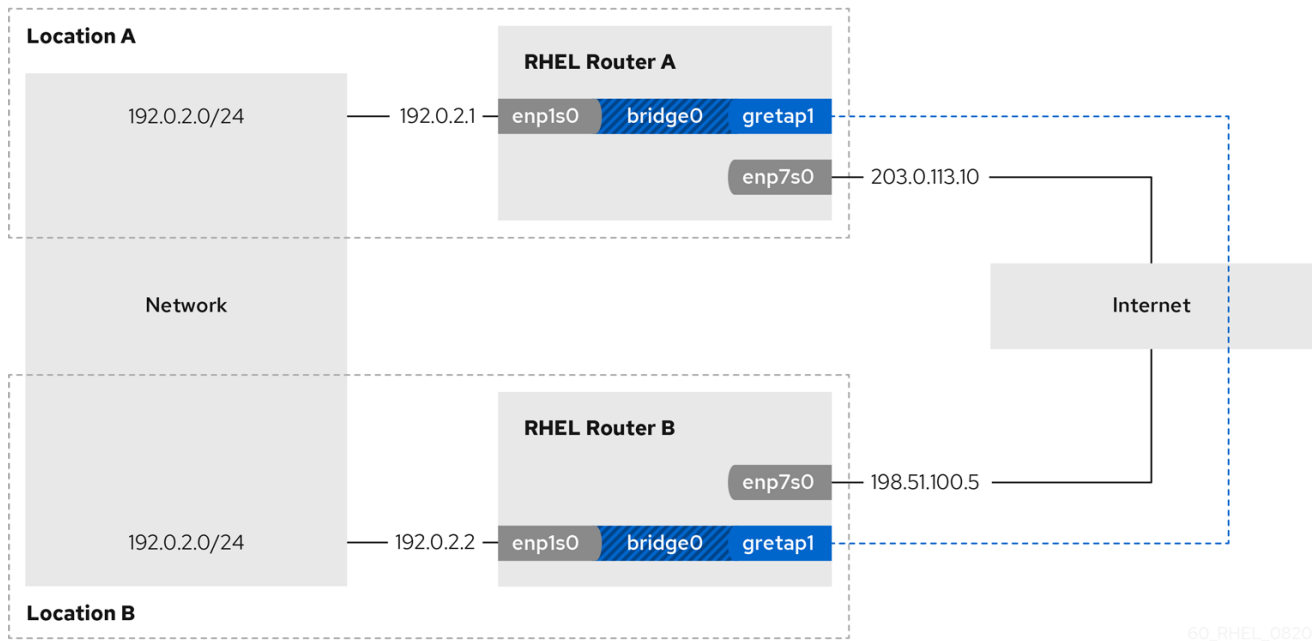
A Generic Routing Encapsulation Terminal Access Point (GRE TAP) tunnel operates on OSI level 2 and encapsulates Ethernet traffic in IPv4 packets as described in [RFC 2784](#).



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:



NOTE

The **gretap0** device name is reserved. Use **gretap1** or a different name for the device.

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 slave-type bridge con-name bridge0-port1
ifname enp1s0 master 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 slave-type bridge
con-name bridge0-port2 ifname gretap1 remote 198.51.100.5 local 203.0.113.10
master 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
```

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-slaves 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 slave-type bridge con-name bridge0-port1
ifname enp1s0 master bridge0
```

- d. Add a new connection profile for the GRETap tunnel interface to the bridge:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap slave-type bridge
con-name bridge0-port2 ifname gretap1 remote 203.0.113.10 local 198.51.100.5
master 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-slaves 1
```

- g. Active the **bridge0** connection:

```
# nmcli connection up bridge0
```

Verification steps

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
```

Additional resources

- **nmcli(1)** man page
- **nm-settings(5)** man page

11.4. ADDITIONAL RESOURCES

- **ip-link(8)** man page

CHAPTER 12. CHANGING A HOSTNAME

The hostname of a system is the name on the system itself. You can set the name when you install RHEL, and you can change it afterwards.

12.1. CHANGING A HOSTNAME USING NMCLI

You can use the **nmcli** utility to update the system hostname. Note that other utilities, might use a different term, such as static or persistent hostname.

Procedure

1. Optional: Display the current hostname setting:

```
# nmcli general hostname  
old-hostname.example.com
```

2. Set the new hostname:

```
# nmcli general hostname new-hostname.example.com
```

3. NetworkManager automatically restarts the **systemd-hostnamed** to activate the new name. However, the following manual actions can be required if you do not want to reboot the host:

- 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:

```
# nmcli general hostname  
new-hostname.example.com
```

12.2. CHANGING A HOSTNAME 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. However, the following manual actions can be required if you do not want to reboot the host:

- 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 13. LEGACY NETWORK SCRIPTS SUPPORT IN RHEL

By default, RHEL uses NetworkManager to configure and manage network connections, and the **/usr/sbin/ifup** and **/usr/sbin/ifdown** scripts use NetworkManager to process **ifcfg** files in the **/etc/sysconfig/network-scripts/** directory.



IMPORTANT

The legacy scripts are deprecated in RHEL 8 and will be removed in a future major version of RHEL. If you still use the legacy network scripts, for example, because you upgraded from an earlier version to RHEL 8, Red Hat recommends that you migrate your configuration to NetworkManager.

13.1. INSTALLING THE LEGACY NETWORK SCRIPTS

If you require the deprecated network scripts that processes the network configuration without using NetworkManager, you can install them. In this case, the **/usr/sbin/ifup** and **/usr/sbin/ifdown** scripts link to the deprecated shell scripts that manage the network configuration.

Procedure

- Install the **network-scripts** package:

```
# yum install network-scripts
```

CHAPTER 14. PORT MIRRORING

Network administrators can use port mirroring to replicate inbound and outbound network traffic being communicated from one network device to another. Administrators use port mirroring to monitor network traffic and collect network data to:

- Debug networking issues and tune the network flow
- Inspect and analyze the network traffic to troubleshoot networking problems
- Detect an intrusion

14.1. MIRRORING A NETWORK INTERFACE USING NMCLI

You can configure port mirroring using NetworkManager. The following procedure mirrors the network traffic from **enp1s0** to **enp7s0** by adding Traffic Control (**tc**) rules and filters to the **enp1s0** network interface.

Prerequisites

- A network interface to mirror the network traffic to.

Procedure

1. Add a network connection profile that you want to mirror the network traffic from:

```
# nmcli connection add type ethernet ifname enp1s0 con-name enp1s0 autoconnect
no
```

2. Attach a **prio qdisc** to **enp1s0** for the egress (outgoing) traffic with the **10:** handle:

```
# nmcli connection modify enp1s0 +tc.qdisc "root prio handle 10:"
```

The **prio qdisc** attached without children allows attaching filters.

3. Add a **qdisc** for the ingress traffic, with the **ffff:** handle:

```
# nmcli connection modify enp1s0 +tc.qdisc "ingress handle ffff:"
```

4. Add the following filters to match packets on the ingress and egress **qdiscs**, and to mirror them to **enp7s0**:

```
# nmcli connection modify enp1s0 +tc.tfilter "parent ffff: matchall action mirrored egress
mirror dev enp7s0"

# nmcli connection modify enp1s0 +tc.tfilter "parent 10: matchall action mirrored egress
mirror dev enp7s0"
```

The **matchall** filter matches all packets, and the **mirrored** action redirects packets to destination.

5. Activate the connection:

```
# nmcli connection up enp1s0
```

Verification steps

1. Install the **tcpdump** utility:

```
# yum install tcpdump
```

2. Display the traffic mirrored on the target device (**enp7s0**):

```
# tcpdump -i enp7s0
```

Additional resources

- [How to capture network packets using **tcpdump**](#)

CHAPTER 15. CONFIGURING NETWORKMANAGER TO IGNORE CERTAIN DEVICES

By default, NetworkManager manages all devices except the **lo** (loopback) device. However, you can set certain devices as **unmanaged** to configure that NetworkManager ignores these devices. With this setting, you can manually manage these devices, for example, using a script.

15.1. 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.

To temporarily configure network devices as **unmanaged**, see [Temporarily configuring a device as unmanaged in NetworkManager](#).

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 semicolon:

3. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

Verification steps

- 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.

Additional resources

- **NetworkManager.conf(5)** man page

15.2. TEMPORARILY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER

You can temporarily configure devices as **unmanaged**.

Use this method, for example, for testing purposes. To permanently configure network devices as **unmanaged**, see [Permanently configuring a device as unmanaged in NetworkManager](#) .

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 steps

- 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.

Additional resources

- **NetworkManager.conf(5)** man page

CHAPTER 16. CONFIGURING NETWORK DEVICES TO ACCEPT TRAFFIC FROM ALL MAC ADDRESSES

Network devices usually intercept and read packets that their controller is programmed to receive. You can configure the network devices to accept traffic from all MAC addresses in a virtual switch or at the port group level.

You can use this network mode to:

- diagnose network connectivity issues,
- monitor network activity for security reasons,
- intercept private data-in-transit or intrusion in the network.

You can enable this mode for any kind of network device, except **InfiniBand**.

16.1. TEMPORARILY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING IPROUTE2

This procedure describes how to configure a network device to accept all traffic regardless of the MAC addresses. Any change made using the **iproute2** utility is temporary and lost after the machine reboots.

Procedure

1. Optional: Display the network interfaces to identify the one for which you want to receive all traffic:

```
# ip a
1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state
DOWN group default qlen 1000
    link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
2: bond0: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc
noqueue state DOWN group default qlen 1000
    link/ether 6a:fd:16:b0:83:5c brd ff:ff:ff:ff:ff:ff
3: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default qlen 1000
...
```

2. Modify the device to enable or disable this property.

- To enable the **accept-all-mac-addresses** mode for **enp1s0**:

```
# ip link set enp1s0 promisc on
```

- To disable the **accept-all-mac-addresses** mode for **enp1s0**:

```
# ip link set enp1s0 promisc off
```

Verification steps

- To verify that the **accept-all-mac-addresses** mode is enabled:

```
# ip link show enp1s0
1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default qlen 1000
    link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
```

The **PROMISC** flag in the device description indicates that the mode is enabled.

16.2. PERMANENTLY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMCLI

This procedure describes how to configure a network device to accept traffic regardless of MAC addresses using the **nmcli** commands.

Procedure

- Optional: Display the network interfaces to identify the one for which you want to receive all traffic:

```
# ip a
1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
2: bond0: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 6a:fd:16:b0:83:5c brd ff:ff:ff:ff:ff:ff
3: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
...
```

You can create a new connection, if you do not have any.

- Modify the network device to enable or disable this property.

- To enable the **ethernet.accept-all-mac-addresses** mode for **enp1s0**:

```
# nmcli connection modify enp1s0 ethernet.accept-all-mac-addresses yes
```

- To disable the **accept-all-mac-addresses** mode for **enp1s0**:

```
# nmcli connection modify enp1s0 ethernet.accept-all-mac-addresses no
```

- To apply the changes, reactivate the connection:

```
# nmcli connection up enp1s0
```

Verification steps

- To verify that the **ethernet.accept-all-mac-addresses** mode is enabled:

```
# nmcli connection show enp1s0
...
802-3-ethernet.accept-all-mac-addresses:1 (true)
```

The **802-3-ethernet.accept-all-mac-addresses: true** indicates that the mode is enabled.

16.3. PERMANENTLY CONFIGURING A NETWORK NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMSTATECTL

This procedure describes how to configure a network device to accept all traffic regardless of MAC addresses using the **nmstatectl** utility.

Prerequisites

- The **nmstate** package is installed.
- The **.yaml** file that you used to configure the device is available.

Procedure

1. Edit the existing **enp1s0.yaml** file for the *enp1s0* connection and add the following content to it.

```
---
interfaces:
  - name: enp1s0
    type: ethernet
    state: up
    accept-all-mac-address: true
```

2. Apply the network settings.

```
# nmstatectl apply ~/enp1s0.yaml
```

Verification steps

- To verify that the **802-3-ethernet.accept-all-mac-addresses** mode is enabled:

```
# nmstatectl show enp1s0
interfaces:
  - name: enp1s0
    type: ethernet
    state: up
    accept-all-mac-addresses: true
...
```

The **802-3-ethernet.accept-all-mac-addresses: true** indicates that the mode is enabled.

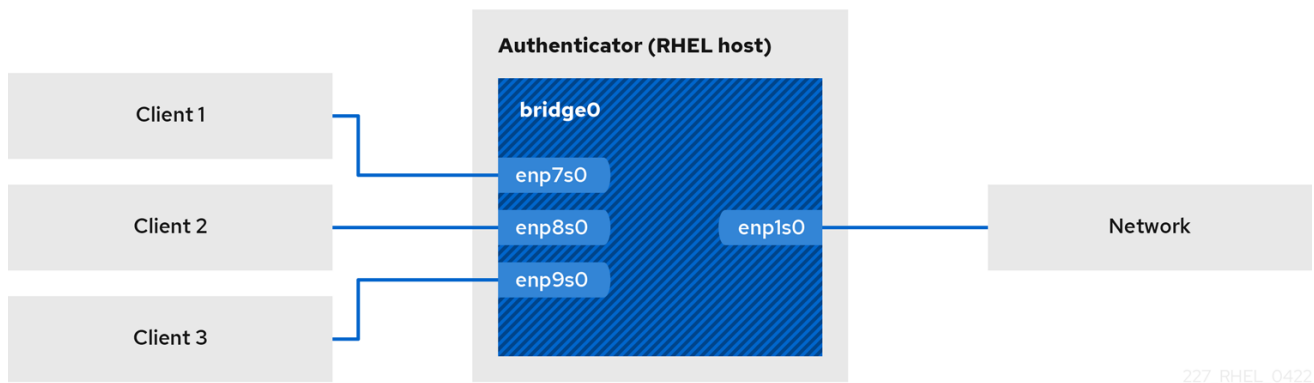
Additional resources

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

CHAPTER 17. SETTING UP AN 802.1X NETWORK AUTHENTICATION SERVICE FOR LAN CLIENTS USING HOSTAPD WITH FREERADIUS BACKEND

The IEEE 802.1X standard defines secure authentication and authorization methods to protect networks from unauthorized clients. Using the **hostapd** service and FreeRADIUS, you can provide network access control (NAC) in your network.

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.



227_RHEL_0422

17.1. PREREQUISITES

- A clean installation of FreeRADIUS.
If the **freeradius** package is already installed, remove the **/etc/raddb/** directory, uninstall and then install the package again. Do not reinstall the package using the **yum reinstall** command, because the permissions and symbolic links in the **/etc/raddb/** directory are then different.

17.2. 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. Create the bridge interface:

```
# nmcli connection add type bridge con-name br0 ifname br0
```

2. Assign the Ethernet interfaces to the bridge:

```
# nmcli connection add type ethernet slave-type bridge con-name br0-port1 ifname enp1s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port2 ifname
```

```

enp7s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port3 ifname
enp8s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port4 ifname
enp9s0 master 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-slaves 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

17.3. CERTIFICATE REQUIREMENTS BY FREERADIUS

For a secure FreeRADIUS service, you require TLS certificates for different purposes:

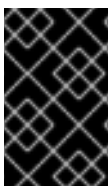
- A TLS server certificate for encrypted connections to the server. Use a trusted certificate authority (CA) to issue the certificate.
The server certificate requires the extended key usage (EKU) field set to **TLS Web Server Authentication**.
- Client certificates issued by the same CA for extended authentication protocol transport layer security (EAP-TLS). EAP-TLS provides certificate-based authentication and is enabled by default.
The client certificates require their EKU field set to **TLS Web Client Authentication**.

**WARNING**

To secure connection, use your company's CA or create your own CA to issue certificates for FreeRADIUS. If you use a public CA, you allow it to authenticate users and issue client certificates for EAP-TLS.

17.4. CREATING A SET OF CERTIFICATES ON A FREERADIUS SERVER FOR TESTING PURPOSES

For testing purposes, the **freeradius** package installs scripts and configuration files in the **/etc/raddb/certs/** directory to create your own certificate authority (CA) and issue certificates.

**IMPORTANT**

If you use the default configuration, certificates generated by these scripts expire after 60 days and keys use an insecure password ("whatever"). However, you can customize the CA, server, and client configuration.

After you perform the procedure, the following files, which you require later in this documentation, are created:

- **/etc/raddb/certs/ca.pem**: CA certificate
- **/etc/raddb/certs/server.key**: Private key of the server certificate
- **/etc/raddb/certs/server.pem**: Server certificate
- **/etc/raddb/certs/client.key**: Private key of the client certificate
- **/etc/raddb/certs/client.pem**: Client certificate

Prerequisites

- You installed the **freeradius** package.

Procedure

1. Change into the **/etc/raddb/certs/** directory:

```
# cd /etc/raddb/certs/
```

2. Optional: Customize the CA configuration:

```
...
[ req ]
default_bits      = 2048
input_password    = ca_password
output_password   = ca_password
...
```



```
[certificate_authority]
countryName      = US
stateOrProvinceName = North Carolina
localityName     = Raleigh
organizationName  = Example Inc.
emailAddress      = admin@example.org
commonName       = "Example Certificate Authority"
...
```

3. Optional: Customize the server configuration:

```
...
[ CA_default ]
default_days      = 730
...
[ req ]
distinguished_name = server
default_bits       = 2048
input_password     = key_password
output_password    = key_password
...
[server]
countryName       = US
stateOrProvinceName = North Carolina
localityName      = Raleigh
organizationName  = Example Inc.
emailAddress      = admin@example.org
commonName        = "Example Server Certificate"
...
```

4. Optional: Customize the client configuration:

```
...
[ CA_default ]
default_days      = 365
...
[ req ]
distinguished_name = client
default_bits       = 2048
input_password     = password_on_private_key
output_password    = password_on_private_key
...
[client]
countryName       = US
stateOrProvinceName = North Carolina
localityName      = Raleigh
organizationName  = Example Inc.
emailAddress      = user@example.org
commonName        = user@example.org
...
```

5. Create the certificates:

```
# make all
```

6. Change the group on the `/etc/raddb/certs/server.pem` file to **radiusd**:

```
# chgrp radiusd /etc/raddb/certs/server.pem*
```

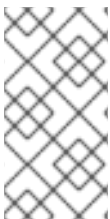
Additional resources

- `/etc/raddb/certs/README.md`

17.5. CONFIGURING FREERADIUS TO AUTHENTICATE NETWORK CLIENTS SECURELY USING EAP

FreeRADIUS supports different methods of the Extensible authentication protocol (EAP). However, for a secure network, this documentation describes how to configure FreeRADIUS to support only the following secure EAP authentication methods:

- EAP-TLS (transport layer security) uses a secure TLS connection to authenticate clients using certificates. To use EAP-TLS, you need TLS client certificates for each network client and a server certificate for the server. Note that the same certificate authority (CA) must have issued the certificates. Always use your own CA to create certificates, because all client certificates issued by the CA you use can authenticate to your FreeRADIUS server.
- EAP-TTLS (tunneled transport layer security) uses a secure TLS connection and authenticates clients using mechanisms, such as password authentication protocol (PAP) or challenge handshake authentication protocol (CHAP). To use EAP-TTLS, you need a TLS server certificate.
- EAP-PEAP (protected extensible authentication protocol) uses a secure TLS connection as the outer authentication protocol to set up the tunnel. The authenticator authenticates the certificate of the RADIUS server. Afterwards, the supplicant authenticates through the encrypted tunnel using Microsoft challenge handshake authentication protocol version 2 (MS-CHAPv2) or other methods.



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** package.
- The configuration files in the `/etc/raddb/` directory are unchanged and as provided by the **freeradius** package.
- The following files exist on the server:
 - TLS private key of the FreeRADIUS host: `/etc/raddb/certs/server.key`
 - TLS server certificate of the FreeRADIUS host: `/etc/raddb/certs/server.pem`
 - TLS CA certificate: `/etc/raddb/certs/ca.pem`

If you store the files in a different location or if they have different names, set the **private_key_file**, **certificate_file**, and **ca_file** parameters in the **/etc/raddb/mods-available/eap** file accordingly.

Procedure

1. If the **/etc/raddb/certs/dh** with Diffie-Hellman (DH) parameters does not exist, create one. 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.

2. Set secure permissions on the TLS private key, server certificate, CA certificate, and the file with DH parameters:

```
# chmod 640 /etc/raddb/certs/server.key /etc/raddb/certs/server.pem
/etc/raddb/certs/ca.pem /etc/raddb/certs/dh
# chown root:radiusd /etc/raddb/certs/server.key /etc/raddb/certs/server.pem
/etc/raddb/certs/ca.pem /etc/raddb/certs/dh
```

3. Edit the **/etc/raddb/mods-available/eap** file:

- a. Set the password of the private key in the **private_key_password** parameter:

```
eap {
    ...
    tls-config tls-common {
        ...
        private_key_password = key_password
        ...
    }
}
```

- b. Depending on your environment, set the **default_eap_type** parameter in the **eap** directive to your primary EAP type you use:

```
eap {
    ...
    default_eap_type = ttls
    ...
}
```

For a secure environment, use only **ttls**, **tls**, or **peap**.

- 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.

4. 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 and disables plain-text authentication methods.

5. 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 = client_password
    ...
}

client localhost_ipv6 {
    ipv6addr = ::1
    secret = client_password
}
```

- b. If RADIUS clients, such as network authenticators, on remote hosts should be able to access the FreeRADIUS service, add corresponding client directives for them:

```
client hostapd.example.org {
    ipaddr = 192.0.2.2/32
    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 an IPv4 and IPv6 address, 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.

- If you want to use EAP-TTLS or EAP-PEAP, add the users to the **/etc/raddb/users** file:

```
example_user    Cleartext-Password := "user_password"
```

For users who should use certificate-based authentication (EAP-TLS), do not add any entry.

- Verify the configuration files:

```
# radiusd -XC
...
Configuration appears to be OK
```

- Enable and start the **radiusd** service:

```
# systemctl enable --now radiusd
```

Verification

- [Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator](#)
- [Testing EAP-TLS authentication against a FreeRADIUS server or authenticator](#)

Troubleshooting

- Stop the **radiusd** service:

```
# systemctl stop radiusd
```

- Start the service in debug mode:

```
# radiusd -X
...
Ready to process requests
```

- Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

Next steps

- Disable unrequired authentication methods and other features you do not use.

17.6. 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=client_password

# RADIUS accounting server
acct_server_addr=192.0.2.1
acct_server_port=1813
acct_server_shared_secret=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

- See:
 - [Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator](#)
 - [Testing EAP-TLS authentication against a FreeRADIUS server or authenticator](#)

Troubleshooting

1. Stop the **hostapd** service:

```
# systemctl stop hostapd
```

2. Start the service in debug mode:

```
# hostapd -d /etc/hostapd/hostapd.conf
```

3. Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

Additional resources

- **hostapd.conf(5)** man page
- **/usr/share/doc/hostapd/hostapd.conf** file

17.7. TESTING EAP-TTLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR

To test if authentication 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/pki/tls/certs/ca.pem** file.

Procedure

1. 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="example_user"
    password="user_password"

    # CA certificate to validate the RADIUS server's identity
    ca_cert="/etc/pki/tls/certs/ca.pem"
}
```

2. 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

17.8. TESTING EAP-TLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR

To test if authentication using extensible authentication protocol (EAP) transport layer security (EAP-TLS) 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/pki/tls/certs/ca.pem** file.
- The CA that issued the client certificate is the same that issued the server certificate of the FreeRADIUS server.
- You stored the client certificate in the **/etc/pki/tls/certs/client.pem** file.
- You stored the private key of the client in the **/etc/pki/tls/private/client.key**

Procedure

1. Create the **/etc/wpa_supplicant/wpa_supplicant-TLS.conf** file with the following content:

```
ap_scan=0

network={
    eap=TLS
    eapol_flags=0
    key_mgmt=IEEE8021X

    identity="user@example.org"
    client_cert="/etc/pki/tls/certs/client.pem"
    private_key="/etc/pki/tls/private/client.key"
    private_key_passwd="password_on_private_key"

    # CA certificate to validate the RADIUS server's identity
    ca_cert="/etc/pki/tls/certs/ca.pem"
}
```

2. To authenticate to:

- A FreeRADIUS server, enter:

```
# eapol_test -c /etc/wpa_supplicant/wpa_supplicant-TLS.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-TLS.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

17.9. 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

if [ "$1" == "xblock_all" ]
then

    nft delete table bridge tr-mgmt-br0 2>/dev/null || true
    nft -f - << EOF
table bridge tr-mgmt-br0 {
    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 ibrname "br0" jump accesscontrol
        }
    }
EOF
    echo "802-1x-tr-mgmt Blocking all traffic through br0. Traffic for given host will be allowed
after 802.1x authentication"

    elif [ "$1" == "xallow_all" ]
    then

        nft delete table bridge tr-mgmt-br0
        echo "802-1x-tr-mgmt Allowed all forwarding again"

    fi

    case ${2:-NOTANEVENT} in

        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 "802-1x-tr-mgmt $1: Denied traffic from $3"
            ;;

        *)
            ;;
    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=/bin/sh -c '/usr/sbin/tc qdisc del dev %i ingress > /dev/null 2>&1'
ExecStartPre=/bin/sh -c '/usr/sbin/tc qdisc del dev %i clsact > /dev/null 2>&1'
ExecStartPre=/usr/sbin/tc qdisc add dev %i clsact
ExecStartPre=/usr/sbin/tc filter add dev %i ingress pref 10000 protocol 0x888e matchall
action ok index 100
ExecStartPre=/usr/sbin/tc filter add dev %i ingress pref 10001 protocol all matchall action
drop index 101
ExecStart=/usr/sbin/hostapd_cli -i %i -a /usr/local/bin/802-1x-tr-mgmt
ExecStopPost=/usr/sbin/tc qdisc del dev %i clsact

[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](#)
 - [Testing EAP-TLS authentication against a FreeRADIUS server or authenticator](#)

Additional resources

- **systemd.service(5)** man page

CHAPTER 18. AUTHENTICATING A RHEL CLIENT TO THE NETWORK 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. The procedures in this section describe different options to configure network authentication.

18.1. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING ETHERNET CONNECTION USING NMCLI

Using the **nmcli** utility, you can configure the client to authenticate itself to the network. This procedure describes how to configure TLS authentication in an existing NetworkManager Ethernet connection profile named **enp1s0** to authenticate to the network.

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. Optionally, 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 `/etc/sysconfig/network-scripts/keys-connection_name` file, that 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 **0x1**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services. In other cases, NetworkManager prompts for the password.

5. Activate the connection profile:

```
# nmcli connection up enp1s0
```

Verification steps

- Access resources on the network that require network authentication.

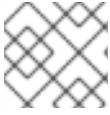
Additional resources

- [Configuring an Ethernet connection](#)
- **nm-settings(5)** man page
- **nmcli(1)** man page

18.2. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING NMSTATECTL

Using the **nmstate** utility, you can create an Ethernet connection that uses the 802.1X standard to authenticate the client. This procedure describes how to add an Ethernet connection for the **enp1s0** interface 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** Extensible Authentication Protocol (EAP)

**NOTE**

The **nmstate** library only supports the **TLS** 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 contents:

```
---
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

```

2. Apply the settings to the system:

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

Verification

- Access resources on the network that require network authentication.

18.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES

Using the **network** RHEL System Role, you can automate the creation of an Ethernet connection that uses the 802.1X standard to authenticate the client. This procedure describes how to remotely add an Ethernet connection for the **enp1s0** interface with the following settings by running an Ansible playbook:

- 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** Extensible Authentication Protocol (EAP)

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.
- The network supports 802.1X network authentication.

- The managed nodes uses NetworkManager.
- 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 Certificate Authority (CA) certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. Create a playbook file, for example **~/enable-802.1x.yml**, with the following content:

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

    - name: Copy client certificate for 802.1X authentication
      copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"

    - 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: "password"
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
```

2. Run the playbook:

```
# ansible-playbook ~/enable-802.1x.yml
```

Additional resources

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

CHAPTER 19. MANAGING THE DEFAULT GATEWAY SETTING

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.

19.1. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMCLI

In most situations, administrators set the default gateway when they create a connection as explained in, for example, [Configuring a static Ethernet connection using nmcli](#).

This section describes how to set or update the default gateway on a previously created connection 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, user must have **root** permissions.

Procedure

1. Set the IP address of the default gateway.

For example, to set the IPv4 address of the default gateway on the **example** connection to **192.0.2.1**:

```
# nmcli connection modify example ipv4.gateway "192.0.2.1"
```

For example, to set the IPv6 address of the default gateway on the **example** connection to **2001:db8:1::1**:

```
# nmcli connection modify example ipv6.gateway "2001:db8:1::1"
```

2. Restart the network connection for changes to take effect. For example, to restart the **example** connection using the command line:

```
# nmcli connection up example
```



WARNING

All connections currently using this network connection are temporarily interrupted during the restart.

3. Optionally, verify that the route is active.
To display the IPv4 default gateway:

ip -4 route

```
default via 192.0.2.1 dev example proto static metric 100
```

To display the IPv6 default gateway:

ip -6 route

```
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- [Configuring a static Ethernet connection using nmcli](#)

19.2. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING THE NMCLI INTERACTIVE MODE

In most situations, administrators set the default gateway when they create a connection as explained in, for example, [Configuring a dynamic Ethernet connection using the nmcli interactive editor](#) .

This section describes how to set or update the default gateway on a previously created connection using the interactive mode of 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. Open the **nmcli** interactive mode for the required connection. For example, to open the **nmcli** interactive mode for the *example* connection:

```
# nmcli connection edit example
```

2. Set the default gateway.

For example, to set the IPv4 address of the default gateway on the *example* connection to **192.0.2.1**:

```
nmcli> set ipv4.gateway 192.0.2.1
```

For example, to set the IPv6 address of the default gateway on the *example* connection to **2001:db8:1::1**:

```
nmcli> set ipv6.gateway 2001:db8:1::1
```

3. Optionally, verify that the default gateway was set correctly:

```
nmcli> print
```

```
...
ipv4.gateway: 192.0.2.1
```

```
...
ipv6.gateway:          2001:db8:1::1
...
```

4. Save the configuration:

```
nmcli> save persistent
```

5. Restart the network connection for changes to take effect:

```
nmcli> activate example
```



WARNING

All connections currently using this network connection are temporarily interrupted during the restart.

6. Leave the **nmcli** interactive mode:

```
nmcli> quit
```

7. Optionally, verify that the route is active.

To display the IPv4 default gateway:

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

To display the IPv6 default gateway:

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- [Configuring a static Ethernet connection using the nmcli interactive editor](#)

19.3. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NM-CONNECTION-EDITOR

In most situations, administrators set the default gateway when they create a connection. This section describes how to set or update the default gateway on a previously created connection using the **nm-connection-editor** application.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
# nm-connection-editor
```

2. Select the connection to modify, and click the gear wheel icon to edit the existing connection.
3. Set the IPv4 default gateway. For example, to set the IPv4 address of the default gateway on the connection to **192.0.2.1**:
 - a. Open the **IPv4 Settings** tab.
 - b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:

Addresses		
Address	Netmask	Gateway
192.0.2.123	24	192.0.2.1

4. Set the IPv6 default gateway. For example, to set the IPv6 address of the default gateway on the connection to **2001:db8:1::1**:
 - a. Open the **IPv6** tab.
 - b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:

Addresses		
Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

5. Click **OK**.
6. Click **Save**.
7. Restart the network connection for changes to take effect. For example, to restart the **example** connection using the command line:

```
# nmcli connection up example
```



WARNING

All connections currently using this network connection are temporarily interrupted during the restart.

8. Optionally, verify that the route is active.
To display the IPv4 default gateway:

ip -4 routedefault via 192.0.2.1 dev *example* proto static metric 100

To display the IPv6 default gateway:

ip -6 routedefault via 2001:db8:1::1 dev *example* proto static metric 100 pref medium**Additional resources**

- [Configuring an Ethernet connection using nm-connection-editor](#)

19.4. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING CONTROL-CENTER

In most situations, administrators set the default gateway when they create a connection. This section describes how to set or update the default gateway on a previously created connection using the **control-center** application.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- The network configuration of the connection is open in the **control-center** application.

Procedure

1. Set the IPv4 default gateway. For example, to set the IPv4 address of the default gateway on the connection to **192.0.2.1**:
 - a. Open the **IPv4** tab.
 - b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:

Addresses		
Address	Netmask	Gateway
192.0.2.123	255.255.255.0	192.0.2.1

2. Set the IPv6 default gateway. For example, to set the IPv6 address of the default gateway on the connection to **2001:db8:1::1**:
 - a. Open the **IPv6** tab.
 - b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:

Addresses		
Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

3. Click **Apply**.

- Back in the **Network** window, disable and re-enable the connection by switching the button for the connection to **Off** and back to **On** for changes to take effect.

**WARNING**

All connections currently using this network connection are temporarily interrupted during the restart.

- Optionally, verify that the route is active.
To display the IPv4 default gateway:

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

To display the IPv6 default gateway:

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- [Configuring an Ethernet connection using control-center](#)

19.5. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMSTATECTL

You can set the default gateway of a network connection using the **nmstatectl** utility. This procedure describes how to set the default gateway of the existing **enp1s0** connection to **192.0.2.1**.

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

- Create a YAML file, for example **~/set-default-gateway.yml**, with the following contents:

```
---
routes:
  config:
```

```
- destination: 0.0.0.0/0
  next-hop-address: 192.0.2.1
  next-hop-interface: enp1s0
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/set-default-gateway.yml
```

Additional resources

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

19.6. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to set the default gateway.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example `~/ethernet-connection.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 default gateway
      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
```

2. Run the playbook:

```
# ansible-playbook ~/ethernet-connection.yml
```

Additional resources

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

19.7. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION WHEN USING THE LEGACY NETWORK SCRIPTS

This procedure describes how to configure a default gateway when you use the legacy network scripts. The example sets the default gateway to **192.0.2.1** that is reachable via the **enp1s0** interface.

Prerequisites

- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.

Procedure

1. Set the **GATEWAY** parameter in the `/etc/sysconfig/network-scripts/ifcfg-enp1s0` file to **192.0.2.1**:

```
GATEWAY=192.0.2.1
```

2. Add the **default** entry in the `/etc/sysconfig/network-scripts/route-enp0s1` file:

```
default via 192.0.2.1
```

3. Restart the network:

```
# systemctl restart network
```

19.8. 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

Connection type	Default metric value
InfiniBand	150
Bond	300
Team	350
VLAN	400
Bridge	425
TUN	450
Wi-Fi	600
IP tunnel	675

Additional resources

- [Configuring policy-based routing to define alternative routes](#)
- [Getting started with Multipath TCP](#)

19.9. 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 the connection as the default route for IPv4 and IPv6 connections:

```
# 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 steps

- 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.

19.10. 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: 192.168.122.1
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 steps

- 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 192.0.2.1 dev enp1s0 proto static metric 101
...
```

- For IPv6, enter:

```
# ip -6 route
default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
...
```

Additional resources

- [Configuring policy-based routing to define alternative routes](#)
- [Getting started with Multipath TCP](#)

CHAPTER 20. CONFIGURING STATIC ROUTES

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.

Section 20.1, “[Example of a network that requires static routes](#)” describes scenarios and how the traffic flows between different networks when you do not configure static routes.

20.1. 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.



In case that the required static routes are not configured the following describes in which situations the communication works and does not work:

- 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 **rp_filter sysctl** setting. 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 can't a server be pinged if net.ipv4.conf.all.rp_filter is set on the server?](#)
- [Enabling IP forwarding](#)

20.2. HOW TO USE THE NMCLI COMMAND 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 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=n**: 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=n**: 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=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=n**: Sets the type of service (TOS) key. Set the value as an integer (0-255).
- **type=value**: Sets the route type. NetworkManager supports the **unicast**, **local**, **blackhole**, **unreachable**, **prohibit**, and **throw** route types. The default is **unicast**.
- **window=n**: Sets the maximal window size for TCP to advertise to these destinations, measured in bytes.

If you use the **ipv4.routes** sub-command, **nmcli** overrides all current settings of this parameter.

To add a route:

```
$ nmcli connection modify connection_name +ipv4.routes "..."
```

Similarly, to remove a specific route:

```
$ nmcli connection modify connection_name -ipv4.routes "..."
```

20.3. CONFIGURING A STATIC ROUTE USING AN NMCLI COMMAND

You can add a static route to an existing NetworkManager connection profile using the **nmcli connection modify** command.

This procedure 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 **example** connection.
- 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 **example** connection.

Prerequisites

- The **example** 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 **example** connection profile:

```
# nmcli connection modify example +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. For example, to add a route to the **198.51.100.0/24** and **203.0.113.0/24** networks, both routed through the **192.0.2.10** gateway, enter:

```
# nmcli connection modify example +ipv4.routes "198.51.100.0/24 192.0.2.10, 203.0.113.0/24 192.0.2.10"
```

2. Add the static IPv6 route to the **example** connection profile:

```
# nmcli connection modify example +ipv6.routes "2001:db8:2::/64 2001:db8:1::10"
```

3. Re-activate the connection:

```
# nmcli connection up example
```

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

- **nmcli(1)** man page
- **nm-settings-nmcli(5)** man page

20.4. CONFIGURING A STATIC ROUTE 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.

This procedure describes how to add 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 deselect checkboxes by using **Space**.

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 20.1. 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>

- Press the **OK** button to return to the **nmtui** main menu.
- Select **Activate a connection** and press **Enter**.
- 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 would 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_name up** command to reactivate the connection in the mentioned scenario.

- Press the **Back** button to return to the main menu.
- 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
```

20.5. CONFIGURING A STATIC ROUTE USING CONTROL-CENTER

You can use **control-center** in GNOME to add a static route to the configuration of a network connection.

This procedure configures the following routes:

- An IPv4 route to the remote **198.51.100.0/24** network. The corresponding gateway has the IP address **192.0.2.10**.
- An IPv6 route to the remote **2001:db8:2::/64** network. The corresponding gateway has the IP address **2001:db8:1::10**.

Prerequisites

- The network is configured.

- This host is in the same IP subnet as the gateways.
- The network configuration of the connection is opened in the **control-center** application. See [Configuring an Ethernet connection using nm-connection-editor](#).

Procedure

1. On the **IPv4** tab:
 - a. Optional: Disable automatic routes by clicking the **On** button in the **Routes** section of the **IPv4** tab to use only static routes. If automatic routes are enabled, Red Hat Enterprise Linux uses static routes and routes received from a DHCP server.
 - b. Enter the address, netmask, gateway, and optionally a metric value of the IPv4 route:

Routes				Automatic <input checked="" type="checkbox"/>
Address	Netmask	Gateway	Metric	
198.51.100.0	24	192.0.2.10		<input type="button" value="x"/>

2. On the **IPv6** tab:
 - a. Optional: Disable automatic routes by clicking the **On** button in the **Routes** section of the **IPv4** tab to use only static routes.
 - b. Enter the address, netmask, gateway, and optionally a metric value of the IPv6 route:

Routes				Automatic <input checked="" type="checkbox"/>
Address	Prefix	Gateway	Metric	
2001:db8:2::	64	2001:db8:1::10		<input type="button" value="x"/>

3. Click **Apply**.
4. Back in the **Network** window, disable and re-enable the connection by switching the button for the connection to **Off** and back to **On** for changes to take effect.



WARNING

Restarting the connection briefly disrupts connectivity on that interface.

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
```

20.6. CONFIGURING A STATIC ROUTE USING NM-CONNECTION-EDITOR

You can use the **nm-connection-editor** application to add a static route to the configuration of a network connection.

This procedure 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 **example** connection.
- 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 **example** connection.

Prerequisites

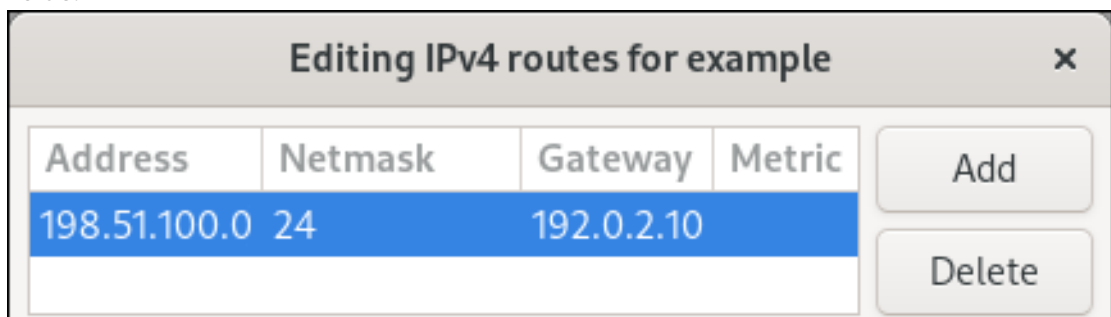
- The network is configured.
- This host is in the same IP subnet as the gateways.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Select the **example** connection profile, and click the gear wheel icon to edit the existing connection.
3. On the **IPv4 Settings** tab:
 - a. Click the **Routes** button.
 - b. Click the **Add** button and enter the address, netmask, gateway, and optionally a metric value.



- c. Click **OK**.
4. On the **IPv6 Settings** tab:
 - a. Click the **Routes** button.

- b. Click the **Add** button and enter the address, netmask, gateway, and optionally a metric value.

Address	Prefix	Gateway	Metric
2001:db8:2::	64	2001:db8:1::10	

Buttons: Add, Delete

- c. Click **OK**.
5. Click **Save**.
6. Restart the network connection for changes to take effect. For example, to restart the **example** connection using the command line:

```
# nmcli connection up example
```

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
```

20.7. CONFIGURING A STATIC ROUTE USING THE NMCLI INTERACTIVE MODE

You can use the interactive mode of the **nmcli** utility to add a static route to the configuration of a network connection.

This procedure 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 **example** connection.
- 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 **example** connection.

Prerequisites

- The **example** connection profile exists and it configures this host to be in the same IP subnet as the gateways.

Procedure

1. Open the **nmcli** interactive mode for the **example** connection:

```
# nmcli connection edit example
```

2. Add the static IPv4 route:

```
nmcli> set ipv4.routes 198.51.100.0/24 192.0.2.10
```

3. Add the static IPv6 route:

```
nmcli> set ipv6.routes 2001:db8:2::/64 2001:db8:1::10
```

4. Optionally, verify that the routes were added correctly to the configuration:

```
nmcli> print
...
ipv4.routes: { ip = 198.51.100.0/24, nh = 192.0.2.10 }
...
ipv6.routes: { ip = 2001:db8:2::/64, nh = 2001:db8:1::10 }
...
```

The **ip** attribute displays the network to route and the **nh** attribute the gateway (next hop).

5. Save the configuration:

```
nmcli> save persistent
```

6. Restart the network connection:

```
nmcli> activate example
```

7. Leave the **nmcli** interactive mode:

```
nmcli> quit
```

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

- **nmcli(1)** man page
- **nm-settings-nmcli(5)** man page

20.8. CONFIGURING A STATIC ROUTE USING NMSTATECTL

You can add a static route to the configuration of a network connection using the **nmstatectl** utility.

This procedure 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 **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.

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 contents:

```
---
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
```

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:

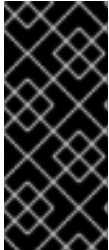
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

Additional resources

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

20.9. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES

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, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

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::fffe**
- 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**

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you to want run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example `~/add-static-routes.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
      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
```

2. Run the playbook:

```
# ansible-playbook ~/add-static-routes.yml
```

Verification steps

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

20.10. CREATING STATIC ROUTES CONFIGURATION FILES IN KEY-VALUE FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS

The legacy network scripts support setting static routes in key-value format.

This procedure configures 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.



NOTE

The legacy network scripts support the key-value format only for static IPv4 routes. For IPv6 routes, use the **ip**-command format. See [Creating static routes configuration files in ip-command format when using the legacy network scripts](#).

Prerequisites

- The gateways for the static route must be directly reachable on the interface.
- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.
- The **network** service is enabled.

Procedure

1. Add the static IPv4 route to the `/etc/sysconfig/network-scripts/route-enp0s1` file:

```
ADDRESS0=198.51.100.0
NETMASK0=255.255.255.0
GATEWAY0=192.0.2.10
```

- The **ADDRESS0** variable defines the network of the first routing entry.
- The **NETMASK0** variable defines the netmask of the first routing entry.
- The **GATEWAY0** variable defines the IP address of the gateway to the remote network or host for the first routing entry.
If you add multiple static routes, increase the number in the variable names. Note that the variables for each route must be numbered sequentially. For example, **ADDRESS0**, **ADDRESS1**, **ADDRESS3**, and so on.

2. Restart the network:

```
# systemctl restart network
```

Verification

- Display the IPv4 routes:

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

Troubleshooting

- Display the journal entries of the **network** unit:

```
# journalctl -u network
```

The following are possible error messages and their causes:

- **Error: Nexthop has invalid gateway:** You specified an IPv4 gateway address in the **route-enp1s0** file that is not in the same subnet as this router.
- **RTNETLINK answers: No route to host:** You specified an IPv6 gateway address in the **route6-enp1s0** file that is not in the same subnet as this router.
- **Error: Invalid prefix for given prefix length:** You specified the remote network in the **route-enp1s0** file by using an IP address within the remote network rather than the network address.

Additional resources

- **/usr/share/doc/network-scripts/sysconfig.txt** file

20.11. CREATING STATIC ROUTES CONFIGURATION FILES IN IP-COMMAND FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS

The legacy network scripts support setting static routes.

This procedure 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 **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.



IMPORTANT

IP addresses of the gateways (next hop) must be in the same IP subnet as the host on which you configure the static routes.

The examples in this procedure use configuration entries in **ip**-command format.

Prerequisites

- The gateways for the static route must be directly reachable on the interface.
- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.

- The **network** service is enabled.

Procedure

1. Add the static IPv4 route to the `/etc/sysconfig/network-scripts/route-enp1s0` file:

```
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

Always specify the network address of the remote network, such as **198.51.100.0**. Setting an IP address within the remote network, such as **198.51.100.1** causes that the network scripts fail to add this route.

2. Add the static IPv6 route to the `/etc/sysconfig/network-scripts/route6-enp1s0` file:

```
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0
```

3. Restart the **network** service:

```
# systemctl restart network
```

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
```

Troubleshooting

- Display the journal entries of the **network** unit:

```
# journalctl -u network
```

The following are possible error messages and their causes:

- **Error: Nexthop has invalid gateway:** You specified an IPv4 gateway address in the `route-enp1s0` file that is not in the same subnet as this router.
- **RTNETLINK answers: No route to host:** You specified an IPv6 gateway address in the `route6-enp1s0` file that is not in the same subnet as this router.
- **Error: Invalid prefix for given prefix length:** You specified the remote network in the `route-enp1s0` file by using an IP address within the remote network rather than the network address.

Additional Resources

- **/usr/share/doc/network-scripts/sysconfig.txt** file

CHAPTER 21. 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.

This section describes of how to configure policy-based routing using NetworkManager.



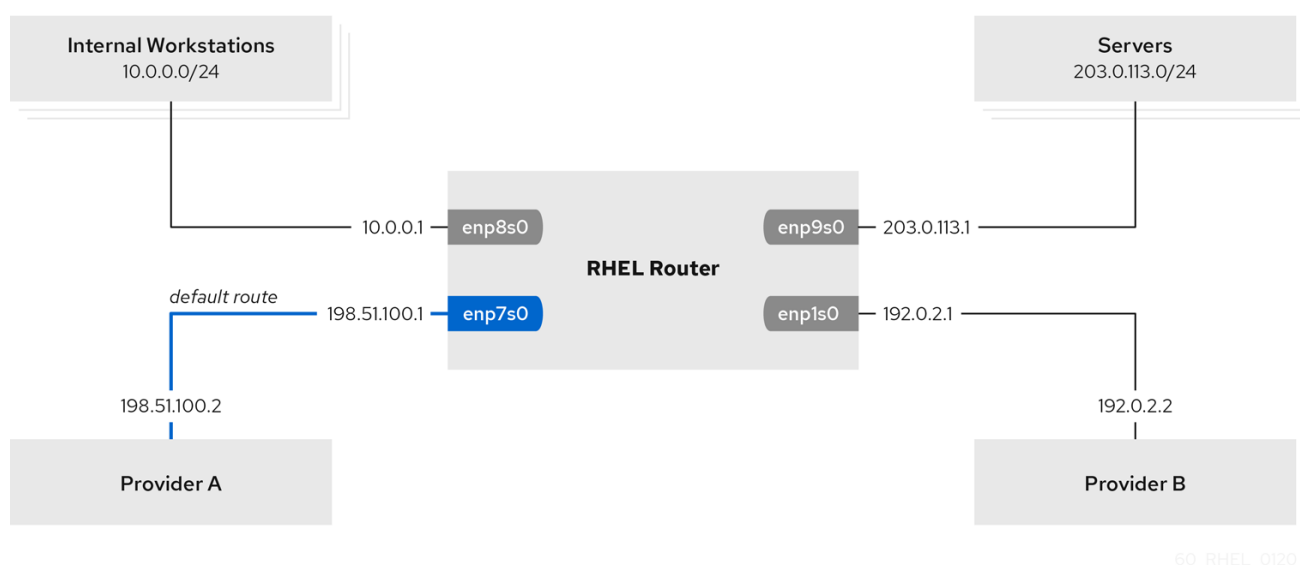
NOTE

On systems that use NetworkManager, only the **nmcli** utility supports setting routing rules and assigning routes to specific tables.

21.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING NETWORKMANAGER

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 following list describes the options of the command:

- **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 steps

1. On a RHEL host in the internal workstation subnet:

- a. Install the **traceroute** package:

```
# yum 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:

```
# yum 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

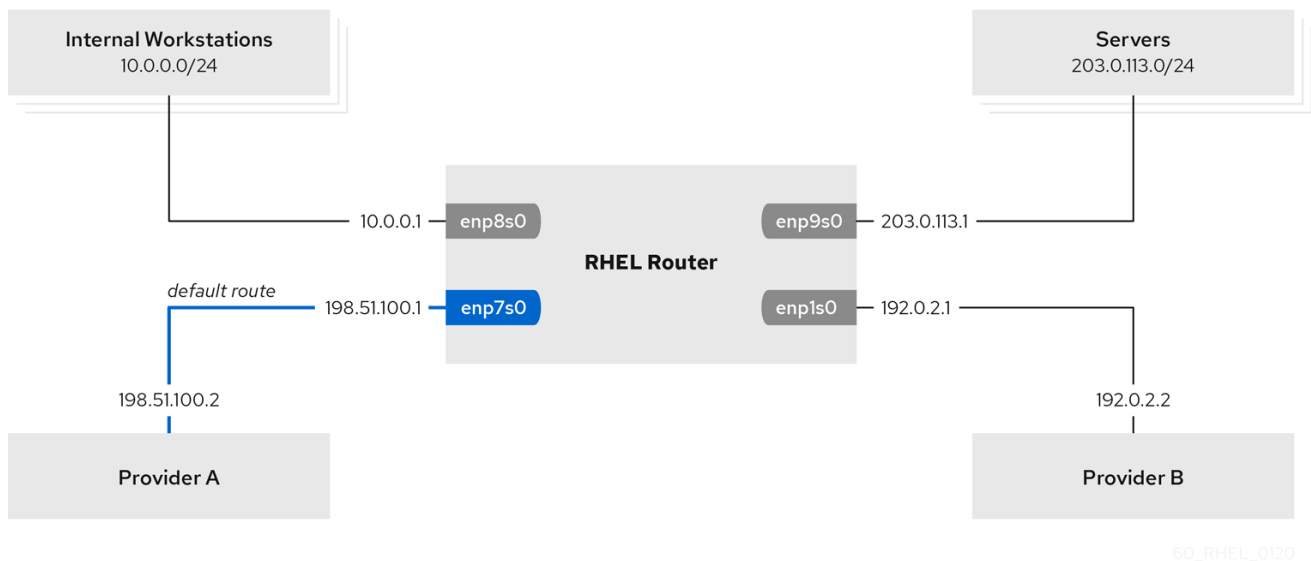
- **nm-settings(5)** man page
- **nmcli(1)** man page
- [Is it possible to set up Policy Based Routing with NetworkManager in RHEL?](#)

21.2. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING RHEL SYSTEM ROLES

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 RHEL **network** System Role. Perform this procedure on the Ansible control node.

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 the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.
- The managed nodes uses the **NetworkManager** and **firewalld** services.
- 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 `~/pbr.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
      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. Run the playbook:

```
# ansible-playbook ~/pbr.yml
```

Verification

1. On a RHEL host in the internal workstation subnet:

- a. Install the **traceroute** package:

```
# yum 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:

```
# yum 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

21.3. OVERVIEW OF CONFIGURATION FILES INVOLVED IN POLICY-BASED ROUTING WHEN USING THE LEGACY NETWORK SCRIPTS

If you use the legacy network scripts instead of NetworkManager to configure your network, you can also configure policy-based routing.



NOTE

Configuring the network using the legacy network scripts provided by the **network-scripts** package is deprecated in RHEL 8. Red Hat recommends that you use NetworkManager to configure policy-based routing. For an example, see [Routing traffic from a specific subnet to a different default gateway using NetworkManager](#).

The following configuration files are involved in policy-based routing when you use the legacy network scripts:

- **/etc/sysconfig/network-scripts/route-interface**: This file defines the IPv4 routes. Use the **table** option to specify the routing table. For example:

```
192.0.2.0/24 via 198.51.100.1 table 1
203.0.113.0/24 via 198.51.100.2 table 2
```

- **/etc/sysconfig/network-scripts/route6-interface**: This file defines the IPv6 routes.
- **/etc/sysconfig/network-scripts/rule-interface**: This file defines the rules for IPv4 source networks for which the kernel routes traffic to specific routing tables. For example:

```
from 192.0.2.0/24 lookup 1
from 203.0.113.0/24 lookup 2
```

- **/etc/sysconfig/network-scripts/rule6-interface**: This file defines the rules for IPv6 source networks for which the kernel routes traffic to specific routing tables.
- **/etc/iproute2/rt_tables**: This file defines the mappings if you want to use names instead of numbers to refer to specific routing tables. For example:

```
1  Provider_A
2  Provider_B
```

Additional resources

- **ip-route(8)** man page
- **ip-rule(8)** man page

21.4. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING THE LEGACY NETWORK SCRIPTS

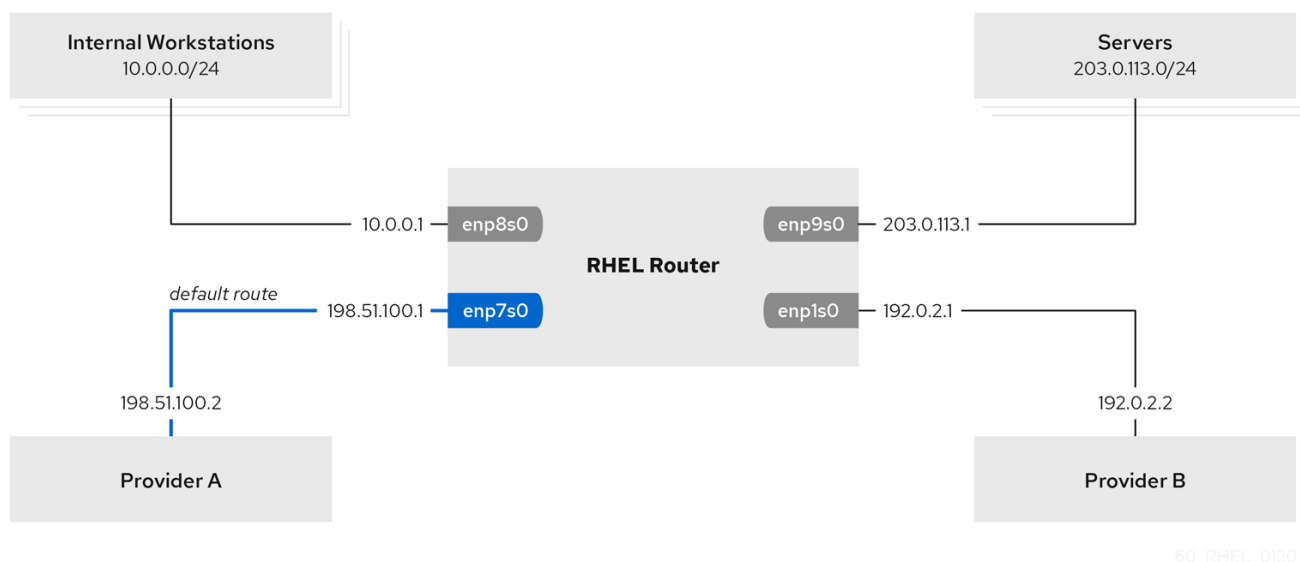
You can use policy-based routing to configure a different default gateway for traffic from certain subnets. For example, you can configure RHEL as a router that, by default, routes all traffic to Internet provider A using the default route. However, traffic received from the internal workstations subnet is routed to provider B.



IMPORTANT

Configuring the network using the legacy network scripts provided by the **network-scripts** package is deprecated in RHEL 8. Follow the procedure in this section only if you use the legacy network scripts instead of NetworkManager on your host. If you use NetworkManager to manage your network settings, see [Routing traffic from a specific subnet to a different default gateway using NetworkManager](#).

The procedure assumes the following network topology:



NOTE

The legacy network scripts process configuration files in alphabetical order. Therefore, you must name the configuration files in a way that ensures that an interface, that is used in rules and routes of other interfaces, are up when a depending interface requires it. To accomplish the correct order, this procedure uses numbers in the **ifcfg-***, **route-***, and **rules-*** files.

Prerequisites

- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.
- 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. Add the configuration for the network interface to provider A by creating the **/etc/sysconfig/network-scripts/ifcfg-1_Provider-A** file with the following content:

```
TYPE=Ethernet
IPADDR=198.51.100.1
PREFIX=30
GATEWAY=198.51.100.2
DNS1=198.51.100.200
DEFROUTE=yes
NAME=1_Provider-A
DEVICE=enp7s0
ONBOOT=yes
ZONE=external
```

The following list describes the parameters used in the configuration file:

- **TYPE=Ethernet**: Defines that the connection type is Ethernet.
 - **IPADDR=IP_address**: Sets the IPv4 address.
 - **PREFIX=subnet_mask**: Sets the subnet mask.
 - **GATEWAY=IP_address**: Sets the default gateway address.
 - **DNS1=IP_of_DNS_server**: Sets the IPv4 address of the DNS server.
 - **DEFROUTE=yes/no**: Defines whether the connection is a default route or not.
 - **NAME=connection_name**: Sets the name of the connection profile. Use a meaningful name to avoid confusion.
 - **DEVICE=network_device**: Sets the network interface.
 - **ONBOOT=yes**: Defines that RHEL starts this connection when the system boots.
 - **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. Add the configuration for the network interface to provider B:
 - a. Create the **/etc/sysconfig/network-scripts/ifcfg-2_Provider-B** file with the following content:

```
TYPE=Ethernet
IPADDR=192.0.2.1
PREFIX=30
```

```

DEFROUTE=no
NAME=2_Provider-B
DEVICE=enp1s0
ONBOOT=yes
ZONE=external

```

Note that the configuration file for this interface does not contain a default gateway setting.

- b. Assign the gateway for the **2_Provider-B** connection to a separate routing table. Therefore, create the **/etc/sysconfig/network-scripts/route-2_Provider-B** file with the following content:

```
0.0.0.0/0 via 192.0.2.2 table 5000
```

This entry assigns the gateway and traffic from all subnets routed through this gateway to table **5000**.

3. Create the configuration for the network interface to the internal workstations subnet:

- a. Create the **/etc/sysconfig/network-scripts/ifcfg-3_Internal-Workstations** file with the following content:

```

TYPE=Ethernet
IPADDR=10.0.0.1
PREFIX=24
DEFROUTE=no
NAME=3_Internal-Workstations
DEVICE=enp8s0
ONBOOT=yes
ZONE=internal

```

- b. Add the routing rule configuration for the internal workstation subnet. Therefore, create the **/etc/sysconfig/network-scripts/rule-3_Internal-Workstations** file with the following content:

```
pri 5 from 10.0.0.0/24 table 5000
```

This configuration defines a routing rule with priority **5** that routes all traffic from the **10.0.0.0/24** subnet to table **5000**. Low values have a high priority.

- c. Create the **/etc/sysconfig/network-scripts/route-3_Internal-Workstations** file with the following content to add a static route to the routing table with ID **5000**:

```
10.0.0.0/24 via 192.0.2.1 table 5000
```

This static route defines that RHEL sends traffic from the **10.0.0.0/24** subnet to the IP of the local network interface to provider B (**192.0.2.1**). This interface is to routing table **5000** and used as the next hop.

4. Add the configuration for the network interface to the server subnet by creating the **/etc/sysconfig/network-scripts/ifcfg-4_Servers** file with the following content:

```

TYPE=Ethernet
IPADDR=203.0.113.1

```

```
PREFIX=24
DEFROUTE=no
NAME=4_Servers
DEVICE=enp9s0
ONBOOT=yes
ZONE=internal
```

5. Restart the network:

```
# systemctl restart network
```

Verification steps

1. On a RHEL host in the internal workstation subnet:

- a. Install the **traceroute** package:

```
# yum 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:

```
# yum 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
```

```
default via 192.0.2.2 dev enp1s0
```

```
10.0.0.0/24 via 192.0.2.1 dev enp1s0
```

3. Display the interfaces and firewall zones:

```
# firewall-cmd --get-active-zones
```

```
external
```

```
  interfaces: enp1s0 enp7s0
```

```
internal
```

```
  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

- [Overview of configuration files involved in policy-based routing when using the legacy network scripts](#)
- **ip-route(8)** man page
- **ip-rule(8)** man page
- **/usr/share/doc/network-scripts/sysconfig.txt** file

CHAPTER 22. CREATING A DUMMY INTERFACE

As a Red Hat Enterprise Linux user, you can create and use dummy network interfaces for debugging and testing purposes. A dummy interface provides a device to route packets without actually transmitting them. It enables you to create additional loopback-like devices managed by NetworkManager and makes an inactive SLIP (Serial Line Internet Protocol) address look like a real address for local programs.

22.1. CREATING A DUMMY INTERFACE WITH BOTH AN IPV4 AND IPV6 ADDRESS USING NMCLI

You can create a dummy interface with various settings. This procedure describes how to create a dummy interface with both an IPv4 and IPv6 address. After creating the dummy interface, NetworkManager automatically assigns it to the default **public** firewall zone.



NOTE

To configure a dummy interface without IPv4 or IPv6 address, set the **ipv4.method** and **ipv6.method** parameters to **disabled**. Otherwise, IP auto-configuration fails, and NetworkManager deactivates the connection and removes the dummy device.

Procedure

1. To create a dummy interface named *dummy0* with static IPv4 and IPv6 addresses, enter:

```
# nmcli connection add type dummy ifname dummy0 ipv4.method manual
  ipv4.addresses 192.0.2.1/24 ipv6.method manual ipv6.addresses 2001:db8:2::1/64
```

2. Optional: To view the dummy interface, enter:

```
# nmcli connection show
NAME          UUID                                  TYPE    DEVICE
enp1s0        db1060e9-c164-476f-b2b5-caec62dc1b05 ethernet ens3
dummy-dummy0  aaf6eb56-73e5-4746-9037-eed42caa8a65 dummy    dummy0
```

Additional resources

- **nm-settings(5)** man page

CHAPTER 23. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE THE NETWORK STATE USING LLDP

Network devices can use the Link Layer Discovery Protocol (LLDP) to advertise their identity, capabilities, and neighbors in a LAN. The **nmstate-autoconf** utility can use this information to automatically configure local network interfaces.



IMPORTANT

The **nmstate-autoconf** utility 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.

23.1. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE NETWORK INTERFACES

The **nmstate-autoconf** utility uses LLDP to identify the VLAN settings of interfaces connected to a switch to configure local devices.

This procedure assumes the following scenario and that the switch broadcasts the VLAN settings using LLDP:

- The **enp1s0** and **enp2s0** interfaces of the RHEL server are connected to switch ports that are configured with VLAN ID **100** and VLAN name **prod-net**.
- The **enp3s0** interface of the RHEL server is connected to a switch port that is configured with VLAN ID **200** and VLAN name **mgmt-net**.

The **nmstate-autoconf** utility then uses this information to create the following interfaces on the server:

- **bond100** - A bond interface with **enp1s0** and **enp2s0** as ports.
- **prod-net** - A VLAN interface on top of **bond100** with VLAN ID **100**.
- **mgmt-net** - A VLAN interface on top of **enp3s0** with VLAN ID **200**

If you connect multiple network interfaces to different switch ports for which LLDP broadcasts the same VLAN ID, **nmstate-autoconf** creates a bond with these interfaces and, additionally, configures the common VLAN ID on top of it.

Prerequisites

- The **nmstate** package is installed.
- LLDP is enabled on the network switch.
- The Ethernet interfaces are up.

Procedure

1. Enable LLDP on the Ethernet interfaces:
 - a. Create a YAML file, for example **~/enable-lldp.yml**, with the following contents:

```
interfaces:
- name: enp1s0
  type: ethernet
  lldp:
    enabled: true
- name: enp2s0
  type: ethernet
  lldp:
    enabled: true
- name: enp3s0
  type: ethernet
  lldp:
    enabled: true
```

- b. Apply the settings to the system:

```
# nmstatectl apply ~/enable-lldp.yml
```

2. Configure the network interfaces using LLDP:
 - a. Optional, start a dry-run to display and verify the YAML configuration that **nmstate-autoconf** generates:

```
# nmstate-autoconf -d enp1s0,enp2s0,enp3s0
---
interfaces:
- name: prod-net
  type: vlan
  state: up
  vlan:
    base-iface: bond100
    id: 100
- name: mgmt-net
  type: vlan
  state: up
  vlan:
    base-iface: enp3s0
    id: 200
- name: bond100
  type: bond
  state: up
  link-aggregation:
    mode: balance-rr
  port:
    - enp1s0
    - enp2s0
```

- b. Use **nmstate-autoconf** to generate the configuration based on information received from LLDP, and apply the settings to the system:

■

```
# nmstate-autoconf enp1s0,enp2s0,enp3s0
```

Next steps

- If there is no DHCP server in your network that provides the IP settings to the interfaces, configure them manual. For details, see:
 - [Configuring an Ethernet connection](#)
 - [Configuring network bonding](#)

Verification

1. Display the settings of the individual interfaces:

```
# nmstatectl show <interface_name>
```

Additional resources

- **nmstate-autoconf(8)** man page

CHAPTER 24. USING LLDP TO DEBUG NETWORK CONFIGURATION PROBLEMS

You can use the Link Layer Discovery Protocol (LLDP) to debug network configuration problems in the topology. This means that, LLDP can report configuration inconsistencies with other hosts or routers and switches.

24.1. DEBUGGING AN INCORRECT VLAN CONFIGURATION USING LLDP INFORMATION

If you configured a switch port to use a certain VLAN and a host does not receive these VLAN packets, you can use the Link Layer Discovery Protocol (LLDP) to debug the problem. Perform this procedure on the host that does not receive the packets.

Prerequisites

- The **nmstate** package is installed.
- The switch supports LLDP.
- LLDP is enabled on neighbor devices.

Procedure

1. Create the **~/enable-LLDP-enp1s0.yml** file with the following content:

```
interfaces:
  - name: enp1s0
    type: ethernet
    lldp:
      enabled: true
```

2. Use the **~/enable-LLDP-enp1s0.yml** file to enable LLDP on interface **enp1s0**:

```
# nmstatectl apply ~/enable-LLDP-enp1s0.yml
```

3. Display the LLDP information:

```
# nmstatectl show enp1s0
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: false
    dhcp: false
  ipv6:
    enabled: false
    autoconf: false
    dhcp: false
  lldp:
    enabled: true
    neighbors:
      - - type: 5
```

```

system-name: Summit300-48
- type: 6
system-description: Summit300-48 - Version 7.4e.1 (Build 5)
05/27/05 04:53:11
- type: 7
system-capabilities:
- MAC Bridge component
- Router
- type: 1
_description: MAC address
chassis-id: 00:01:30:F9:AD:A0
chassis-id-type: 4
- type: 2
_description: Interface name
port-id: 1/1
port-id-type: 5
- type: 127
ieee-802-1-vlans:
- name: v2-0488-03-0505
vid: 488
oui: 00:80:c2
subtype: 3
- type: 127
ieee-802-3-mac-phy-conf:
autoneg: true
operational-mau-type: 16
pmd-autoneg-cap: 27648
oui: 00:12:0f
subtype: 1
- type: 127
ieee-802-1-ppvids:
- 0
oui: 00:80:c2
subtype: 2
- type: 8
management-addresses:
- address: 00:01:30:F9:AD:A0
address-subtype: MAC
interface-number: 1001
interface-number-subtype: 2
- type: 127
ieee-802-3-max-frame-size: 1522
oui: 00:12:0f
subtype: 4
mac-address: 82:75:BE:6F:8C:7A
mtu: 1500

```

4. Verify the output to ensure that the settings match your expected configuration. For example, the LLDP information of the interface connected to the switch shows that the switch port this host is connected to uses VLAN ID **448**:

```

- type: 127
ieee-802-1-vlans:
- name: v2-0488-03-0505
vid: 488

```

If the network configuration of the **enp1s0** interface uses a different VLAN ID, change it accordingly.

Additional resources

[Configuring VLAN tagging](#)

CHAPTER 25. MANUALLY CREATING NETWORKMANAGER PROFILES IN KEYFILE FORMAT

NetworkManager supports profiles stored in the keyfile format. However, by default, if you use NetworkManager utilities, such as **nmcli**, the **network** RHEL System Role, or the **nmstate** API to manage profiles, NetworkManager still uses profiles in the **ifcfg** format.

In the next major RHEL release, the keyfile format will be the default.

25.1. THE KEYFILE FORMAT OF NETWORKMANAGER PROFILES

NetworkManager uses the INI-style keyfile format when it stores connection profiles on disk.

Example of an Ethernet connection profile in keyfile format

```
[connection]
id=example_connection
uuid=82c6272d-1ff7-4d56-9c7c-0eb27c300029
type=ethernet
autoconnect=true

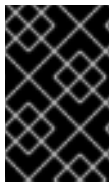
[ipv4]
method=auto

[ipv6]
method=auto

[ethernet]
mac-address=00:53:00:8f:fa:66
```

Each section corresponds to a NetworkManager setting name as described in the **nm-settings(5)** and **nm-settings-keyfile(5)** man pages. 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.



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** and that are only readable and writable by **root**.

Depending on the purpose of the connection profile, save it in one of the following directories:

- **/etc/NetworkManager/system-connections/**: The general location for persistent profiles created by the user that can also be edited. NetworkManager copies them automatically to **/etc/NetworkManager/system-connections/**.
- **/run/NetworkManager/system-connections/**: For temporary profiles that are automatically removed when you reboot the system.

- **/usr/lib/NetworkManager/system-connections/**: For pre-deployed immutable profiles. When you edit such a profile using the NetworkManager API, NetworkManager copies this profile to either the persistent or temporary storage.

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.

25.2. CREATING A NETWORKMANAGER PROFILE IN KEYFILE FORMAT

This section explains a general procedure on how to manually create a NetworkManager connection profile in keyfile format.



NOTE

Manually creating or updating the configuration files can result in an unexpected or non-functional network configuration. Red Hat recommends that you use NetworkManager utilities, such as **nmcli**, the **network** RHEL System Role, or the **nmstate** API to manage NetworkManager connections.

Procedure

1. If you create a profile for a hardware interface, such as Ethernet, display the MAC address of this interface:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:53:00:8f:fa:66 brd ff:ff:ff:ff:ff:ff
```

2. Create a connection profile. For example, for a connection profile of an 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

[ipv4]
method=auto

[ipv6]
method=auto

[ethernet]
mac-address=00:53:00:8f:fa:66
```




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.

3. 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
```

4. Reload the connection profiles:

```
# nmcli connection reload
```

5. 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.

6. Optional: If you set the **autoconnect** variable in the profile to **false**, activate the connection:

```
# nmcli connection up example_connection
```

Verification

1. Display the connection profile:

```
# nmcli connection show example_connection
```

2. Display the IP settings of the interface:

```
# ip address show enp1s0
```

Additional resources

- [nm-settings-keyfile \(5\)](#)

25.3. MIGRATING NETWORKMANAGER PROFILES FROM IFCFG TO KEYFILE FORMAT

You can use the **nmcli connection migrate** command to migrate your existing **ifcfg** connection profiles to the keyfile format. This way, all your connection profiles will be in one location and in the preferred format.

Prerequisites

- You have connection profiles in **ifcfg** format in the **/etc/sysconfig/network-scripts/** directory.

Procedure

- Migrate the connection profiles:

```
# nmcli connection migrate
Connection 'enp1s0' (43ed18ab-f0c4-4934-af3d-2b3333948e45) successfully migrated.
Connection 'enp2s0' (883333e8-1b87-4947-8ceb-1f8812a80a9b) successfully migrated.
...
```

Verification

- Optionally, you can verify that you successfully migrated all your connection profiles:

```
# nmcli -f TYPE,FILENAME,NAME connection
TYPE      FILENAME                                     NAME
ethernet  /etc/NetworkManager/system-connections/enp1s0.nmconnection  enp1s0
ethernet  /etc/NetworkManager/system-connections/enp2s0.nmconnection  enp2s0
...
```

Additional resources

- nm-settings-keyfile(5)**
- nm-settings-ifcfg-rh(5)**
- nmcli(1)**

25.4. USING NMCLI TO CREATE KEYFILE CONNECTION PROFILES IN OFFLINE MODE

Red Hat recommends using 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 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 if:

- 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.

You can create the following connection profile types:

- static Ethernet connection

- dynamic Ethernet connection
- network bond
- network bridge
- VLAN or any kind of supported connections



WARNING

Manually creating or updating the configuration files can result in an unexpected or non-functional network configuration.

Prerequisites

- The **NetworkManager** service is stopped.

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/output.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/output.nmconnection
# chown root:root /etc/NetworkManager/system-connections/output.nmconnection
```

3. Start the **NetworkManager** service:

```
# systemctl start NetworkManager.service
```

4. Optional: 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; 1 min 40s ago
     Docs: man:NetworkManager(8)
    Main PID: 7138 (NetworkManager)
      Tasks: 3 (limit: 22901)
     Memory: 4.4M
    CGroup: /system.slice/NetworkManager.service
            └─7138 /usr/sbin/NetworkManager --no-daemon

Aug 03 13:08:33 example.com NetworkManager[7138]: <info> [1659524913.3600] device
(vlan20): state change: secondaries -> activated (reason 'none', sys-iface-state: 'assume')
Aug 03 13:08:33 example.com NetworkManager[7138]: <info> [1659524913.3607] device
(vlan20): Activation: successful, device activated.
...
```

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/output.nmconnection Example-
Connection
ethernet  /etc/sysconfig/network-scripts/ifcfg-enp1s0      enp1s0
...
```

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-keyfile(5)**
- [The keyfile format of NetworkManager profiles](#)

- [Configuring a static Ethernet connection using nmcli](#)
- [Configuring a dynamic Ethernet connection using nmcli](#)
- [Configuring VLAN tagging using nmcli commands](#)
- [Configuring a network bridge using nmcli commands](#)
- [Configuring a network bond using nmcli commands](#)

CHAPTER 26. SYSTEMD NETWORK TARGETS AND SERVICES

NetworkManager configures the network during the system boot process. However, when booting with a remote root (/), such as if the root directory is stored on an iSCSI device, the network settings are applied in the initial RAM disk (**initrd**) before RHEL is started. For example, if the network configuration is specified on the kernel command line using **rd.neednet=1** or a configuration is specified to mount remote file systems, then the network settings are applied on **initrd**.

This section describes different targets such as **network**, **network-online**, and **NetworkManager-wait-online** service that are used while applying network settings, and how to configure the **systemd** service to start after the **network-online** service is started.

26.1. 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 starts. 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.

26.2. OVERVIEW OF NETWORKMANAGER-WAIT-ONLINE

The synchronous legacy network scripts iterate through all configuration files to set up devices. They apply all network-related configurations and ensure that the network is online.

The **NetworkManager-wait-online** service waits with a timeout for the network to be configured. This network configuration involves plugging-in an Ethernet device, scanning for a Wi-Fi device, and so forth. NetworkManager automatically activates suitable profiles that are configured to start automatically. The failure of the automatic activation process due to a DHCP timeout or similar event might keep NetworkManager busy for an extended period of time. Depending on the configuration, NetworkManager retries activating the same profile or a different profile.

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)** man page

26.3. 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. To open the service file in the editor, enter:
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 27. 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.

This section explains queuing disciplines and describes how to update the default **qdiscs** in RHEL.

27.1. OVERVIEW OF QUEUING DISCIPLINES

Queuing 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)** man page
- **tc-actions(8)** man page

27.2. 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 27.1. Available schedulers in RHEL

qdisc name	Included in	Offload support
Asynchronous Transfer Mode (ATM)	kernel-modules-extra	
Class-Based Queueing	kernel-modules-extra	
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	
Deficit Round Robin (DRR)	kernel-modules-extra	
Differentiated Services marker (DSMARK)	kernel-modules-extra	
Enhanced Transmission Selection (ETS)	kernel-modules-extra	Yes
Fair Queue (FQ)	kernel-core	
Fair Queueing Controlled Delay (FQ_CODEL)	kernel-core	
Generalized Random Early Detection (GRED)	kernel-modules-extra	
Hierarchical Fair Service Curve (HSFC)	kernel-core	
Heavy-Hitter Filter (HHF)	kernel-core	
Hierarchy Token Bucket (HTB)	kernel-core	
INGRESS	kernel-core	Yes
Multi Queue Priority (MQPRIO)	kernel-modules-extra	Yes
Multiqueue (MULTIQ)	kernel-modules-extra	Yes

qdisc name	Included in	Offload support
Network Emulator (NETEM)	kernel-modules-extra	
Proportional Integral-controller Enhanced (PIE)	kernel-core	
PLUG	kernel-core	
Quick Fair Queueing (QFQ)	kernel-modules-extra	
Random Early Detection (RED)	kernel-modules-extra	Yes
Stochastic Fair Blue (SFB)	kernel-modules-extra	
Stochastic Fairness Queueing (SFQ)	kernel-core	
Token Bucket Filter (TBF)	kernel-core	Yes
Trivial Link Equalizer (TEQL)	kernel-modules-extra	



IMPORTANT

The **qdisc** offload requires hardware and driver support on NIC.

Additional resources

- **tc(8)** man page

27.3. INSPECTING QDISCS OF A NETWORK INTERFACE USING THE TC UTILITY

By default, Red Hat Enterprise Linux systems use **fq_codel qdisc**. This procedure describes how to inspect **qdisc** counters.

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

27.4. UPDATING THE DEFAULT QDISC

If you observe networking packet losses with the current **qdisc**, you can change the **qdisc** based on your network-requirements. You can select the **qdisc**, which meets your network requirements. This procedure describes how to change the default **qdisc** in Red Hat Enterprise Linux.

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:
rmmmod NETWORKDRIVERNAME

modprobe NETWORKDRIVERNAME
5. Start the network interface:
ip link set enp0s1 up

Verification steps

- 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
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](#)

27.5. TEMPORARILY SETTING THE CURRENT QDISK OF A NETWORK INTERFACE USING THE TC UTILITY

You can update the current **qdisc** without changing the default one. This procedure describes how to change the current **qdisc** in Red Hat Enterprise Linux.

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 step

- 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
```

27.6. PERMANENTLY SETTING THE CURRENT QDISK OF A NETWORK INTERFACE 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 steps

- 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 1
```

```
qdisc ingress ffff: parent ffff:fff1 -----
```

Additional resources

- **nm-settings(5)** man page

CHAPTER 28. 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.

28.1. 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, we highly recommend you review the *Additional resources*.

Additional resources

- [Understanding Multipath TCP: High availability for endpoints and the networking highway of the future](#)
- [RFC8684: TCP Extensions for Multipath Operation with Multiple Addresses](#)
- [Multipath TCP on Red Hat Enterprise Linux 8.3: From 0 to 1 subflows](#)

28.2. PREPARING RHEL TO ENABLE MPTCP SUPPORT

By default the MPTCP support is disabled in RHEL. Enable MPTCP so that applications that support this feature can use it. Additionally, you have to configure user space applications to force use MPTCP sockets if those applications have TCP sockets by default.

This procedure describes how to use the **sysctl** tool to enable MPTCP support and prepare RHEL for enabling MPTCP for applications system-wide using a **SystemTap** script.

Prerequisites

The following packages are installed:

- **systemtap**
- **iperf3**

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. Verify that MPTCP is enabled in the kernel:

```
# sysctl -a | grep mptcp.enabled
net.mptcp.enabled = 1
```

3. Create a **mptcp-app.stap** file with the following content:

```
#!/usr/bin/env stap

%{
#include <linux/in.h>
#include <linux/ip.h>
%}

/* RSI contains 'type' and RDX contains 'protocol'.
*/

function mptcpify () %{
    if (CONTEXT->kregs->si == SOCK_STREAM &&
        (CONTEXT->kregs->dx == IPPROTO_TCP ||
         CONTEXT->kregs->dx == 0)) {
        CONTEXT->kregs->dx = IPPROTO_MPTCP;
        STAP_RETVALUE = 1;
    } else {
        STAP_RETVALUE = 0;
    }
}%

probe kernel.function("__sys_socket") {
    if (mptcpify() == 1) {
        printf("command %16s mptcpified\n", execname());
    }
}
```

4. Force user space applications to create MPTCP sockets instead of TCP ones:

```
# stap -vg mptcp-app.stap
```

Note: This operation affects all TCP sockets which are started after the command. The applications will continue using TCP sockets after you interrupt the command above with **Ctrl+C**.

5. Alternatively, to allow MPTCP usage to only specific application, you can modify the **mptcp-app.stap** file with the following content:

```
#!/usr/bin/env stap

%{
```

```
#include <linux/in.h>
#include <linux/ip.h>
%}

/* according to [1], RSI contains 'type' and RDX
 * contains 'protocol'.
 * [1] https://github.com/torvalds/linux/blob/master/arch/x86/entry/entry\_64.S#L79
 */

function mptcpify () %{
    if (CONTEXT->kregs->si == SOCK_STREAM &&
        (CONTEXT->kregs->dx == IPPROTO_TCP ||
         CONTEXT->kregs->dx == 0)) {
        CONTEXT->kregs->dx = IPPROTO_MPTCP;
        STAP_RETVALUE = 1;
    } else {
        STAP_RETVALUE = 0;
    }
}%}

probe kernel.function("__sys_socket") {
    cur_proc = execname()
    if ((cur_proc == @1) && (mptcpify() == 1)) {
        printf("command %16s mptcpified\n", cur_proc);
    }
}
```

- In case of alternative choice, assuming, you want to force the **iperf3** tool to use MPTCP instead of TCP. To do so, enter the following command:

```
# stap -vg mptcp-app.stap iperf3
```

- After the **mptcp-app.stap** script installs the kernel probe, the following warnings appear in the kernel **dmesg** output

```
# dmesg
...
[ 1752.694072] Kprobes globally unoptimized
[ 1752.730147] stap_1ade3b3356f3e68765322e26dec00c3d_1476: module_layout: kernel
tainted.
[ 1752.732162] Disabling lock debugging due to kernel taint
[ 1752.733468] stap_1ade3b3356f3e68765322e26dec00c3d_1476: loading out-of-tree
module taints kernel.
[ 1752.737219] stap_1ade3b3356f3e68765322e26dec00c3d_1476: module verification
failed: signature and/or required key missing - tainting kernel
[ 1752.737219] stap_1ade3b3356f3e68765322e26dec00c3d_1476 (mptcp-app.stap):
systemtap: 4.5/0.185, base: ffffffff05500000, memory:
224data/32text/57ctx/65638net/367alloc kb, probes: 1
```

- Start the **iperf3** server:

```
# iperf3 -s
```

```
Server listening on 5201
```


9. Connect the client to the server:

```
# iperf3 -c 127.0.0.1 -t 3
```

10. After the connection is established, verify the **ss** output to see the subflow-specific status:

```
# ss -nti '( dport :5201 )'
```

```
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
ESTAB 0    0    127.0.0.1:41842  127.0.0.1:5201
cubic wscale:7,7 rto:205 rtt:4.455/8.878 ato:40 mss:21888 pmtu:65535 rcvmss:536
advms:65483 cwnd:10 bytes_sent:141 bytes_acked:142 bytes_received:4 segs_out:8
segs_in:7 data_segs_out:3 data_segs_in:3 send 393050505bps lastsnd:2813 lastrcv:2772
lastack:2772 pacing_rate 785946640bps delivery_rate 10944000000bps delivered:4
busy:41ms rcv_space:43690 rcv_ssthresh:43690 minrtt:0.008 tcp-ulp-mptcp flags:Mmec
token:0000(id:0)/2ff053ec(id:0) seq:3e2cbea12d7673d4 sfseq:3 ssnoff:ad3d00f4 maplen:2
```

11. Verify MPTCP counters by using **nstat MPTcp*** command:

```
# nstat MPTcp*

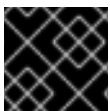
#kernel
MPTcpExtMPCapableSYNRX      2          0.0
MPTcpExtMPCapableSYNTAX     2          0.0
MPTcpExtMPCapableSYNACKRX   2          0.0
MPTcpExtMPCapableACKRX      2          0.0
```

Additional resources

- [How can I download or install debuginfo packages for RHEL systems?](#)
- **tcp(7)** man page
- **mptcpize(8)** man page

28.3. USING IPROUTE2 TO TEMPORARILY CONFIGURE AND ENABLE MULTIPLE PATHS FOR MPTCP APPLICATIONS

Each MPTCP connection uses a single subflow similar to plain TCP. To leverage the MPTCP benefits, specify a higher limit for maximum number of subflows for each MPTCP connection. Then configure additional endpoints to create those subflows.



IMPORTANT

The configuration in this procedure will not persist after rebooting your machine.

Note that MPTCP does not yet support mixed IPv6 and IPv4 endpoints for the same socket. Use endpoints belonging to the same address family.

Prerequisites

- The **iperf3** package is installed

- Server network interface settings:
 - enp4s0: 192.0.2.1/24
 - enp1s0: 198.51.100.1/24
- Client network interface settings:
 - enp4s0f0: 192.0.2.2/24
 - enp4s0f1: 198.51.100.2/24

Procedure

1. Set the per connection additional subflow limits to **1** on the server:

```
# ip mptcp limits set subflow 1
```

Note, that sets a maximum number of *additional* subflows which each connection can have, excluding the initial one.

2. Set the per connection and additional subflow limits to **1** on the client:

```
# ip mptcp limits set subflow 1 add_addr_accepted 1
```

3. Add IP address **198.51.100.1** as a new MPTCP endpoint on the server:

```
# ip mptcp endpoint add 198.51.100.1 dev enp1s0 signal
```

The **signal** option ensures that the **ADD_ADDR** packet is sent after the three-way-handshake.

4. Start the **iperf3** server:

```
# iperf3 -s
```

```
Server listening on 5201
```

5. Connect the client to the server:

```
# iperf3 -c 192.0.2.1 -t 3
```

Verification steps

1. Verify the connection is established:

```
# ss -nti '( sport :5201 )'
```

2. Verify the connection and IP address limit:

```
# ip mptcp limit show
```

3. Verify the newly added endpoint:

```
# ip mptcp endpoint show
```

- 4. Verify MPTCP counters by using the **nstat MPTcp*** command on a server:

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX      2          0.0
MPTcpExtMPCapableACKRX      2          0.0
MPTcpExtMPJoinSynRx         2          0.0
MPTcpExtMPJoinAckRx         2          0.0
MPTcpExtEchoAdd             2          0.0
```

Additional resources

- **ip-mptcp(8)** man page
- **mptcpize(8)** man page

28.4. PERMANENTLY CONFIGURING MULTIPLE PATHS FOR MPTCP APPLICATIONS

You can configure MultiPath TCP (MPTCP) 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**
- **enp4s0f1: 198.51.101.2/24**
- **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. Set the per connection additional subflow limits to **2** on the server:

- a. Create the `/etc/systemd/system/set_mptcp_limit.service` file with the following content:

```
[Unit]
Description=Set MPTCP subflow limit to 2
After=network.target

[Service]
ExecStart=ip mptcp limits set subflows 2
Type=oneshot

[Install]
WantedBy=multi-user.target
```

In RHEL 8, it is not possible to set the subflow limits using the **nmcli** utility. Nor is the subflow limit defined in the RHEL kernel configuration. Instead, use a similar **oneshot** unit that executes the **ip mptcp limits set subflows 2** command after your network (**network.target**) is operational during every boot process.

The **ip mptcp limits set subflows 2** command sets the maximum number of *additional* subflows for each connection, so 3 in total.

- 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 also 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:

```
# ip mptcp limit show
add_addr_accepted 1 subflows 2
```

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
...
```

Additional resources

- **nm-settings-nmcli(5)**
- **ip-mptcp(8)**
- [Section 28.1, “Understanding MPTCP”](#)
- [Understanding Multipath TCP: High availability for endpoints and the networking highway of the future](#)
- [RFC8684: TCP Extensions for Multipath Operation with Multiple Addresses](#)
- [Using Multipath TCP to better survive outages and increase bandwidth](#)

28.5. MONITORING MPTCP SUB-FLOWS

The life cycle of a multipath TCP (MPTCP) socket can be complex: The main MPTCP socket is created, the MPTCP path is validated, one or more sub-flows are created and eventually removed. Finally, the MPTCP socket is terminated.

The MPTCP protocol allows monitoring MPTCP-specific events related to socket and sub-flow creation and deletion, using the **ip** utility provided by the **iproute** package. This utility uses the **netlink** interface to monitor MPTCP events.

This procedure demonstrates how to monitor MPTCP events. For that, it simulates a MPTCP server application, and a client connects to this service. The involved clients in this example use the following interfaces and IP addresses:

- Server: **192.0.2.1**
- Client (Ethernet connection): **192.0.2.2**
- Client (WiFi connection): **192.0.2.3**

To simplify this example, all interfaces are within the same subnet. This is not a requirement. However, it is important that routing has been configured correctly, and the client can reach the server via both interfaces.

Prerequisites

- A RHEL client with two network interfaces, such as a laptop with Ethernet and WiFi
- The client can connect to the server via both interfaces
- A RHEL server
- Both the client and the server run RHEL 8.6 or later

Procedure

1. Set the per connection additional subflow limits to **1** on both client and server:

```
# ip mptcp limits set add_addr_accepted 0 subflows 1
```

2. On the server, to simulate a MPTCP server application, start **netcat (nc)** in listen mode with enforced MPTCP sockets instead of TCP sockets:

```
# nc -l -k -p 12345
```

The **-k** option causes that **nc** does not close the listener after the first accepted connection. This is required to demonstrate the monitoring of sub-flows.

3. On the client:
 - a. Identify the interface with the lowest metric:

```
# ip -4 route
192.0.2.0/24 dev enp1s0 proto kernel scope link src 192.0.2.2 metric 100
192.0.2.0/24 dev wlp1s0 proto kernel scope link src 192.0.2.3 metric 600
```

The **enp1s0** interface has a lower metric than **wlp1s0**. Therefore, RHEL uses **enp1s0** by default.

- b. On the first terminal, start the monitoring:

```
# ip mptcp monitor
```

- c. On the second terminal, start a MPTCP connection to the server:

```
# nc 192.0.2.1 12345
```

RHEL uses the **enp1s0** interface and its associated IP address as a source for this connection.

On the monitoring terminal, the ``ip mptcp monitor`` command now logs:

```
[   CREATED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2 daddr4=192.0.2.1
sport=36444 dport=12345
```

The token identifies the MPTCP socket as a unique ID, and later it enables you to correlate MPTCP events on the same socket.

- d. On the terminal with the running **nc** connection to the server, press **Enter**. This first data packet fully establishes the connection. Note that, as long as no data has been sent, the connection is not established.

On the monitoring terminal, **ip mptcp monitor** now logs:

```
[ ESTABLISHED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2
daddr4=192.0.2.1 sport=36444 dport=12345
```

- e. Optional: Display the connections to port **12345** on the server:

```
# ss -taunp | grep ":12345"
tcp ESTAB 0 0      192.0.2.2:36444 192.0.2.1:12345
```

At this point, only one connection to the server has been established.

- f. On a third terminal, create another endpoint:

```
# ip mptcp endpoint add dev wlp1s0 192.0.2.3 subflow
```

This command sets the name and IP address of the WiFi interface of the client in this command.

On the monitoring terminal, **ip mptcp monitor** now logs:

```
[SF_ESTABLISHED] token=63c070d2 remid=0 locid=2 saddr4=192.0.2.3
daddr4=192.0.2.1 sport=53345 dport=12345 backup=0 ifindex=3
```

The **locid** field displays the local address ID of the new sub-flow and identifies this sub-flow even if the connection uses network address translation (NAT). The **saddr4** field matches the endpoint's IP address from the **ip mptcp endpoint add** command.

- g. Optional: Display the connections to port **12345** on the server:

```
# ss -taunp | grep ":12345"
tcp ESTAB 0 0      192.0.2.2:36444 192.0.2.1:12345
tcp ESTAB 0 0 192.0.2.3%wlp1s0:53345 192.0.2.1:12345
```

The command now displays two connections:

- The connection with source address **192.0.2.2** corresponds to the first MPTCP sub-flow that you established previously.
- The connection from the sub-flow over the **wlp1s0** interface with source address **192.0.2.3**.

- h. On the third terminal, delete the endpoint:

```
# ip mptcp endpoint delete id 2
```

Use the ID from the **locid** field from the **ip mptcp monitor** output, or retrieve the endpoint ID using the **ip mptcp endpoint show** command.

On the monitoring terminal, **ip mptcp monitor** now logs:

```
[ SF_CLOSED] token=63c070d2 remid=0 locid=2 saddr4=192.0.2.3 daddr4=192.0.2.1
sport=53345 dport=12345 backup=0 ifindex=3
```

- i. On the first terminal with the **nc** client, press **Ctrl+C** to terminate the session.
On the monitoring terminal, **ip mptcp monitor** now logs:

```
[ CLOSED] token=63c070d2
```

Additional resources

- **ip-mptcp(1)** man page
- [How NetworkManager manages multiple default gateways](#)

28.6. DISABLING MULTIPATH TCP IN THE KERNEL

This procedure describes how to disable the MPTCP option in the kernel.

Procedure

- Disable the **mptcp.enabled** option.

```
# echo "net.mptcp.enabled=0" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

Verification steps

- Verify whether the **mptcp.enabled** is disabled in the kernel.

```
# sysctl -a | grep mptcp.enabled
net.mptcp.enabled = 0
```


CHAPTER 29. 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**.

29.1. 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** parameter 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
- [Using different DNS servers for different domains](#)

29.2. 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

This section describes how to 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

29.3. SETTING THE DNS PRIORITY OF A NETWORKMANAGER CONNECTION

This section describes how to define the order of DNS servers when NetworkManager 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** for the **Example_con_1** connection:

```
# nmcli connection modify Example_con_1 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 Example_con_1
```

Verification steps

- Display the contents of the **/etc/resolv.conf** file to verify that the DNS server order is correct:

```
# cat /etc/resolv.conf
```

CHAPTER 30. CONFIGURING IP NETWORKING WITH IFCFG FILES

This section describes how to configure a network interface manually by editing the **ifcfg** files.



IMPORTANT

NetworkManager supports profiles stored in the keyfile format. However, by default, NetworkManager uses the **ifcfg** format when you use the NetworkManager API to create or update profiles.

In a future major RHEL release, the keyfile format will be default. Consider using the keyfile format if you want to manually create and manage configuration files. For details, see [Manually creating NetworkManager profiles in keyfile format](#).

Interface configuration (**ifcfg**) files control the software interfaces for individual network devices. As the system boots, it uses these files to determine what interfaces to bring up and how to configure them. These files are usually named **ifcfg-*name***, where the suffix *name* refers to the name of the device that the configuration file controls. By convention, the **ifcfg** file's suffix is the same as the string given by the **DEVICE** directive in the configuration file itself.

30.1. CONFIGURING AN INTERFACE WITH STATIC NETWORK SETTINGS USING IFCFG FILES

This procedure describes how to configure a network interface using **ifcfg** files.

Procedure

- To configure an interface with static network settings using **ifcfg** files, for an interface with the name **enp1s0**, create a file with the name **ifcfg-enp1s0** in the **/etc/sysconfig/network-scripts/** directory that contains:

- For **IPv4** configuration:

```
DEVICE=enp1s0
BOOTPROTO=none
ONBOOT=yes
PREFIX=24
IPADDR=10.0.1.27
GATEWAY=10.0.1.1
```

- For **IPv6** configuration:

```
DEVICE=enp1s0
BOOTPROTO=none
ONBOOT=yes
IPV6INIT=yes
IPV6ADDR=2001:db8:1::2/64
```

Additional resources

- **nm-settings-ifcfg-rh(5)** man page

30.2. CONFIGURING AN INTERFACE WITH DYNAMIC NETWORK SETTINGS USING IFCFG FILES

This procedure describes how to configure a network interface with dynamic network settings using **ifcfg** files.

Procedure

1. To configure an interface named *em1* with dynamic network settings using **ifcfg** files, create a file with the name **ifcfg-em1** in the **/etc/sysconfig/network-scripts/** directory that contains:

```
DEVICE=em1
BOOTPROTO=dhcp
ONBOOT=yes
```

2. To configure an interface to send:

- A different host name to the **DHCP** server, add the following line to the **ifcfg** file:

```
DHCP_HOSTNAME=hostname
```

- A different fully qualified domain name (FQDN) to the **DHCP** server, add the following line to the **ifcfg** file:

```
DHCP_FQDN=fully.qualified.domain.name
```



NOTE

You can use only one of these settings. If you specify both **DHCP_HOSTNAME** and **DHCP_FQDN**, only **DHCP_FQDN** is used.

3. To configure an interface to use particular **DNS** servers, add the following lines to the **ifcfg** file:

```
PEERDNS=no
DNS1=ip-address
DNS2=ip-address
```

where *ip-address* is the address of a **DNS** server. This will cause the network service to update **/etc/resolv.conf** with the specified **DNS** servers specified. Only one **DNS** server address is necessary, the other is optional.

30.3. MANAGING SYSTEM-WIDE AND PRIVATE CONNECTION PROFILES WITH IFCFG FILES

This procedure describes how to configure **ifcfg** files to manage the system-wide and private connection profiles.

Procedure

The permissions correspond to the **USERS** directive in the **ifcfg** files. If the **USERS** directive is not present, the network profile will be available to all users.

- As an example, modify the **ifcfg** file with the following row, which will make the connection available only to the users listed:

```
USERS="joe bob alice"
```

CHAPTER 31. USING NETWORKMANAGER TO DISABLE IPV6 FOR A SPECIFIC CONNECTION

This section describes how to disable the **IPv6** protocol on a system that uses NetworkManager to manage network interfaces. If you disable **IPv6**, NetworkManager automatically sets the corresponding **sysctl** values in the Kernel.



NOTE

If disabling IPv6 using kernel tunables or kernel boot parameters, additional consideration must be given to system configuration. For more information, see the [How do I disable or enable the IPv6 protocol in RHEL?](#) article.

Prerequisites

- The system uses NetworkManager to manage network interfaces, which is the default on Red Hat Enterprise Linux.

31.1. DISABLING IPV6 ON A CONNECTION USING NMCLI

This procedure describes how to disable the **IPv6** protocol using the **nmcli** utility.

Procedure

1. Optionally, display the list of network connections:

```
# nmcli connection show
NAME UUID TYPE DEVICE
Example 7a7e0151-9c18-4e6f-89ee-65bb2d64d365 ethernet enp1s0
...
```

2. Set the **ipv6.method** parameter of the connection to **disabled**:

```
# nmcli connection modify Example ipv6.method "disabled"
```

3. Restart the network connection:

```
# nmcli connection up Example
```

Verification steps

1. Enter the **ip address show** command to display the IP settings of the device:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
link/ether 52:54:00:6b:74:be brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.10.2.255 scope global noprefixroute enp1s0
    valid_lft forever preferred_lft forever
```

If no **inet6** entry is displayed, **IPv6** is disabled on the device.

2. Verify that the `/proc/sys/net/ipv6/conf/enp1s0/disable_ipv6` file now contains the value **1**:

```
# cat /proc/sys/net/ipv6/conf/enp1s0/disable_ipv6
1
```

The value **1** means that **IPv6** is disabled for the device.

CHAPTER 32. MANUALLY CONFIGURING THE /ETC/RESOLV.CONF FILE

By default, NetworkManager on Red Hat Enterprise Linux (RHEL) 8 dynamically updates the **/etc/resolv.conf** file with the DNS settings from active NetworkManager connection profiles. This section describes different options on how to disable this feature to manually configure DNS settings in **/etc/resolv.conf**.

32.1. DISABLING DNS PROCESSING IN THE NETWORKMANAGER CONFIGURATION

This section describes how to disable DNS processing in the NetworkManager configuration to manually configure the **/etc/resolv.conf** file.

Procedure

1. As the root user, create the **/etc/NetworkManager/conf.d/90-dns-none.conf** file with the following content by using a text editor:

```
[main]
dns=none
```

2. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```



NOTE

After you reload the service, NetworkManager no longer updates the **/etc/resolv.conf** file. However, the last contents of the file are preserved.

3. Optionally, remove the **Generated by NetworkManager** comment from **/etc/resolv.conf** to avoid confusion.

Verification steps

1. Edit the **/etc/resolv.conf** file and manually update the configuration.
2. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

3. Display the **/etc/resolv.conf** file:

```
# cat /etc/resolv.conf
```

If you successfully disabled DNS processing, NetworkManager did not override the manually configured settings.

Additional resources

- **NetworkManager.conf(5)** man page

32.2. REPLACING /ETC/RESOLV.CONF WITH A SYMBOLIC LINK TO MANUALLY CONFIGURE DNS SETTINGS

NetworkManager does not automatically update the DNS configuration if **/etc/resolv.conf** is a symbolic link. This section describes how to replace **/etc/resolv.conf** with a symbolic link to an alternative file with the DNS configuration.

Prerequisites

- The **rc-manager** option is not set to **file**. To verify, use the **NetworkManager --print-config** command.

Procedure

1. Create a file, such as **/etc/resolv.conf.manually-configured**, and add the DNS configuration for your environment to it. Use the same parameters and syntax as in the original **/etc/resolv.conf**.
2. Remove the **/etc/resolv.conf** file:

```
# rm /etc/resolv.conf
```

3. Create a symbolic link named **/etc/resolv.conf** that refers to **/etc/resolv.conf.manually-configured**:

```
# ln -s /etc/resolv.conf.manually-configured /etc/resolv.conf
```

Additional resources

- **resolv.conf(5)** man page
- **NetworkManager.conf(5)** man page

CHAPTER 33. MONITORING AND TUNING NIC RING BUFFERS

Receive ring buffers are shared between the device driver and network interface controller (NIC). The card assigns a transmit (TX) and receive (RX) ring buffer. As the name implies, the ring buffer is a circular buffer where an overflow overwrites existing data. There are two ways to move data from the NIC to the kernel, hardware interrupts and software interrupts, also called SoftIRQs.

The kernel uses the RX ring buffer to store incoming packets until they can be processed by the device driver. The device driver drains the RX ring, typically using SoftIRQs, which puts the incoming packets into a kernel data structure called an **sk_buff** or **skb** to begin its journey through the kernel and up to the application which owns the relevant socket.

The kernel uses the TX ring buffer to hold outgoing packets which are destined for the wire.

These ring buffers reside at the bottom of the stack and are a crucial point at which packet drop can occur, which in turn will adversely affect network performance.

33.1. DISPLAYING THE NUMBER OF DROPPED PACKETS

The **ethtool** utility enables administrators to query, configure, or control network driver settings.

The exhaustion of the ring buffers causes an increment in the counters, such as "discard" or "drop" in the output of **ethtool -S interface_name**. The discarded packets indicate that the available buffer is filling up faster than the kernel can process the packets.

Procedure

- Display the drop counters for the **enp1s0** interface:

```
$ ethtool -S enp1s0
```

33.2. INCREASING THE RING BUFFERS TO REDUCE A HIGH PACKET DROP RATE

You can increase the size of an Ethernet device's ring buffers if the packet drop rate causes applications to report:

- a loss of data
- cluster fence
- slow performance
- timeouts
- failed large data transfers, such as backups

Identify the number of dropped packets, and increase the TX and RX ring buffer to reduce a high packet drop rate.

Procedure

1. Display the maximum ring buffer sizes:

ethtool -g enp1s0Ring parameters for *enp1s0*:

Pre-set maximums:

RX: 4096

RX Mini: 0

RX Jumbo: 16320

TX: 4096

Current hardware settings:

RX: 255

RX Mini: 0

RX Jumbo: 0

TX: 255

2. If the values in the **Pre-set maximums** section are higher than in the **Current hardware settings** section increase the ring buffers:

- To increase the RX ring buffer, enter:

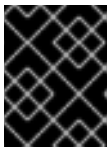
```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

- To increase the TX ring buffer, enter:

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

3. Reload the NetworkManager connection:

```
# nmcli connection up Example-Connection
```

**IMPORTANT**

Depending on the driver your network interface card uses, changing in the ring buffer can shortly interrupt the network connection.

Additional resources

- [ifconfig and ip commands report packet drops](#)
- [Should I be concerned about a 0.05% packet drop rate?](#)
- **ethtool(8)** man page

CHAPTER 34. CONFIGURING 802.3 LINK SETTINGS

34.1. UNDERSTANDING AUTO-NEGOTIATION

Auto-negotiation is a feature of the IEEE 802.3u Fast Ethernet protocol. It targets the device ports to provide an optimal performance of speed, duplex mode, and flow control for information exchange over a link. Using the auto-negotiation protocol, you have optimal performance of data transfer over the Ethernet.



NOTE

To utilize maximum performance of auto-negotiation, use the same configuration on both sides of a link.

34.2. CONFIGURING 802.3 LINK SETTINGS USING THE NMCLI UTILITY

To configure the 802.3 link settings of an Ethernet connection, modify the following configuration parameters:

- **802-3-ethernet.auto-negotiate**
- **802-3-ethernet.speed**
- **802-3-ethernet.duplex**

Procedure

1. Display the current settings of the connection:

```
# nmcli connection show Example-connection
...
802-3-ethernet.speed: 0
802-3-ethernet.duplex: --
802-3-ethernet.auto-negotiate: no
...
```

You can use these values if you need to reset the parameters in case of any problems.

2. Set the speed and duplex link settings:

```
# nmcli connection modify Example-connection 802-3-ethernet.auto-negotiate no 802-3-ethernet.speed 10000 802-3-ethernet.duplex full
```

This command disables auto-negotiation and sets the speed of the connection to **10000** Mbit full duplex.

3. Reactivate the connection:

```
# nmcli connection up Example-connection
```

Verification

- Use the **ethtool** utility to verify the values of Ethernet interface **enp1s0**:

ethtool enp1s0

Settings for enp1s0:

```
...  
Advertised auto-negotiation: No  
...  
Speed: 10000Mb/s  
Duplex: Full  
Auto-negotiation: off  
...  
Link detected: yes
```

Additional resources

- [Network interface speed is 100Mbps and should be 1Gbps](#)
- **nm-settings(5)** man page

CHAPTER 35. CONFIGURING ETHTOOL OFFLOAD FEATURES

Network interface cards can use the TCP offload engine (TOE) to offload processing certain operations to the network controller to improve the network throughput.

This section describes how to set offload features.

35.1. OFFLOAD FEATURES SUPPORTED BY NETWORKMANAGER

You can set the following **ethtool** offload features using NetworkManager:

- **ethtool.feature-esp-hw-offload**
- **ethtool.feature-esp-tx-csum-hw-offload**
- **ethtool.feature-fcoe-mtu**
- **ethtool.feature-gro**
- **ethtool.feature-gso**
- **ethtool.feature-highdma**
- **ethtool.feature-hw-tc-offload**
- **ethtool.feature-l2-fwd-offload**
- **ethtool.feature-loopback**
- **ethtool.feature-lro**
- **ethtool.feature-macsec-hw-offload**
- **ethtool.feature-ntuple**
- **ethtool.feature-rx**
- **ethtool.feature-rx-all**
- **ethtool.feature-rx-fcs**
- **ethtool.feature-rx-gro-hw**
- **ethtool.feature-rx-gro-list**
- **ethtool.feature-rx-udp_tunnel-port-offload**
- **ethtool.feature-rx-udp-gro-forwarding**
- **ethtool.feature-rx-vlan-filter**
- **ethtool.feature-rx-vlan-stag-filter**
- **ethtool.feature-rx-vlan-stag-hw-parse**
- **ethtool.feature-rxhash**

- `ethtool.feature-rxvlan`
- `ethtool.feature-sg`
- `ethtool.feature-tls-hw-record`
- `ethtool.feature-tls-hw-rx-offload`
- `ethtool.feature-tls-hw-tx-offload`
- `ethtool.feature-tso`
- `ethtool.feature-tx`
- `ethtool.feature-tx-checksum-fcoe-crc`
- `ethtool.feature-tx-checksum-ip-generic`
- `ethtool.feature-tx-checksum-ipv4`
- `ethtool.feature-tx-checksum-ipv6`
- `ethtool.feature-tx-checksum-sctp`
- `ethtool.feature-tx-esp-segmentation`
- `ethtool.feature-tx-fcoe-segmentation`
- `ethtool.feature-tx-gre-csum-segmentation`
- `ethtool.feature-tx-gre-segmentation`
- `ethtool.feature-tx-gso-list`
- `ethtool.feature-tx-gso-partial`
- `ethtool.feature-tx-gso-robust`
- `ethtool.feature-tx-ipxip4-segmentation`
- `ethtool.feature-tx-ipxip6-segmentation`
- `ethtool.feature-tx-nocache-copy`
- `ethtool.feature-tx-scatter-gather`
- `ethtool.feature-tx-scatter-gather-fraglist`
- `ethtool.feature-tx-sctp-segmentation`
- `ethtool.feature-tx-tcp-ecn-segmentation`
- `ethtool.feature-tx-tcp-mangleid-segmentation`
- `ethtool.feature-tx-tcp-segmentation`
- `ethtool.feature-tx-tcp6-segmentation`

- **ethtool.feature-tx-tunnel-remcsum-segmentation**
- **ethtool.feature-tx-udp-segmentation**
- **ethtool.feature-tx-udp_tnl-csum-segmentation**
- **ethtool.feature-tx-udp_tnl-segmentation**
- **ethtool.feature-tx-vlan-stag-hw-insert**
- **ethtool.feature-txvlan**

For details about the individual offload features, see the documentation of the **ethtool** utility and the kernel documentation.

35.2. CONFIGURING AN ETHTOOL OFFLOAD FEATURE USING NETWORKMANAGER

This section describes how to enable and disable **ethtool** offload features using NetworkManager, as well as how to remove the setting for a feature from a NetworkManager connection profile.

Procedure

1. For example, to enable the RX offload feature and disable TX offload in the **enp1s0** connection profile, enter:

```
# nmcli con modify enp1s0 ethtool.feature-rx on ethtool.feature-tx off
```

This command explicitly enables RX offload and disables TX offload.

2. To remove the setting of an offload feature that you previously enabled or disabled, set the feature's parameter to **ignore**. For example, to remove the configuration for TX offload, enter:

```
# nmcli con modify enp1s0 ethtool.feature-tx ignore
```

3. Reactivate the network profile:

```
# nmcli connection up enp1s0
```

Verification steps

- Use the **ethtool -k** command to display the current offload features of a network device:

```
# ethtool -k network_device
```

Additional resources

- [Offload features supported by NetworkManager](#)

35.3. USING RHEL SYSTEM ROLES TO SET ETHTOOL FEATURES

You can use the **network** RHEL System Role to configure **ethtool** features of a NetworkManager connection.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static **IPv4** address – **198.51.100.20** with a **/24** subnet mask
- A static **IPv6** address – **2001:db8:1::1** with a **/64** subnet mask
- An **IPv4** default gateway – **198.51.100.254**
- An **IPv6** default gateway – **2001:db8:1::fffe**
- An **IPv4** DNS server – **198.51.100.200**
- An **IPv6** DNS server – **2001:db8:1::ffbb**
- A DNS search domain – **example.com**
- **ethtool** features:
 - Generic receive offload (GRO): disabled
 - Generic segmentation offload (GSO): enabled
 - TX stream control transmission protocol (SCTP) segmentation: disabled

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example **~/configure-ethernet-device-with-ethtool-features.yml**, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool features
```

```

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
      ethtool:
        features:
          gro: "no"
          gso: "yes"
          tx_sctp_segmentation: "no"
      state: up

```

2. Run the playbook:

```
# ansible-playbook ~/configure-ethernet-device-with-ethtool-features.yml
```

Additional resources

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

CHAPTER 36. CONFIGURING ETHTOOL COALESCE SETTINGS

Using interrupt coalescing, the system collects network packets and generates a single interrupt for multiple packets. This increases the amount of data sent to the kernel with one hardware interrupt, which reduces the interrupt load, and maximizes the throughput.

This section provides different options to set the **ethtool** coalesce settings.

36.1. COALESCE SETTINGS SUPPORTED BY NETWORKMANAGER

You can set the following **ethtool** coalesce settings using NetworkManager:

- **coalesce-adaptive-rx**
- **coalesce-adaptive-tx**
- **coalesce-pkt-rate-high**
- **coalesce-pkt-rate-low**
- **coalesce-rx-frames**
- **coalesce-rx-frames-high**
- **coalesce-rx-frames-irq**
- **coalesce-rx-frames-low**
- **coalesce-rx-usecs**
- **coalesce-rx-usecs-high**
- **coalesce-rx-usecs-irq**
- **coalesce-rx-usecs-low**
- **coalesce-sample-interval**
- **coalesce-stats-block-usecs**
- **coalesce-tx-frames**
- **coalesce-tx-frames-high**
- **coalesce-tx-frames-irq**
- **coalesce-tx-frames-low**
- **coalesce-tx-usecs**
- **coalesce-tx-usecs-high**
- **coalesce-tx-usecs-irq**
- **coalesce-tx-usecs-low**

36.2. CONFIGURING ETHTOOL COALESCE SETTINGS USING NETWORKMANAGER

This section describes how to set **ethtool** coalesce settings using NetworkManager, as well as how you remove the setting from a NetworkManager connection profile.

Procedure

1. For example, to set the maximum number of received packets to delay to **128** in the **enp1s0** connection profile, enter:

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames 128
```

2. To remove a coalesce setting, set the setting to **ignore**. For example, to remove the **ethtool.coalesce-rx-frames** setting, enter:

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames ignore
```

3. To reactivate the network profile:

```
# nmcli connection up enp1s0
```

Verification steps

1. Use the **ethtool -c** command to display the current offload features of a network device:

```
# ethtool -c network_device
```

Additional resources

- [Coalesce settings supported by NetworkManager](#)

36.3. USING RHEL SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

You can use the **network** RHEL System Role to configure **ethtool** coalesce settings of a NetworkManager connection.



IMPORTANT

When you run a play that uses the **network** RHEL System Role, the system role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address – **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address – **2001:db8:1::1** with a **/64** subnet mask

- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- **ethtool** coalesce settings:
 - RX frames: **128**
 - TX frames: **128**

Perform this procedure on the Ansible control node.

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 hosts or host groups on which you want to run this playbook are listed in the Ansible inventory file.

Procedure

1. Create a playbook file, for example `~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml`, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool coalesce settings
      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
ethtool:
  coalesce:
    rx_frames: 128
    tx_frames: 128
  state: up
```

2. Run the playbook:

```
# ansible-playbook ~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml
```

Additional resources

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

CHAPTER 37. 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.

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)
- Internet Protocol version 4 / 6 (**IPv4 / IPv6**) and
- any traffic over IP such as TCP or UDP

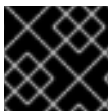
MACsec encrypts and authenticates all traffic in LANs, by default with the GCM-AES-128 algorithm, and uses a pre-shared key to establish the connection between the participant hosts. If you want to change the pre-shared key, you need to update the NM configuration on all hosts in the network that uses MACsec.

A MACsec connection uses an Ethernet device, such as an Ethernet network card, VLAN, or tunnel device, as parent. You can either set an IP configuration only on the MACsec device to communicate with other hosts only 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, there are 2 common methods to configure MACsec;

- host to host and
- host to switch then switch to other host(s)



IMPORTANT

You can use MACsec only between hosts that are in the same (physical or virtual) LAN.

37.1. CONFIGURING A MACSEC CONNECTION USING NMCLI

You can configure Ethernet interfaces to use MACsec using the **nmcli** utility. This procedure describes how to 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 steps

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
```

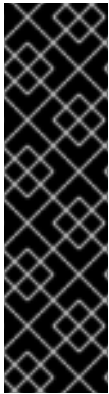
37.2. ADDITIONAL RESOURCES

- [MACsec: a different solution to encrypt network traffic](#) blog.

CHAPTER 38. USING DIFFERENT DNS SERVERS FOR DIFFERENT DOMAINS

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 can configure one DNS server to resolve queries for **example.com** and another DNS server to resolve queries for **example.net**. For all other DNS requests, RHEL uses the DNS server configured in the connection with the default gateway.



IMPORTANT

The **systemd-resolved** service 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.

38.1. SENDING DNS REQUESTS FOR A SPECIFIC DOMAIN TO A SELECTED DNS SERVER

This section configures **systemd-resolved** service and NetworkManager to send DNS queries for a specific domain to a selected DNS server.

If you complete the procedure in this section, RHEL uses the DNS service provided by **systemd-resolved** in the **/etc/resolv.conf** file. The **systemd-resolved** service starts a DNS service that listens on port **53** IP address **127.0.0.53**. The service dynamically routes DNS requests to the corresponding DNS servers specified in NetworkManager.



NOTE

The **127.0.0.53** address is only reachable from the local system and not from the network.

Prerequisites

- The system has multiple NetworkManager connections configured.
- A DNS server and search domain are configured in the NetworkManager connections that are responsible for resolving a specific domain
For example, if the DNS server specified in a VPN connection should resolve queries for the **example.com** domain, the VPN connection profile must have:
 - Configured a DNS server that can resolve **example.com**
 - Configured the search domain to **example.com** in the **ipv4.dns-search** and **ipv6.dns-search** parameters

Procedure

1. Start and enable the **systemd-resolved** service:

```
# systemctl --now enable systemd-resolved
```

2. Edit the **/etc/NetworkManager/NetworkManager.conf** file, and set the following entry in the **[main]** section:

```
dns=systemd-resolved
```

3. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

Verification steps

1. Verify that the **nameserver** entry in the **/etc/resolv.conf** file refers to **127.0.0.53**:

```
# cat /etc/resolv.conf
nameserver 127.0.0.53
```

2. Verify that the **systemd-resolved** service listens on port **53** on the local IP address **127.0.0.53**:

```
# ss -tulpn | grep "127.0.0.53"
udp UNCONN 0 0 127.0.0.53%lo:53 0.0.0.0:* users:(("systemd-
resolve",pid=1050,fd=12))
tcp LISTEN 0 4096 127.0.0.53%lo:53 0.0.0.0:* users:(("systemd-
resolve",pid=1050,fd=13))
```

Additional resources

- **NetworkManager.conf(5)** man page

CHAPTER 39. GETTING STARTED WITH IPVLAN

IPVLAN is a driver for a virtual network device that can be used in container environment to access the host network. IPVLAN exposes a single MAC address to the external network regardless the number of IPVLAN device created inside the host network. This means that a user can have multiple IPVLAN devices in multiple containers and the corresponding switch reads a single MAC address. IPVLAN driver is useful when the local switch imposes constraints on the total number of MAC addresses that it can manage.

39.1. IPVLAN MODES

The following modes are available for IPVLAN:

- L2 mode**
 In IPVLAN **L2 mode**, virtual devices receive and respond to address resolution protocol (ARP) requests. The **netfilter** framework runs only inside the container that owns the virtual device. No **netfilter** chains are executed in the default namespace on the containerized traffic. Using **L2 mode** provides good performance, but less control on the network traffic.
- L3 mode**
 In **L3 mode**, virtual devices process only **L3** traffic and above. Virtual devices do not respond to ARP request and users must configure the neighbour entries for the IPVLAN IP addresses on the relevant peers manually. The egress traffic of a relevant container is landed on the **netfilter** POSTROUTING and OUTPUT chains in the default namespace while the ingress traffic is threaded in the same way as **L2 mode**. Using **L3 mode** provides good control but decreases the network traffic performance.
- L3S mode**
 In **L3S mode**, virtual devices process the same way as in **L3 mode**, except that both egress and ingress traffics of a relevant container are landed on **netfilter** chain in the default namespace. **L3S mode** behaves in a similar way to **L3 mode** but provides greater control of the network.



NOTE

The IPVLAN virtual device does not receive broadcast and multicast traffic in case of **L3** and **L3S** modes.

39.2. COMPARISON OF IPVLAN AND MACVLAN

The following table shows the major differences between MACVLAN and IPVLAN.

MACVLAN	IPVLAN
Uses MAC address for each MACVLAN device. The overlimit of MAC addresses of MAC table in switch might cause losing the connectivity.	Uses single MAC address which does not limit the number of IPVLAN devices.
Netfilter rules for global namespace cannot affect traffic to or from MACVLAN device in a child namespace.	It is possible to control traffic to or from IPVLAN device in L3 mode and L3S mode .

Note that both IPVLAN and MACVLAN do not require any level of encapsulation.

39.3. CREATING AND CONFIGURING THE IPVLAN DEVICE USING IPROUTE2

This procedure shows how to set up the IPVLAN device using **iproute2**.

Procedure

1. To create an IPVLAN device, enter the following command:

```
# ip link add link real_NIC_device name IPVLAN_device type ipvlan mode I2
```

Note that network interface controller (NIC) is a hardware component which connects a computer to a network.

Example 39.1. Creating an IPVLAN device

```
# ip link add link enp0s31f6 name my_ipvlan type ipvlan mode I2
# ip link
47: my_ipvlan@enp0s31f6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state
DOWN mode DEFAULT group default qlen 1000 link/ether e8:6a:6e:8a:a2:44 brd
ff:ff:ff:ff:ff:ff
```

2. To assign an **IPv4** or **IPv6** address to the interface, enter the following command:

```
# ip addr add dev IPVLAN_device IP_address/subnet_mask_prefix
```

3. In case of configuring an IPVLAN device in **L3 mode** or **L3S mode**, make the following setups:
 - a. Configure the neighbor setup for the remote peer on the remote host:

```
# ip neigh add dev peer_device IPVLAN_device_IP_address lladdr MAC_address
```

where *MAC_address* is the MAC address of the real NIC on which an IPVLAN device is based on.

- b. Configure an IPVLAN device for **L3 mode** with the following command:

```
# ip route add dev <real_NIC_device> <peer_IP_address/32>
```

For **L3S mode**:

```
# ip route add dev real_NIC_device peer_IP_address/32
```

where IP-address represents the address of the remote peer.

4. To set an IPVLAN device active, enter the following command:

```
# ip link set dev IPVLAN_device up
```

5. To check if the IPVLAN device is active, execute the following command on the remote host:

ping *IP_address*

where the *IP_address* uses the IP address of the IPVLAN device.

CHAPTER 40. REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

With Virtual routing and forwarding (VRF), administrators can use multiple routing tables simultaneously on the same host. For that, VRF partitions a network at layer 3. This enables the administrator to isolate traffic using separate and independent route tables per VRF domain. This technique is similar to virtual LANs (VLAN), which partitions a network at layer 2, where the operating system uses different VLAN tags to isolate traffic sharing the same physical medium.

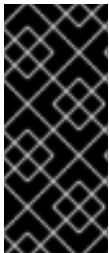
One benefit of VRF over partitioning on layer 2 is that routing scales better considering the number of peers involved.

Red Hat Enterprise Linux uses a virtual **vrt** device for each VRF domain and adds routes to a VRF domain by adding existing network devices to a VRF device. Addresses and routes previously attached to the original device will be moved inside the VRF domain.

Note that each VRF domain is isolated from each other.

40.1. PERMANENTLY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

This procedure describes how to permanently use the same IP address on different interfaces in one server by using the VRF feature.



IMPORTANT

To enable remote peers to contact both VRF interfaces while reusing the same IP address, the network interfaces must belong to different broadcasting domains. A broadcast domain in a network is a set of nodes, which receive broadcast traffic sent by any of them. In most configurations, all nodes connected to the same switch belong to the same broadcasting domain.

Prerequisites

- You are logged in as the **root** user.
- The network interfaces are not configured.

Procedure

1. Create and configure the first VRF device:
 - a. Create a connection for the VRF device and assign it to a routing table. For example, to create a VRF device named **vrf0** that is assigned to the **1001** routing table:

```
# nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1001 ipv4.method disabled ipv6.method disabled
```

- b. Enable the **vrf0** device:

```
# nmcli connection up vrf0
```


- c. Assign a network device to the VRF just created. For example, to add the **enp1s0** Ethernet device to the **vrf0** VRF device and assign an IP address and the subnet mask to **enp1s0**, enter:

```
# nmcli connection add type ethernet con-name vrf.enp1s0 ifname enp1s0 master
vrf0 ipv4.method manual ipv4.address 192.0.2.1/24
```

- d. Activate the **vrf.enp1s0** connection:

```
# nmcli connection up vrf.enp1s0
```

2. Create and configure the next VRF device:

- a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **vrf1** that is assigned to the **1002** routing table, enter:

```
# nmcli connection add type vrf ifname vrf1 con-name vrf1 table 1002 ipv4.method
disabled ipv6.method disabled
```

- b. Activate the **vrf1** device:

```
# nmcli connection up vrf1
```

- c. Assign a network device to the VRF just created. For example, to add the **enp7s0** Ethernet device to the **vrf1** VRF device and assign an IP address and the subnet mask to **enp7s0**, enter:

```
# nmcli connection add type ethernet con-name vrf.enp7s0 ifname enp7s0 master
vrf1 ipv4.method manual ipv4.address 192.0.2.1/24
```

- d. Activate the **vrf.enp7s0** device:

```
# nmcli connection up vrf.enp7s0
```

40.2. TEMPORARILY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

The procedure in this section describes how to temporarily use the same IP address on different interfaces in one server by using the virtual routing and forwarding (VRF) feature. Use this procedure only for testing purposes, because the configuration is temporary and lost after you reboot the system.



IMPORTANT

To enable remote peers to contact both VRF interfaces while reusing the same IP address, the network interfaces must belong to different broadcasting domains. A broadcast domain in a network is a set of nodes which receive broadcast traffic sent by any of them. In most configurations, all nodes connected to the same switch belong to the same broadcasting domain.

Prerequisites

- You are logged in as the **root** user.

- The network interfaces are not configured.

Procedure

1. Create and configure the first VRF device:

- a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **blue** that is assigned to the **1001** routing table:

```
# ip link add dev blue type vrf table 1001
```

- b. Enable the **blue** device:

```
# ip link set dev blue up
```

- c. Assign a network device to the VRF device. For example, to add the **enp1s0** Ethernet device to the **blue** VRF device:

```
# ip link set dev enp1s0 master blue
```

- d. Enable the **enp1s0** device:

```
# ip link set dev enp1s0 up
```

- e. Assign an IP address and subnet mask to the **enp1s0** device. For example, to set it to **192.0.2.1/24**:

```
# ip addr add dev enp1s0 192.0.2.1/24
```

2. Create and configure the next VRF device:

- a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **red** that is assigned to the **1002** routing table:

```
# ip link add dev red type vrf table 1002
```

- b. Enable the **red** device:

```
# ip link set dev red up
```

- c. Assign a network device to the VRF device. For example, to add the **enp7s0** Ethernet device to the **red** VRF device:

```
# ip link set dev enp7s0 master red
```

- d. Enable the **enp7s0** device:

```
# ip link set dev enp7s0 up
```

- e. Assign the same IP address and subnet mask to the **enp7s0** device as you used for **enp1s0** in the **blue** VRF domain:

```
# ip addr add dev enp7s0 192.0.2.1/24
```

3. Optionally, create further VRF devices as described above.

40.3. ADDITIONAL RESOURCES

- `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vrf.txt` from the `kernel-doc` package

CHAPTER 41. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK

With virtual routing and forwarding (VRF), you can create isolated networks with a routing table that is different to the main routing table of the operating system. You can then start services and applications so that they have only access to the network defined in that routing table.

41.1. CONFIGURING A VRF DEVICE

To use virtual routing and forwarding (VRF), you create a VRF device and attach a physical or virtual network interface and routing information to it.



WARNING

To prevent that you lock out yourself out remotely, perform this procedure on the local console or remotely over a network interface that you do not want to assign to the VRF device.

Prerequisites

- You are logged in locally or using a network interface that is different to the one you want to assign to the VRF device.

Procedure

- Create the **vrf0** connection with a same-named virtual device, and attach it to routing table **1000**:

```
# nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1000 ipv4.method disabled ipv6.method disabled
```

- Add the **enp1s0** device to the **vrf0** connection, and configure the IP settings:

```
# nmcli connection add type ethernet con-name enp1s0 ifname enp1s0 master vrf0 ipv4.method manual ipv4.address 192.0.2.1/24 ipv4.gateway 192.0.2.254
```

This command creates the **enp1s0** connection as a port of the **vrf0** connection. Due to this configuration, the routing information are automatically assigned to the routing table **1000** that is associated with the **vrf0** device.

- If you require static routes in the isolated network:

- Add the static routes:

```
# nmcli connection modify enp1s0 +ipv4.routes "198.51.100.0/24 192.0.2.2"
```

This adds a route to the **198.51.100.0/24** network that uses **192.0.2.2** as the router.

- Activate the connection:

```
# nmcli connection up enp1s0
```

Verification

1. Display the IP settings of the device that is associated with **vrf0**:

```
# ip -br addr show vrf vrf0
enp1s0  UP   192.0.2.15/24
```

2. Display the VRF devices and their associated routing table:

```
# ip vrf show
Name          Table
-----
vrf0         1000
```

3. Display the main routing table:

```
# ip route show
default via 192.168.0.1 dev enp1s0 proto static metric 100
```

4. Display the routing table **1000**:

```
# ip route show table 1000
default via 192.0.2.254 dev enp1s0 proto static metric 101
broadcast 192.0.2.0 dev enp1s0 proto kernel scope link src 192.0.2.1
192.0.2.0/24 dev enp1s0 proto kernel scope link src 192.0.2.1 metric 101
local 192.0.2.1 dev enp1s0 proto kernel scope host src 192.0.2.1
broadcast 192.0.2.255 dev enp1s0 proto kernel scope link src 192.0.2.1
198.51.100.0/24 via 192.0.2.2 dev enp1s0 proto static metric 101
```

The **default** entry indicates that services that use this routing table, use **192.0.2.254** as their default gateway and not the default gateway in the main routing table.

5. Execute the **traceroute** utility in the network associated with **vrf0** to verify that the utility uses the route from table **1000**:

```
# ip vrf exec vrf0 traceroute 203.0.113.1
traceroute to 203.0.113.1 (203.0.113.1), 30 hops max, 60 byte packets
1 192.0.2.254 (192.0.2.254) 0.516 ms 0.459 ms 0.430 ms
...
```

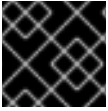
The first hop is the default gateway that is assigned to the routing table **1000** and not the default gateway from the system's main routing table.

Additional resources

- **ip-vrf(8)** man page

41.2. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK

You can configure a service, such as the Apache HTTP Server, to start within an isolated virtual routing and forwarding (VRF) network.



IMPORTANT

Services can only bind to local IP addresses that are in the same VRF network.

Prerequisites

- You configured the **vrf0** device.
- You configured Apache HTTP Server to listen only on the IP address that is assigned to the interface associated with the **vrf0** device.

Procedure

1. Display the content of the **httpd** systemd service:

```
# systemctl cat httpd
...
[Service]
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
...
```

You require the content of the **ExecStart** parameter in a later step to run the same command within the isolated VRF network.

2. Create the **/etc/systemd/system/httpd.service.d/** directory:

```
# mkdir /etc/systemd/system/httpd.service.d/
```

3. Create the **/etc/systemd/system/httpd.service.d/override.conf** file with the following content:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/ip vrf exec vrf0 /usr/sbin/httpd $OPTIONS -DFOREGROUND
```

To override the **ExecStart** parameter, you first need to unset it and then set it to the new value as shown.

4. Reload systemd.

```
# systemctl daemon-reload
```

5. Restart the **httpd** service.

```
# systemctl restart httpd
```

Verification

1. Display the process IDs (PID) of **httpd** processes:

```
# pidof -c httpd
1904 ...
```

2. Display the VRF association for the PIDs, for example:

-

```
# ip vrf identify 1904
vrf0
```

3. Display all PIDs associated with the **vrf0** device:

```
# ip vrf pids vrf0
1904 httpd
...
```

Additional resources

- **ip-vrf(8)** man page

CHAPTER 42. RUNNING DHCLIENT EXIT HOOKS USING NETWORKMANAGER A DISPATCHER SCRIPT

You can use a NetworkManager dispatcher script to execute **dhclient** exit hooks.

42.1. THE CONCEPT OF NETWORKMANAGER DISPATCHER SCRIPTS

The **NetworkManager-dispatcher** service executes user-provided scripts in alphabetical order when network events happen. These scripts are typically shell scripts, but can be any executable script or application. You can use dispatcher scripts, for example, to adjust network-related settings that you cannot manage with NetworkManager.

You can store dispatcher scripts in the following directories:

- **/etc/NetworkManager/dispatcher.d/**: The general location for dispatcher scripts the **root** user can edit.
- **/usr/lib/NetworkManager/dispatcher.d/**: For pre-deployed immutable dispatcher scripts.

For security reasons, the **NetworkManager-dispatcher** service executes scripts only if the following conditions met:

- The script is owned by the **root** user.
- The script is only readable and writable by **root**.
- The **setuid** bit is not set on the script.

The **NetworkManager-dispatcher** service runs each script with two arguments:

1. The interface name of the device the operation happened on.
2. The action, such as **up**, when the interface has been activated.

The **Dispatcher scripts** section in the **NetworkManager(8)** man page provides an overview of actions and environment variables you can use in scripts.

The **NetworkManager-dispatcher** service runs one script at a time, but asynchronously from the main NetworkManager process. Note that, if a script is queued, the service will always run it, even if a later event makes it obsolete. However, the **NetworkManager-dispatcher** service runs scripts that are symbolic links referring to files in **/etc/NetworkManager/dispatcher.d/no-wait.d/** immediately, without waiting for the termination of previous scripts, and in parallel.

Additional resources

- **NetworkManager(8)** man page

42.2. CREATING A NETWORKMANAGER DISPATCHER SCRIPT THAT RUNS DHCLIENT EXIT HOOKS

This section explains how to write a NetworkManager dispatcher script that runs **dhclient** exit hooks stored in the **/etc/dhcp/dhclient-exit-hooks.d/** directory when an IPv4 address is assigned or updated from a DHCP server.

Prerequisites

Prerequisites

- The **dhclient** exit hooks are stored in the `/etc/dhcp/dhclient-exit-hooks.d/` directory.

Procedure

1. Create the `/etc/NetworkManager/dispatcher.d/12-dhclient-down` file with the following content:

```
#!/bin/bash
# Run dhclient.exit-hooks.d scripts

if [ -n "$DHCP4_DHCP_LEASE_TIME" ] ; then
  if [ "$2" = "dhcp4-change" ] || [ "$2" = "up" ] ; then
    if [ -d /etc/dhcp/dhclient-exit-hooks.d ] ; then
      for f in /etc/dhcp/dhclient-exit-hooks.d/*.sh ; do
        if [ -x "${f}" ] ; then
          . "${f}"
        fi
      done
    fi
  fi
fi
```

2. Set the **root** user as owner of the file:

```
# chown root:root /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

3. Set the permissions so that only the root user can execute it:

```
# chmod 0700 /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

4. Restore the SELinux context:

```
# restorecon /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

Additional resources

- **NetworkManager(8)** man page

CHAPTER 43. INTRODUCTION TO NETWORKMANAGER DEBUGGING

Increasing the log levels for all or certain domains helps to log more details of the operations NetworkManager performs. Administrators can use this information to troubleshoot problems. NetworkManager provides different levels and domains to produce logging information. The `/etc/NetworkManager/NetworkManager.conf` file is the main configuration file for NetworkManager. The logs are stored in the journal.

This section provides information on enabling debug logging for NetworkManager and using different logging levels and domains to configure the amount of logging details.

43.1. DEBUGGING LEVELS AND DOMAINS

You can use the **levels** and **domains** parameters to manage the debugging for NetworkManager. The level defines the verbosity level, whereas the domains define the category of the messages to record the logs with given severity (**level**).

Log levels	Description
OFF	Does not log any messages about NetworkManager
ERR	Logs only critical errors
WARN	Logs warnings that can reflect the operation
INFO	Logs various informational messages that are useful for tracking state and operations
DEBUG	Enables verbose logging for debugging purposes
TRACE	Enables more verbose logging than the DEBUG level

Note that subsequent levels log all messages from earlier levels. For example, setting the log level to **INFO** also logs messages contained in the **ERR** and **WARN** log level.

Additional resources

- **NetworkManager.conf(5)** man page

43.2. SETTING THE NETWORKMANAGER LOG LEVEL

By default, all the log domains are set to record the **INFO** log level. Disable rate-limiting before collecting debug logs. With rate-limiting, **systemd-journald** drops messages if there are too many of them in a short time. This can occur when the log level is **TRACE**.

This procedure disables rate-limiting and enables recording debug logs for the all (ALL) domains.

Procedure

1. To disable rate-limiting, edit the `/etc/systemd/journald.conf` file, uncomment the **RateLimitBurst** parameter in the **[Journal]** section, and set its value as **0**:

```
RateLimitBurst=0
```

2. Restart the **systemd-journald** service.

```
# systemctl restart systemd-journald
```

3. Create the `/etc/NetworkManager/conf.d/95-nm-debug.conf` file with the following content:

```
[logging]
domains=ALL:TRACE
```

The **domains** parameter can contain multiple comma-separated **domain:level** pairs.

4. Restart the NetworkManager service.

```
# systemctl restart NetworkManager
```

Verification

- Query the **systemd** journal to display the journal entries of the **NetworkManager** unit:

```
# journalctl -u NetworkManager
...
Jun 30 15:24:32 server NetworkManager[164187]: <debug> [1656595472.4939] active-
connection[0x5565143c80a0]: update activation type from assume to managed
Jun 30 15:24:32 server NetworkManager[164187]: <trace> [1656595472.4939]
device[55b33c3bdb72840c] (enp1s0): sys-iface-state: assume -> managed
Jun 30 15:24:32 server NetworkManager[164187]: <trace> [1656595472.4939]
l3cfg[4281fdf43e356454,ifindex=3]: commit type register (type "update", source "device",
existing a369f23014b9ede3) -> a369f23014b9ede3
Jun 30 15:24:32 server NetworkManager[164187]: <info> [1656595472.4940] manager:
NetworkManager state is now CONNECTED_SITE
...
```

43.3. TEMPORARILY SETTING LOG LEVELS AT RUN TIME USING NMCLI

You can change the log level at run time using **nmcli**. However, Red Hat recommends to enable debugging using configuration files and restart NetworkManager. Updating debugging **levels** and **domains** using the **.conf** file helps to debug boot issues and captures all the logs from the initial state.

Procedure

1. Optional: Display the current logging settings:

```
# nmcli general logging
LEVEL DOMAINS
INFO
PLATFORM,RFKILL,ETHER,WIFI,BT,MB,DHCP4,DHCP6,PPP,WIFI_SCAN,IP4,IP6,A
UTOIP4,DNS,VPN,SHARING,SUPPLICANT,AGENTS,SETTINGS,SUSPEND,CORE,DEVIC
```

```
E,OLPC,  
WIMAX,INFINIBAND,FIREWALL,ADSL,BOND,VLAN,BRIDGE,DBUS_PROPS,TEAM,CONC  
HECK,DC  
B,DISPATCH
```

2. To modify the logging level and domains, use the following options:

- To set the log level for all domains to the same **LEVEL**, enter:

```
# nmcli general logging level LEVEL domains ALL
```

- To change the level for specific domains, enter:

```
# nmcli general logging level LEVEL domains DOMAINS
```

Note that updating the logging level using this command disables logging for all the other domains.

- To change the level of specific domains and preserve the level of all other domains, enter:

```
# nmcli general logging level KEEP domains DOMAIN:LEVEL,DOMAIN:LEVEL
```

43.4. VIEWING NETWORKMANAGER LOGS

You can view the NetworkManager logs for troubleshooting.

Procedure

- To view the logs, enter:

```
# journalctl -u NetworkManager -b
```

Additional resources

- **NetworkManager.conf(5)** man page
- **journalctl(1)** man page

CHAPTER 44. INTRODUCTION TO NMSTATE

Nmstate is a declarative network manager API. The **nmstate** package provides the **libnmstate** Python library and a command-line utility, **nmstatectl**, to manage NetworkManager on RHEL. When you use Nmstate, you describe the expected networking state 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
- Provides plug-in support to enable administrators to use their own plug-ins

44.1. 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** package to use this library.

Example 44.1. Querying the network state using the libnmstate library

The following Python code imports the **libnmstate** library and displays the available network interfaces and their state:

```
import json
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])
```

44.2. UPDATING THE CURRENT NETWORK CONFIGURATION 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.

This procedure describes how to 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.

44.3. 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.

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
- [Introduction to Nmstate](#)

44.4. ADDITIONAL RESOURCES

- [/usr/share/doc/nmstate/README.md](#)
- [/usr/share/doc/nmstate/examples/](#)

CHAPTER 45. CAPTURING NETWORK PACKETS

To debug network issues and communications, you can capture network packets. The following sections provide instructions and additional information about capturing network packets.

45.1. USING XDPDUMP TO CAPTURE NETWORK PACKETS INCLUDING PACKETS DROPPED BY XDP PROGRAMS

The **xdpdump** utility captures network packets. Unlike the **tcpdump** utility, **xdpdump** uses an extended Berkeley Packet Filter (eBPF) program for this task. This enables **xdpdump** to also capture packets dropped by Express Data Path (XDP) programs. User-space utilities, such as **tcpdump**, are not able to capture these dropped packages, as well as original packets modified by an XDP program.

You can use **xdpdump** to debug XDP programs that are already attached to an interface. Therefore, the utility can capture packets before an XDP program is started and after it has finished. In the latter case, **xdpdump** also captures the XDP action. By default, **xdpdump** captures incoming packets at the entry of the XDP program.



IMPORTANT

On other architectures than AMD and Intel 64-bit, the **xdpdump** utility 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 **xdpdump** has no packet filter or decode capabilities. However, you can use it in combination with **tcpdump** for packet decoding.

The procedure describes how to capture all packets on the **enp1s0** interface and write them to the **/root/capture.pcap** file.

Prerequisites

- A network driver that supports XDP programs.
- An XDP program is loaded to the **enp1s0** interface. If no program is loaded, **xdpdump** captures packets in a similar way **tcpdump** does, for backward compatibility.

Procedure

1. To capture packets on the **enp1s0** interface and write them to the **/root/capture.pcap** file, enter:

```
# xdpdump -i enp1s0 -w /root/capture.pcap
```

2. To stop capturing packets, press **Ctrl+C**.

Additional resources

- **xdpdump(8)** man page
- If you are a developer and you are interested in the source code of **xdpdump**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

45.2. ADDITIONAL RESOURCES

- [How to capture network packets with tcpdump?](#)

CHAPTER 46. USING AND CONFIGURING FIREWALLD

A *firewall* is a way to protect machines from any unwanted traffic from outside. It enables users to control incoming network traffic on host machines by defining a set of *firewall rules*. These rules are used to sort the incoming traffic and either block it or allow through.

firewalld is a firewall service daemon that provides a dynamic customizable host-based firewall with a D-Bus interface. Being dynamic, it enables creating, changing, and deleting the rules without the necessity to restart the firewall daemon each time the rules are changed.

firewalld uses the concepts of zones and services, that simplify the traffic management. Zones are predefined sets of rules. Network interfaces and sources can be assigned to a zone. The traffic allowed depends on the network your computer is connected to and the security level this network is assigned. Firewall services are predefined rules that cover all necessary settings to allow incoming traffic for a specific service and they apply within a zone.

Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic by default.

Note that **firewalld** with **nftables** backend does not support passing custom **nftables** rules to **firewalld**, using the **--direct** option.

46.1. GETTING STARTED WITH FIREWALLD

This section provides information about **firewalld**.

46.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance-critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.



IMPORTANT

To prevent the different firewall services from influencing each other, run only one of them on a RHEL host, and disable the other services.

46.1.2. Zones

firewalld can be used to separate networks into different zones according to the level of trust that the user has decided to place on the interfaces and traffic within that network. A connection can only be part of one zone, but a zone can be used for many network connections.

NetworkManager notifies **firewalld** of the zone of an interface. You can assign zones to interfaces with:

- **NetworkManager**
- **firewall-config** tool
- **firewall-cmd** command-line tool
- The RHEL web console

The latter three can only edit the appropriate **NetworkManager** configuration files. If you change the zone of the interface using the web console, **firewall-cmd** or **firewall-config**, the request is forwarded to **NetworkManager** and is not handled by **firewalld**.

The predefined zones are stored in the **/usr/lib/firewalld/zones/** directory and can be instantly applied to any available network interface. These files are copied to the **/etc/firewalld/zones/** directory only after they are modified. The default settings of the predefined zones are as follows:

block

Any incoming network connections are rejected with an icmp-host-prohibited message for **IPv4** and icmp6-adm-prohibited for **IPv6**. Only network connections initiated from within the system are possible.

dmz

For computers in your demilitarized zone that are publicly-accessible with limited access to your internal network. Only selected incoming connections are accepted.

drop

Any incoming network packets are dropped without any notification. Only outgoing network connections are possible.

external

For use on external networks with masquerading enabled, especially for routers. You do not trust the other computers on the network to not harm your computer. Only selected incoming connections are accepted.

home

For use at home when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

internal

For use on internal networks when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

public

For use in public areas where you do not trust other computers on the network. Only selected incoming connections are accepted.

trusted

All network connections are accepted.

work

For use at work where you mostly trust the other computers on the network. Only selected incoming connections are accepted.

One of these zones is set as the *default* zone. When interface connections are added to **NetworkManager**, they are assigned to the default zone. On installation, the default zone in **firewalld** is set to be the **public** zone. The default zone can be changed.



NOTE

The network zone names should be self-explanatory and to allow users to quickly make a reasonable decision. To avoid any security problems, review the default zone configuration and disable any unnecessary services according to your needs and risk assessments.

Additional resources

- The **firewalld.zone(5)** man page.

46.1.3. Predefined services

A service can be a list of local ports, protocols, source ports, and destinations, as well as a list of firewall helper modules automatically loaded if a service is enabled. Using services saves users time because they can achieve several tasks, such as opening ports, defining protocols, enabling packet forwarding and more, in a single step, rather than setting up everything one after another.

Service configuration options and generic file information are described in the **firewalld.service(5)** man page. The services are specified by means of individual XML configuration files, which are named in the following format: **service-name.xml**. Protocol names are preferred over service or application names in **firewalld**.

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**.

Alternatively, you can edit the XML files in the **/etc/firewalld/services/** directory. If a service is not added or changed by the user, then no corresponding XML file is found in **/etc/firewalld/services/**. The files in the **/usr/lib/firewalld/services/** directory can be used as templates if you want to add or change a service.

Additional resources

- The **firewalld.service(5)** man page

46.1.4. Starting firewalld

Procedure

1. To start **firewalld**, enter the following command as **root**:

```
# systemctl unmask firewalld
# systemctl start firewalld
```

2. To ensure **firewalld** starts automatically at system start, enter the following command as **root**:

```
# systemctl enable firewalld
```

46.1.5. Stopping firewalld

Procedure

1. To stop **firewalld**, enter the following command as **root**:

```
# systemctl stop firewalld
```

2. To prevent **firewalld** from starting automatically at system start:

```
# systemctl disable firewalld
```

3. To make sure firewalld is not started by accessing the **firewalld D-Bus** interface and also if other services require **firewalld**:

```
# systemctl mask firewalld
```

46.1.6. Verifying the permanent firewalld configuration

In certain situations, for example after manually editing **firewalld** configuration files, administrators want to verify that the changes are correct. This section describes how to verify the permanent configuration of the **firewalld** service.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Verify the permanent configuration of the **firewalld** service:

```
# firewall-cmd --check-config
success
```

If the permanent configuration is valid, the command returns **success**. In other cases, the command returns an error with further details, such as the following:

```
# firewall-cmd --check-config
Error: INVALID_PROTOCOL: 'public.xml': 'tcpx' not from {'tcp'|'udp'|'sctp'|'dccp'}
```

46.2. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD

This section covers information about viewing current status, allowed services, and current settings of **firewalld**.

46.2.1. Viewing the current status of firewalld

The firewall service, **firewalld**, is installed on the system by default. Use the **firewalld** CLI interface to check that the service is running.

Procedure

1. To see the status of the service:

```
# firewall-cmd --state
```

2. For more information about the service status, use the **systemctl status** sub-command:

```
# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor pr
  Active: active (running) since Mon 2017-12-18 16:05:15 CET; 50min ago
    Docs: man:firewalld(1)
  Main PID: 705 (firewalld)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/firewalld.service
           └─705 /usr/bin/python3 -Es /usr/sbin/firewalld --nofork --nopid
```

46.2.2. Viewing allowed services using GUI

To view the list of services using the graphical **firewall-config** tool, press the **Super** key to enter the Activities Overview, type **firewall**, and press **Enter**. The **firewall-config** tool appears. You can now view the list of services under the **Services** tab.

You can start the graphical firewall configuration tool using the command-line.

Prerequisites

- You installed the **firewall-config** package.

Procedure

- To start the graphical firewall configuration tool using the command-line:

```
$ firewall-config
```

The **Firewall Configuration** window opens. Note that this command can be run as a normal user, but you are prompted for an administrator password occasionally.

46.2.3. Viewing firewalld settings using CLI

With the CLI client, it is possible to get different views of the current firewall settings. The **--list-all** option shows a complete overview of the **firewalld** settings.

firewalld uses zones to manage the traffic. If a zone is not specified by the **--zone** option, the command is effective in the default zone assigned to the active network interface and connection.

Procedure

- To list all the relevant information for the default zone:

```
# firewall-cmd --list-all
public
target: default
icmp-block-inversion: no
interfaces:
sources:
services: ssh dhcpv6-client
ports:
protocols:
masquerade: no
forward-ports:
```

```
source-ports:
icmp-blocks:
rich rules:
```

- To specify the zone for which to display the settings, add the **--zone=zone-name** argument to the **firewall-cmd --list-all** command, for example:

```
# firewall-cmd --list-all --zone=home
home
target: default
icmp-block-inversion: no
interfaces:
sources:
services: ssh mdns samba-client dhcpv6-client
... [trimmed for clarity]
```

- To see the settings for particular information, such as services or ports, use a specific option. See the **firewalld** manual pages or get a list of the options using the command help:

```
# firewall-cmd --help
```

- To see which services are allowed in the current zone:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```



NOTE

Listing the settings for a certain subpart using the CLI tool can sometimes be difficult to interpret. For example, you allow the **SSH** service and **firewalld** opens the necessary port (22) for the service. Later, if you list the allowed services, the list shows the **SSH** service, but if you list open ports, it does not show any. Therefore, it is recommended to use the **--list-all** option to make sure you receive a complete information.

46.3. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

This section covers information about controlling network traffic using **firewalld**.

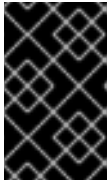
46.3.1. Disabling all traffic in case of emergency using CLI

In an emergency situation, such as a system attack, it is possible to disable all network traffic and cut off the attacker.

Procedure

1. To immediately disable networking traffic, switch panic mode on:

```
# firewall-cmd --panic-on
```



IMPORTANT

Enabling panic mode stops all networking traffic. For this reason, it should be used only when you have the physical access to the machine or if you are logged in using a serial console.

2. Switching off panic mode reverts the firewall to its permanent settings. To switch panic mode off, enter:

```
# firewall-cmd --panic-off
```

Verification

- To see whether panic mode is switched on or off, use:

```
# firewall-cmd --query-panic
```

46.3.2. Controlling traffic with predefined services using CLI

The most straightforward method to control traffic is to add a predefined service to **firewalld**. This opens all necessary ports and modifies other settings according to the *service definition file*.

Procedure

1. Check that the service is not already allowed:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```

2. List all predefined services:

```
# firewall-cmd --get-services
RH-Satellite-6 amanda-client amanda-k5-client bacula bacula-client bitcoin bitcoin-rpc
bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine condor-collector ctdb dhcp dhcpv6
dhcpv6-client dns docker-registry ...
[trimmed for clarity]
```

3. Add the service to the allowed services:

```
# firewall-cmd --add-service=<service-name>
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.3.3. Controlling traffic with predefined services using GUI

This procedure describes how to control the network traffic with predefined services using graphical user interface.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. To enable or disable a predefined or custom service:
 - a. Start the **firewall-config** tool and select the network zone whose services are to be configured.
 - b. Select the **Zones** tab and then the **Services** tab below.
 - c. Select the check box for each type of service you want to trust or clear the check box to block a service in the selected zone.
2. To edit a service:
 - a. Start the **firewall-config** tool.
 - b. Select **Permanent** from the menu labeled **Configuration**. Additional icons and menu buttons appear at the bottom of the **Services** window.
 - c. Select the service you want to configure.

The **Ports**, **Protocols**, and **Source Port** tabs enable adding, changing, and removing of ports, protocols, and source port for the selected service. The modules tab is for configuring **Netfilter** helper modules. The **Destination** tab enables limiting traffic to a particular destination address and Internet Protocol (**IPv4** or **IPv6**).



NOTE

It is not possible to alter service settings in the **Runtime** mode.

46.3.4. Adding new services

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**. Alternatively, you can edit the XML files in **/etc/firewalld/services/**. If a service is not added or changed by the user, then no corresponding XML file are found in **/etc/firewalld/services/**. The files **/usr/lib/firewalld/services/** can be used as templates if you want to add or change a service.



NOTE

Service names must be alphanumeric and can, additionally, include only **_** (underscore) and **-** (dash) characters.

Procedure

To add a new service in a terminal, use **firewall-cmd**, or **firewall-offline-cmd** in case of not active **firewalld**.

1. Enter the following command to add a new and empty service:

```
$ firewall-cmd --new-service=service-name --permanent
```

2. To add a new service using a local file, use the following command:

```
$ firewall-cmd --new-service-from-file=service-name.xml --permanent
```

You can change the service name with the additional **--name=*service-name*** option.

3. As soon as service settings are changed, an updated copy of the service is placed into **/etc/firewalld/services/**.

As **root**, you can enter the following command to copy a service manually:

```
# cp /usr/lib/firewalld/services/service-name.xml /etc/firewalld/services/service-name.xml
```

firewalld loads files from **/usr/lib/firewalld/services** in the first place. If files are placed in **/etc/firewalld/services** and they are valid, then these will override the matching files from **/usr/lib/firewalld/services**. The overridden files in **/usr/lib/firewalld/services** are used as soon as the matching files in **/etc/firewalld/services** have been removed or if **firewalld** has been asked to load the defaults of the services. This applies to the permanent environment only. A reload is needed to get these fallbacks also in the runtime environment.

46.3.5. Opening ports using GUI

To permit traffic through the firewall to a certain port, you can open the port in the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Ports** tab and click the **Add** button on the right-hand side. The **Port and Protocol** window opens.
3. Enter the port number or range of ports to permit.
4. Select **tcp** or **udp** from the list.

46.3.6. Controlling traffic with protocols using GUI

To permit traffic through the firewall using a certain protocol, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Protocols** tab and click the **Add** button on the right-hand side. The **Protocol** window opens.
3. Either select a protocol from the list or select the **Other Protocol** check box and enter the protocol in the field.

46.3.7. Opening source ports using GUI

To permit traffic through the firewall from a certain port, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Source Port** tab and click the **Add** button on the right-hand side. The **Source Port** window opens.
3. Enter the port number or range of ports to permit. Select **tcp** or **udp** from the list.

46.4. CONTROLLING PORTS USING CLI

Ports are logical devices that enable an operating system to receive and distinguish network traffic and forward it accordingly to system services. These are usually represented by a daemon that listens on the port, that is it waits for any traffic coming to this port.

Normally, system services listen on standard ports that are reserved for them. The **httpd** daemon, for example, listens on port 80. However, system administrators by default configure daemons to listen on different ports to enhance security or for other reasons.

46.4.1. Opening a port

Through open ports, the system is accessible from the outside, which represents a security risk. Generally, keep ports closed and only open them if they are required for certain services.

Procedure

To get a list of open ports in the current zone:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```

2. Add a port to the allowed ports to open it for incoming traffic:

```
# firewall-cmd --add-port=port-number/port-type
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

46.4.2. Closing a port

When an open port is no longer needed, close that port in **firewalld**. It is highly recommended to close all unnecessary ports as soon as they are not used because leaving a port open represents a security risk.

Procedure

To close a port, remove it from the list of allowed ports:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```



WARNING

This command will only give you a list of ports that have been opened as ports. You will not be able to see any open ports that have been opened as a service. Therefore, you should consider using the **--list-all** option instead of **--list-ports**.

2. Remove the port from the allowed ports to close it for the incoming traffic:

```
# firewall-cmd --remove-port=port-number/port-type
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.5. WORKING WITH FIREWALLD ZONES

Zones represent a concept to manage incoming traffic more transparently. The zones are connected to networking interfaces or assigned a range of source addresses. You manage firewall rules for each zone independently, which enables you to define complex firewall settings and apply them to the traffic.

46.5.1. Listing zones

This procedure describes how to list zones using the command line.

Procedure

1. To see which zones are available on your system:

```
# firewall-cmd --get-zones
```

The **firewall-cmd --get-zones** command displays all zones that are available on the system, but it does not show any details for particular zones.

2. To see detailed information for all zones:

■

```
# firewall-cmd --list-all-zones
```

3. To see detailed information for a specific zone:

```
# firewall-cmd --zone=zone-name --list-all
```

46.5.2. Modifying firewalld settings for a certain zone

The [Controlling traffic with predefined services using cli](#) and [Controlling ports using cli](#) explain how to add services or modify ports in the scope of the current working zone. Sometimes, it is required to set up rules in a different zone.

Procedure

- To work in a different zone, use the **--zone=zone-name** option. For example, to allow the **SSH** service in the zone *public*:

```
# firewall-cmd --add-service=ssh --zone=public
```

46.5.3. Changing the default zone

System administrators assign a zone to a networking interface in its configuration files. If an interface is not assigned to a specific zone, it is assigned to the default zone. After each restart of the **firewalld** service, **firewalld** loads the settings for the default zone and makes it active.

Procedure

To set up the default zone:

1. Display the current default zone:

```
# firewall-cmd --get-default-zone
```

2. Set the new default zone:

```
# firewall-cmd --set-default-zone zone-name
```



NOTE

Following this procedure, the setting is a permanent setting, even without the **--permanent** option.

46.5.4. Assigning a network interface to a zone

It is possible to define different sets of rules for different zones and then change the settings quickly by changing the zone for the interface that is being used. With multiple interfaces, a specific zone can be set for each of them to distinguish traffic that is coming through them.

Procedure

To assign the zone to a specific interface:

1. List the active zones and the interfaces assigned to them:

—

```
# firewall-cmd --get-active-zones
```

2. Assign the interface to a different zone:

```
# firewall-cmd --zone=zone_name --change-interface=interface_name --permanent
```

46.5.5. Assigning a zone to a connection using nmcli

This procedure describes how to add a **firewalld** zone to a **NetworkManager** connection using the **nmcli** utility.

Procedure

1. Assign the zone to the **NetworkManager** connection profile:

```
# nmcli connection modify profile connection.zone zone_name
```

2. Activate the connection:

```
# nmcli connection up profile
```

46.5.6. Manually assigning a zone to a network connection in an ifcfg file

When the connection is managed by **NetworkManager**, it must be aware of a zone that it uses. For every network connection, a zone can be specified, which provides the flexibility of various firewall settings according to the location of the computer with portable devices. Thus, zones and settings can be specified for different locations, such as company or home.

Procedure

- To set a zone for a connection, edit the **/etc/sysconfig/network-scripts/ifcfg-*connection_name*** file and add a line that assigns a zone to this connection:

```
ZONE=zone_name
```

46.5.7. Creating a new zone

To use custom zones, create a new zone and use it just like a predefined zone. New zones require the **--permanent** option, otherwise the command does not work.

Procedure

1. Create a new zone:

```
# firewall-cmd --permanent --new-zone=zone-name
```

2. Check if the new zone is added to your permanent settings:

```
# firewall-cmd --get-zones
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.5.8. Zone configuration files

Zones can also be created using a *zone configuration file*. This approach can be helpful when you need to create a new zone, but want to reuse the settings from a different zone and only alter them a little.

A **firewalld** zone configuration file contains the information for a zone. These are the zone description, services, ports, protocols, icmp-blocks, masquerade, forward-ports and rich language rules in an XML file format. The file name has to be **zone-name.xml** where the length of *zone-name* is currently limited to 17 chars. The zone configuration files are located in the **/usr/lib/firewalld/zones/** and **/etc/firewalld/zones/** directories.

The following example shows a configuration that allows one service (**SSH**) and one port range, for both the **TCP** and **UDP** protocols:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>My Zone</short>
  <description>Here you can describe the characteristic features of the zone.</description>
  <service name="ssh"/>
  <port protocol="udp" port="1025-65535"/>
  <port protocol="tcp" port="1025-65535"/>
</zone>
```

To change settings for that zone, add or remove sections to add ports, forward ports, services, and so on.

Additional resources

- **firewalld.zone** manual page

46.5.9. Using zone targets to set default behavior for incoming traffic

For every zone, you can set a default behavior that handles incoming traffic that is not further specified. Such behavior is defined by setting the target of the zone. There are four options:

- **ACCEPT**: Accepts all incoming packets except those disallowed by specific rules.
- **REJECT**: Rejects all incoming packets except those allowed by specific rules. When **firewalld** rejects packets, the source machine is informed about the rejection.
- **DROP**: Drops all incoming packets except those allowed by specific rules. When **firewalld** drops packets, the source machine is not informed about the packet drop.
- **default**: Similar behavior as for **REJECT**, but with special meanings in certain scenarios. For details, see the **Options to Adapt and Query Zones and Policies** section in the **firewall-cmd(1)** man page.

Procedure

To set a target for a zone:

1. List the information for the specific zone to see the default target:

```
# firewall-cmd --zone=zone-name --list-all
```

2. Set a new target in the zone:

```
# firewall-cmd --permanent --zone=zone-name --set-target=  
<default|ACCEPT|REJECT|DROP>
```

Additional resources

- **firewall-cmd(1)** man page

46.6. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

You can use zones to manage incoming traffic based on its source. That enables you to sort incoming traffic and route it through different zones to allow or disallow services that can be reached by that traffic.

If you add a source to a zone, the zone becomes active and any incoming traffic from that source will be directed through it. You can specify different settings for each zone, which is applied to the traffic from the given sources accordingly. You can use more zones even if you only have one network interface.

46.6.1. Adding a source

To route incoming traffic into a specific zone, add the source to that zone. The source can be an IP address or an IP mask in the classless inter-domain routing (CIDR) notation.



NOTE

In case you add multiple zones with an overlapping network range, they are ordered alphanumerically by zone name and only the first one is considered.

- To set the source in the current zone:

```
# firewall-cmd --add-source=<source>
```

- To set the source IP address for a specific zone:

```
# firewall-cmd --zone=zone-name --add-source=<source>
```

The following procedure allows all incoming traffic from *192.168.2.15* in the **trusted** zone:

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
```

2. Add the source IP to the trusted zone in the permanent mode:

```
# firewall-cmd --zone=trusted --add-source=192.168.2.15
```


3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.6.2. Removing a source

Removing a source from the zone cuts off the traffic coming from it.

Procedure

1. List allowed sources for the required zone:

```
# firewall-cmd --zone=zone-name --list-sources
```

2. Remove the source from the zone permanently:

```
# firewall-cmd --zone=zone-name --remove-source=<source>
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.6.3. Adding a source port

To enable sorting the traffic based on a port of origin, specify a source port using the **--add-source-port** option. You can also combine this with the **--add-source** option to limit the traffic to a certain IP address or IP range.

Procedure

- To add a source port:

```
# firewall-cmd --zone=zone-name --add-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

46.6.4. Removing a source port

By removing a source port you disable sorting the traffic based on a port of origin.

Procedure

- To remove a source port:

```
# firewall-cmd --zone=zone-name --remove-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

46.6.5. Using zones and sources to allow a service for only a specific domain

To allow traffic from a specific network to use a service on a machine, use zones and source. The following procedure allows only HTTP traffic from the **192.0.2.0/24** network while any other traffic is blocked.

**WARNING**

When you configure this scenario, use a zone that has the **default** target. Using a zone that has the target set to **ACCEPT** is a security risk, because for traffic from **192.0.2.0/24**, all network connections would be accepted.

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

2. Add the IP range to the **internal** zone to route the traffic originating from the source through the zone:

```
# firewall-cmd --zone=internal --add-source=192.0.2.0/24
```

3. Add the **http** service to the **internal** zone:

```
# firewall-cmd --zone=internal --add-service=http
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

Verification

- Check that the **internal** zone is active and that the service is allowed in it:

```
# firewall-cmd --zone=internal --list-all
internal (active)
target: default
icmp-block-inversion: no
interfaces:
sources: 192.0.2.0/24
services: cockpit dhcpv6-client mdns samba-client ssh http
...
```

Additional resources

- **firewalld.zones(5)** man page

46.7. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

With a policy object, users can group different identities that require similar permissions in the policy. You can apply policies depending on the direction of the traffic.

The policy objects feature provides forward and output filtering in firewalld. The following describes the usage of firewalld to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

46.7.1. The relationship between policy objects and zones

Policy objects allow the user to attach firewalld's primitives' such as services, ports, and rich rules to the policy. You can apply the policy objects to traffic that passes between zones in a stateful and unidirectional manner.

```
# firewall-cmd --permanent --new-policy myOutputPolicy

# firewall-cmd --permanent --policy myOutputPolicy --add-ingress-zone HOST

# firewall-cmd --permanent --policy myOutputPolicy --add-egress-zone ANY
```

HOST and **ANY** are the symbolic zones used in the ingress and egress zone lists.

- The **HOST** symbolic zone allows policies for the traffic originating from or has a destination to the host running firewalld.
- The **ANY** symbolic zone applies policy to all the current and future zones. **ANY** symbolic zone acts as a wildcard for all zones.

46.7.2. Using priorities to sort policies

Multiple policies can apply to the same set of traffic, therefore, priorities should be used to create an order of precedence for the policies that may be applied.

To set a priority to sort the policies:

```
# firewall-cmd --permanent --policy mypolicy --set-priority -500
```

In the above example **-500** is a lower priority value but has higher precedence. Thus, **-500** will execute before **-100**. Higher priority values have precedence over lower values.

The following rules apply to policy priorities:

- Policies with negative priorities apply before rules in zones.
- Policies with positive priorities apply after rules in zones.
- Priority 0 is reserved and hence is unusable.

46.7.3. Using policy objects to filter traffic between locally hosted Containers and a network physically connected to the host

The policy objects feature allows users to filter their container and virtual machine traffic.

Procedure

1. Create a new policy.

```
# firewall-cmd --permanent --new-policy podmanToHost
```

2. Block all traffic.

```
# firewall-cmd --permanent --policy podmanToHost --set-target REJECT
# firewall-cmd --permanent --policy podmanToHost --add-service dhcp
# firewall-cmd --permanent --policy podmanToHost --add-service dns
```



NOTE

Red Hat recommends that you block all traffic to the host by default and then selectively open the services you need for the host.

3. Define the ingress zone to use with the policy.

```
# firewall-cmd --permanent --policy podmanToHost --add-ingress-zone podman
```

4. Define the egress zone to use with the policy.

```
# firewall-cmd --permanent --policy podmanToHost --add-egress-zone ANY
```

Verification

- Verify information about the policy.

```
# firewall-cmd --info-policy podmanToHost
```

46.7.4. Setting the default target of policy objects

You can specify `--set-target` options for policies. The following targets are available:

- **ACCEPT** - accepts the packet
- **DROP** - drops the unwanted packets
- **REJECT** - rejects unwanted packets with an ICMP reply
- **CONTINUE (default)** - packets will be subject to rules in following policies and zones.

```
# firewall-cmd --permanent --policy mypolicy --set-target CONTINUE
```

Verification

- Verify information about the policy

```
# firewall-cmd --info-policy mypolicy
```

46.8. CONFIGURING NAT USING FIREWALLD

With **firewalld**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

46.8.1. NAT types

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.

Masquerading and SNAT are very similar to one another. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

46.8.2. Configuring IP address masquerading

The following procedure describes how to enable IP masquerading on your system. IP masquerading hides individual machines behind a gateway when accessing the Internet.

Procedure

1. To check if IP masquerading is enabled (for example, for the **external** zone), enter the following command as **root**:

```
# firewall-cmd --zone=external --query-masquerade
```

The command prints **yes** with exit status **0** if enabled. It prints **no** with exit status **1** otherwise. If **zone** is omitted, the default zone will be used.

2. To enable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --add-masquerade
```

3. To make this setting persistent, pass the **--permanent** option to the command.
4. To disable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --remove-masquerade
```

To make this setting permanent, pass the **--permanent** option to the command.

46.9. USING DNAT TO FORWARD HTTPS TRAFFIC TO A DIFFERENT HOST

If your web server runs in a DMZ with private IP addresses, you can configure destination network address translation (DNAT) to enable clients on the internet to connect to this web server. In this case, the host name of the web server resolves to the public IP address of the router. When a client establishes a connection to a defined port on the router, the router forwards the packets to the internal web server.

Prerequisites

- The DNS server resolves the host name of the web server to the router's IP address.
- You know the following settings:
 - The private IP address and port number that you want to forward
 - The IP protocol to be used
 - The destination IP address and port of the web server where you want to redirect the packets

Procedure

1. Create a firewall policy:

```
# firewall-cmd --permanent --new-policy ExamplePolicy
```

The policies, as opposed to zones, allow packet filtering for input, output, and forwarded traffic. This is important, because forwarding traffic to endpoints on locally run web servers, containers, or virtual machines requires such capability.

2. Configure symbolic zones for the ingress and egress traffic to also enable the router itself to connect to its local IP address and forward this traffic:

```
# firewall-cmd --permanent --policy=ExamplePolicy --add-ingress-zone=HOST  
# firewall-cmd --permanent --policy=ExamplePolicy --add-egress-zone=ANY
```

The **--add-ingress-zone=HOST** option refers to packets generated locally, which are transmitted out of the local host. The **--add-egress-zone=ANY** option refers to traffic destined to any zone.

3. Add a rich rule that forwards traffic to the web server:

```
# firewall-cmd --permanent --policy=ExamplePolicy --add-rich-rule='rule family="ipv4"
destination address="192.0.2.1" forward-port port="443" protocol="tcp" to-port="443"
to-addr="192.51.100.20"
```

The rich rule forwards TCP traffic from port 443 on the router's IP address 192.0.2.1 to port 443 of the web server's IP 198.51.100.20. The rule uses the **ExamplePolicy** to ensure that the router can also connect to its local IP address.

4. Reload the firewall configuration files:

```
# firewall-cmd --reload
success
```

5. Activate routing of 127.0.0.0/8 in the kernel:

```
# echo "net.ipv4.conf.all.route_localnet=1" > /etc/sysctl.d/90-enable-route-localnet.conf
# sysctl -p /etc/sysctl.d/90-enable-route-localnet.conf
```

Verification

1. Connect to the router's IP address and port that you have forwarded to the web server:

```
# curl https://192.0.2.1:443
```

2. Optional: Verify that **net.ipv4.conf.all.route_localnet** is active:

```
# sysctl net.ipv4.conf.all.route_localnet
net.ipv4.conf.all.route_localnet = 1
```

3. Verify that **ExamplePolicy** is active and contains the settings you need. Especially the source IP address and port, protocol to be used, and the destination IP address and port:

```
# firewall-cmd --info-policy=ExamplePolicy
ExamplePolicy (active)
priority: -1
target: CONTINUE
ingress-zones: HOST
egress-zones: ANY
services:
ports:
protocols:
masquerade: no
forward-ports:
source-ports:
icmp-blocks:
rich rules:
rule family="ipv4" destination address="192.0.2.1" forward-port port="443" protocol="tcp" to-
port="443" to-addr="192.51.100.20"
```

Additional resources

- **firewall-cmd(1)**

- **firewalld.policies(5)**
- **firewalld.richlanguage(5)**
- **sysctl(8)**
- **sysctl.conf(5)**
- [Using configuration files in /etc/sysctl.d/ to adjust kernel parameters](#)

46.10. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol (ICMP)** is a supporting protocol that is used by various network devices to send error messages and operational information indicating a connection problem, for example, that a requested service is not available. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

Unfortunately, it is possible to use the **ICMP** messages, especially **echo-request** and **echo-reply**, to reveal information about your network and misuse such information for various kinds of fraudulent activities. Therefore, **firewalld** enables blocking the **ICMP** requests to protect your network information.

46.10.1. Listing and blocking ICMP requests

Listing ICMP requests

The **ICMP** requests are described in individual XML files that are located in the **/usr/lib/firewalld/icmptypes/** directory. You can read these files to see a description of the request. The **firewall-cmd** command controls the **ICMP** requests manipulation.

- To list all available **ICMP** types:

```
# firewall-cmd --get-icmptypes
```

- The **ICMP** request can be used by IPv4, IPv6, or by both protocols. To see for which protocol the **ICMP** request has used:

```
# firewall-cmd --info-icmptype=<icmptype>
```

- The status of an **ICMP** request shows **yes** if the request is currently blocked or **no** if it is not. To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```

Blocking or unblocking ICMP requests

When your server blocks **ICMP** requests, it does not provide the information that it normally would. However, that does not mean that no information is given at all. The clients receive information that the particular **ICMP** request is being blocked (rejected). Blocking the **ICMP** requests should be considered carefully, because it can cause communication problems, especially with IPv6 traffic.

- To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```


- To block an **ICMP** request:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

- To remove the block for an **ICMP** request:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

Blocking ICMP requests without providing any information at all

Normally, if you block **ICMP** requests, clients know that you are blocking it. So, a potential attacker who is sniffing for live IP addresses is still able to see that your IP address is online. To hide this information completely, you have to drop all **ICMP** requests.

- To block and drop all **ICMP** requests:
- Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

Now, all traffic, including **ICMP** requests, is dropped, except traffic which you have explicitly allowed.

To block and drop certain **ICMP** requests and allow others:

1. Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

2. Add the ICMP block inversion to block all **ICMP** requests at once:

```
# firewall-cmd --add-icmp-block-inversion
```

3. Add the ICMP block for those **ICMP** requests that you want to allow:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The *block inversion* inverts the setting of the **ICMP** requests blocks, so all requests, that were not previously blocked, are blocked because of the target of your zone changes to **DROP**. The requests that were blocked are not blocked. This means that if you want to unblock a request, you must use the blocking command.

To revert the block inversion to a fully permissive setting:

1. Set the target of your zone to **default** or **ACCEPT**:

```
# firewall-cmd --permanent --set-target=default
```

2. Remove all added blocks for **ICMP** requests:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

3. Remove the **ICMP** block inversion:

```
# firewall-cmd --remove-icmp-block-inversion
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

46.10.2. Configuring the ICMP filter using GUI

- To enable or disable an **ICMP** filter, start the **firewall-config** tool and select the network zone whose messages are to be filtered. Select the **ICMP Filter** tab and select the check box for each type of **ICMP** message you want to filter. Clear the check box to disable a filter. This setting is per direction and the default allows everything.
- To enable inverting the **ICMP Filter**, click the **Invert Filter** check box on the right. Only marked **ICMP** types are now accepted, all other are rejected. In a zone using the DROP target, they are dropped.

46.11. SETTING AND CONTROLLING IP SETS USING FIREWALLD

To see the list of IP set types supported by **firewalld**, enter the following command as root.

```
~]# firewall-cmd --get-ipset-types
hash:ip hash:ip,mark hash:ip,port hash:ip,port,ip hash:ip,port,net hash:mac hash:net hash:net,iface
hash:net,net hash:net,port hash:net,port,net
```

46.11.1. Configuring IP set options using CLI

IP sets can be used in **firewalld** zones as sources and also as sources in rich rules. In Red Hat Enterprise Linux, the preferred method is to use the IP sets created with **firewalld** in a direct rule.

- To list the IP sets known to **firewalld** in the permanent environment, use the following command as **root**:

```
# firewall-cmd --permanent --get-ipsets
```

- To add a new IP set, use the following command using the permanent environment as **root**:

```
# firewall-cmd --permanent --new-ipset=test --type=hash:net
success
```

The previous command creates a new IP set with the name *test* and the **hash:net** type for **IPv4**. To create an IP set for use with **IPv6**, add the **--option=family=inet6** option. To make the new setting effective in the runtime environment, reload **firewalld**.

- List the new IP set with the following command as **root**:

```
# firewall-cmd --permanent --get-ipsets
test
```

- To get more information about the IP set, use the following command as **root**:

```
# firewall-cmd --permanent --info-ipset=test
test
type: hash:net
options:
entries:
```

Note that the IP set does not have any entries at the moment.

- To add an entry to the *test* IP set, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --add-entry=192.168.0.1
success
```

The previous command adds the IP address *192.168.0.1* to the IP set.

- To get the list of current entries in the IP set, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --get-entries
192.168.0.1
```

- Generate a file containing a list of IP addresses, for example:

```
# cat > iplist.txt <<EOL
192.168.0.2
192.168.0.3
192.168.1.0/24
192.168.2.254
EOL
```

The file with the list of IP addresses for an IP set should contain an entry per line. Lines starting with a hash, a semi-colon, or empty lines are ignored.

- To add the addresses from the *iplist.txt* file, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --add-entries-from-file=iplist.txt
success
```

- To see the extended entries list of the IP set, use the following command as **root**:

```
# firewall-cmd --permanent --ipset=test --get-entries
192.168.0.1
192.168.0.2
192.168.0.3
192.168.1.0/24
192.168.2.254
```

- To remove the addresses from the IP set and to check the updated entries list, use the following commands as **root**:

```
# firewall-cmd --permanent --ipset=test --remove-entries-from-file=iplist.txt
success
# firewall-cmd --permanent --ipset=test --get-entries
```

192.168.0.1

- You can add the IP set as a source to a zone to handle all traffic coming in from any of the addresses listed in the IP set with a zone. For example, to add the *test* IP set as a source to the *drop* zone to drop all packets coming from all entries listed in the *test* IP set, use the following command as **root**:

```
# firewall-cmd --permanent --zone=drop --add-source=ipset:test
success
```

The **ipset:** prefix in the source shows **firewalld** that the source is an IP set and not an IP address or an address range.

Only the creation and removal of IP sets is limited to the permanent environment, all other IP set options can be used also in the runtime environment without the **--permanent** option.



WARNING

Red Hat does not recommend using IP sets that are not managed through **firewalld**. To use such IP sets, a permanent direct rule is required to reference the set, and a custom service must be added to create these IP sets. This service needs to be started before **firewalld** starts, otherwise **firewalld** is not able to add the direct rules using these sets. You can add permanent direct rules with the **/etc/firewalld/direct.xml** file.

46.12. PRIORITIZING RICH RULES

By default, rich rules are organized based on their rule action. For example, **deny** rules have precedence over **allow** rules. The **priority** parameter in rich rules provides administrators fine-grained control over rich rules and their execution order.

46.12.1. How the priority parameter organizes rules into different chains

You can set the **priority** parameter in a rich rule to any number between **-32768** and **32767**, and lower values have higher precedence.

The **firewalld** service organizes rules based on their priority value into different chains:

- Priority lower than 0: the rule is redirected into a chain with the **_pre** suffix.
- Priority higher than 0: the rule is redirected into a chain with the **_post** suffix.
- Priority equals 0: based on the action, the rule is redirected into a chain with the **_log**, **_deny**, or **_allow** the action.

Inside these sub-chains, **firewalld** sorts the rules based on their priority value.

46.12.2. Setting the priority of a rich rule

The procedure describes an example of how to create a rich rule that uses the **priority** parameter to log all traffic that is not allowed or denied by other rules. You can use this rule to flag unexpected traffic.

Procedure

1. Add a rich rule with a very low precedence to log all traffic that has not been matched by other rules:

```
# firewall-cmd --add-rich-rule='rule priority=32767 log prefix="UNEXPECTED: " limit
value="5/m"'
```

The command additionally limits the number of log entries to **5** per minute.

2. Optionally, display the **nftables** rule that the command in the previous step created:

```
# nft list chain inet firewalld filter_IN_public_post
table inet firewalld {
  chain filter_IN_public_post {
    log prefix "UNEXPECTED: " limit rate 5/minute
  }
}
```

46.13. CONFIGURING FIREWALL LOCKDOWN

Local applications or services are able to change the firewall configuration if they are running as **root** (for example, **libvirt**). With this feature, the administrator can lock the firewall configuration so that either no applications or only applications that are added to the lockdown allow list are able to request firewall changes. The lockdown settings default to disabled. If enabled, the user can be sure that there are no unwanted configuration changes made to the firewall by local applications or services.

46.13.1. Configuring lockdown using CLI

This procedure describes how to enable or disable lockdown using the command line.

- To query whether lockdown is enabled, use the following command as **root**:

```
# firewall-cmd --query-lockdown
```

The command prints **yes** with exit status **0** if lockdown is enabled. It prints **no** with exit status **1** otherwise.

- To enable lockdown, enter the following command as **root**:

```
# firewall-cmd --lockdown-on
```

- To disable lockdown, use the following command as **root**:

```
# firewall-cmd --lockdown-off
```

46.13.2. Configuring lockdown allowlist options using CLI

The lockdown allowlist can contain commands, security contexts, users and user IDs. If a command entry on the allowlist ends with an asterisk "*", then all command lines starting with that command will match. If the "*" is not there then the absolute command including arguments must match.

- The context is the security (SELinux) context of a running application or service. To get the context of a running application use the following command:

```
$ ps -e --context
```

That command returns all running applications. Pipe the output through the **grep** tool to get the application of interest. For example:

```
$ ps -e --context | grep example_program
```

- To list all command lines that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-commands
```

- To add a command *command* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
```

- To remove a command *command* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-command='/usr/bin/python3 -Es  
/usr/bin/command'
```

- To query whether the command *command* is in the allowlist, enter the following command as **root**:

```
# firewall-cmd --query-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
```

The command prints **yes** with exit status **0** if true. It prints **no** with exit status **1** otherwise.

- To list all security contexts that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-contexts
```

- To add a context *context* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-context=context
```

- To remove a context *context* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-context=context
```

- To query whether the context *context* is in the allowlist, enter the following command as **root**:

```
# firewall-cmd --query-lockdown-whitelist-context=context
```

Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user IDs that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-uids
```

- To add a user ID *uid* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-uid=uid
```

- To remove a user ID *uid* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-uid=uid
```

- To query whether the user ID *uid* is in the allowlist, enter the following command:

```
$ firewall-cmd --query-lockdown-whitelist-uid=uid
```

Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user names that are in the allowlist, enter the following command as **root**:

```
# firewall-cmd --list-lockdown-whitelist-users
```

- To add a user name *user* to the allowlist, enter the following command as **root**:

```
# firewall-cmd --add-lockdown-whitelist-user=user
```

- To remove a user name *user* from the allowlist, enter the following command as **root**:

```
# firewall-cmd --remove-lockdown-whitelist-user=user
```

- To query whether the user name *user* is in the allowlist, enter the following command:

```
$ firewall-cmd --query-lockdown-whitelist-user=user
```

Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

46.13.3. Configuring lockdown allowlist options using configuration files

The default allowlist configuration file contains the **NetworkManager** context and the default context of **libvirt**. The user ID 0 is also on the list.

+ The allowlist configuration files are stored in the **/etc/firewalld/** directory.

```
<?xml version="1.0" encoding="utf-8"?>
<whitelist>
  <selinux context="system_u:system_r:NetworkManager_t:s0"/>
  <selinux context="system_u:system_r:virt_d_t:s0-s0:c0.c1023"/>
  <user id="0"/>
</whitelist>
```

Following is an example allowlist configuration file enabling all commands for the **firewall-cmd** utility, for a user called *user* whose user ID is **815**:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<whitelist>
  <command name="/usr/libexec/platform-python -s /bin/firewall-cmd"/>
  <selinux context="system_u:system_r:NetworkManager_t:s0"/>
  <user id="815"/>
  <user name="user"/>
</whitelist>
```

This example shows both **user id** and **user name**, but only one option is required. Python is the interpreter and is prepended to the command line. You can also use a specific command, for example:

```
/usr/bin/python3 /bin/firewall-cmd --lockdown-on
```

In that example, only the **--lockdown-on** command is allowed.

In Red Hat Enterprise Linux, all utilities are placed in the **/usr/bin/** directory and the **/bin/** directory is sym-linked to the **/usr/bin/** directory. In other words, although the path for **firewall-cmd** when entered as **root** might resolve to **/bin/firewall-cmd**, **/usr/bin/firewall-cmd** can now be used. All new scripts should use the new location. But be aware that if scripts that run as **root** are written to use the **/bin/firewall-cmd** path, then that command path must be added in the allowlist in addition to the **/usr/bin/firewall-cmd** path traditionally used only for non-**root** users.

The ***** at the end of the name attribute of a command means that all commands that start with this string match. If the ***** is not there then the absolute command including arguments must match.

46.14. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE

Intra-zone forwarding is a **firewalld** feature that enables traffic forwarding between interfaces or sources within a **firewalld** zone.

46.14.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT

When intra-zone forwarding is enabled, the traffic within a single **firewalld** zone can flow from one interface or source to another interface or source. The zone specifies the trust level of interfaces and sources. If the trust level is the same, communication between interfaces or sources is possible.

Note that, if you enable intra-zone forwarding in the default zone of **firewalld**, it applies only to the interfaces and sources added to the current default zone.

The **trusted** zone of **firewalld** uses a default target set to **ACCEPT**. This zone accepts all forwarded traffic, and intra-zone forwarding is not applicable for it.

As for other default target values, forwarded traffic is dropped by default, which applies to all standard zones except the trusted zone.

46.14.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same **firewalld** zone. For example, use this feature to forward traffic between an Ethernet network connected to **enp1s0** and a Wi-Fi network connected to **wlp0s20**.

Procedure

1. Enable packet forwarding in the kernel:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf

# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. Ensure that interfaces between which you want to enable intra-zone forwarding are not assigned to a zone different than the **internal** zone:

```
# firewall-cmd --get-active-zones
```

3. If the interface is currently assigned to a zone other than **internal**, reassign it:

```
# firewall-cmd --zone=internal --change-interface=interface_name --permanent
```

4. Add the **enp1s0** and **wlp0s20** interfaces to the **internal** zone:

```
# firewall-cmd --zone=internal --add-interface=enp1s0 --add-interface=wlp0s20
```

5. Enable intra-zone forwarding:

```
# firewall-cmd --zone=internal --add-forward
```

Verification

The following verification steps require that the **nmap-ncat** package is installed on both hosts.

1. Log in to a host that is in the same network as the **enp1s0** interface of the host you enabled zone forwarding on.
2. Start an echo service with **ncat** to test connectivity:

```
# ncat -e /usr/bin/cat -l 12345
```

3. Log in to a host that is in the same network as the **wlp0s20** interface.
4. Connect to the echo server running on the host that is in the same network as the **enp1s0**:

```
# ncat <other host> 12345
```

5. Type something and press **Enter**, and verify the text is sent back.

Additional resources

- **firewalld.zones(5)** man page

46.15. CONFIGURING FIREWALLD USING SYSTEM ROLES

You can use the **firewall** System Role to configure settings of the **firewalld** service on multiple clients at once. This solution:

- Provides an interface with efficient input settings.

- Keeps all intended **firewalld** parameters in one place.

After you run the **firewall** role on the control node, the System Role applies the **firewalld** parameters to the managed node immediately and makes them persistent across reboots.

46.15.1. Introduction to the firewall RHEL System Role

RHEL System Roles is a set of contents for the Ansible automation utility. This content together with the Ansible automation utility provides a consistent configuration interface to remotely manage multiple systems.

The **rhel-system-roles.firewall** role from the RHEL System Roles was introduced for automated configurations of the **firewalld** service. The **rhel-system-roles** package contains this System Role, and also the reference documentation.

To apply the **firewalld** parameters on one or more systems in an automated fashion, use the **firewall** System Role variable in a playbook. A playbook is a list of one or more plays that is written in the text-based YAML format.

You can use an inventory file to define a set of systems that you want Ansible to configure.

With the **firewall** role you can configure many different **firewalld** parameters, for example:

- Zones.
- The services for which packets should be allowed.
- Granting, rejection, or dropping of traffic access to ports.
- Forwarding of ports or port ranges for a zone.

Additional resources

- **README.md** and **README.html** files in the `/usr/share/doc/rhel-system-roles/firewall/` directory
- [Working with playbooks](#)
- [How to build your inventory](#)

46.15.2. Resetting the firewalld settings using the firewall RHEL System Role

With the **firewall** RHEL system role, you can reset the **firewalld** settings to their default state. If you add the **previous:replaced** parameter to the variable list, the System Role removes all existing user-defined settings and resets **firewalld** to the defaults. If you combine the **previous:replaced** parameter with other settings, the **firewall** role removes all existing settings before applying new ones.

Run this procedure on Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, you must have appropriate sudo permissions on the managed node.

- One or more managed nodes that you configure with the **firewall** RHEL System Role.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/reset-firewalld.yml** playbook with the following content:

```
- name: Reset firewalld example
  hosts: node.example.com
  tasks:
    - name: Reset firewalld
      include_role:
        name: rhel-system-roles.firewall
  vars:
    firewall:
      - previous: replaced
```

3. Run the playbook:
 - a. To connect as root user to the managed node:

```
# ansible-playbook -u root ~/reset-firewalld.yml
```

- b. To connect as a user to the managed node:

```
# ansible-playbook -u user_name --ask-become-pass ~/reset-firewalld.yml
```

The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the sudo password of the user defined in the **-u user_name** option.

If you do not specify the **-u user_name** option, **ansible-playbook** connects to the managed node as the user that is currently logged in to the control node.

Verification

- Run this command as **root** on the managed node to check all the zones:

```
# firewall-cmd --list-all-zones
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md**
- **ansible-playbook(1)**
- **firewalld(1)**

46.15.3. Forwarding incoming traffic from one local port to a different local port

With the **firewall** role you can remotely configure **firewalld** parameters with persisting effect on multiple managed hosts.

Perform this procedure on the Ansible control node.

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 the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Forward incoming traffic on port 8080 to 443
      include_role:
        name: rhel-system-roles.firewall

  vars:
    firewall:
      - { forward_port: 8080/tcp;443;, state: enabled, runtime: true, permanent: true }
```

2. Run the playbook:

```
# ansible-playbook ~/port_forwarding.yml
```

Verification

- On the managed host, display the **firewalld** settings:

```
# firewall-cmd --list-forward-ports
```

Additional resources

- [/usr/share/ansible/roles/rhel-system-roles.firewall/README.md](#)

46.15.4. Configuring ports using System Roles

You can use the RHEL **firewall** System Role to open or close ports in the local firewall for incoming traffic and make the new configuration persist across reboots. The example describes how to configure the default zone to permit incoming traffic for the HTTPS service.

Perform this procedure on the Ansible control node.

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 the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Allow incoming HTTPS traffic to the local host
      include_role:
        name: rhel-system-roles.firewall

  vars:
    firewall:
      - port: 443/tcp
        service: http
        state: enabled
        runtime: true
        permanent: true
```

The **permanent: true** option makes the new settings persistent across reboots.

2. Run the playbook:

```
# ansible-playbook ~/opening-a-port.yml
```

Verification

- On the managed node, verify that the **443/tcp** port associated with the **HTTPS** service is open:

```
# firewall-cmd --list-ports
443/tcp
```

Additional resources

- </usr/share/ansible/roles/rhel-system-roles.firewall/README.md>

46.15.5. Configuring a DMZ firewalld zone by using the firewalld RHEL System Role

As a system administrator, you can use the **firewall** System Role to configure a **dmz** zone on the **enp1s0** interface to permit **HTTPS** traffic to the zone. In this way, you enable external users to access your web servers.

Perform this procedure on the Ansible control node.

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 the them.
- The hosts or host groups on which you want run this playbook are listed in the Ansible inventory file.

Procedure

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

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
      include_role:
        name: rhel-system-roles.firewall

  vars:
    firewall:
      - zone: dmz
        interface: enp1s0
        service: https
        state: enabled
        runtime: true
        permanent: true
```


2. Run the playbook:

```
# ansible-playbook ~/configuring-a-dmz.yml
```

Verification

- On the managed node, view detailed information about the **dmz** zone:

```
# firewall-cmd --zone=dmz --list-all
dmz (active)
target: default
icmp-block-inversion: no
interfaces: enp1s0
sources:
services: https ssh
ports:
protocols:
forward: no
masquerade: no
```



```
forward-ports:
source-ports:
icmp-blocks:
```

Additional resources

- [/usr/share/ansible/roles/rhel-system-roles.firewall/README.md](#)

46.16. ADDITIONAL RESOURCES

- **firewalld(1)** man page
- **firewalld.conf(5)** man page
- **firewall-cmd(1)** man page
- **firewall-config(1)** man page
- **firewall-offline-cmd(1)** man page
- **firewalld.icmptype(5)** man page
- **firewalld.ipset(5)** man page
- **firewalld.service(5)** man page
- **firewalld.zone(5)** man page
- **firewalld.direct(5)** man page
- **firewalld.lockdown-whitelist(5)**
- **firewalld.richlanguage(5)**
- **firewalld.zones(5)** man page
- **firewalld.dbus(5)** man page

CHAPTER 47. GETTING STARTED WITH NFTABLES

The **nftables** framework classifies packets and it is the successor to the **iptables**, **ip6tables**, **arptables**, **ebtables**, and **ipset** utilities. It offers numerous improvements in convenience, features, and performance over previous packet-filtering tools, most notably:

- Built-in lookup tables instead of linear processing
- A single framework for both the **IPv4** and **IPv6** protocols
- All rules applied atomically instead of fetching, updating, and storing a complete rule set
- Support for debugging and tracing in the rule set (**nftrace**) and monitoring trace events (in the **nft** tool)
- More consistent and compact syntax, no protocol-specific extensions
- A Netlink API for third-party applications

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **nft** utility replaces all tools from the previous packet-filtering frameworks. You can use the **libnftnl** library for low-level interaction with **nftables** Netlink API through the **libmnl** library.

To display the effect of rule set changes, use the **nft list ruleset** command. Because these utilities add tables, chains, rules, sets, and other objects to the **nftables** rule set, be aware that **nftables** rule-set operations, such as the **nft flush ruleset** command, might affect rule sets installed using the **iptables** command.

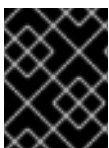
47.1. MIGRATING FROM IPTABLES TO NFTABLES

If your firewall configuration still uses **iptables** rules, you can migrate your **iptables** rules to **nftables**.

47.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance-critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.



IMPORTANT

To prevent the different firewall services from influencing each other, run only one of them on a RHEL host, and disable the other services.

47.1.2. Converting iptables and ip6tables rule sets to nftables

Use the **iptables-restore-translate** and **ip6tables-restore-translate** utilities to translate **iptables** and **ip6tables** rule sets to **nftables**.

Prerequisites

- The **nftables** and **iptables** packages are installed.
- The system has **iptables** and **ip6tables** rules configured.

Procedure

1. Write the **iptables** and **ip6tables** rules to a file:

```
# iptables-save >/root/iptables.dump
# ip6tables-save >/root/ip6tables.dump
```

2. Convert the dump files to **nftables** instructions:

```
# iptables-restore-translate -f /root/iptables.dump > /etc/nftables/ruleset-migrated-
from-iptables.nft
# ip6tables-restore-translate -f /root/ip6tables.dump > /etc/nftables/ruleset-migrated-
from-ip6tables.nft
```

3. Review and, if needed, manually update the generated **nftables** rules.
4. To enable the **nftables** service to load the generated files, add the following to the **/etc/sysconfig/nftables.conf** file:

```
include "/etc/nftables/ruleset-migrated-from-iptables.nft"
include "/etc/nftables/ruleset-migrated-from-ip6tables.nft"
```

5. Stop and disable the **iptables** service:

```
# systemctl disable --now iptables
```

If you used a custom script to load the **iptables** rules, ensure that the script no longer starts automatically and reboot to flush all tables.

6. Enable and start the **nftables** service:

```
# systemctl enable --now nftables
```

Verification

- Display the **nftables** rule set:

```
# nft list ruleset
```

Additional resources

- [Automatically loading nftables rules when the system boots](#)

47.1.3. Converting single iptables and ip6tables rules to nftables

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** utilities to convert an **iptables** or **ip6tables** rule into the equivalent one for **nftables**.

Prerequisites

- The **nftables** package is installed.

Procedure

- Use the **iptables-translate** or **ip6tables-translate** utility instead of **iptables** or **ip6tables** to display the corresponding **nftables** rule, for example:

```
# iptables-translate -A INPUT -s 192.0.2.0/24 -j ACCEPT
nft add rule ip filter INPUT ip saddr 192.0.2.0/24 counter accept
```

Note that some extensions lack translation support. In these cases, the utility prints the untranslated rule prefixed with the **#** sign, for example:

```
# iptables-translate -A INPUT -j CHECKSUM --checksum-fill
nft # -A INPUT -j CHECKSUM --checksum-fill
```

Additional resources

- **iptables-translate --help**

47.1.4. Comparison of common iptables and nftables commands

The following is a comparison of common **iptables** and **nftables** commands:

- Listing all rules:

iptables	nftables
iptables-save	nft list ruleset

- Listing a certain table and chain:

iptables	nftables
iptables -L	nft list table ip filter
iptables -L INPUT	nft list chain ip filter INPUT
iptables -t nat -L PREROUTING	nft list chain ip nat PREROUTING

The **nft** command does not pre-create tables and chains. They exist only if a user created them manually.

Example: Listing rules generated by firewalld

```
# nft list table inet firewallld
# nft list table ip firewallld
# nft list table ip6 firewallld
```

47.1.5. Additional resources

- [iptables: The two variants and their relationship with nftables](#)

47.2. WRITING AND EXECUTING NFTABLES SCRIPTS

The major benefit of using the **nftables** framework is that the execution of scripts is atomic. This means that the system either applies the whole script or prevents the execution if an error occurs. This guarantees that the firewall is always in a consistent state.

Additionally, with the **nftables** script environment, you can:

- Add comments
- Define variables
- Include other rule-set files

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates ***.nft** scripts in the **/etc/nftables/** directory. These scripts contain commands that create tables and empty chains for different purposes.

47.2.1. Supported nftables script formats

You can write scripts in the **nftables** scripting environment in the following formats:

- The same format as the **nft list ruleset** command displays the rule set:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

table inet example_table {
  chain example_chain {
    # Chain for incoming packets that drops all packets that
    # are not explicitly allowed by any rule in this chain
    type filter hook input priority 0; policy drop;

    # Accept connections to port 22 (ssh)
    tcp dport ssh accept
  }
}
```

- The same syntax as for **nft** commands:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset
```

```
# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets
# that are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0 ; policy drop ; }

# Add a rule that accepts connections to port 22 (ssh)
add rule inet example_table example_chain tcp dport ssh accept
```

47.2.2. Running nftables scripts

You can run an **nftables** script either by passing it to the **nft** utility or by executing the script directly.

Procedure

- To run an **nftables** script by passing it to the **nft** utility, enter:

```
# nft -f /etc/nftables/<example_firewall_script>.nft
```

- To run an **nftables** script directly:
 - For the single time that you perform this:
 - Ensure that the script starts with the following shebang sequence:

```
#!/usr/sbin/nft -f
```



IMPORTANT

If you omit the **-f** parameter, the **nft** utility does not read the script and displays: **Error: syntax error, unexpected newline, expecting string.**

- Optional: Set the owner of the script to **root**:

```
# chown root /etc/nftables/<example_firewall_script>.nft
```

- Make the script executable for the owner:

```
# chmod u+x /etc/nftables/<example_firewall_script>.nft
```

- Run the script:

```
# /etc/nftables/<example_firewall_script>.nft
```

If no output is displayed, the system executed the script successfully.



IMPORTANT

Even if **nft** executes the script successfully, incorrectly placed rules, missing parameters, or other problems in the script can cause that the firewall behaves not as expected.

Additional resources

- **chown(1)** man page
- **chmod(1)** man page
- [Automatically loading nftables rules when the system boots](#)

47.2.3. Using comments in nftables scripts

The **nftables** scripting environment interprets everything to the right of a **#** character to the end of a line as a comment.

Example 47.1. Comments in an nftables script

Comments can start at the beginning of a line, or next to a command:

```
...
# Flush the rule set
flush ruleset

add table inet example_table # Create a table
...
```

47.2.4. Using variables in nftables script

To define a variable in an **nftables** script, use the **define** keyword. You can store single values and anonymous sets in a variable. For more complex scenarios, use sets or verdict maps.

Variables with a single value

The following example defines a variable named **INET_DEV** with the value **enp1s0**:

```
define INET_DEV = enp1s0
```

You can use the variable in the script by entering the **\$** sign followed by the variable name:

```
...
add rule inet example_table example_chain iifname $INET_DEV tcp dport ssh accept
...
```

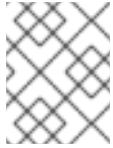
Variables that contain an anonymous set

The following example defines a variable that contains an anonymous set:

```
define DNS_SERVERS = { 192.0.2.1, 192.0.2.2 }
```

You can use the variable in the script by writing the **\$** sign followed by the variable name:

```
add rule inet example_table example_chain ip daddr $DNS_SERVERS accept
```

**NOTE**

Curly braces have special semantics when you use them in a rule because they indicate that the variable represents a set.

Additional resources

- [Using sets in nftables commands](#)
- [Using verdict maps in nftables commands](#)

47.2.5. Including files in nftables scripts

In the **nftables** scripting environment, you can include other scripts by using the **include** statement.

If you specify only a file name without an absolute or relative path, **nftables** includes files from the default search path, which is set to **/etc** on Red Hat Enterprise Linux.

Example 47.2. Including files from the default search directory

To include a file from the default search directory:

```
include "example.nft"
```

Example 47.3. Including all *.nft files from a directory

To include all files ending with ***.nft** that are stored in the **/etc/nftables/rulesets/** directory:

```
include "/etc/nftables/rulesets/*.nft"
```

Note that the **include** statement does not match files beginning with a dot.

Additional resources

- The **Include files** section in the **nft(8)** man page

47.2.6. Automatically loading nftables rules when the system boots

The **nftables** systemd service loads firewall scripts that are included in the **/etc/sysconfig/nftables.conf** file.

Prerequisites

- The **nftables** scripts are stored in the **/etc/nftables/** directory.

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.

- If you modified the ***.nft** scripts that were created in **/etc/nftables/** with the installation of the **nftables** package, uncomment the **include** statement for these scripts.
- If you wrote new scripts, add **include** statements to include these scripts. For example, to load the **/etc/nftables/example.nft** script when the **nftables** service starts, add:

```
include "/etc/nftables/example.nft"
```

2. Optional: Start the **nftables** service to load the firewall rules without rebooting the system:

```
# systemctl start nftables
```

3. Enable the **nftables** service.

```
# systemctl enable nftables
```

Additional resources

- [Supported nftables script formats](#)

47.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES

You can display **nftables** rule sets and manage them.

47.3.1. Basics of nftables tables

A table in **nftables** is a namespace that contains a collection of chains, rules, sets, and other objects.

Each table must have an address family assigned. The address family defines the packet types that this table processes. You can set one of the following address families when you create a table:

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.
- **ip6**: Matches only IPv6 packets.
- **inet**: Matches both IPv4 and IPv6 packets.
- **arp**: Matches IPv4 address resolution protocol (ARP) packets.
- **bridge**: Matches packets that pass through a bridge device.
- **netdev**: Matches packets from ingress.

If you want to add a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {  
}
```

- In shell scripts, use:

```
nft add table <table_address_family> <table_name>
```

-

47.3.2. Basics of nftables chains

Tables consist of chains which in turn are containers for rules. The following two rule types exists:

- **Base chain:** You can use base chains as an entry point for packets from the networking stack.
- **Regular chain:** You can use regular chains as a **jump** target to better organize rules.

If you want to add a base chain to a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {  
  chain <chain_name> {  
    type <type> hook <hook> priority <priority>  
    policy <policy> ;  
  }  
}
```

- In shell scripts, use:

```
nft add chain <table_address_family> <table_name> <chain_name> { type <type> hook  
<hook> priority <priority> \; policy <policy> \; }
```

To avoid that the shell interprets the semicolons as the end of the command, place the \ escape character in front of the semicolons.

Both examples create **base chains**. To create a **regular chain**, do not set any parameters in the curly brackets.

Chain types

The following are the chain types and an overview with which address families and hooks you can use them:

Type	Address families	Hooks	Description
filter	all	all	Standard chain type
nat	ip, ip6, inet	prerouting, input, output, postrouting	Chains of this type perform native address translation based on connection tracking entries. Only the first packet traverses this chain type.
route	ip, ip6	output	Accepted packets that traverse this chain type cause a new route lookup if relevant parts of the IP header have changed.

Chain priorities

The priority parameter specifies the order in which packets traverse chains with the same hook value. You can set this parameter to an integer value or use a standard priority name.

The following matrix is an overview of the standard priority names and their numeric values, and with which address families and hooks you can use them:

Textual value	Numeric value	Address families	Hooks
raw	-300	ip, ip6, inet	all
mangle	-150	ip, ip6, inet	all
dstnat	-100	ip, ip6, inet	prerouting
	-300	bridge	prerouting
filter	0	ip, ip6, inet, arp, netdev	all
	-200	bridge	all
security	50	ip, ip6, inet	all
srcnat	100	ip, ip6, inet	postrouting
	300	bridge	postrouting
out	100	bridge	output

Chain policies

The chain policy defines whether **nftables** should accept or drop packets if rules in this chain do not specify any action. You can set one of the following policies in a chain:

- **accept** (default)
- **drop**

47.3.3. Basics of nftables rules

Rules define actions to perform on packets that pass a chain that contains this rule. If the rule also contains matching expressions, **nftables** performs the actions only if all previous expressions apply.

If you want to add a rule to a chain, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
    chain <chain_name> {
        type <type> hook <hook> priority <priority> ; policy <policy> ;
        <rule>
    }
}
```

- In shell scripts, use:

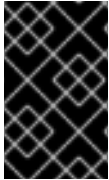
■

```
nft add rule <table_address_family> <table_name> <chain_name> <rule>
```

This shell command appends the new rule at the end of the chain. If you prefer to add a rule at the beginning of the chain, use the **nft insert** command instead of **nft add**.

47.3.4. Managing tables, chains, and rules using nft commands

To manage an **nftables** firewall on the command line or in shell scripts, use the **nft** utility.



IMPORTANT

The commands in this procedure do not represent a typical workflow and are not optimized. This procedure only demonstrates how to use **nft** commands to manage tables, chains, and rules in general.

Procedure

1. Create a table named **nftables_svc** with the **inet** address family so that the table can process both IPv4 and IPv6 packets:

```
# nft add table inet nftables_svc
```

2. Add a base chain named **INPUT**, that processes incoming network traffic, to the **inet nftables_svc** table:

```
# nft add chain inet nftables_svc INPUT { type filter hook input priority filter; policy accept; }
```

To avoid that the shell interprets the semicolons as the end of the command, escape the semicolons using the **** character.

3. Add rules to the **INPUT** chain. For example, allow incoming TCP traffic on port 22 and 443, and, as the last rule of the **INPUT** chain, reject other incoming traffic with an Internet Control Message Protocol (ICMP) port unreachable message:

```
# nft add rule inet nftables_svc INPUT tcp dport 22 accept
# nft add rule inet nftables_svc INPUT tcp dport 443 accept
# nft add rule inet nftables_svc INPUT reject with icmpx type port-unreachable
```

If you enter the **nft add rule** commands as shown, **nft** adds the rules in the same order to the chain as you run the commands.

4. Display the current rule set including handles:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

5. Insert a rule before the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 636, enter:

```
# nft insert rule inet nftables_svc INPUT position 3 tcp dport 636 accept
```

6. Append a rule after the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 80, enter:

```
# nft add rule inet nftables_svc INPUT position 3 tcp dport 80 accept
```

7. Display the rule set again with handles. Verify that the later added rules have been added to the specified positions:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    tcp dport 80 accept # handle 6
    reject # handle 4
  }
}
```

8. Remove the rule with handle 6:

```
# nft delete rule inet nftables_svc INPUT handle 6
```

To remove a rule, you must specify the handle.

9. Display the rule set, and verify that the removed rule is no longer present:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

10. Remove all remaining rules from the **INPUT** chain:

```
# nft flush chain inet nftables_svc INPUT
```

11. Display the rule set, and verify that the **INPUT** chain is empty:

```
# nft list table inet nftables_svc
table inet nftables_svc {
  chain INPUT {
```

```
type filter hook input priority filter; policy accept
}
}
```

12. Delete the **INPUT** chain:

```
# nft delete chain inet nftables_svc INPUT
```

You can also use this command to delete chains that still contain rules.

13. Display the rule set, and verify that the **INPUT** chain has been deleted:

```
# nft list table inet nftables_svc
table inet nftables_svc {
}
```

14. Delete the **nftables_svc** table:

```
# nft delete table inet nftables_svc
```

You can also use this command to delete tables that still contain chains.



NOTE

To delete the entire rule set, use the **nft flush ruleset** command instead of manually deleting all rules, chains, and tables in separate commands.

Additional resources

nft(8) man page

47.4. CONFIGURING NAT USING NFTABLES

With **nftables**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect



IMPORTANT

You can only use real interface names in **iifname** and **oifname** parameters, and alternative names (**altname**) are not supported.

47.4.1. NAT types

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.

Masquerading and SNAT are very similar to one another. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

47.4.2. Configuring masquerading using nftables

Masquerading enables a router to dynamically change the source IP of packets sent through an interface to the IP address of the interface. This means that if the interface gets a new IP assigned, **nftables** automatically uses the new IP when replacing the source IP.

Replace the source IP of packets leaving the host through the **ens3** interface to the IP set on **ens3**.

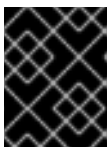
Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **prerouting** chain, the **nftables** framework requires this chain to match incoming packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that matches outgoing packets on the **ens3** interface:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

47.4.3. Configuring source NAT using nftables

On a router, Source NAT (SNAT) enables you to change the IP of packets sent through an interface to a specific IP address.

The following procedure describes how to replace the source IP of packets leaving the router through the **ens3** interface to **192.0.2.1**.

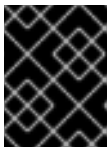
Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that replaces the source IP of outgoing packets through **ens3** with **192.0.2.1**:

```
# nft add rule nat postrouting oifname "ens3" snat to 192.0.2.1
```

Additional resources

- [Forwarding incoming packets on a specific local port to a different host](#)

47.4.4. Configuring destination NAT using nftables

Destination NAT enables you to redirect traffic on a router to a host that is not directly accessible from the Internet.

The following procedure describes how to redirect incoming traffic sent to port **80** and **443** on the router to a web server with the IP address **192.0.2.1**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }  
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic to port **80** and **443** on the **ens3** interface of the router to the web server with the IP address **192.0.2.1**:

```
# nft add rule nat prerouting iifname ens3 tcp dport { 80, 443 } dnat to 192.0.2.1
```

4. Depending on your environment, add either a SNAT or masquerading rule to change the source address for packets returning from the web server to the sender:

- a. If the **ens3** interface uses a dynamic IP addresses, add a masquerading rule:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

- b. If the **ens3** interface uses a static IP address, add a SNAT rule. For example, if the **ens3** uses the **198.51.100.1** IP address:

```
# nft add rule nat postrouting oifname "ens3" snat to 198.51.100.1
```

5. 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
```

Additional resources

- [NAT types](#)

47.4.5. Configuring a redirect using nftables

The **redirect** feature is a special case of destination network address translation (DNAT) that redirects packets to the local machine depending on the chain hook.

The following procedure describes how to redirect incoming and forwarded traffic sent to port **22** of the local host to port **2222**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** chain to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
```

Note that you must pass the `--` option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic on port **22** to port **2222**:

```
# nft add rule nat prerouting tcp dport 22 redirect to 2222
```

Additional resources

- [NAT types](#)

47.5. USING SETS IN NFTABLES COMMANDS

The **nftables** framework natively supports sets. You can use sets, for example, if a rule should match multiple IP addresses, port numbers, interfaces, or any other match criteria.

47.5.1. Using anonymous sets in nftables

An anonymous set contains comma-separated values enclosed in curly brackets, such as `{ 22, 80, 443 }`, that you use directly in a rule. You can use anonymous sets also for IP addresses and any other match criteria.

The drawback of anonymous sets is that if you want to change the set, you must replace the rule. For a dynamic solution, use named sets as described in [Using named sets in nftables](#).

Prerequisites

- The **example_chain** chain and the **example_table** table in the **inet** family exists.

Procedure

1. For example, to add a rule to **example_chain** in **example_table** that allows incoming traffic to port **22**, **80**, and **443**:

```
# nft add rule inet example_table example_chain tcp dport { 22, 80, 443 } accept
```

2. Optional: Display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport { ssh, http, https } accept
  }
}
```

47.5.2. Using named sets in nftables

The **nftables** framework supports mutable named sets. A named set is a list or range of elements that you can use in multiple rules within a table. Another benefit over anonymous sets is that you can update a named set without replacing the rules that use the set.

When you create a named set, you must specify the type of elements the set contains. You can set the following types:

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as **192.0.2.1** or **192.0.2.0/24**.
- **ipv6_addr** for a set that contains IPv6 addresses or ranges, such as **2001:db8:1::1** or **2001:db8:1::1/64**.
- **ether_addr** for a set that contains a list of media access control (MAC) addresses, such as **52:54:00:6b:66:42**.
- **inet_proto** for a set that contains a list of Internet protocol types, such as **tcp**.
- **inet_service** for a set that contains a list of Internet services, such as **ssh**.
- **mark** for a set that contains a list of packet marks. Packet marks can be any positive 32-bit integer value (**0** to **2147483647**).

Prerequisites

- The **example_chain** chain and the **example_table** table exists.

Procedure

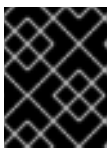
1. Create an empty set. The following examples create a set for IPv4 addresses:

- To create a set that can store multiple individual IPv4 addresses:

```
# nft add set inet example_table example_set { type ipv4_addr \;
```

- To create a set that can store IPv4 address ranges:

```
# nft add set inet example_table example_set { type ipv4_addr \; flags interval \;
```



IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

2. Optional: Create rules that use the set. For example, the following command adds a rule to the **example_chain** in the **example_table** that will drop all packets from IPv4 addresses in **example_set**.

```
# nft add rule inet example_table example_chain ip saddr @example_set drop
```

Because **example_set** is still empty, the rule has currently no effect.

3. Add IPv4 addresses to **example_set**:

- If you create a set that stores individual IPv4 addresses, enter:

```
# nft add element inet example_table example_set { 192.0.2.1, 192.0.2.2 }
```

- If you create a set that stores IPv4 ranges, enter:

—

```
# nft add element inet example_table example_set { 192.0.2.0-192.0.2.255 }
```

When you specify an IP address range, you can alternatively use the Classless Inter-Domain Routing (CIDR) notation, such as **192.0.2.0/24** in the above example.

47.5.3. Additional resources

- The **Sets** section in the **nft(8)** man page

47.6. USING VERDICT MAPS IN NFTABLES COMMANDS

Verdict maps, which are also known as dictionaries, enable **nft** to perform an action based on packet information by mapping match criteria to an action.

47.6.1. Using anonymous maps in nftables

An anonymous map is a **{ match_criteria : action }** statement that you use directly in a rule. The statement can contain multiple comma-separated mappings.

The drawback of an anonymous map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named maps as described in [Using named maps in nftables](#).

The example describes how to use an anonymous map to route both TCP and UDP packets of the IPv4 and IPv6 protocol to different chains to count incoming TCP and UDP packets separately.

Procedure

1. Create a new table:

```
# nft add table inet example_table
```

2. Create the **tcp_packets** chain in **example_table**:

```
# nft add chain inet example_table tcp_packets
```

3. Add a rule to **tcp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table tcp_packets counter
```

4. Create the **udp_packets** chain in **example_table**

```
# nft add chain inet example_table udp_packets
```

5. Add a rule to **udp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table udp_packets counter
```

6. Create a chain for incoming traffic. For example, to create a chain named **incoming_traffic** in **example_table** that filters incoming traffic:

```
# nft add chain inet example_table incoming_traffic { type filter hook input priority 0 \; }
```

7. Add a rule with an anonymous map to **incoming_traffic**:

```
# nft add rule inet example_table incoming_traffic ip protocol vmap { tcp : jump tcp_packets,
udp : jump udp_packets }
```

The anonymous map distinguishes the packets and sends them to the different counter chains based on their protocol.

8. To list the traffic counters, display **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain tcp_packets {
    counter packets 36379 bytes 2103816
  }

  chain udp_packets {
    counter packets 10 bytes 1559
  }

  chain incoming_traffic {
    type filter hook input priority filter; policy accept;
    ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }
  }
}
```

The counters in the **tcp_packets** and **udp_packets** chain display both the number of received packets and bytes.

47.6.2. Using named maps in nftables

The **nftables** framework supports named maps. You can use these maps in multiple rules within a table. Another benefit over anonymous maps is that you can update a named map without replacing the rules that use it.

When you create a named map, you must specify the type of elements:

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as **192.0.2.1**.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as **2001:db8:1::1**.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as **52:54:00:6b:66:42**.
- **inet_proto** for a map whose match part contains an Internet protocol type, such as **tcp**.
- **inet_service** for a map whose match part contains an Internet services name port number, such as **ssh** or **22**.
- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (**0** to **2147483647**).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.

- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

The example describes how to allow or drop incoming packets based on their source IP address. Using a named map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map. The procedure also describes how to add and remove entries from the map.

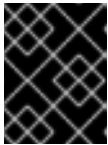
Procedure

1. Create a table. For example, to create a table named **example_table** that processes IPv4 packets:

```
# nft add table ip example_table
```

2. Create a chain. For example, to create a chain named **example_chain** in **example_table**:

```
# nft add chain ip example_table example_chain { type filter hook input priority 0 \; }
```



IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

3. Create an empty map. For example, to create a map for IPv4 addresses:

```
# nft add map ip example_table example_map { type ipv4_addr : verdict \; }
```

4. Create rules that use the map. For example, the following command adds a rule to **example_chain** in **example_table** that applies actions to IPv4 addresses which are both defined in **example_map**:

```
# nft add rule example_table example_chain ip saddr vmap @example_map
```

5. Add IPv4 addresses and corresponding actions to **example_map**:

```
# nft add element ip example_table example_map { 192.0.2.1 : accept, 192.0.2.2 : drop }
```

This example defines the mappings of IPv4 addresses to actions. In combination with the rule created above, the firewall accepts packet from **192.0.2.1** and drops packets from **192.0.2.2**.

6. Optional: Enhance the map by adding another IP address and action statement:

```
# nft add element ip example_table example_map { 192.0.2.3 : accept }
```

7. Optional: Remove an entry from the map:

```
# nft delete element ip example_table example_map { 192.0.2.1 }
```

8. Optional: Display the rule set:

```
# nft list ruleset
```

```

table ip example_table {
    map example_map {
        type ipv4_addr : verdict
        elements = { 192.0.2.2 : drop, 192.0.2.3 : accept }
    }

    chain example_chain {
        type filter hook input priority filter; policy accept;
        ip saddr vmap @example_map
    }
}

```

47.6.3. Additional resources

- The **Maps** section in the **nft(8)** man page

47.7. EXAMPLE: PROTECTING A LAN AND DMZ USING AN NFTABLES SCRIPT

Use the **nftables** framework on a RHEL router to write and install a firewall script that protects the network clients in an internal LAN and a web server in a DMZ from unauthorized access from the Internet and from other networks.



IMPORTANT

This example is only for demonstration purposes and describes a scenario with specific requirements.

Firewall scripts highly depend on the network infrastructure and security requirements. Use this example to learn the concepts of **nftables** firewalls when you write scripts for your own environment.

47.7.1. Network conditions

The network in this example has the following conditions:

- The router is connected to the following networks:
 - The Internet through interface **enp1s0**
 - The internal LAN through interface **enp7s0**
 - The DMZ through **enp8s0**
- The Internet interface of the router has both a static IPv4 address (**203.0.113.1**) and IPv6 address (**2001:db8:a::1**) assigned.
- The clients in the internal LAN use only private IPv4 addresses from the range **10.0.0.0/24**. Consequently, traffic from the LAN to the Internet requires source network address translation (SNAT).
- The administrator PCs in the internal LAN use the IP addresses **10.0.0.100** and **10.0.0.200**.
- The DMZ uses public IP addresses from the ranges **198.51.100.0/24** and **2001:db8:b::/56**.

- The web server in the DMZ uses the IP addresses **198.51.100.5** and **2001:db8:b::5**.
- The router acts as a caching DNS server for hosts in the LAN and DMZ.

47.7.2. Security requirements to the firewall script

The following are the requirements to the **nftables** firewall in the example network:

- The router must be able to:
 - Recursively resolve DNS queries.
 - Perform all connections on the loopback interface.
- Clients in the internal LAN must be able to:
 - Query the caching DNS server running on the router.
 - Access the HTTPS server in the DMZ.
 - Access any HTTPS server on the Internet.
- The PCs of the administrators must be able to access the router and every server in the DMZ using SSH.
- The web server in the DMZ must be able to:
 - Query the caching DNS server running on the router.
 - Access HTTPS servers on the Internet to download updates.
- Hosts on the Internet must be able to:
 - Access the HTTPS servers in the DMZ.
- Additionally, the following security requirements exists:
 - Connection attempts that are not explicitly allowed should be dropped.
 - Dropped packets should be logged.

47.7.3. Configuring logging of dropped packets to a file

By default, **systemd** logs kernel messages, such as for dropped packets, to the journal. Additionally, you can configure the **rsyslog** service to log such entries to a separate file. To ensure that the log file does not grow infinitely, configure a rotation policy.

Prerequisites

- The **rsyslog** package is installed.
- The **rsyslog** service is running.

Procedure

1. Create the **/etc/rsyslog.d/nftables.conf** file with the following content:

```
:msg, startswith, "nft drop" -/var/log/nftables.log  
& stop
```

Using this configuration, the **rsyslog** service logs dropped packets to the **/var/log/nftables.log** file instead of **/var/log/messages**.

- Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

- Create the **/etc/logrotate.d/nftables** file with the following content to rotate **/var/log/nftables.log** if the size exceeds 10 MB:

```
/var/log/nftables.log {  
  size +10M  
  maxage 30  
  sharedscripts  
  postrotate  
    /usr/bin/systemctl kill -s HUP rsyslog.service >/dev/null 2>&1 || true  
  endscript  
}
```

The **maxage 30** setting defines that **logrotate** removes rotated logs older than 30 days during the next rotation operation.

Additional resources

- **rsyslog.conf(5)** man page
- **logrotate(8)** man page

47.7.4. Writing and activating the nftables script

This example describes an **nftables** firewall script that runs on a RHEL router and protects the clients in an internal LAN and a web server in a DMZ. For details about the network and the requirements for the firewall used in the example, see [Network conditions](#) and [Security requirements to the firewall script](#).



WARNING

The **nftables** firewall script in this section is only for demonstration purposes. Do not use it without adopting it to your environments and security requirements.

Prerequisites

- The network is configured as described in [Network conditions](#).

Procedure

- Create the **/etc/nftables/firewall.nft** script with the following content:

```

# Remove all rules
flush ruleset

# Table for both IPv4 and IPv6 rules
table inet nftables_svc {

    # Define variables for the interface name
    define INET_DEV = enp1s0
    define LAN_DEV = enp7s0
    define DMZ_DEV = enp8s0

    # Set with the IPv4 addresses of admin PCs
    set admin_pc_ipv4 {
        type ipv4_addr
        elements = { 10.0.0.100, 10.0.0.200 }
    }

    # Chain for incoming traffic. Default policy: drop
    chain INPUT {
        type filter hook input priority filter
        policy drop

        # Accept packets in established and related state, drop invalid packets
        ct state vmap { established:accept, related:accept, invalid:drop }

        # Accept incoming traffic on loopback interface
        iifname lo accept

        # Allow request from LAN and DMZ to local DNS server
        iifname { $LAN_DEV, $DMZ_DEV } meta l4proto { tcp, udp } th dport 53 accept

        # Allow admins PCs to access the router using SSH
        iifname $LAN_DEV ip saddr @admin_pc_ipv4 tcp dport 22 accept

        # Last action: Log blocked packets
        # (packets that were not accepted in previous rules in this chain)
        log prefix "nft drop IN : "
    }

    # Chain for outgoing traffic. Default policy: drop
    chain OUTPUT {
        type filter hook output priority filter
        policy drop

        # Accept packets in established and related state, drop invalid packets
        ct state vmap { established:accept, related:accept, invalid:drop }

        # Accept outgoing traffic on loopback interface
        oifname lo accept

        # Allow local DNS server to recursively resolve queries
        oifname $INET_DEV meta l4proto { tcp, udp } th dport 53 accept
    }
}

```



```

# Last action: Log blocked packets
log prefix "nft drop OUT: "
}

# Chain for forwarding traffic. Default policy: drop
chain FORWARD {
    type filter hook forward priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # IPv4 access from LAN and Internet to the HTTPS server in the DMZ
    iifname { $LAN_DEV, $INET_DEV } oifname $DMZ_DEV ip daddr 198.51.100.5 tcp dport
    443 accept

    # IPv6 access from Internet to the HTTPS server in the DMZ
    iifname $INET_DEV oifname $DMZ_DEV ip6 daddr 2001:db8:b::5 tcp dport 443 accept

    # Access from LAN and DMZ to HTTPS servers on the Internet
    iifname { $LAN_DEV, $DMZ_DEV } oifname $INET_DEV tcp dport 443 accept

    # Last action: Log blocked packets
    log prefix "nft drop FWD: "
}

# Postrouting chain to handle SNAT
chain postrouting {
    type nat hook postrouting priority srcnat; policy accept;

    # SNAT for IPv4 traffic from LAN to Internet
    iifname $LAN_DEV oifname $INET_DEV snat ip to 203.0.113.1
}
}

```

2. Include the `/etc/nftables/firewall.nft` script in the `/etc/sysconfig/nftables.conf` file:

```
include "/etc/nftables/firewall.nft"
```

3. Enable IPv4 forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

4. Enable and start the **nftables** service:

```
# systemctl enable --now nftables
```

Verification

1. Optional: Verify the **nftables** rule set:

—

```
# nft list ruleset
```

```
...
```

2. Try to perform an access that the firewall prevents. For example, try to access the router using SSH from the DMZ:

```
# ssh router.example.com
```

```
ssh: connect to host router.example.com port 22: Network is unreachable
```

3. Depending on your logging settings, search:

- The **systemd** journal for the blocked packets:

```
# journalctl -k -g "nft drop"
```

```
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
```

```
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```

- The **/var/log/nftables.log** file for the blocked packets:

```
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
```

```
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```

47.8. CONFIGURING PORT FORWARDING USING NFTABLES

Port forwarding enables administrators to forward packets sent to a specific destination port to a different local or remote port.

For example, if your web server does not have a public IP address, you can set a port forwarding rule on your firewall that forwards incoming packets on port **80** and **443** on the firewall to the web server. With this firewall rule, users on the internet can access the web server using the IP or host name of the firewall.

47.8.1. Forwarding incoming packets to a different local port

This section describes an example of how to forward incoming IPv4 packets on port **8022** to port **22** on the local system.

Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; }
```



NOTE

Pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **8022** to the local port **22**:

```
# nft add rule ip nat prerouting tcp dport 8022 redirect to :22
```

47.8.2. Forwarding incoming packets on a specific local port to a different host

You can use a destination network address translation (DNAT) rule to forward incoming packets on a local port to a remote host. This enables users on the Internet to access a service that runs on a host with a private IP address.

The procedure describes how to forward incoming IPv4 packets on the local port **443** to the same port number on the remote system with the **192.0.2.1** IP address.

Prerequisites

- You are logged in as the **root** user on the system that should forward the packets.

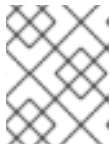
Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain ip nat postrouting { type nat hook postrouting priority 100 \; }
```



NOTE

Pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **443** to the same port on **192.0.2.1**:

```
# nft add rule ip nat prerouting tcp dport 443 dnat to 192.0.2.1
```

4. Add a rule to the **postrouting** chain to masquerade outgoing traffic:

```
# nft add rule ip nat postrouting daddr 192.0.2.1 masquerade
```

5. 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
```

47.9. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

You can use **nftables** to limit the number of connections or to block IP addresses that attempt to establish a given amount of connections to prevent them from using too many system resources.

47.9.1. Limiting the number of connections using nftables

The **ct count** parameter of the **nft** utility enables administrators to limit the number of connections. The procedure describes a basic example of how to limit incoming connections.

Prerequisites

- The base **example_chain** in **example_table** exists.

Procedure

1. Create a dynamic set for IPv4 addresses:

```
# nft add set inet example_table example_meter { type ipv4_addr\; flags dynamic \;}
```

2. Add a rule that allows only two simultaneous connections to the SSH port (22) from an IPv4 address and rejects all further connections from the same IP:

```
# nft add rule ip example_table example_chain tcp dport ssh meter example_meter { ip saddr ct count over 2 } counter reject
```

3. Optional: Display the set created in the previous step:

```
# nft list set inet example_table example_meter
table inet example_table {
  meter example_meter {
    type ipv4_addr
    size 65535
    elements = { 192.0.2.1 ct count over 2 , 192.0.2.2 ct count over 2 }
  }
}
```

The **elements** entry displays addresses that currently match the rule. In this example, **elements** lists IP addresses that have active connections to the SSH port. Note that the output does not display the number of active connections or if connections were rejected.

47.9.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute

This section explains how you temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute.

Procedure

1. Create the **filter** table with the **ip** address family:

```
# nft add table ip filter
```

2. Add the **input** chain to the **filter** table:

```
# nft add chain ip filter input { type filter hook input priority 0 \; }
```

3. Add a rule that drops all packets from source addresses that attempt to establish more than ten TCP connections within one minute:

```
# nft add rule ip filter input ip protocol tcp ct state new, untracked meter ratemeter { ip saddr  
timeout 5m limit rate over 10/minute } drop
```

The **timeout 5m** parameter defines that **nftables** automatically removes entries after five minutes to prevent that the meter fills up with stale entries.

Verification

- To display the meter's content, enter:

```
# nft list meter ip filter ratemeter  
table ip filter {  
  meter ratemeter {  
    type ipv4_addr  
    size 65535  
    flags dynamic,timeout  
    elements = { 192.0.2.1 limit rate over 10/minute timeout 5m expires 4m58s224ms }  
  }  
}
```

47.10. DEBUGGING NFTABLES RULES

The **nftables** framework provides different options for administrators to debug rules and if packets match them. This section describes these options.

47.10.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter. This section describes how to create a new rule with a counter.

- For more information on a procedure that adds a counter to an existing rule, see [Adding a counter to an existing rule](#).

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. Add a new rule with the **counter** parameter to the chain. The following example adds a rule with a counter that allows TCP traffic on port 22 and counts the packets and traffic that match this rule:

```
# nft add rule inet example_table example_chain tcp dport 22 counter accept
```

2. To display the counter values:

```
# nft list ruleset
```

```

table inet example_table {
    chain example_chain {
        type filter hook input priority filter; policy accept;
        tcp dport ssh counter packets 6872 bytes 105448565 accept
    }
}

```

47.10.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter. This section describes how to add a counter to an existing rule.

- For more information on a procedure that adds a new rule with a counter, see [Creating a rule with the counter](#).

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```

# nft --handle list chain inet example_table example_chain
table inet example_table {
    chain example_chain { # handle 1
        type filter hook input priority filter; policy accept;
        tcp dport ssh accept # handle 4
    }
}

```

2. Add the counter by replacing the rule but with the **counter** parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```

# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept

```

3. To display the counter values:

```

# nft list ruleset
table inet example_table {
    chain example_chain {
        type filter hook input priority filter; policy accept;
        tcp dport ssh counter packets 6872 bytes 105448565 accept
    }
}

```

47.10.3. Monitoring packets that match an existing rule

The tracing feature in **nftables** in combination with the **nft monitor** command enables administrators to display packets that match a rule. The procedure describes how to enable tracing for a rule as well as monitoring packets that match this rule.

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the **meta nfttrace set 1** parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nfttrace set 1 accept
```

3. Use the **nft monitor** command to display the tracing. The following example filters the output of the command to display only entries that contain **inet example_table example_chain**:

```
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr
52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp
cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport
ssh tcp flags == syn tcp window 64240
trace id 3c5eb15e inet example_table example_chain rule tcp dport ssh nfttrace set 1 accept
(verdict accept)
...
```



WARNING

Depending on the number of rules with tracing enabled and the amount of matching traffic, the **nft monitor** command can display a lot of output. Use **grep** or other utilities to filter the output.

47.11. BACKING UP AND RESTORING THE NFTABLES RULE SET

This section describes how to backup **nftables** rules to a file, as well as restoring rules from a file.

Administrators can use a file with the rules to, for example, transfer the rules to a different server.

47.11.1. Backing up the nftables rule set to a file

This section describes how to back up the **nftables** rule set to a file.

Procedure

- To backup **nftables** rules:
 - In a format produced by **nft list ruleset** format:

```
# nft list ruleset > file.nft
```

- In JSON format:

```
# nft -j list ruleset > file.json
```

47.11.2. Restoring the nftables rule set from a file

This section describes how to restore the **nftables** rule set.

Procedure

- To restore **nftables** rules:
 - If the file to restore is in the format produced by **nft list ruleset** or contains **nft** commands directly:

```
# nft -f file.nft
```

- If the file to restore is in JSON format:

```
# nft -j -f file.json
```

47.12. ADDITIONAL RESOURCES

- [Using nftables in Red Hat Enterprise Linux 8](#)
- [What comes after iptables? Its successor, of course: nftables](#)
- [Firewalld: The Future is nftables](#)

CHAPTER 48. USING XDP-FILTER FOR HIGH-PERFORMANCE TRAFFIC FILTERING TO PREVENT DDOS ATTACKS

Compared to packet filters, such as **nftables**, Express Data Path (XDP) processes and drops network packets right at the network interface. Therefore, XDP determines the next step for the package before it reaches a firewall or other applications. As a result, XDP filters require less resources and can process network packets at a much higher rate than conventional packet filters to defend against distributed denial of service (DDoS) attacks. For example, during testing, Red Hat dropped 26 million network packets per second on a single core, which is significantly higher than the drop rate of **nftables** on the same hardware.

The **xdp-filter** utility allows or drops incoming network packets using XDP. You can create rules to filter traffic to or from specific:

- IP addresses
- MAC addresses
- Ports

Note that, even if **xdp-filter** has a significantly higher packet-processing rate, it does not have the same capabilities as, for example, **nftables**. Consider **xdp-filter** a conceptual utility to demonstrate packet filtering using XDP. Additionally, you can use the code of the utility for a better understanding of how to write your own XDP applications.



IMPORTANT

On other architectures than AMD and Intel 64-bit, the **xdp-filter** utility 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.

48.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to drop network packets:

- To a specific destination port
- From a specific IP address
- From a specific MAC address

The **allow** policy of **xdp-filter** defines that all traffic is allowed and the filter drops only network packets that match a particular rule. For example, use this method if you know the source IP addresses of packets you want to drop.

Prerequisites

- The **xdp-tools** package is installed.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process incoming packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0
```

By default, **xdp-filter** uses the **allow** policy, and the utility drops only traffic that matches any rule.

Optionally, use the **-f feature** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of package processing. To enable multiple features, separate them with a comma.

If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to drop packets that match them. For example:

- To drop incoming packets to port **22**, enter:

```
# xdp-filter port 22
```

This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, use the **-p protocol** option.

- To drop incoming packets from **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1 -m src
```

Note that **xdp-filter** does not support IP ranges.

- To drop incoming packets from MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE -m src
```

Verification steps

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

Additional resources

- **xdp-filter(8)** man page
- If you are a developer and interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

48.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to allow only network packets:

- From and to a specific destination port
- From and to a specific IP address
- From and to specific MAC address

To do so, use the **deny** policy of **xdp-filter** which defines that the filter drops all network packets except the ones that match a particular rule. For example, use this method if you do not know the source IP addresses of packets you want to drop.



WARNING

If you set the default policy to **deny** when you load **xdp-filter** on an interface, the kernel immediately drops all packets from this interface until you create rules that allow certain traffic. To avoid being locked out from the system, enter the commands locally or connect through a different network interface to the host.

Prerequisites

- The **xdp-tools** package is installed.
- You are logged in to the host either locally or using a network interface for which you do not plan to filter the traffic.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0 -p deny
```

Optionally, use the **-f feature** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of package processing. To enable multiple features, separate them with a comma.

If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to allow packets that match them. For example:

- To allow packets to port **22**, enter:

```
# xdp-filter port 22
```

This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, pass the **-p protocol** option to the command.

- To allow packets to **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1
```

Note that **xdp-filter** does not support IP ranges.

- To allow packets to MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE
```



IMPORTANT

The **xdp-filter** utility does not support stateful packet inspection. This requires that you either do not set a mode using the **-m mode** option or you add explicit rules to allow incoming traffic that the machine receives in reply to outgoing traffic.

Verification steps

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

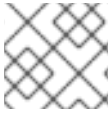
Additional resources

- **xdp-filter(8)** man page.
- If you are a developer and you are interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

CHAPTER 49. GETTING STARTED WITH DPDK

The data plane development kit (DPDK) provides libraries and network drivers to accelerate package processing in user space.

Administrators use DPDK, for example, in virtual machines to use Single Root I/O Virtualization (SR-IOV) to reduce latencies and increase I/O throughput.



NOTE

Red Hat does not support experimental DPDK APIs.

49.1. INSTALLING THE DPDK PACKAGE

This section describes how to install the **dpdk** package.

Prerequisites

- Red Hat Enterprise Linux is installed.
- A valid subscription is assigned to the host.

Procedure

- Use the **yum** utility to install the **dpdk** package:

```
# yum install dpdk
```

49.2. ADDITIONAL RESOURCES

- [Network Adapter Fast Datapath Feature Support Matrix](#)

CHAPTER 50. UNDERSTANDING THE EBPF NETWORKING FEATURES IN RHEL 8

The extended Berkeley Packet Filter (eBPF) is an in-kernel virtual machine that allows code execution in the kernel space. This code runs in a restricted sandbox environment with access only to a limited set of functions.

In networking, you can use eBPF to complement or replace kernel packet processing. Depending on the hook you use, eBPF programs have, for example:

- Read and write access to packet data and metadata
- Can look up sockets and routes
- Can set socket options
- Can redirect packets

50.1. OVERVIEW OF NETWORKING EBPF FEATURES IN RHEL 8

You can attach extended Berkeley Packet Filter (eBPF) networking programs to the following hooks in RHEL:

- **eXpress Data Path (XDP)**: Provides early access to received packets before the kernel networking stack processes them.
- **tc** eBPF classifier with direct-action flag: Provides powerful packet processing on ingress and egress.
- **Control Groups version 2 (cgroup v2)**: Enables filtering and overriding socket-based operations performed by programs in a control group.
- **Socket filtering**: Enables filtering of packets received from sockets. This feature was also available in the classic Berkeley Packet Filter (cBPF), but has been extended to support eBPF programs.
- **Stream parser**: Enables splitting up streams to individual messages, filtering, and redirecting them to sockets.
- **SO_REUSEPORT** socket selection: Provides a programmable selection of a receiving socket from a **reuseport** socket group.
- **Flow dissector**: Enables overriding the way the kernel parses packet headers in certain situations.
- **TCP congestion control callbacks**: Enables implementing a custom TCP congestion control algorithm.
- **Routes with encapsulation**: Enables creating custom tunnel encapsulation.

Note that Red Hat does not support all of the eBPF functionality that is available in RHEL and described here. For further details and the support status of the individual hooks, see the [RHEL 8 Release Notes](#) and the following overview.

XDP

You can attach programs of the **BPF_PROG_TYPE_XDP** type to a network interface. The kernel then

executes the program on received packets before the kernel network stack starts processing them. This allows fast packet forwarding in certain situations, such as fast packet dropping to prevent distributed denial of service (DDoS) attacks and fast packet redirects for load balancing scenarios.

You can also use XDP for different forms of packet monitoring and sampling. The kernel allows XDP programs to modify packets and to pass them for further processing to the kernel network stack.

The following XDP modes are available:

- **Native (driver) XDP:** The kernel executes the program from the earliest possible point during packet reception. At this moment, the kernel did not parse the packet and, therefore, no metadata provided by the kernel is available. This mode requires that the network interface driver supports XDP but not all drivers support this native mode.
- **Generic XDP:** The kernel network stack executes the XDP program early in the processing. At that time, kernel data structures have been allocated, and the packet has been pre-processed. If a packet should be dropped or redirected, it requires a significant overhead compared to the native mode. However, the generic mode does not require network interface driver support and works with all network interfaces.
- **Offloaded XDP:** The kernel executes the XDP program on the network interface instead of on the host CPU. Note that this requires specific hardware, and only certain eBPF features are available in this mode.

On RHEL, load all XDP programs using the **libxdp** library. This library enables system-controlled usage of XDP.



NOTE

Currently, there are some system configuration limitations for XDP programs. For example, you must disable certain hardware offload features on the receiving interface. Additionally, not all features are available with all drivers that support the native mode.

In RHEL 8.7, Red Hat supports the XDP feature only if all of the following conditions apply:

- You load the XDP program on an AMD or Intel 64-bit architecture.
- You use the **libxdp** library to load the program into the kernel.
- The XDP program does not use the XDP hardware offloading.

Additionally, Red Hat provides the following usage of XDP features as unsupported Technology Preview:

- Loading XDP programs on architectures other than AMD and Intel 64-bit. Note that the **libxdp** library is not available for architectures other than AMD and Intel 64-bit.
- The XDP hardware offloading.

AF_XDP

Using an XDP program that filters and redirects packets to a given **AF_XDP** socket, you can use one or more sockets from the **AF_XDP** protocol family to quickly copy packets from the kernel to the user space.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Traffic Control

The Traffic Control (**tc**) subsystem offers the following types of eBPF programs:

- **BPF_PROG_TYPE_SCHED_CLS**
- **BPF_PROG_TYPE_SCHED_ACT**

These types enable you to write custom **tc** classifiers and **tc** actions in eBPF. Together with the parts of the **tc** ecosystem, this provides the ability for powerful packet processing and is the core part of several container networking orchestration solutions.

In most cases, only the classifier is used, as with the direct-action flag, the eBPF classifier can execute actions directly from the same eBPF program. The **clsact** Queueing Discipline (**qdisc**) has been designed to enable this on the ingress side.

Note that using a flow dissector eBPF program can influence operation of some other **qdiscs** and **tc** classifiers, such as **flower**.

The eBPF for **tc** feature is fully supported in RHEL 8.2 and later.

Socket filter

Several utilities use or have used the classic Berkeley Packet Filter (cBPF) for filtering packets received on a socket. For example, the **tcpdump** utility enables the user to specify expressions, which **tcpdump** then translates into cBPF code.

As an alternative to cBPF, the kernel allows eBPF programs of the **BPF_PROG_TYPE_SOCKET_FILTER** type for the same purpose.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Control Groups

In RHEL, you can use multiple types of eBPF programs that you can attach to a cgroup. The kernel executes these programs when a program in the given cgroup performs an operation. Note that you can use only cgroups version 2.

The following networking-related cgroup eBPF programs are available in RHEL:

- **BPF_PROG_TYPE SOCK_OPS**: The kernel calls this program on various TCP events. The program can adjust the behavior of the kernel TCP stack, including custom TCP header options, and so on.
- **BPF_PROG_TYPE_CGROUP_SOCK_ADDR**: The kernel calls this program during **connect**, **bind**, **sendto**, **recvmsg**, **getpeername**, and **getsockname** operations. This program allows changing IP addresses and ports. This is useful when you implement socket-based network address translation (NAT) in eBPF.
- **BPF_PROG_TYPE_CGROUP_SOCKOPT**: The kernel calls this program during **setsockopt** and **getsockopt** operations and allows changing the options.
- **BPF_PROG_TYPE_CGROUP_SOCK**: The kernel calls this program during socket creation, socket releasing, and binding to addresses. You can use these programs to allow or deny the operation, or only to inspect socket creation for statistics.
- **BPF_PROG_TYPE_CGROUP_SKB**: This program filters individual packets on ingress and egress, and can accept or reject packets.

- **BPF_PROG_TYPE_CGROUP_SYSCTL**: This program allows filtering of access to system controls (**sysctl**).
- **BPF_CGROUP_INET4_GETPEERNAME**, **BPF_CGROUP_INET6_GETPEERNAME**, **BPF_CGROUP_INET4_GETSOCKNAME**, and **BPF_CGROUP_INET6_GETSOCKNAME**: Using these programs, you can override the result of **getsockname** and **getpeername** system calls. This is useful when you implement socket-based network address translation (NAT) in eBPF.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Stream Parser

A stream parser operates on a group of sockets that are added to a special eBPF map. The eBPF program then processes packets that the kernel receives or sends on those sockets.

The following stream parser eBPF programs are available in RHEL:

- **BPF_PROG_TYPE_SK_SKB**: An eBPF program parses packets received from the socket into individual messages, and instructs the kernel to drop those messages or send them to another socket in the group.
- **BPF_PROG_TYPE_SK_MSG**: This program filters egress messages. An eBPF program parses the packets into individual messages and either approves or rejects them.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

SO_REUSEPORT socket selection

Using this socket option, you can bind multiple sockets to the same IP address and port. Without eBPF, the kernel selects the receiving socket based on a connection hash. With the **BPF_PROG_TYPE_SK_REUSEPORT** program, the selection of the receiving socket is fully programmable.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Flow dissector

When the kernel needs to process packet headers without going through the full protocol decode, they are **dissected**. For example, this happens in the **tc** subsystem, in multipath routing, in bonding, or when calculating a packet hash. In this situation the kernel parses the packet headers and fills internal structures with the information from the packet headers. You can replace this internal parsing using the **BPF_PROG_TYPE_FLOW_DISSECTOR** program. Note that you can only dissect TCP and UDP over IPv4 and IPv6 in eBPF in RHEL.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

TCP Congestion Control

You can write a custom TCP congestion control algorithm using a group of **BPF_PROG_TYPE_STRUCT_OPS** programs that implement **struct tcp_congestion_oops** callbacks. An algorithm that is implemented this way is available to the system alongside the built-in kernel algorithms.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Routes with encapsulation

You can attach one of the following eBPF program types to routes in the routing table as a tunnel encapsulation attribute:

- **BPF_PROG_TYPE_LWT_IN**

- **BPF_PROG_TYPE_LWT_OUT**
- **BPF_PROG_TYPE_LWT_XMIT**

The functionality of such an eBPF program is limited to specific tunnel configurations and does not allow creating a generic encapsulation or decapsulation solution.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

Socket lookup

To bypass limitations of the **bind** system call, use an eBPF program of the **BPF_PROG_TYPE_SK_LOOKUP** type. Such programs can select a listening socket for new incoming TCP connections or an unconnected socket for UDP packets.

In RHEL 8.7, Red Hat provides this feature as an unsupported Technology Preview.

50.2. OVERVIEW OF XDP FEATURES IN RHEL 8 BY NETWORK CARDS

The following is an overview of XDP-enabled network cards and the XDP features you can use with them:

Network card	Driver	Basic	Redirect	Target	HW offload	Zero-copy
Amazon Elastic Network Adapter	ena	yes	yes	yes ^[a]	no	no
Broadcom NetXtreme-C/E 10/25/40/50 gigabit Ethernet	bnxt_en	yes	yes	yes ^[a]	no	no
Cavium Thunder Virtual function	nicvf	yes	no	no	no	no
Intel® 10GbE PCI Express Virtual Function Ethernet	ixgbev	yes	no	no	no	no
Intel® 10GbE PCI Express adapters	ixgbe	yes	yes	yes ^[a]	no	yes
Intel® Ethernet Connection E800 Series	ice	yes	yes	yes ^[a] ^[b]	no	yes
Intel® Ethernet Controller I225-LM/I225-V family	igc	yes	yes	yes	no	yes
Intel® Ethernet Controller XL710 Family	i40e	yes	yes	yes ^[a] ^[b]	no	yes
Intel® PCI Express Gigabit adapters	igb	yes	yes	yes ^[a]	no	no
Mellanox 5th generation network adapters (ConnectX series)	mlx5_core	yes	yes	yes ^[b]	no	yes

Network card	Driver	Basic	Redirect	Target	HW offload	Zero-copy
Mellanox Technologies 1/10/40Gbit Ethernet	mlx4_en	yes	yes	no	no	no
Microsoft Azure Network Adapter	mana	yes	no	no	no	no
Microsoft Hyper-V virtual network	hv_netvsc	yes	yes	yes	no	no
Netronome® NFP4000/NFP6000 NIC	nfp	yes	no	no	yes	no
QEMU Virtio network	virtio_net	yes	yes	yes ^[a] _[b]	no	no
QLogic QED 25/40/100Gb Ethernet NIC	qed	yes	yes	yes	no	no
Solarflare SFC9000/SFC9100/EF100-family	sfc	yes	yes	yes ^[b]	no	no
Universal TUN/TAP device	tun	yes	yes	yes	no	no
Virtual Ethernet pair device	veth	yes	yes	yes	no	no
<p>[a] Only if an XDP program is loaded on the interface.</p> <p>[b] Requires a number of XDP TX queues allocated that is larger or equal to the largest CPU index.</p>						

Legend:

- Basic: Supports basic return codes: **DROP**, **PASS**, **ABORTED**, and **TX**.
- Redirect: Supports the **REDIRECT** return code.
- Target: Can be a target of a **REDIRECT** return code.
- HW offload: Supports XDP hardware offload.
- Zero-copy: Supports the zero-copy mode for the **AF_XDP** protocol family.

CHAPTER 51. NETWORK TRACING USING THE BPF COMPILER COLLECTION

This section explains what the BPF Compiler Collection (BCC) is, how you install the BCC, as well as how to perform different network tracing operations using the pre-created scripts provided by the **bcc-tools** package. All of these scripts support the **--ebpf** parameter to display the eBPF code the utility uploads to the kernel. You can use the code to learn more about writing eBPF scripts.

51.1. AN INTRODUCTION TO BCC

BPF Compiler Collection (BCC) is a library, which facilitates the creation of the extended Berkeley Packet Filter (eBPF) programs. The main utility of eBPF programs is analyzing OS performance and network performance without experiencing overhead or security issues.

BCC removes the need for users to know deep technical details of eBPF, and provides many out-of-the-box starting points, such as the **bcc-tools** package with pre-created eBPF programs.



NOTE

The eBPF programs are triggered on events, such as disk I/O, TCP connections, and process creations. It is unlikely that the programs should cause the kernel to crash, loop or become unresponsive because they run in a safe virtual machine in the kernel.

51.2. INSTALLING THE BCC-TOOLS PACKAGE

This section describes how to install the **bcc-tools** package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Procedure

1. Install **bcc-tools**:

```
# yum install bcc-tools
```

The BCC tools are installed in the **/usr/share/bcc/tools/** directory.

2. Optionally, inspect the tools:

```
# ll /usr/share/bcc/tools/
...
-rwxr-xr-x. 1 root root 4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root 3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root 7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root 8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root 7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root 6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

The **doc** directory in the listing above contains documentation for each tool.

51.3. DISPLAYING TCP CONNECTIONS ADDED TO THE KERNEL'S ACCEPT QUEUE

After the kernel receives the **ACK** packet in a TCP 3-way handshake, the kernel moves the connection from the **SYN** queue to the **accept** queue after the connection's state changes to **ESTABLISHED**. Therefore, only successful TCP connections are visible in this queue.

The **tcpaccept** utility uses eBPF features to display all connections the kernel adds to the **accept** queue. The utility is lightweight because it traces the **accept()** function of the kernel instead of capturing packets and filtering them. For example, use **tcpaccept** for general troubleshooting to display new connections the server has accepted.

Procedure

1. Enter the following command to start the tracing the kernel **accept** queue:

```
# /usr/share/bcc/tools/tcpaccept
PID COMM  IP RADDR  RPORT LADDR  LPORT
843  sshd    4  192.0.2.17  50598  192.0.2.1  22
1107 ns-slapd 4  198.51.100.6 38772  192.0.2.1  389
1107 ns-slapd 4  203.0.113.85 38774  192.0.2.1  389
...
```

Each time the kernel accepts a connection, **tcpaccept** displays the details of the connections.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpaccept(8)** man page
- **/usr/share/bcc/tools/doc/tcpaccept_example.txt** file

51.4. TRACING OUTGOING TCP CONNECTION ATTEMPTS

The **tcpconnect** utility uses eBPF features to trace outgoing TCP connection attempts. The output of the utility also includes connections that failed.

The **tcpconnect** utility is lightweight because it traces, for example, the **connect()** function of the kernel instead of capturing packets and filtering them.

Procedure

1. Enter the following command to start the tracing process that displays all outgoing connections:

```
# /usr/share/bcc/tools/tcpconnect
PID COMM  IP SADDR  DADDR  DPORT
31346 curl    4  192.0.2.1 198.51.100.16 80
31348 telnet  4  192.0.2.1 203.0.113.231 23
31361 isc-worker00 4  192.0.2.1 192.0.2.254 53
...
```

Each time the kernel processes an outgoing connection, **tcpconnect** displays the details of the connections.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpconnect(8)** man page
- `/usr/share/bcc/tools/doc/tcpconnect_example.txt` file

51.5. MEASURING THE LATENCY OF OUTGOING TCP CONNECTIONS

The TCP connection latency is the time taken to establish a connection. This typically involves the kernel TCP/IP processing and network round trip time, and not the application runtime.

The **tcpconnl**at utility uses eBPF features to measure the time between a sent **SYN** packet and the received response packet.

Procedure

1. Start measuring the latency of outgoing connections:

```
# /usr/share/bcc/tools/tcpconnl  
PID COMM      IP SADDR  DADDR      DPORT LAT(ms)  
32151 isc-worker00 4 192.0.2.1 192.0.2.254 53 0.60  
32155 ssh        4 192.0.2.1 203.0.113.190 22 26.34  
32319 curl      4 192.0.2.1 198.51.100.59 443 188.96  
...
```

Each time the kernel processes an outgoing connection, **tcpconnl**at displays the details of the connection after the kernel receives the response packet.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpconnl**at(8) man page
- `/usr/share/bcc/tools/doc/tcpconnl_example.txt` file

51.6. DISPLAYING DETAILS ABOUT TCP PACKETS AND SEGMENTS THAT WERE DROPPED BY THE KERNEL

The **tcpdrop** utility enables administrators to display details about TCP packets and segments that were dropped by the kernel. Use this utility to debug high rates of dropped packets that can cause the remote system to send timer-based retransmits. High rates of dropped packets and segments can impact the performance of a server.

Instead of capturing and filtering packets, which is resource-intensive, the **tcpdrop** utility uses eBPF features to retrieve the information directly from the kernel.

Procedure

1. Enter the following command to start displaying details about dropped TCP packets and segments:

```
# /usr/share/bcc/tools/tcpdrop
TIME   PID  IP SADDR:SPORT  > DADDR:DPORT  STATE (FLAGS)
13:28:39 32253 4 192.0.2.85:51616 > 192.0.2.1:22 CLOSE_WAIT (FIN|ACK)
b'tcp_drop+0x1'
b'tcp_data_queue+0x2b9'
...

13:28:39 1    4 192.0.2.85:51616 > 192.0.2.1:22 CLOSE (ACK)
b'tcp_drop+0x1'
b'tcp_rcv_state_process+0xe2'
...
```

Each time the kernel drops TCP packets and segments, **tcpdrop** displays the details of the connection, including the kernel stack trace that led to the dropped package.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpdrop(8)** man page
- **/usr/share/bcc/tools/doc/tcpdrop_example.txt** file

51.7. TRACING TCP SESSIONS

The **tcplife** utility uses eBPF to trace TCP sessions that open and close, and prints a line of output to summarize each one. Administrators can use **tcplife** to identify connections and the amount of transferred traffic.

The example in this section describes how to display connections to port **22** (SSH) to retrieve the following information:

- The local process ID (PID)
- The local process name
- The local IP address and port number
- The remote IP address and port number
- The amount of received and transmitted traffic in KB.
- The time in milliseconds the connection was active

Procedure

1. Enter the following command to start the tracing of connections to the local port **22**:

```
/usr/share/bcc/tools/tcplife -L 22
PID COMM  LADDR  LPORT RADDR  RPORT TX_KB RX_KB  MS
19392 sshd  192.0.2.1 22 192.0.2.17 43892 53 52 6681.95
19431 sshd  192.0.2.1 22 192.0.2.245 43902 81 249381 7585.09
19487 sshd  192.0.2.1 22 192.0.2.121 43970 6998 7 16740.35
...
```

Each time a connection is closed, **tcplife** displays the details of the connections.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcplife(8)** man page
- `/usr/share/bcc/tools/doc/tcplife_example.txt` file

51.8. TRACING TCP RETRANSMISSIONS

The **tcpretrans** utility displays details about TCP retransmissions, such as the local and remote IP address and port number, as well as the TCP state at the time of the retransmissions.

The utility uses eBPF features and, therefore, has a very low overhead.

Procedure

1. Use the following command to start displaying TCP retransmission details:

```
# /usr/share/bcc/tools/tcpretrans
TIME    PID IP LADDR:LPORT  T> RADDR:RPORT    STATE
00:23:02 0   4 192.0.2.1:22  R> 198.51.100.0:26788 ESTABLISHED
00:23:02 0   4 192.0.2.1:22  R> 198.51.100.0:26788 ESTABLISHED
00:45:43 0   4 192.0.2.1:22  R> 198.51.100.0:17634 ESTABLISHED
...
```

Each time the kernel calls the TCP retransmit function, **tcpretrans** displays the details of the connection.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpretrans(8)** man page
- `/usr/share/bcc/tools/doc/tcpretrans_example.txt` file

51.9. DISPLAYING TCP STATE CHANGE INFORMATION

During a TCP session, the TCP state changes. The **tcpstates** utility uses eBPF functions to trace these state changes, and prints details including the duration in each state. For example, use **tcpstates** to identify if connections spend too much time in the initialization state.

Procedure

1. Use the following command to start to start tracing TCP state changes:

```
# /usr/share/bcc/tools/tcpstates
SKADDR      C-PID C-COMM  LADDR  LPORT RADDR  RPORT OLDSTATE  ->
NEWSTATE  MS
ffff9cd377b3af80 0  swapper/1 0.0.0.0 22  0.0.0.0 0  LISTEN  -> SYN_RECV
0.000
```



```

ffff9cd377b3af80 0  swapper/1 192.0.2.1 22  192.0.2.45 53152 SYN_RECV  ->
ESTABLISHED 0.067
ffff9cd377b3af80 818  sssd_nss 192.0.2.1 22  192.0.2.45 53152 ESTABLISHED ->
CLOSE_WAIT 65636.773
ffff9cd377b3af80 1432  sshd 192.0.2.1 22  192.0.2.45 53152 CLOSE_WAIT ->
LAST_ACK 24.409
ffff9cd377b3af80 1267  pulseaudio 192.0.2.1 22  192.0.2.45 53152 LAST_ACK ->
CLOSE 0.376
...

```

Each time a connection changes its state, **tcpstates** displays a new line with updated connection details.

If multiple connections change their state at the same time, use the socket address in the first column (**SKADDR**) to determine which entries belong to the same connection.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpstates(8)** man page
- `/usr/share/bcc/tools/doc/tcpstates_example.txt` file

51.10. SUMMARIZING AND AGGREGATING TCP TRAFFIC SENT TO SPECIFIC SUBNETS

The **tcpsubnet** utility summarizes and aggregates IPv4 TCP traffic that the local host sends to subnets and displays the output on a fixed interval. The utility uses eBPF features to collect and summarize the data to reduce the overhead.

By default, **tcpsubnet** summarizes traffic for the following subnets:

- **127.0.0.1/32**
- **10.0.0.0/8**
- **172.16.0.0/12**
- **192.0.2.0/24/16**
- **0.0.0.0/0**

Note that the last subnet (**0.0.0.0/0**) is a catch-all option. The **tcpsubnet** utility counts all traffic for subnets different than the first four in this catch-all entry.

Follow the procedure to count the traffic for the **192.0.2.0/24** and **198.51.100.0/24** subnets. Traffic to other subnets will be tracked in the **0.0.0.0/0** catch-all subnet entry.

Procedure

1. Start monitoring the amount of traffic sent to the **192.0.2.0/24**, **198.51.100.0/24**, and other subnets:

```
# /usr/share/bcc/tools/tcpsubnet 192.0.2.0/24,198.51.100.0/24,0.0.0.0/0
```

```
Tracing... Output every 1 secs. Hit Ctrl-C to end
[02/21/20 10:04:50]
192.0.2.0/24      856
198.51.100.0/24   7467
[02/21/20 10:04:51]
192.0.2.0/24      1200
198.51.100.0/24   8763
0.0.0.0/0         673
...
```

This command displays the traffic in bytes for the specified subnets once per second.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcpsubnet(8)** man page
- `/usr/share/bcc/tools/doc/tcpsubnet.txt` file

51.11. DISPLAYING THE NETWORK THROUGHPUT BY IP ADDRESS AND PORT

The **tcptop** utility displays TCP traffic the host sends and receives in kilobytes. The report automatically refreshes and contains only active TCP connections. The utility uses eBPF features and, therefore, has only a very low overhead.

Procedure

1. To monitor the sent and received traffic, enter:

```
# /usr/share/bcc/tools/tcptop
13:46:29 loadavg: 0.10 0.03 0.01 1/215 3875

PID  COMM    LADDR      RADDR      RX_KB  TX_KB
3853  3853     192.0.2.1:22 192.0.2.165:41838 32    102626
1285  sshd     192.0.2.1:22 192.0.2.45:39240 0      0
...
```

The output of the command includes only active TCP connections. If the local or remote system closes a connection, the connection is no longer visible in the output.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcptop(8)** man page
- `/usr/share/bcc/tools/doc/tcptop.txt` file

51.12. TRACING ESTABLISHED TCP CONNECTIONS

The **tcptracer** utility traces the kernel functions that connect, accept, and close TCP connections. The utility uses eBPF features and, therefore, has a very low overhead.

Procedure

1. Use the following command to start the tracing process:

```
# /usr/share/bcc/tools/tcptracer
Tracing TCP established connections. Ctrl-C to end.
T PID  COMM      IP SADDR      DADDR      SPORT DPORT
A 1088 ns-slapd  4 192.0.2.153 192.0.2.1  0    65535
A 845  sshd      4 192.0.2.1   192.0.2.67 22    42302
X 4502 sshd      4 192.0.2.1   192.0.2.67 22    42302
...
```

Each time the kernel connects, accepts, or closes a connection, **tcptracer** displays the details of the connections.

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **tcptracer(8)** man page
- `/usr/share/bcc/tools/doc/tcptracer_example.txt` file

51.13. TRACING IPV4 AND IPV6 LISTEN ATTEMPTS

The **solisten** utility traces all IPv4 and IPv6 listen attempts. It traces the listen attempts including that ultimately fail or the listening program that does not accept the connection. The utility traces function that the kernel calls when a program wants to listen for TCP connections.

Procedure

1. Enter the following command to start the tracing process that displays all listen TCP attempts:

```
# /usr/share/bcc/tools/solisten
PID  COMM      PROTO  BACKLOG  PORT  ADDR
3643 nc        TCPv4   1        4242  0.0.0.0
3659 nc        TCPv6   1        4242  2001:db8:1::1
4221 redis-server TCPv6   128     6379  ::
4221 redis-server TCPv4   128     6379  0.0.0.0
....
```

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **solisten(9)** man page
- `/usr/share/bcc/tools/doc/solisten_example.txt` file

51.14. SUMMARIZING THE SERVICE TIME OF SOFT INTERRUPTS

The **softirqs** utility summarizes the time spent servicing soft interrupts (soft IRQs) and shows this time as either totals or histogram distributions. This utility uses the **irq:softirq_enter** and **irq:softirq_exit** kernel tracepoints, which is a stable tracing mechanism.

Procedure

1. Enter the following command to start the tracing **soft irq** event time:

```
# /usr/share/bcc/tools/softirqs
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ      TOTAL_usecs
tasklet      166
block        9152
net_rx       12829
rcu          53140
sched        182360
timer        306256
```

2. Press **Ctrl+C** to stop the tracing process.

Additional resources

- **softirqs(8)** man page
- **/usr/share/bcc/tools/doc/softirqs_example.txt** file
- **mpstat(1)** man page

51.15. ADDITIONAL RESOURCES

- **/usr/share/doc/bcc/README.md** file

CHAPTER 52. GETTING STARTED WITH TIPC

Transparent Inter-process Communication (TIPC), which is also known as **Cluster Domain Sockets**, is an Inter-process Communication (IPC) service for cluster-wide operation.

Applications that are running in a high-available and dynamic cluster environment have special needs. The number of nodes in a cluster can vary, routers can fail, and, due to load balancing considerations, functionality can be moved to different nodes in the cluster. TIPC minimizes the effort by application developers to deal with such situations, and maximizes the chance that they are handled in a correct and optimal way. Additionally, TIPC provides a more efficient and fault-tolerant communication than general protocols, such as TCP.

52.1. THE ARCHITECTURE OF TIPC

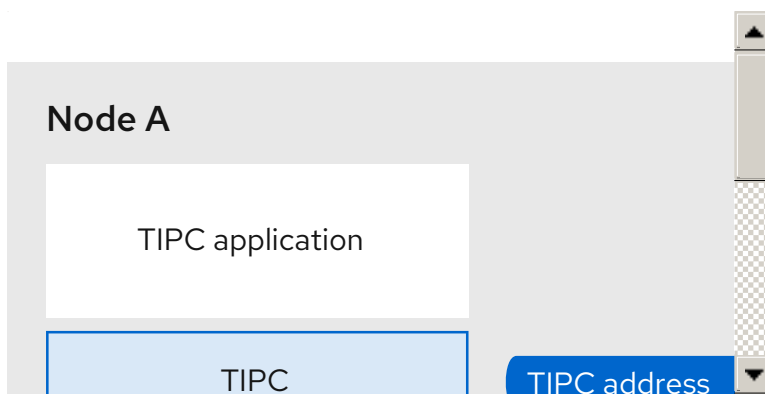
TIPC is a layer between applications using TIPC and a packet transport service (**bearer**), and spans the level of transport, network, and signaling link layers. However, TIPC can use a different transport protocol as bearer, so that, for example, a TCP connection can serve as a bearer for a TIPC signaling link.

TIPC supports the following bearers:

- Ethernet
- InfiniBand
- UDP protocol

TIPC provides a reliable transfer of messages between TIPC ports, that are the endpoints of all TIPC communication.

The following is a diagram of the TIPC architecture:



52.2. LOADING THE TIPC MODULE WHEN THE SYSTEM BOOTS

Before you can use the TIPC protocol, load the **tipc** kernel module. This section explains how to configure that RHEL loads this module automatically when the system boots.

Procedure

1. Create the **/etc/modules-load.d/tipc.conf** file with the following content:

```
tipc
```

2. Restart the **systemd-modules-load** service to load the module without rebooting the system:

```
# systemctl start systemd-modules-load
```

Verification steps

1. Use the following command to verify that RHEL loaded the **tipc** module:

```
# lsmod | grep tipc
tipc    311296 0
```

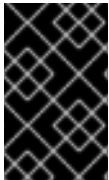
If the command shows no entry for the **tipc** module, RHEL failed to load it.

Additional resources

- **modules-load.d(5)** man page

52.3. CREATING A TIPC NETWORK

This section describes how to create a TIPC network.



IMPORTANT

The commands configure the TIPC network only temporarily. To permanently configure TIPC on a node, use the commands of this procedure in a script, and configure RHEL to execute that script when the system boots.

Prerequisites

- The **tipc** module has been loaded. For details, see [Loading the tipc module when the system boots](#)

Procedure

1. Optional: Set a unique node identity, such as a UUID or the node's host name:

```
# tipc node set identity host_name
```

The identity can be any unique string consisting of a maximum 16 letters and numbers.

You cannot set or change an identity after this step.

2. Add a bearer. For example, to use Ethernet as media and **enp0s1** device as physical bearer device, enter:

```
# tipc bearer enable media eth device enp1s0
```

3. Optional: For redundancy and better performance, attach further bearers using the command from the previous step. You can configure up to three bearers, but not more than two on the same media.
4. Repeat all previous steps on each node that should join the TIPC network.

Verification steps

1. Display the link status for cluster members:

```
# tipc link list
broadcast-link: up
5254006b74be:enp1s0-525400df55d1:enp1s0: up
```

This output indicates that the link between bearer **enp1s0** on node **5254006b74be** and bearer **enp1s0** on node **525400df55d1** is **up**.

2. Display the TIPC publishing table:

```
# tipc nametable show
Type   Lower   Upper   Scope  Port   Node
0      1795222054 1795222054 cluster 0      5254006b74be
0      3741353223 3741353223 cluster 0      525400df55d1
1      1         1       node   2399405586 5254006b74be
2      3741353223 3741353223 node    0      5254006b74be
```

- The two entries with service type **0** indicate that two nodes are members of this cluster.
- The entry with service type **1** represents the built-in topology service tracking service.
- The entry with service type **2** displays the link as seen from the issuing node. The range limit **3741353223** represents peer endpoint's address (a unique 32-bit hash value based on the node identity) in decimal format.

Additional resources

- **tipc-bearer(8)** man page
- **tipc-namespace(8)** man page

52.4. ADDITIONAL RESOURCES

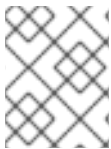
- Red Hat recommends to use other bearer level protocols to encrypt the communication between nodes based on the transport media. For example:
 - MACSec: See [Using MACsec to encrypt layer 2 traffic](#)
 - IPsec: See [Configuring a VPN with IPsec](#)
- For examples of how to use TIPC, clone the upstream GIT repository using the **git clone git://git.code.sf.net/p/tipc/tipcutils** command. This repository contains the source code of demos and test programs that use TIPC features. Note that this repository is not provided by Red Hat.
- **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/output/networking/tipc.html** provided by the **kernel-doc** package.

CHAPTER 53. AUTOMATICALLY CONFIGURING NETWORK INTERFACES IN PUBLIC CLOUDS USING NM-CLOUD-SETUP

Normally, a virtual machine (VM) has only one interface that is configurable by DHCP. However, some VMs might have multiple network interfaces, IP addresses, and IP subnets on one interface that is not configurable by DHCP. Also, administrators can reconfigure the network while the machine is running. The **nm-cloud-setup** utility automatically retrieves configuration information from the metadata server of the cloud service provider and updates the network configurations of VM in public clouds.

53.1. CONFIGURING AND PRE-DEPLOYING NM-CLOUD-SETUP

To enable and configure network interfaces in public clouds, run **nm-cloud-setup** as a timer and service. The following procedure describes how to use **nm-cloud-setup** for Amazon EC2.



NOTE

On Red Hat Enterprise Linux On Demand and AWS golden images, **nm-cloud-setup** is already enabled and no action is required.

Prerequisite

- A network connection exists.
- The connection uses DHCP.
By default, NetworkManager creates a connection profile which uses DHCP. If no profile was created because you set the **no-auto-default** parameter in **/etc/NetworkManager/NetworkManager.conf**, create this initial connection manually.

Procedure

1. Install the **nm-cloud-setup** package:

```
# yum install NetworkManager-cloud-setup
```

2. Create and run the snap-in file for the **nm-cloud-setup** service:

- a. Use the following command to start editing the snap-in file:

```
# systemctl edit nm-cloud-setup.service
```

It is important to either start the service explicitly or reboot the system to make configuration settings effective.

- b. Use the **systemd** snap-in file to configure the cloud provider in **nm-cloud-setup**. For example, to use Amazon EC2, enter:

```
[Service]
Environment=NM_CLOUD_SETUP_EC2=yes
```

You can set the following environment variables to enable the cloud provide you use:

- **NM_CLOUD_SETUP_AZURE** for Microsoft Azure

- **NM_CLOUD_SETUP_EC2** for Amazon EC2 (AWS)
- **NM_CLOUD_SETUP_GCP** for Google Cloud Platform(GCP)
- **NM_CLOUD_SETUP_ALIYUN** for Alibaba Cloud (Aliyun)

c. Save the file and quit the editor.

3. Reload the **systemd** configuration:

```
# systemctl daemon-reload
```

4. Enable and start the **nm-cloud-setup** service:

```
# systemctl enable --now nm-cloud-setup.service
```

5. Enable and start the **nm-cloud-setup** timer:

```
# systemctl enable --now nm-cloud-setup.timer
```

Additional resources

- **nm-cloud-setup(8)** man page
- [Configuring an Ethernet connection](#)