



Red Hat Enterprise Linux 9

配置和管理网络

管理网络接口和高级网络功能

法律通告

Copyright © 2024 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.

摘要

利用 Red Hat Enterprise Linux (RHEL) 的网络功能，您可以配置主机以满足您组织的网络和安全要求。例如：您可以配置绑定、VLAN、网桥、隧道和其他网络类型，以将主机连接到网络。IPsec 和 WireGuard 在主机和网络之间提供安全 VPN。RHEL 还支持高级网络功能，如基于策略的路由和多路径 TCP (MPTCP)。

目录

对红帽文档提供反馈	8
第 1 章 实施一致的网络接口命名	9
1.1. UDEV 设备管理器如何重命名网络接口	9
1.2. 网络接口命名策略	10
1.3. 网络接口命名方案	10
1.4. 切换到不同的网络接口命名方案	11
1.5. 在安装过程中自定义以太网接口的前缀	12
1.6. 使用 UDEV 规则配置用户定义的网络接口名称	13
1.7. 使用 SYSTEMD 链接文件配置用户定义的网络接口名称	15
1.8. 使用 SYSTEMD 链接文件为网络接口分配替代名称	17
第 2 章 配置以太网连接	19
2.1. 使用 NMCLI 配置以太网连接	19
2.2. 使用 NMCLI 互动编辑器配置以太网连接	22
2.3. 使用 NMTUI 配置以太网连接	24
2.4. 使用 CONTROL-CENTER 配置以太网连接	28
2.5. 使用 NM-CONNECTION-EDITOR 配置以太网连接	29
2.6. 使用 NMSTATECTL 配置带有静态 IP 地址的以太网连接	32
2.7. 使用网络 RHEL 系统角色和接口名称，使用静态 IP 地址配置以太网连接	34
2.8. 使用网络 RHEL 系统角色和设备路径，使用静态 IP 地址配置以太网连接	35
2.9. 使用 NMSTATECTL 配置带有动态 IP 地址的以太网连接	37
2.10. 使用网络 RHEL 系统角色和接口名称，使用动态 IP 地址配置以太网连接	39
2.11. 使用网络 RHEL 系统角色和设备路径，使用动态 IP 地址配置以太网连接	40
2.12. 按接口名称使用单个连接配置文件配置多个以太网接口	41
2.13. 使用 PCI ID 为多个以太网接口配置一个连接配置文件	42
第 3 章 配置网络绑定	44
3.1. 了解控制器和端口接口的默认行为	44
3.2. 依赖绑定模式的上游交换机配置	44
3.3. 使用 NMCLI 配置网络绑定	45
3.4. 使用 RHEL WEB 控制台配置网络绑定	47
3.5. 使用 NMTUI 配置网络绑定	50
3.6. 使用 NM-CONNECTION-EDITOR 配置网络绑定	54
3.7. 使用 NMSTATECTL 配置网络绑定	56
3.8. 使用 NETWORK RHEL 系统角色配置网络绑定	58
3.9. 创建网络绑定以便在不中断 VPN 的情况下在以太网和无线连接间进行切换	60
3.10. 不同的网络绑定模式	62
3.11. XMIT_HASH_POLICY BONDING 参数	64
第 4 章 配置网络团队 (TEAM)	66
4.1. 将网络团队配置迁移到网络绑定	66
4.2. 了解控制器和端口接口的默认行为	69
4.3. 了解 TEAMD 服务、运行程序和 LINK-WATCHERS	69
4.4. 使用 NMCLI 配置网络团队	70
4.5. 使用 RHEL WEB 控制台配置网络团队	73
4.6. 使用 NM-CONNECTION-EDITOR 配置网络团队	76
第 5 章 配置 VLAN 标记	79
5.1. 使用 NMCLI 配置 VLAN 标记	79
5.2. 使用 NMCLI 配置嵌套 VLAN	81
5.3. 使用 RHEL WEB 控制台配置 VLAN 标记	83

5.4. 使用 NMTUI配置 VLAN 标记	84
5.5. 使用 NM-CONNECTION-EDITOR 配置 VLAN 标记	88
5.6. 使用 NMSTATECTL配置 VLAN 标记	90
5.7. 使用 NETWORK RHEL 系统角色配置 VLAN 标记	92
第 6 章 配置网络桥接	94
6.1. 使用 NMCLI配置网络桥接	94
6.2. 使用 RHEL WEB 控制台配置网桥	97
6.3. 使用 NMTUI配置网桥	99
6.4. 使用 NM-CONNECTION-EDITOR 配置网桥	103
6.5. 使用 NMSTATECTL配置网络桥接	105
6.6. 使用 NETWORK RHEL 系统角色配置网桥	107
第 7 章 设置 IPSEC VPN	110
7.1. 使用 CONTROL-CENTER 配置 VPN 连接	110
7.2. 使用 NM-CONNECTION-EDITOR 配置 VPN 连接	113
7.3. 配置自动检测和使用 ESP 硬件卸载来加速 IPSEC 连接	116
7.4. 在绑定中配置 ESP 硬件卸载以加快 IPSEC 连接	117
7.5. 使用 NMSTATECTL 配置基于 IPSEC 的 VPN 连接	119
第 8 章 设置 WIREGUARD VPN	130
8.1. WIREGUARD 使用的协议和原语	130
8.2. WIREGUARD 如何使用隧道 IP 地址、公钥和远程端点	130
8.3. 使用 NAT 和防火墙后面的 WIREGUARD 客户端	131
8.4. 创建在 WIREGUARD 连接中使用的私钥和公钥	131
8.5. 使用 NMCLI 配置 WIREGUARD 服务器	132
8.6. 使用 NMTUI 配置 WIREGUARD 服务器	135
8.7. 使用 RHEL WEB 控制台配置 WIREGUARD 服务器	138
8.8. 使用 NM-CONNECTION-EDITOR 配置 WIREGUARD 服务器	140
8.9. 使用 WG-QUICK 服务配置 WIREGUARD 服务器	142
8.10. 使用命令行在 WIREGUARD 服务器上配置 FIREWALLD	144
8.11. 使用 RHEL WEB 控制台在 WIREGUARD 服务器上配置 FIREWALLD	145
8.12. 使用图形界面在 WIREGUARD 服务器上配置 FIREWALLD	146
8.13. 使用 NMCLI 配置 WIREGUARD 客户端	146
8.14. 使用 NMTUI 配置 WIREGUARD 客户端	149
8.15. 使用 RHEL WEB 控制台配置 WIREGUARD 客户端	153
8.16. 使用 NM-CONNECTION-EDITOR 配置 WIREGUARD 客户端	155
8.17. 使用 WG-QUICK 服务配置 WIREGUARD 客户端	158
第 9 章 配置 IP 隧道	161
9.1. 使用 NMCLI 配置 IPIP 隧道来封装 IPV4 数据包中的 IPV4 流量	161
9.2. 使用 NMCLI 配置 GRE 隧道来封装 IPV4 数据包中的第 3 层流量	164
9.3. 配置 GRETAP 隧道来通过 IPV4 传输以太网帧	166
9.4. 其他资源	169
第 10 章 使用 VXLAN 为虚拟机创建虚拟第 2 层域	170
10.1. VXLAN 的优点	170
10.2. 在主机上配置以太网接口	171
10.3. 创建附加了 VXLAN 的网桥	171
10.4. 在带有现有网桥的 LIBVIRT 中创建一个虚拟网络	173
10.5. 配置虚拟机以使用 VXLAN	174
第 11 章 管理 WIFI 连接	176
11.1. 支持的 WIFI 安全类型	176
11.2. 使用 NMCLI连接到 WIFI 网络	176

11.3. 使用 GNOME 系统菜单连接到 WIFI 网络	178
11.4. 使用 GNOME 设置应用程序连接到 WIFI 网络	179
11.5. 使用 NMTUI配置 WIFI 连接	180
11.6. 使用 NM-CONNECTION-EDITOR 配置 WIFI 连接	182
11.7. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的 WIFI 连接	183
11.8. 使用 NMCLI在现有配置集中配置带有 802.1X 网络身份验证的 WIFI 连接	185
11.9. 手动设置无线规范域	186
第 12 章 将 RHEL 配置为 WPA2 或 WPA3 个人访问令牌	188
第 13 章 使用 MACSEC 加密同一物理网络中的第 2 层流量	191
13.1. 使用 NMCLI配置 MACSEC 连接	191
13.2. 使用 NMSTATECTL配置 MACSEC 连接	192
13.3. 其他资源	195
第 14 章 开始使用 IPVLAN	196
14.1. IPVLAN 模式	196
14.2. IPVLAN 和 MACVLAN 的比较	196
14.3. 使用 IPROUTE2 创建和配置 IPVLAN 设备	196
第 15 章 配置 NETWORKMANAGER 以忽略某些设备	198
15.1. 使用 NMCLI配置回环接口	198
15.2. 永久将设备配置为网络管理器 (NETWORKMANAGER) 中非受管设备	199
15.3. 将设备临时配置为在 NETWORKMANAGER 中不被管理	200
第 16 章 创建 DUMMY 接口	202
16.1. 使用 NMCLI使用 IPV4 和 IPV6 地址创建 DUMMY 接口	202
第 17 章 使用 NETWORKMANAGER 为特定连接禁用 IPV6	203
17.1. 使用 NMCLI在连接中禁用 IPV6	203
第 18 章 更改主机名	205
18.1. 使用 NMCLI更改主机名	205
18.2. 使用 HOSTNAMECTL 更改主机名	205
第 19 章 配置 NETWORKMANAGER DHCP 设置	207
19.1. 更改 NETWORKMANAGER 的 DHCP 客户端	207
19.2. 配置 NETWORKMANAGER 连接的 DHCP 行为	207
第 20 章 使用 NETWORKMANAGER 的一个分配程序脚本运行 HCLIENT EXIT HOOKS	209
20.1. NETWORKMANAGER 分配程序脚本的概念	209
20.2. 创建运行 DHCLIENT EXIT HOOKS 的 NETWORKMANAGER 分配程序脚本	209
第 21 章 手动配置 /ETC/RESOLV.CONF 文件	211
21.1. 在 NETWORKMANAGER 配置中禁用 DNS 处理	211
21.2. 使用符号链接替换 /ETC/RESOLV.CONF 来手动配置 DNS 设置	212
第 22 章 配置 DNS 服务器顺序	213
22.1. NETWORKMANAGER 如何在 /ETC/RESOLV.CONF 中对 DNS 服务器进行排序	213
22.2. 设置 NETWORKMANAGER 范围默认 DNS 服务器优先级值	214
22.3. 设置网络管理器连接的 DNS 优先级	214
第 23 章 在不同域中使用不同的 DNS 服务器	216
23.1. 在 NETWORKMANAGER 中使用 DNSMASQ 将特定域的 DNS 请求发送到所选的 DNS 服务器	216
23.2. 使用 NETWORKMANAGER 中的 SYSTEMD-RESOLVED 将特定域的 DNS 请求发送到所选 DNS 服务器	218
第 24 章 管理默认网关设置	221

24.1. 使用 NMCLI在现有连接上设置默认网关	221
24.2. 使用 NMCLI 交互模式在现有连接上设置默认网关	222
24.3. 使用 NM-CONNECTION-EDITOR 在现有连接上设置默认网关	223
24.4. 使用 CONTROL-CENTER 在现有连接上设置默认网关	224
24.5. 使用 NMSTATECTL在现有连接上设置默认网关	225
24.6. 使用 NETWORK RHEL 系统角色在现有连接上设置默认网关	226
24.7. NETWORKMANAGER 如何管理多个默认网关	228
24.8. 配置 NETWORKMANAGER 以避免使用特定配置集提供默认网关	229
24.9. 修复因为多个默认网关导致的意外路由行为	229
第 25 章 配置静态路由	232
25.1. 需要静态路由的网络示例	232
25.2. 如何使用 NMCLI 工具配置静态路由	234
25.3. 使用 NMCLI配置静态路由	235
25.4. 使用 NMTUI配置静态路由	236
25.5. 使用 CONTROL-CENTER 配置静态路由	238
25.6. 使用 NM-CONNECTION-EDITOR 配置静态路由	239
25.7. 使用 NMCLI 交互模式配置静态路由	240
25.8. 使用 NMSTATECTL配置静态路由	242
25.9. 使用 NETWORK RHEL 系统角色配置静态路由	243
第 26 章 配置基于策略的路由以定义其他路由	246
26.1. 使用 NMCLI将特定子网的流量路由到不同的默认网关	246
26.2. 使用 NETWORK RHEL 系统角色将特定子网的流量路由到不同的默认网关	249
第 27 章 在不同的接口上重复使用相同的 IP 地址	254
27.1. 在不同接口上永久重复使用相同的 IP 地址	254
27.2. 在不同接口中临时重复使用相同的 IP 地址	255
27.3. 其他资源	256
第 28 章 在隔离的 VRF 网络内启动服务	257
28.1. 配置 VRF 设备	257
28.2. 在隔离的 VRF 网络内启动服务	258
第 29 章 在 NETWORKMANAGER 连接配置文件中配置 ETHTOOL 设置	261
29.1. 使用 NMCLI配置 ETHTOOL OFFLOAD 功能	261
29.2. 使用 NETWORK RHEL 系统角色配置 ETHTOOL OFFLOAD 功能	262
29.3. 使用 NMCLI配置 ETHTOOL COALESCE 设置	263
29.4. 使用 NETWORK RHEL 系统角色配置 ETHTOOL COALESCE 设置	264
29.5. 使用 NMCLI增加环缓冲的大小, 以减少数据包丢弃率	266
29.6. 使用 NETWORK RHEL 系统角色增加环缓冲的大小, 以减少数据包丢弃率	267
29.7. 使用 NMCLI配置 ETHTOOL 频道设置	269
第 30 章 网络管理器调试介绍	271
30.1. NETWORKMANAGER 重新应用方法的简介	271
30.2. 设置 NETWORKMANAGER 日志级别	273
30.3. 使用 NMCLI在运行时临时设置日志级别	274
30.4. 查看 NETWORKMANAGER 日志	275
30.5. 调试级别和域	275
第 31 章 使用 LLDP 来调试网络配置问题	276
31.1. 使用 LLDP 信息调试不正确的 VLAN 配置	276
第 32 章 LINUX 流量控制	278
32.1. 排队规则概述	278

32.2. 连接跟踪简介	278
32.3. 使用 TC 工具检查网络接口的 QDISCS	279
32.4. 更新默认的 QDISC	279
32.5. 使用 TC 工具临时设置网络接口的当前 QDISC	280
32.6. 使用 NETWORKMANAGER 永久设置网络接口的当前 QDISC	281
32.7. 使用 TC-CTINFO 工具配置数据包的速率限制	281
32.8. RHEL 中可用的 QDISCS	285
第 33 章 使用带有存储在文件系统上证书的 802.1X 标准将 RHEL 客户端认证到网络	288
33.1. 使用 NMCLI 在现有以太网连接中配置 802.1X 网络身份验证	288
33.2. 使用 NMSTATECTL 配置带有 802.1X 网络身份验证的静态以太网连接	289
33.3. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的静态以太网连接	291
33.4. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的 WIFI 连接	293
第 34 章 使用带有 FREERADIUS 后端的 HOSTAPD 为 LAN 客户端设置 802.1X 网络身份验证服务	295
34.1. 先决条件	295
34.2. 在 AUTHENTICATOR 中设置桥接	295
34.3. FREERADIUS 的证书要求	296
34.4. 在 FREERADIUS 服务器上, 出于测试目的创建一组证书	297
34.5. 配置 FREERADIUS 以验证使用 EAP 保护的客户端	299
34.6. 在有线网络中将 HOSTAPD 配置为验证器	302
34.7. 针对 FREERADIUS 服务器或验证器测试 EAP-TTLS 身份验证	304
34.8. 针对 FREERADIUS 服务器或验证器测试 EAP-TLS 身份验证	306
34.9. 根据 HOSTAPD 身份验证事件阻止和允许流量	307
第 35 章 多路径 TCP 入门	310
35.1. 了解 MPTCP	310
35.2. 准备 RHEL 启用 MPTCP 支持	310
35.3. 使用 IPROUTE2 为 MPTCP 应用程序临时配置和启用多个路径	311
35.4. 为 MPTCP 应用程序永久配置多路径	313
35.5. 监控 MPTCP 子流	315
35.6. 在内核中禁用多路径 TCP	317
第 36 章 管理 MPTCPD 服务	318
36.1. 配置 MPTCPD	318
36.2. 使用 MPTCPIZE 工具管理应用程序	318
36.3. 使用 MPTCPIZE 程序为服务启用 MPTCP 套接字	319
第 37 章 密钥文件格式的 NETWORKMANAGER 连接配置文件	320
37.1. NETWORKMANAGER 配置文件的密钥文件格式	320
37.2. 使用 NMCLI 在离线模式下创建密钥文件连接配置集	321
37.3. 手动创建 KEYFILE 格式的 NETWORKMANAGER 配置文件	323
37.4. IFCFG 和 KEYFILE 格式的使用配置文件重命名接口的差异	324
37.5. 将 NETWORKMANAGER 配置集从 IFCFG 迁移到 KEYFILE 格式	325
第 38 章 SYSTEMD 网络目标和服务	326
38.1. NETWORK 和 NETWORK-ONLINE SYSTEMD TARGET 的不同	326
38.2. NETWORKMANAGER-WAIT-ONLINE 概述	326
38.3. 将 SYSTEMD 服务配置为在网络已启动后再启动	327
第 39 章 NMSTATE 简介	328
39.1. 在 PYTHON 应用程序中使用 LIBNMSTATE 库	328
39.2. 使用 NMSTATECTL 更新当前的网络配置	328
39.3. NMSTATE SYSTEMD 服务	329
39.4. 网络 RHEL 系统角色的网络状态	329

39.5. 其他资源	330
第 40 章 捕获网络数据包	331
40.1. 使用 XDPDUMP 捕获包括 XDP 程序丢弃的数据包在内的网络数据包	331
40.2. 其他资源	331
第 41 章 了解 RHEL 9 中的 EBPf 网络功能	332
41.1. RHEL 9 中网络 EBPf 功能概述	332
41.2. 使用网卡 RHEL 9 中的 XDP 功能概述	334
第 42 章 使用 BPF 编译器集合进行网络追踪	337
42.1. 安装 BCC-TOOLS 软件包	337
42.2. 显示添加到内核的接受队列中的 TCP 连接	337
42.3. 追踪出去的 TCP 连接尝试	338
42.4. 测量出站 TCP 连接的延迟	338
42.5. 显示被内核丢弃的 TCP 数据包和片段详情	339
42.6. 追踪 TCP 会话	339
42.7. 追踪 TCP 重新传输	340
42.8. 显示 TCP 状态更改信息	341
42.9. 聚合发送到特定子网的 TCP 流量	341
42.10. 通过 IP 地址和端口显示网络吞吐量	342
42.11. 追踪已建立的 TCP 连接	343
42.12. 追踪 IPV4 和 IPV6 侦听尝试	343
42.13. 软中断的服务时间概述	344
42.14. 总结在网络接口上数据包大小和计数	344
42.15. 其他资源	345
第 43 章 配置网络设备以接受来自所有 MAC 地址的流量	346
43.1. 临时配置设备以接受所有流量	346
43.2. 使用 NMCLI 永久配置网络设备以接受所有流量	346
43.3. 使用 NMSTATECTL 永久配置网络设备以接受所有流量	347
第 44 章 使用 NMCLI 对网络接口进行镜像	349
第 45 章 使用 NMSTATE-AUTOCONF 自动配置使用 LLDP 的网络状态	351
45.1. 使用 NMSTATE-AUTOCONF 来自动配置网络接口	351
第 46 章 配置 802.3 链路设置	354
46.1. 使用 NMCLI 工具配置 802.3 链路设置	354
第 47 章 DPDK 入门	356
47.1. 安装 DPDK 软件包	356
47.2. 其他资源	356
第 48 章 TIPC 入门	357
48.1. TIPC 的构架	357
48.2. 系统引导时载入 TIPC 模块	357
48.3. 创建 TIPC 网络	358
48.4. 其他资源	359
第 49 章 使用 NM-CLOUD-SETUP 在公有云中自动配置网络接口	360
49.1. 配置和预部署 NM-CLOUD-SETUP	360
49.2. 了解 RHEL EC2 实例中 IMDSV2 和 NM-CLOUD-SETUP 的角色	361

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 [Jira](#) 网站。
2. 点顶部导航栏中的 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 实施一致的网络接口命名

udev 设备管理器在 Red Hat Enterprise Linux 中实施一致的设备命名。设备管理器支持不同的命名方案，默认情况下，根据固件、拓扑和位置信息分配固定的名称。

如果没有一致的设备命名，Linux 内核通过组合固定的前缀和索引来为网络接口分配名称。当内核初始化网络设备时，索引会增加。例如：**eth0** 代表启动时探测到的第一个以太网设备。如果您在系统中添加了另一个网络接口控制器，则内核设备名称的分配不再是固定的，因为重启后，设备可以以不同的方式初始化。在这种情况下，内核可以以不同的方式命名设备。

要解决这个问题，**udev** 分配一致的设备名称。它有以下优点：

- 设备名称在重启时是稳定的。
- 即使添加或删除硬件，设备名称也会保持不变。
- 因此，有问题的硬件可以被无缝地替换。
- 网络命名是无状态的，不需要显式的配置文件。



警告

通常，红帽不支持禁用了一致设备命名的系统。有关例外情况，请参阅 [设置 net.ifnames=0 安全吗](#) 解决方案。

1.1. UDEV 设备管理器如何重命名网络接口

要为网络接口实施一致的命名方案，**udev** 设备管理器按列出的顺序处理以下规则文件：

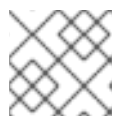
1. 可选：`/usr/lib/udev/rules.d/60-net.rules`

只有在安装 **initscripts-rename-device** 软件包时，该文件才存在。定义了弃用的 `/usr/lib/udev/rename_device` 助手工具的 `/usr/lib/udev/rules.d/60-net.rules` 文件在 `/etc/sysconfig/network-scripts/ifcfg-*` 文件中查找 **HWADDR** 参数。如果变量中设置的值与接口的 MAC 地址匹配，则助手工具会将接口重命名为 `ifcfg` 文件的 **DEVICE** 参数中设置的名称。

如果系统只使用 keyfile 格式的 NetworkManager 连接配置文件，则 **udev** 会跳过这一步。

2. 只在 Dell 系统上：`/usr/lib/udev/rules.d/71-biosdevname.rules`

只有安装了 **biosdevname** 软件包，该文件才存在，如果没有在之前的步骤中重命名，定义了 **biosdevname** 工具的规则文件会根据其命名策略重命名接口。



注意

仅在 Dell 系统上安装和使用 **biosdevname**。

3. `/usr/lib/udev/rules.d/75-net-description.rules`

此文件定义 **udev** 如何检查网络接口，并在 `udev-internal` 变量中设置属性。然后，这些变量被 `/usr/lib/udev/rules.d/80-net-setup-link.rules` 文件在下一步中处理。其中一些属性可以是未定义。

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

此文件调用 `udev` 服务的 `net_setup_link` 内置，并且 `udev` 根据 `/usr/lib/systemd/network/99-default.link` 文件中 `NamePolicy` 参数中策略的顺序来重命名接口。详情请查看 [网络接口命名策略](#)。

如果没有应用任何策略，则 `udev` 不会重命名接口。

其他资源

- [为什么 systemd 网络接口名称在主 RHEL 版本之间有所不同](#) 解决方案

1.2. 网络接口命名策略

默认情况下，`udev` 设备管理器使用 `/usr/lib/systemd/network/99-default.link` 文件来确定在重命名接口时要应用哪个设备命名策略。此文件中的 `NamePolicy` 参数定义 `udev` 使用哪些策略以及按什么顺序：

```
NamePolicy=keep kernel database onboard slot path
```

下表根据哪个策略与 `NamePolicy` 参数指定的首先匹配，描述了 `udev` 的不同操作：

policy	描述	名称示例
keep	如果设备在用户空间中已有分配的名称，则 <code>udev</code> 不会重命名此设备。例如，如果在创建设备期间或通过重命名操作分配了名称，则会出现这种情况。	
kernel	如果内核表示设备名称是可预测的，则 <code>udev</code> 不会重命名这个设备。	<code>lo</code>
database	此策略根据 <code>udev</code> 硬件数据库中的映射分配名称。详情请查看 hwdb (7) 手册页。	<code>idrac</code>
onboard	设备名称包含固件或者 BIOS 提供的索引号，用于板上的设备。	<code>eno1</code>
slot	设备名称包含固件或 BIOS 提供的 PCI Express (PCIe) 热插拔 slot-index 号。	<code>ens1</code>
path	设备名称包含硬件连接器的物理位置。	<code>enp1s0</code>
mac	设备名称包含 MAC 地址。默认情况下，Red Hat Enterprise Linux 不使用此策略，但管理员可以启用它。	<code>enx525400d5e0fb</code>

其他资源

- [udev 设备管理器如何重命名网络接口](#)
- [systemd.link\(5\) 手册页](#)

1.3. 网络接口命名方案

udev 设备管理器使用设备驱动程序提供的某些稳定接口属性来生成一致的设备名称。

如果新的 **udev** 版本更改了服务为特定接口创建名称的方式，红帽会添加新的方案版本，并在 **systemd.net-naming-scheme (7)** 手册页中记录了详情。默认情况下，Red Hat Enterprise Linux (RHEL) 9 使用 **rhel-9.0** 命名方案，即使您安装或升级到更新的 RHEL 次版本。

为了防止新驱动程序为网络接口提供更多或其他属性，**rhel-net-naming-sysattr** 软件包提供了 **/usr/lib/udev/hwdb.d/50-net-naming-sysattr-allowlist.hwdb** 数据库。此数据库定义 **udev** 服务可用于创建网络接口名称的 **sysfs** 值。数据库中的条目也会被方案版本进行版本控制和影响。



注意

在 RHEL 9.4 及更高版本中，您还可以使用所有 **rhel-8463** 命名方案。

如果要使用默认方案，您可以 [切换网络接口命名方案](#)。

有关不同设备类型和平台的命名方案的详情，请查看 **systemd.net-naming-scheme (7)** 手册页。

1.4. 切换到不同的网络接口命名方案

默认情况下，Red Hat Enterprise Linux (RHEL) 9 使用 **rhel-9.0** 命名方案，即使您安装或升级到更新的 RHEL 次版本。虽然大多数情况下默认命名方案适合，但可能会有原因切换到不同的方案版本，例如：

- 将新方案添加到接口名称时，如果设备添加了附加属性，如插槽号，则它可以帮助更好地识别设备。
- 新方案可以防止 **udev** 回退到内核分配的设备名称(**eth***)。如果驱动程序没有为两个或多个接口提供足够的唯一属性，以便为它们生成唯一名称。

先决条件

- 您可以访问服务器的控制台。

步骤

1. 列出网络接口：

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

记录接口的 MAC 地址。

2. 可选：显示网络接口的 **ID_NET_NAMING_SCHEME** 属性来识别 RHEL 当前使用的命名方案：

```
# udevadm info --query=property --property=ID_NET_NAMING_SCHEME
/sys/class/net/eno1'
ID_NET_NAMING_SCHEME=rhel-9.0
```

请注意，这个属性在 **lo** loopback 设备上不可用。

3. 将 **net.naming-scheme=<scheme>** 选项附加到所有安装的内核的命令行中，例如：

```
# grubby --update-kernel=ALL --args=net.naming-scheme=rhel-9.4
```

4. 重启系统。

```
# reboot
```

5. 根据您记录的 MAC 地址，识别因不同的命名方案而更改的网络接口的新名称：

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

切换方案后，在这个示例中，**udev** 名称为 MAC 地址为 **00:00:5e:00:53:1a eno1np0** 的设备，而在之前被命名为 **eno1**。

6. 确定哪个 NetworkManager 连接配置集使用与之前名称的接口：

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

7. 将连接配置文件中的 **connection.interface-name** 属性设置为新接口名称：

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

8. 重新激活连接配置文件：

```
# nmcli connection up example_profile
```

验证

- 通过显示网络接口的 **ID_NET_NAMING_SCHEME** 属性来识别 RHEL 现在使用的命名方案：

```
# udevadm info --query=property --property=ID_NET_NAMING_SCHEME
/sys/class/net/eno1np0'
ID_NET_NAMING_SCHEME=_rhel-9.4
```

其他资源

- [网络接口命名方案](#)

1.5. 在安装过程中自定义以太网接口的前缀

如果您不想将默认的设备命名策略用于以太网接口，您可以在 Red Hat Enterprise Linux (RHEL) 安装过程中设置一个自定义设备前缀。



重要

只有您在 RHEL 安装过程中设置了前缀时，红帽才支持带有自定义以太网前缀的系统。不支持在已部署的系统上使用 `prefixdevname` 工具。

如果您在安装过程中设置了设备前缀，则 `udev` 服务会在安装后为以太网接口使用 `<prefix><index>` 格式。例如，如果您设置了前缀 `net`，服务会将名称 `net0`、`net1` 等分配给以太网接口。

`udev` 服务将索引附加到自定义前缀，并保留已知以太网接口的索引值。如果您添加了一个接口，`udev` 会为新接口分配一个比之前分配的索引值大的索引值。

先决条件

- 前缀由 ASCII 字符组成。
- 前缀是一个字母数字字符串。
- 前缀少于 16 个字符。
- 前缀不会与任何其他已知的网络接口前缀冲突，如 `eth`、`eno`、`ens`、`em`。

步骤

1. 引导 Red Hat Enterprise Linux 安装介质。
2. 在引导管理器中，按照以下步骤操作：
 - a. 选择 **Install Red Hat Enterprise Linux <version>** 条目。
 - b. 按 **Tab** 编辑条目。
 - c. 将 `net.ifnames.prefix=<prefix>` 追加到在内核选项中。
 - d. 按 **Enter** 键启动安装程序。
3. 安装 Red Hat Enterprise Linux。

验证

- 要验证接口名称，显示网络接口：

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

其他资源

- [执行标准的 RHEL 9 安装](#)

1.6. 使用 UDEV 规则配置用户定义的网络接口名称

您可以使用 `udev` 规则来实现反映您机构的要求的自定义网络接口名称。

流程

1. 识别您要重命名的网络接口：

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

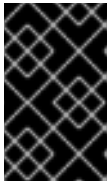
记录接口的 MAC 地址。

2. 显示接口的设备类型 ID：

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

3. 创建 `/etc/udev/rules.d/70-persistent-net.rules` 文件，并为您要重命名的每个接口添加一个规则：

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



重要

如果您在引导过程中需要一致的设备名称，则只使用 `70-persistent-net.rules` 作为文件名。如果您重新生成 RAM 磁盘镜像，则 `dracut` 工具会在 `initrd` 镜像中添加一个具有此名称的文件。

例如，使用以下规则将 MAC 地址为 `00:00:5e:00:53:1a` 的接口重命名为 `provider0`：

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

4. 可选：重新生成 `initrd` RAM 磁盘镜像：

```
# dracut -f
```

只有在 RAM 磁盘中需要网络功能时才需要这一步。例如，如果根文件系统存储在网络设备上，如 iSCSI，则会出现这种情况。

5. 确定哪个 NetworkManager 连接配置文件使用您要重命名的接口：

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

6. 在连接配置文件中取消 `connection.interface-name` 属性的设置：

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

7. 临时配置连接配置文件，以匹配新的和以前的接口名称：

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

8. 重启系统：

```
# reboot
```

9. 验证具有您在链接文件中指定的 MAC 地址的设备是否已被重命名为 **provider0**：

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

10. 配置连接配置文件，以仅匹配新接口名称：

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

现在，您已经从连接配置文件中删除了旧接口名称。

11. 重新激活连接配置文件：

```
# nmcli connection up example_profile
```

其他资源

- [udev\(7\) 手册页](#)

1.7. 使用 SYSTEMD 链接文件配置用户定义的网络接口名称

您可以使用 **systemd** 链接文件来实现反映您机构要求的自定义网络接口名称。

先决条件

- 您必须满足这些条件之一：NetworkManager 不管理这个接口，或者对应的连接配置文件使用 [keyfile 格式](#)。

流程

1. 识别您要重命名的网络接口：

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

记录接口的 MAC 地址。

2. 如果其不存在，请创建 `/etc/systemd/network/` 目录：

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

- 对于您要重命名的每个接口，请在 `/etc/systemd/network/` 目录中创建一个具有以下内容的 `70-*.link` 文件：

```
[Match]
MACAddress=<MAC_address>

[Link]
Name=<new_interface_name>
```



重要

使用一个具有 **70** 前缀的文件名，使文件名与 **udev** 规则的解决方案一致。

例如，使用以下内容创建 `/etc/systemd/network/70-provider0.link` 文件，将 MAC 地址为 `00:00:5e:00:53:1a` 的接口重命名为 **provider0**：

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

[Link]
Name=provider0
```

- 可选：重新生成 **initrd** RAM 磁盘镜像：

```
# dracut -f
```

只有在 RAM 磁盘中需要网络功能时才需要这一步。例如，如果根文件系统存储在网络设备上，如 iSCSI，则会出现这种情况。

- 确定哪个 NetworkManager 连接配置文件使用您要重命名的接口：

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

- 在连接配置文件中取消 `connection.interface-name` 属性的设置：

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

- 临时配置连接配置文件，以匹配新的和以前的接口名称：

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

- 重启系统：

```
# reboot
```

- 验证具有您在链接文件中指定的 MAC 地址的设备是否已被重命名为 **provider0**：

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

- 配置连接配置文件，以仅匹配新接口名称：

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

现在，您已经从连接配置文件中删除了旧接口名称。

- 重新激活连接配置文件。

```
# nmcli connection up example_profile
```

其他资源

- [systemd.link\(5\) 手册页](#)

1.8. 使用 SYSTEMD 链接文件为网络接口分配替代名称

通过替代接口命名，内核可以为网络接口分配额外的名称。您可以以同样的方式将这些替代名称在需要网络接口名称的命令中用作普通的接口名称。

先决条件

- 对于替代名称，您必须使用 ASCII 字符。
- 备用名称必须小于 128 个字符。

流程

- 显示网络接口名称及其 MAC 地址：

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

记录您要为其分配替代名称的接口的 MAC 地址。

- 如果其不存在，请创建 `/etc/systemd/network/` 目录：

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

- 对于必须有替代名称的每个接口，请在 `/etc/systemd/network/` 目录中创建一个具有以下内容的 `*.link` 文件：

```
[Match]
MACAddress=<MAC_address>
```

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

例如，创建具有以下内容的 `/etc/systemd/network/70-altname.link` 文件，来将 **provider** 作为一个替代名称分配给 MAC 地址为 **00:00:5e:00:53:1a** 的接口：

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

[Link]
AlternativeName=provider
```

4. 重新生成 **initrd** RAM 磁盘镜像：

```
# dracut -f
```

5. 重启系统：

```
# reboot
```

验证

- 使用替代接口名称。例如，显示替代名称为 **provider** 的设备的 IP 地址设置：

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

其他资源

- [接口命名方案中的 alternativesNamesPolicy 是什么？](#)

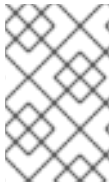
第 2 章 配置以太网连接

NetworkManager 为主机上安装的每个以太网适配器创建一个连接配置文件。默认情况下，此配置文件将 DHCP 用于 IPv4 和 IPv6 连接。修改此自动创建的配置文件，或在以下情况下添加一个新配置文件：

- 网络需要自定义设置，如静态 IP 地址配置。
- 您需要多个配置文件，因为主机在不同的网络间漫游。

Red Hat Enterprise Linux 为管理员提供不同的选项来配置以太网连接。例如：

- 在命令行中使用 **nmcli** 配置连接。
- 使用 **nmtui** 在基于文本的用户界面中配置连接。
- 使用 GNOME Settings 菜单或 **nm-connection-editor** 应用程序在图形界面中配置连接。
- 使用 **nmstatectl** 通过 Nmstate API 配置连接。
- 使用 RHEL 系统角色自动化一个或多个主机上连接的配置。



注意

如果要对运行在 Microsoft Azure 云中的主机手动配置以太网连接，请禁用 **cloud-init** 服务或将其配置为忽略从云环境检索到的网络设置。否则，**cloud-init** 将在下次重启时覆盖您手动配置的网络设置。

2.1. 使用 NMCLI配置以太网连接

如果您通过以太网将主机连接到网络，您可以使用 **nmcli** 工具在命令行上管理连接的设置。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。

流程

1. 列出 NetworkManager 连接配置文件：

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

默认情况下，NetworkManager 为主机中的每个 NIC 创建一个配置文件。如果您计划仅将这个 NIC 连接到特定的网络，请调整自动创建的配置文件。如果您计划将这个 NIC 连接到具有不同设置的网络，请为每个网络创建单独的配置文件。

2. 如果要创建额外的连接配置文件，请输入：

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

跳过此步骤以修改现有配置文件。

3. 可选：重命名连接配置文件：

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

在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。

4. 显示连接配置文件的当前设置：

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

5. 配置 IPv4 设置：

- 要使用 DHCP，请输入：

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

如果 `ipv4.method` 已设置为 `auto`（默认），请跳过这一步。

- 要设置静态 IPv4 地址、网络掩码、默认网关、DNS 服务器和搜索域，请输入：

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

6. 配置 IPv6 设置：

- 要使用无状态地址自动配置(SLAAC)，请输入：

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

如果 `ipv6.method` 已设置为 `auto`（默认），请跳过这一步。

- 要设置静态 IPv6 地址、网络掩码、默认网关、DNS 服务器和搜索域，请输入：

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

7. 要在配置文件中自定义其他设置，请使用以下命令：

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

将值用空格或分号括在引号中。

8. 激活配置文件：

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


验证

1. 显示 NIC 的 IP 设置：

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

2. 显示 IPv4 默认网关：

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

3. 显示 IPv6 默认网关：

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

4. 显示 DNS 设置：

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

5. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

故障排除

- 验证网线是否插入到主机和交换机。
- 检查链路失败是否只存在于此主机上，或者也存在于其它连接到同一交换机的主机上。
- 验证网络电缆和网络接口是否如预期工作。执行硬件诊断步骤并替换有缺陷的电缆和网络接口卡。
- 如果磁盘中的配置与设备中的配置不匹配，则启动或重启 NetworkManager 会创建一个代表该设备的配置的内存连接。有关详情以及如何避免这个问题，请参阅 [NetworkManager 服务重启后会复制一个连接](#) 解决方案。

其他资源

- [nm-settings\(5\)](#) 手册页

2.2. 使用 NMCLI 互动编辑器配置以太网连接

如果您通过以太网将主机连接到网络，您可以使用 **nmcli** 工具在命令行上管理连接的设置。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。

步骤

1. 列出 NetworkManager 连接配置文件：

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

默认情况下，NetworkManager 为主机中的每个 NIC 创建一个配置文件。如果您计划仅将这个 NIC 连接到特定的网络，请调整自动创建的配置文件。如果您计划将这个 NIC 连接到具有不同设置的网络，请为每个网络创建单独的配置文件。

2. 以交互模式启动 **nmcli**：

- 要创建额外的连接配置文件，请输入：

```
# nmcli connection edit type ethernet con-name "<connection-name>"
```

- 要修改现有的连接配置文件，请输入：

```
# nmcli connection edit con-name "<connection-name>"
```

3. 可选：重命名连接配置文件：

```
nmcli> set connection.id Internal-LAN
```

在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。

不要使用引号来设置包含空格的 ID，以避免 **nmcli** 将引号作为名称的一部分。例如，要将 **Example Connection** 设置为 ID，请输入 **set connection.id Example Connection**。

4. 显示连接配置文件的当前设置：

```
nmcli> print
...
connection.interface-name:  enp1s0
connection.autoconnect:    yes
ipv4.method:                auto
ipv6.method:                auto
...
```

5. 如果创建新连接配置文件，请设置网络接口：

```
nmcli> set connection.interface-name enp1s0
```

6. 配置 IPv4 设置：

- 要使用 DHCP，请输入：

```
nmcli> set ipv4.method auto
```

如果 **ipv4.method** 已设置为 **auto**（默认），请跳过这一步。

- 要设置静态 IPv4 地址、网络掩码、默认网关、DNS 服务器和搜索域，请输入：

```
nmcli> ipv4.addresses 192.0.2.1/24
Do you also want to set 'ipv4.method' to 'manual'? [yes]: yes
nmcli> ipv4.gateway 192.0.2.254
nmcli> ipv4.dns 192.0.2.200
nmcli> ipv4.dns-search example.com
```

7. 配置 IPv6 设置：

- 要使用无状态地址自动配置(SLAAC)，请输入：

```
nmcli> set ipv6.method auto
```

如果 **ipv6.method** 已设置为 **auto**（默认），请跳过这一步。

- 要设置静态 IPv6 地址、网络掩码、默认网关、DNS 服务器和搜索域，请输入：

```
nmcli> ipv6.addresses 2001:db8:1::ffe/64
Do you also want to set 'ipv6.method' to 'manual'? [yes]: yes
nmcli> ipv6.gateway 2001:db8:1::ffe
nmcli> ipv6.dns 2001:db8:1::ffbb
nmcli> ipv6.dns-search example.com
```

8. 保存并激活连接：

```
nmcli> save persistent
```

9. 保留为互动模式：

```
nmcli> quit
```

验证

1. 显示 NIC 的 IP 设置：

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

2. 显示 IPv4 默认网关：

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

3. 显示 IPv6 默认网关：

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

4. 显示 DNS 设置：

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

5. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

故障排除

- 验证网线是否插入到主机和交换机。
- 检查链路失败是否只存在于此主机上，或者也存在于其它连接到同一交换机的主机上。
- 验证网络电缆和网络接口是否如预期工作。执行硬件诊断步骤并替换有缺陷的电缆和网络接口卡。
- 如果磁盘中的配置与设备中的配置不匹配，则启动或重启 NetworkManager 会创建一个代表该设备的配置的内存连接。有关详情以及如何避免这个问题，请参阅 [NetworkManager 服务重启后会复制一个连接](#) 解决方案

其他资源

- [nm-settings\(5\)](#) 手册页
- [nmcli\(1\)](#) 手册页

2.3. 使用 NMTUI 配置以太网连接

如果通过以太网将主机连接到网络，您可以使用 **nmtui** 工具在基于文本的用户界面中管理连接的设置。使用 **nmtui** 创建新配置文件，并在没有图形界面的主机上更新现有配置文件。



注意

在 **nmcli** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。

流程

1. 如果您不知道要在连接中使用的网络设备名称，显示可用的设备：

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

2. 启动 **nmcli**：

```
# nmcli
```

3. 选择 **Edit a connection**，然后按 **Enter**。
4. 选择是否添加一个新连接配置文件或修改现有连接配置文件：
 - 要创建一个新配置文件：
 - i. 按 **添加**。
 - ii. 从网络类型列表中选择 **Ethernet**，然后按 **Enter** 键。
 - 要修改现有的配置文件，请从列表中选择配置文件，然后按 **Enter**。
5. 可选：更新连接配置文件的名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
6. 如果创建新连接配置文件，请在 **Device** 字段中输入网络设备名称。
7. 根据您的环境，相应地在 **IPv4 configuration** 和 **IPv6 configuration** 区中配置 IP 地址。为此，请按这些区域旁边的按钮，并选择：
 - **Disabled**，如果此连接不需要 IP 地址。
 - **Automatic**，如果 DHCP 服务器为这个 NIC 动态分配了一个 IP 地址。
 - **Manual**，如果网络需要静态 IP 地址设置。在这种情况下，您必须填写更多字段：
 - i. 在您要配置的协议旁边按 **Show** 以显示其他字段。
 - ii. 按 **Addresses** 旁边的 **Add**，并以无类别域间路由(CIDR)格式输入 IP 地址和子网掩码。

如果没有指定子网掩码，NetworkManager 会为 IPv4 地址设置 /32 子网掩码，为 IPv6 地址设置 /64 子网掩码。

- iii. 输入默认网关的地址。
- iv. 按 **DNS 服务器** 旁边的 **Add**，并输入 DNS 服务器地址。
- v. 按 **搜索域** 旁边的 **Add**，并输入 DNS 搜索域。

图 2.1. 具有静态 IP 地址设置的以太网连接示例

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::ffffe`

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>

8. 按 **OK** 创建并自动激活新连接。
9. 按 **Back** 返回到主菜单。
10. 选择 **Quit**，然后按 **Enter** 关闭 **nmtui** 应用程序。

验证

1. 显示 NIC 的 IP 设置：

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

2. 显示 IPv4 默认网关：

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

3. 显示 IPv6 默认网关：

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

4. 显示 DNS 设置：

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

5. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

故障排除

- 验证网线是否插入到主机和交换机。
- 检查链路失败是否只存在于此主机上，或者也存在于其它连接到同一交换机的主机上。
- 验证网络电缆和网络接口是否如预期工作。执行硬件诊断步骤并替换有缺陷的电缆和网络接口卡。
- 如果磁盘中的配置与设备中的配置不匹配，则启动或重启 NetworkManager 会创建一个代表该设备的配置的内存连接。有关详情以及如何避免这个问题，请参阅 [NetworkManager 服务重启后会复制一个连接](#) 解决方案。

其他资源

- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)

- [配置 DNS 服务器顺序](#)

2.4. 使用 CONTROL-CENTER 配置以太网连接

如果您通过以太网将主机连接到网络，您可以使用 GNOME Settings 菜单通过图形界面管理连接的设置。

请注意，**control-center** 不支持与 **nm-connection-editor** 应用程序或 **nmcli** 实用程序一样多的配置选项。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。
- 已安装了 GNOME。

步骤

1. 按 **Super** 键，输入 **Settings**，然后按 **Enter** 键。
2. 在左侧导航中选择 **Network**。
3. 选择是否添加一个新连接配置文件或修改现有连接配置文件：
 - 要创建一个新配置文件，请单击 **Ethernet** 条目旁边的 **+** 按钮。
 - 要修改现有配置文件，请点击配置文件条目旁的齿轮图标。
4. 可选：在 **Identity** 选项卡中，更新连接配置文件的名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
5. 根据您的环境，相应地在 **IPv4** 和 **IPv6** 标签页中配置 IP 地址设置：
 - 要使用 DHCP 或 IPv6 无状态地址自动配置(SLAAC)，请选择 **Automatic (DHCP)** 作为方法（默认）。
 - 要设置静态 IP 地址、网络掩码、默认网关、DNS 服务器和搜索域，请选择 **Manual** 作为方法，并在标签页中填写字段：

The image shows two side-by-side screenshots of the 'New Profile' dialog box in GNOME Settings. Both screenshots have 'Cancel' and 'Add' buttons at the top. The left screenshot is for IPv4 configuration. It has tabs for 'Identity', 'IPv4', 'IPv6', and 'Security'. Under 'IPv4 Method', 'Manual' is selected. Below, there are fields for 'Addresses' with columns for 'Address', 'Netmask', and 'Gateway'. The first row contains '192.0.2.1', '24', and '192.0.2.254'. There is also a 'DNS' field with '192.0.2.1' and an 'Automatic' toggle switch. The right screenshot is for IPv6 configuration. It has tabs for 'Identity', 'IPv4', 'IPv6', and 'Security'. Under 'IPv6 Method', 'Manual' is selected. Below, there are fields for 'Addresses' with columns for 'Address', 'Prefix', and 'Gateway'. The first row contains '2001:db8:1::1', '64', and '2001:db8:1::fff3'. There is also a 'DNS' field with '2001:db8:1::ffff' and an 'Automatic' toggle switch. Both screenshots have a note at the bottom: 'Separate IP addresses with commas'.

6. 根据您是否添加或修改连接配置文件，点 **Add** 或 **Apply** 按钮保存连接。
GNOME **control-center** 会自动激活连接。

验证

1. 显示 NIC 的 IP 设置：

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

2. 显示 IPv4 默认网关：

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

3. 显示 IPv6 默认网关：

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

4. 显示 DNS 设置：

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

5. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

故障排除步骤

- 验证网线是否插入到主机和交换机。
- 检查链路失败是否只存在于此主机上，或者也存在于其它连接到同一交换机的主机上。
- 验证网络电缆和网络接口是否如预期工作。执行硬件诊断步骤并替换有缺陷的电缆和网络接口卡。
- 如果磁盘中的配置与设备中的配置不匹配，则启动或重启 NetworkManager 会创建一个代表该设备的配置的内存连接。有关详情以及如何避免这个问题，请参阅 [NetworkManager 服务重启后会复制一个连接](#) 解决方案。

2.5. 使用 NM-CONNECTION-EDITOR 配置以太网连接

如果通过以太网将主机连接到网络，您可以使用 **nm-connection-editor** 应用程序使用图形界面管理连接的设置。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。
- 已安装了 GNOME。

流程

1. 打开终端窗口，输入：

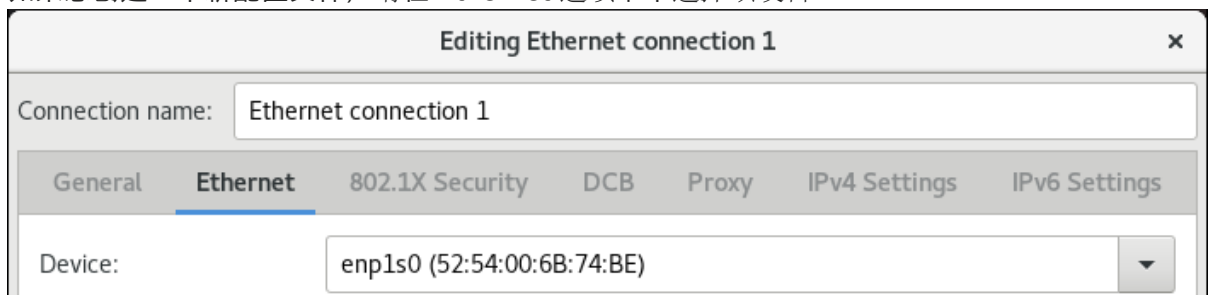
```
$ nm-connection-editor
```

2. 选择是否添加一个新连接配置文件或修改现有连接配置文件：

- 要创建一个新配置文件：
 - i. 点 **+** 按钮
 - ii. 选择 **Ethernet** 作为连接类型，然后单击 **Create**。
- 要修改现有配置文件，请双击配置文件条目。

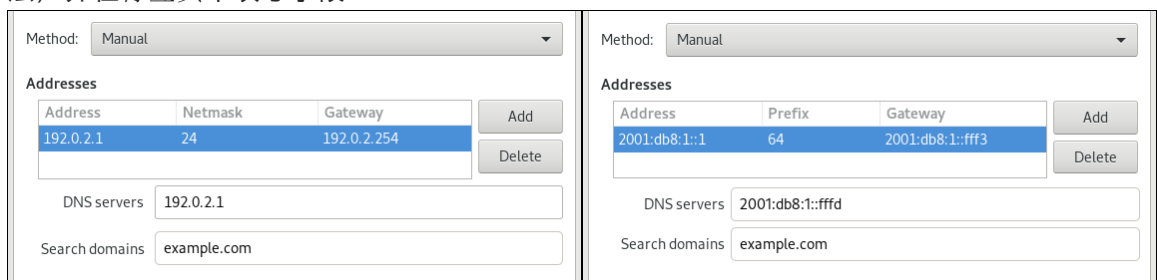
3. 可选：在 **Connection Name** 字段中更新配置文件的名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。

4. 如果您创建一个新配置文件，请在 **Ethernet** 选项卡中选择该设备：



5. 根据您的环境，相应地在 **IPv4 Settings** 和 **IPv6 Settings** 选项卡中配置 IP 地址设置：

- 要使用 DHCP 或 IPv6 无状态地址自动配置(SLAAC)，请选择 **Automatic (DHCP)** 作为方法（默认）。
- 要设置静态 IP 地址、网络掩码、默认网关、DNS 服务器和搜索域，请选择 **Manual** 作为方法，并在标签页中填写字段：



6. 点 **Save**。
7. 关闭 **nm-connection-editor**。

验证

1. 显示 NIC 的 IP 设置：

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

2. 显示 IPv4 默认网关：

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

3. 显示 IPv6 默认网关：

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

4. 显示 DNS 设置：

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

5. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

故障排除步骤

- 验证网线是否插入到主机和交换机。
- 检查链路失败是否只存在于此主机上，或者也存在于其它连接到同一交换机的主机上。
- 验证网络电缆和网络接口是否如预期工作。执行硬件诊断步骤并替换有缺陷的电缆和网络接口卡。
- 如果磁盘中的配置与设备中的配置不匹配，则启动或重启 NetworkManager 会创建一个代表该设备的配置的内存连接。有关详情以及如何避免这个问题，请参阅 [NetworkManager 服务重启后会复制一个连接](#) 解决方案。

其它资源

- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)
- [配置 DNS 服务器顺序](#)

2.6. 使用 NMSTATECTL 配置带有静态 IP 地址的以太网连接

使用 `nmstatectl` 工具通过 Nmstate API 配置以太网连接。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，`nmstatectl` 会自动回滚更改以避免系统处于不正确的状态。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。
- `nmstate` 软件包已安装。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 `~/create-ethernet-profile.yml`：

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

这些设置使用以下设置为 `enp1s0` 设备定义一个以太网连接配置文件：

- 静态 IPv4 地址 - `192.0.2.1`，子网掩码为 `/24`
- 静态 IPv6 地址 - `2001:db8:1::1`，子网掩码为 `/64`

- IPv4 默认网关 - **192.0.2.254**
 - IPv6 默认网关 - **2001:db8:1::fffe**
 - IPv4 DNS 服务器 - **192.0.2.200**
 - IPv6 DNS 服务器 - **2001:db8:1::ffbb**
 - DNS 搜索域 - **example.com**
2. 可选：您可以在 **interfaces** 属性中定义 **identifier: mac-address** 和 **mac-address: <mac_address>** 属性，通过其 MAC 地址而不是其名称来识别网络接口卡，例如：

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

3. 将设置应用到系统：

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

验证

1. 以 YAML 格式显示当前状态：

```
# nmstatectl show enp1s0
```

2. 显示 NIC 的 IP 设置：

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

3. 显示 IPv4 默认网关：

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

4. 显示 IPv6 默认网关：

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

5. 显示 DNS 设置：

■

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

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

6. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

2.7. 使用网络 RHEL 系统角色和接口名称，使用静态 IP 地址配置以太网连接

您可以使用 **network** RHEL 系统角色远程配置以太网连接。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 服务器配置中有一个物理或者虚拟以太网设备。
- 受管节点使用 NetworkManager 配置网络。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            interface_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
state: up

```

这些设置使用以下设置为 **enp1s0** 设备定义一个以太网连接配置文件：

- 静态 IPv4 地址 - **192.0.2.1** 和 /24 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 /64 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

2.8. 使用网络 RHEL 系统角色和设备路径，使用静态 IP 地址配置以太网连接

您可以使用 **network** RHEL 系统角色远程配置以太网连接。

您可以使用以下命令识别设备路径：

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

前提条件

- 您已准备好控制节点和受管节点

- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 服务器配置中有一个物理或者虚拟以太网设备。
- 受管节点使用 NetworkManager 配置网络。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

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

这些设置使用以下设置定义以太网连接配置文件：

- 静态 IPv4 地址 - **192.0.2.1** 和 **/24** 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**

本例中的 **match** 参数定义了 Ansible 将脚本应用到与 PCI ID **0000:00:0[1-3].0** 匹配的设备，但没有 **0000:00:02.0** 设备。有关可以使用的特殊修饰符和通配符的详情，请查看 [/usr/share/ansible/roles/rhel-system-roles.network/README.md](#) 文件中的 **match** 参数描述。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

2.9. 使用 NMSTATECTL 配置带有动态 IP 地址的以太网连接

使用 **nmstatectl** 工具通过 Nmstate API 配置以太网连接。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，**nmstatectl** 会自动回滚更改以避免系统处于不正确的状态。

先决条件

- 服务器配置中存在物理或虚拟以太网网络接口控制器(NIC)。
- 网络中有 DHCP 服务器。
- **nmstate** 软件包已安装。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 **~/create-ethernet-profile.yml**：

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

```

auto-routes: true
autoconf: true
dhcp: true

```

这些设置为 **enp1s0** 设备定义一个以太网连接配置文件。连接从 DHCP 服务器和 IPv6 无状态地址自动配置(SLAAC)检索 IPv4 地址、IPv6 地址、默认网关、路由、DNS 服务器和搜索域。

2. 可选：您可以在 **interfaces** 属性中定义 **identifier: mac-address** 和 **mac-address: <mac_address>** 属性，通过其 MAC 地址而不是其名称来识别网络接口卡，例如：

```

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

```

3. 将设置应用到系统：

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

验证

1. 以 YAML 格式显示当前状态：

```
# nmstatectl show enp1s0
```

2. 显示 NIC 的 IP 设置：

```

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

```

3. 显示 IPv4 默认网关：

```

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

```

4. 显示 IPv6 默认网关：

```

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

```

5. 显示 DNS 设置：

```

# cat /etc/resolv.conf
search example.com

```

```
nameserver 192.0.2.200
nameserver 2001:db8:1::ffbb
```

如果多个连接配置文件同时处于活动状态，则 **nameserver** 条目的顺序取决于这些配置文件中的 DNS 优先级值和连接类型。

6. 使用 **ping** 程序来验证这个主机是否可以向其它主机发送数据包：

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

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

2.10. 使用网络 RHEL 系统角色和接口名称，使用动态 IP 地址配置以太网连接

您可以使用 **network** RHEL 系统角色远程配置以太网连接。对于具有动态 IP 地址设置的连接，NetworkManager 会为来自 DHCP 服务器的连接请求 IP 设置。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 服务器配置中有一个物理或者虚拟以太网设备。
- 网络中有 DHCP 服务器
- 受管节点使用 NetworkManager 配置网络。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

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

```

dhcp4: yes
auto6: yes
state: up

```

这些设置为 **enp1s0** 设备定义一个以太网连接配置文件。连接从 DHCP 服务器和 IPv6 无状态地址自动配置(SLAAC)检索 IPv4 地址、IPv6 地址、默认网关、路由、DNS 服务器和搜索域。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

2.11. 使用网络 RHEL 系统角色和设备路径，使用动态 IP 地址配置以太网连接

您可以使用 **network** RHEL 系统角色远程配置以太网连接。对于具有动态 IP 地址设置的连接，NetworkManager 会为来自 DHCP 服务器的连接请求 IP 设置。

您可以使用以下命令识别设备路径：

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

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 服务器配置中有一个物理或者虚拟以太网设备。
- 网络中有 DHCP 服务器。
- 受管主机使用 NetworkManager 配置网络。

流程

1. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml`：

```

---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:

```

```

- name: Configure an Ethernet connection with dynamic IP
  ansible.builtin.include_role:
    name: rhel-system-roles.network
  vars:
    network_connections:
      - name: example
        match:
          path:
            - pci-0000:00:0[1-3].0
            - &!pci-0000:00:02.0
          type: ethernet
          autoconnect: yes
          ip:
            dhcp4: yes
            auto6: yes
          state: up

```

这些设置定义一个以太网连接配置文件。连接从 DHCP 服务器和 IPv6 无状态地址自动配置 (SLAAC) 检索 IPv4 地址、IPv6 地址、默认网关、路由、DNS 服务器和搜索域。

match 参数定义 Ansible 将 play 应用到与 PCI ID **0000:00:0[1-3].0**，而不是 **0000:00:02.0** 匹配的设备。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

2.12. 按接口名称使用单个连接配置文件配置多个以太网接口

在大多数情况下，一个连接配置文件包含一个网络设备的设置。但是，当您在连接配置文件中设置接口名称时，NetworkManager 也支持通配符。如果主机在具有动态 IP 地址分配的以太网之间漫游，则您可以使用此功能创建可用于多个以太网接口的单一连接配置文件。

前提条件

- 服务器配置中存在多个物理或虚拟以太网设备。
- 网络中有 DHCP 服务器。
- 主机上不存在连接配置文件。

流程

1. 添加可应用于以 **enp** 开头的所有接口名称的连接配置文件：

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

验证

1. 显示单个连接配置文件的所有设置：

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

3 表示同一时间在连接配置文件上活跃的接口数量，而不是连接配置文件中的网络接口数量。连接配置文件使用与 **match.interface-name** 参数中的模式匹配的所有设备，因此连接配置文件具有相同的通用唯一识别符(UUID)。

2. 显示连接的状态：

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

其他资源

- [nmcli\(1\) 手册页](#)
- [nm-settings\(5\) 手册页](#)

2.13. 使用 PCI ID 为多个以太网接口配置一个连接配置文件

PCI ID 是连接到系统的设备的唯一标识符。连接配置文件根据 PCI ID 列表按匹配的接口来添加多个设备。您可以使用这个流程将多个设备 PCI ID 连接到一个连接配置文件。

先决条件

- 服务器配置中存在多个物理或虚拟以太网设备。
- 网络中有 DHCP 服务器。
- 主机上不存在连接配置文件。

步骤

1. 识别设备路径。例如，要显示以 **enp** 开头的所有接口的设备路径，请输入：

```
# 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. 添加可应用于匹配 **0000:00:0[7-8].0** 表达式的所有 PCI ID 的连接配置文件：

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

验证

1. 显示连接的状态：

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

2. 显示连接配置集的所有设置：

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

此连接配置文件使用 PCI ID 与 **match.path** 参数中的模式匹配的所有设备，因此连接配置文件具有相同的全局唯一标识符(UUID)。

其他资源

- [nmcli\(1\) 手册页](#)
- [nm-settings\(5\) 手册页](#)

第 3 章 配置网络绑定

网络绑定是一种组合或聚合物理和虚拟网络接口的方法，以提供高吞吐量或冗余的逻辑接口。在绑定中，内核只处理所有操作。您可以在不同类型的设备中创建绑定，如以太网设备或 VLAN。

Red Hat Enterprise Linux 为管理员提供不同的选项来配置团队设备。例如：

- 使用 **nmcli** 使用命令行配置绑定连接。
- 通过 RHEL web 控制台使用 Web 浏览器配置绑定连接。
- 使用 **nmtui** 在基于文本的用户界面中配置绑定连接。
- 使用 **nm-connection-editor** 应用程序在图形界面中配置绑定连接。
- 使用 **nmstatectl** 通过 Nmstate API 配置绑定连接。
- 使用 RHEL 系统角色自动化一个或多个主机上的绑定配置。

3.1. 了解控制器和端口接口的默认行为

在使用 **NetworkManager** 服务管理或排除团队或绑定端口接口故障排除时，请考虑以下默认行为：

- 启动控制器接口不会自动启动端口接口。
- 启动端口接口总会启动控制器接口。
- 停止控制器接口也会停止端口接口。
- 没有端口的控制器可以启动静态 IP 连接。
- 没有端口的控制器在启动 DHCP 连接时会等待端口。
- 当您添加具有载体的端口时，等待端口且具有 DHCP 连接的控制器会完成。
- 当您添加不具有载体的端口时，等待端口且具有 DHCP 连接的控制器将继续等待。

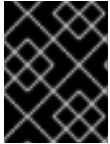
3.2. 依赖绑定模式的上游交换机配置

根据您要使用的绑定模式，您必须在交换机上配置端口：

绑定模式	交换机上的配置
0 - balance-rr	需要启用静态 EtherChannel，而不是链路聚合控制协议(LACP)协商。
1 - active-backup	交换机上不需要任何配置。
2 - balance-xor	需要启用静态 EtherChannel，而不是 LACP 协商。
3 - broadcast	需要启用静态 EtherChannel，而不是 LACP 协商。
4 - 802.3ad	需要启用 LACP 协商的 EtherChannel。

绑定模式	交换机上的配置
5 - balance-tlb	交换机上不需要任何配置。
6 - balance-alb	交换机上不需要任何配置。

有关如何配置交换机的详情，请查看交换机文档。



重要

某些网络绑定的功能，比如故障切换机制，不支持不通过网络交换机的直接电缆连接。详情请查看[是否支持直接连接的绑定？KCS 解决方案](#)。

3.3. 使用 NMCLI配置网络绑定

要在命令行上配置网络绑定，请使用 `nmcli` 工具。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作绑定的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bridge 或 VLAN 设备作为绑定的端口，您可以在创建绑定时创建这些设备，或者预先创建它们，如：
 - [使用 nmcli 配置网络团队](#)
 - [使用 nmcli 配置网桥](#)
 - [使用 nmcli 配置 VLAN 标记](#)

步骤

1. 创建绑定接口：

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

这个命令会创建一个使用 `active-backup` 模式、名为 `bond0` 的绑定。

要额外设置介质独立接口(MII)监控间隔，请在 `bond.options` 属性中添加 `miimon=interval` 选项，例如：

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

2. 显示网络接口以及您要添加到绑定中的接口名称：

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

```
enp8s0 ethernet disconnected --
bridge0 bridge connected bridge0
bridge1 bridge connected bridge1
...
```

在本例中：

- 没有配置 **enp7s0** 和 **enp8s0**。要将这些设备用作端口，请在下一步中添加连接配置集。
- **bridge0** 和 **bridge1** 都有现有的连接配置文件。要将这些设备用作端口，请在下一步中修改其配置集。

3. 为绑定分配接口：

- a. 如果没有配置您要分配给绑定的接口，为其创建新的连接配置集：

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

这些命令为 **enp7s0** 和 **enp8s0** 创建配置文件，并将它们添加到 **bond0** 连接中。

- b. 要将现有的连接配置文件分配给绑定：

- i. 将这些连接的 **controller** 参数设置为 **bond0**：

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

这些命令将名为 **bridge0** 和 **bridge1** 的现有连接配置文件分配给 **bond0** 连接。

- ii. 重新激活连接：

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

4. 配置 IPv4 设置：

- 要将这个绑定设备用作其它设备的端口，请输入：

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

- 要使用 DHCP，不需要进行任何操作。
- 要为 **bond0** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

5. 配置 IPv6 设置：

- 要将这个绑定设备用作其它设备的端口，请输入：

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

- 要使用无状态地址自动配置(SLAAC)，不需要任何操作。
- 要为 **bond0** 连接设置静态 IPv6 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

6. 可选：如果要对绑定端口设置任何参数，请使用以下命令：

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

7. 激活连接：

```
# nmcli connection up bond0
```

8. 验证端口是否已连接，并且 **CONNECTION** 列是否显示端口的连接名称：

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

当您激活连接的任何端口时，NetworkManager 也激活绑定，但不会激活它的其它端口。您可以配置 Red Hat Enterprise Linux 在启用绑定时自动启用所有端口：

- a. 启用绑定连接的 **connection.autoconnect-ports** 参数：

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

- b. 重新激活桥接：

```
# nmcli connection up bond0
```

验证

1. 从其中一个网络设备中临时删除网络电缆，并检查绑定中是否有其他设备处理流量。请注意，无法使用软件工具正确测试链路失败事件。停用连接的工具（如 **nmcli**），只显示绑定驱动程序可以处理端口配置的更改，而不是实际的链接失败事件。
2. 显示绑定状态：

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

3.4. 使用 RHEL WEB 控制台配置网络绑定

如果您希望使用基于 Web 浏览器的界面管理网络设置，请使用 RHEL web 控制台配置网络绑定。

先决条件

- 已登陆到 RHEL web 控制台。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作绑定的成员，必须在服务器中安装物理或者虚拟以太网设备。
- 要将 team、bridge 或 VLAN 设备用作绑定成员，请预先创建它们，如：
 - [使用 RHEL web 控制台配置网络团队](#)
 - [使用 RHEL web 控制台配置网桥](#)
 - [使用 RHEL web 控制台配置 VLAN 标记](#)

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add bond**。
3. 输入您要创建的绑定设备名称。
4. 选择应该是绑定成员的接口。
5. 选择绑定模式。
如果您选择 **Active backup**，Web 控制台会显示额外的 **Primary** 字段，您可以在其中选择首选的活动设备。
6. 设置链路监控模式。例如，当您使用 **Adaptive 负载均衡** 模式时，将它设置为 **ARP**。
7. 可选：调整监控间隔、链接延迟和连接延迟设置。通常，您只需要更改默认值以进行故障排除。

Bond settings

Name

Interfaces enp7s0
 enp8s0

MAC

Mode

Primary

Link monitoring

Monitoring interval

Link up delay

Link down delay

8. 点应用。

9. 默认情况下，绑定使用动态 IP 地址。如果要设置静态 IP 地址：

- a. 在 **Interfaces** 部分点绑定的名称。
- b. 点您要配置的协议旁的 **Edit**。
- c. 选择 **Addresses** 旁的 **Manual**，并输入 IP 地址、前缀和默认网关。
- d. 在 **DNS** 部分，点 **+** 按钮，并输入 DNS 服务器的 IP 地址。重复此步骤来设置多个 DNS 服务器。
- e. 在 **DNS search domains** 部分中，点 **+** 按钮并输入搜索域。

- f. 如果接口需要静态路由，请在 **Routes** 部分配置它们。

IPv4 settings ✕

Addresses Manual ▾ +

Address	Prefix length or netmask	Gateway	-
<input type="text" value="192.0.2.1"/>	<input type="text" value="24"/>	<input type="text" value="192.0.2.254"/>	-

DNS Automatic +

Server -

DNS search domains Automatic +

Search domain -

Routes Automatic +

Apply Cancel

- g. 点应用

验证

1. 在屏幕左侧的导航中选择 **Networking** 选项卡，并检查接口上是否有传入和传出流量：

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

2. 从其中一个网络设备中临时删除网络电缆，并检查绑定中是否有其他设备处理流量。请注意，无法使用软件工具正确测试链路失败事件。取消激活连接的工具（如 Web 控制台）只显示处理成员配置更改且没有实际链路失败事件的能力。
3. 显示绑定状态：

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

3.5. 使用 NMTUI 配置网络绑定

nmtui 应用程序为 NetworkManager 提供了一个基于文本的用户界面。您可以使用 **nmtui** 在没有图形界面的主机上配置网络绑定。



注意

在 **nmtui** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作绑定的端口，必须在服务器中安装物理或者虚拟以太网设备。

步骤

1. 如果您不知道您要在其上配置网络绑定的网络设备名称，请显示可用的设备：

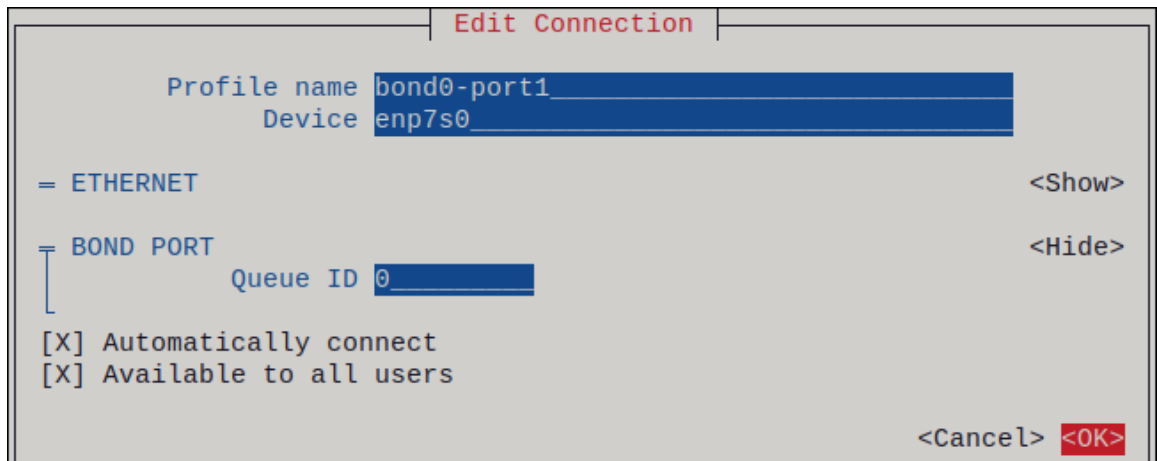
```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

2. 启动 **nmtui**：

```
# nmtui
```

3. 选择 **Edit a connection**，然后按 **Enter**。
4. 按 **添加**。
5. 从网络类型列表中选择 **Bond**，然后按 **Enter** 键。
6. 可选：为要创建的 NetworkManager 配置文件输入一个名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
7. 在 **Device** 字段中输入要创建的绑定设备名称。
8. 为要创建的绑定添加端口：
 - a. 按 **Slaves** 列表旁边的 **Add**。
 - b. 选择您要作为端口添加到绑定的接口类型，例如 **Ethernet**。
 - c. 可选：为这个绑定端口输入要创建的 NetworkManager 配置文件的名称。
 - d. 在 **Device** 字段中输入端口的设备名称。
 - e. 按 **OK** 返回绑定设置窗口。

图 3.1. 将以太网设备作为端口添加到绑定



- f. 重复这些步骤，来向绑定添加更多的端口。
9. 设置绑定模式。根据您设置的值，**nmtui** 会显示与所选模式相关的设置的额外字段。
 10. 根据您的环境，相应地在 **IPv4 配置**和 **IPv6 配置区**中配置 IP 地址设置。为此，请按这些区域旁边的按钮，并选择：
 - **Disabled**，如果绑定不需要 IP 地址。
 - **Automatic**，如果 DHCP 服务器或无状态地址自动配置(SLAAC)动态为绑定分配一个 IP 地址。
 - **Manual**，如果网络需要静态 IP 地址设置。在这种情况下，您必须填写更多字段：
 - i. 在您要配置的协议旁边按 **Show** 以显示其他字段。
 - ii. 按 **Addresses** 旁边的 **Add**，并以无类别域间路由(CIDR)格式输入 IP 地址和子网掩码。如果没有指定子网掩码，NetworkManager 会为 IPv4 地址设置 **/32** 子网掩码，为 IPv6 地址设置 **/64** 子网掩码。
 - iii. 输入默认网关的地址。
 - iv. 按 **DNS 服务器** 旁边的 **Add**，并输入 DNS 服务器地址。
 - v. 按 **搜索域** 旁边的 **Add**，并输入 DNS 搜索域。

图 3.2. 具有静态 IP 地址设置的绑定连接的示例

Edit Connection

Profile name

Device

BOND Slaves <Hide>

↑
 ↓

<Add>
<Edit...>
<Delete>

Mode

Primary

Link monitoring

Monitoring frequency ms

Link up delay ms

Link down delay ms

Cloned MAC address

IPv4 CONFIGURATION <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 <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>

11. 按 OK 创建并自动激活新连接。

- 按 **Back** 返回到主菜单。
- 选择 **Quit**，然后按 **Enter** 关闭 **nmtui** 应用程序。

验证

- 从其中一个网络设备中临时删除网络电缆，并检查绑定中是否有其他设备处理流量。请注意，无法使用软件工具正确测试链路失败事件。停用连接的工具（如 **nmcli**），只显示绑定驱动程序可以处理端口配置的更改，而不是实际的链接失败事件。
- 显示绑定状态：

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

3.6. 使用 NM-CONNECTION-EDITOR 配置网络绑定

如果使用带有图形界面的 Red Hat Enterprise Linux，您可以使用 **nm-connection-editor** 应用程序配置网络绑定。

请注意：**nm-connection-editor** 只能向绑定添加新端口。要使用现有连接配置文件作为端口，请使用 **nmcli** 工具创建绑定，如 [使用 nmcli 配置网络绑定](#) 中所述。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作绑定的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bond 或 VLAN 设备作为绑定的端口，请确保这些设备还没有配置。

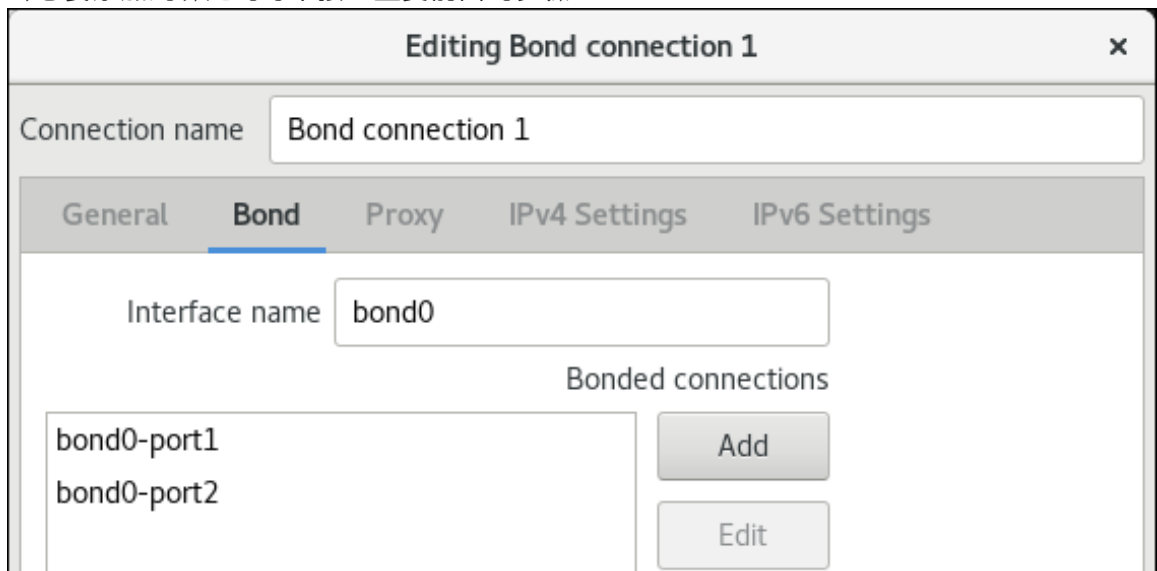
步骤

- 打开一个终端，输入 **nm-connection-editor**：

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

- 点击 **+** 按钮来添加一个新的连接。
- 选择 **Bond** 连接类型，然后单击 **Create**。
- 在 **Bond** 选项卡中：
 - 可选：在 **Interface name** 字段中设置绑定接口的名称。
 - 点 **Add** 按钮将网络接口作为端口添加到绑定。
 - 选择接口的连接类型。例如，为有线连接选择 **Ethernet**。
 - 可选：为端口设置连接名称。
 - 如果您为以太网设备创建了一个连接配置文件，请打开 **Ethernet** 选项卡，在 **Device** 字段中选择您要作为端口添加到绑定的网络接口。如果您选择了不同的设备类型，请相应地进行配置。请注意，您只能在没有配置的绑定中使用以太网接口。
 - 点 **Save**。

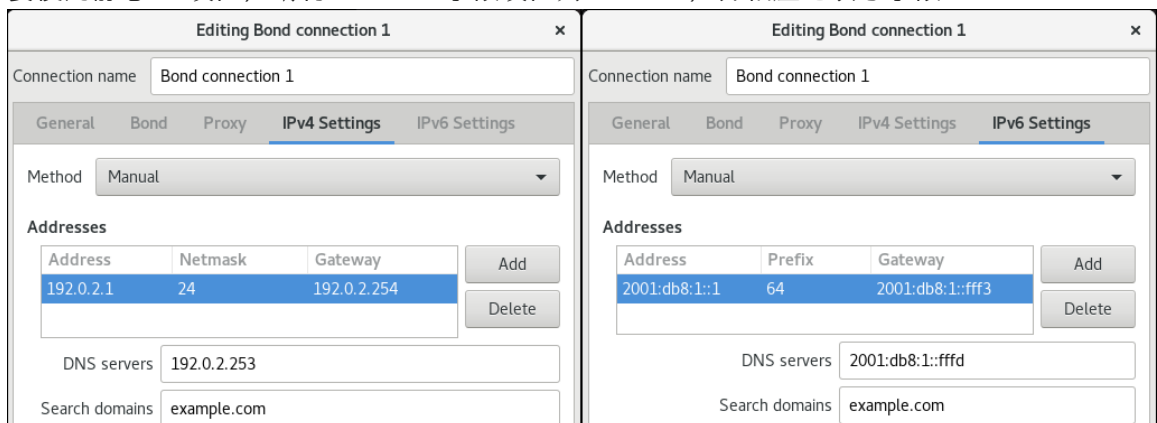
- c. 对您要添加到绑定的每个接口重复前面的步骤：



- d. 可选：设置其他选项，如介质独立接口（MII）监控间隔。

5. 在 **IPv4 Settings** 和 **IPv6 Settings** 标签页中配置 IP 地址设置：

- 要将这个网桥设备用作其他设备的端口，请将 **Method** 字段设置为 **Disabled**。
- 要使用 DHCP，请将 **Method** 字段保留为默认值 **Automatic (DHCP)**。
- 要使用静态 IP 设置，请将 **Method** 字段设置为 **Manual**，并相应地填写字段：



6. 点 **Save**。

7. 关闭 **nm-connection-editor**。

验证

1. 从其中一个网络设备中临时删除网络电缆，并检查绑定中是否有其他设备处理流量。请注意，无法使用软件工具正确测试链路失败事件。停用连接的工具（如 **nmcli**），只显示绑定驱动程序可以处理端口配置的改变，而不是实际的链接失败事件。
2. 显示绑定状态：

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

其他资源

- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)
- [使用 nm-connection-editor 配置网络团队](#)
- [使用 nm-connection-editor 配置网桥](#)
- [使用 nm-connection-editor 配置 VLAN 标记](#)

3.7. 使用 NMSTATECTL 配置网络绑定

使用 `nmstatectl` 工具通过 Nmstate API 配置网络绑定。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，`nmstatectl` 会自动回滚更改以避免系统处于不正确的状态。

根据您的环境，相应地调整 YAML 文件。例如，要在绑定中使用与以太网适配器不同的设备，请调整您在绑定中使用的端口的 `base-iface` 属性和 `type` 属性。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作绑定中的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要在绑定中使用团队、网桥或 VLAN 设备作为端口，请在 `port` 列表中设置接口名称，并定义相应的接口。
- `nmstate` 软件包已安装。

步骤

1. 创建包含一个以下内容的 YAML 文件，如 `~/create-bond.yml`：

```
---
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

```

这些设置使用以下设置定义网络绑定：

- 绑定中的网络接口：**enp1s0** 和 **enp7s0**
- 模式：**active-backup**
- 静态 IPv4 地址：**192.0.2.1**，子网掩码为 **/24**
- 静态 IPv6 地址：**2001:db8:1::1**子网掩码为 **/64**
- IPv4 默认网关：**192.0.2.254**
- IPv6 默认网关：**2001:db8:1::fffe**
- IPv4 DNS 服务器：**192.0.2.200**
- IPv6 DNS 服务器：**2001:db8:1::ffbb**
- DNS 搜索域：**example.com**

2. 将设置应用到系统：

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

验证

1. 显示设备和连接的状态：

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

2. 显示连接配置集的所有设置：

```
# 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. 以 YAML 格式显示连接设置：

```
# nmstatectl show bond0
```

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

3.8. 使用 NETWORK RHEL 系统角色配置网络绑定

您可以使用 **network** RHEL 系统角色远程配置网络绑定。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。

流程

1. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml`：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bond that uses two Ethernet ports
      ansible.builtin.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

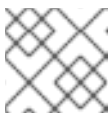
# Add an Ethernet profile to the bond
- name: bond0-port1
  interface_name: enp7s0
  type: ethernet
  controller: bond0
  state: up

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

```

这些设置使用以下设置定义网络绑定：

- 静态 IPv4 地址 - **192.0.2.1** 和 /24 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 /64 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- 绑定的端口 - **enp7s0** 和 **enp8s0**
- 绑定模式 - **active-backup**



注意

在绑定上设置 IP 配置，而不是在 Linux 绑定的端口上设置。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

3.9. 创建网络绑定以便在不中断 VPN 的情况下在以太网和无线连接间进行切换

将工作站连接到公司网络的 RHEL 用户通常会使用 VPN 访问远程资源。然而，如果工作站在以太网和 Wi-Fi 连接间切换，例如：如果您是从带以太网连接的 docking 站中释放的笔记本电脑，VPN 连接就中断。要避免这个问题，您可以在 **active-backup** 模式中创建使用以太网和 Wi-Fi 连接的网络绑定。

先决条件

- 主机包含以太网和 Wi-Fi 设备。
- 已创建以太网和 Wi-Fi 网络管理器连接配置集，且两个连接都可以独立工作。此流程使用以下连接配置文件来创建名为 **bond0** 的网络绑定：
 - 与 `enp11s0u1` 以太网设备关联的 **Docking_station**
 - **Wi-Fi** 与 `wlp1s0` Wi-Fi 设备关联

流程

1. 在 **active-backup** 模式中创建一个绑定接口：

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

这个命令将接口和连接配置文件命名为 **bond0**。

2. 配置绑定的 IPv4 设置：

- 如果您的网络中的 DHCP 服务器为主机分配 IPv4 地址，则不需要任何操作。
- 如果您的本地网络需要静态 IPv4 地址，请将地址、网络掩码、默认网关、DNS 服务器和 DNS 搜索域设为 **bond0** 连接：

```
# 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. 配置绑定的 IPv6 设置：

- 如果您的网络中的路由器或者 DHCP 服务器为主机分配 IPv6 地址，则不需要任何操作。
- 如果您的本地网络需要静态 IPv6 地址，请将地址、网络掩码、默认网关、DNS 服务器和 DNS 搜索域设为 **bond0** 连接：


```
# 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. 显示连接配置集：

```
# 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
...
```

下一步需要连接配置集的名称和以太网设备名称。

5. 为绑定分配以太网连接的配置：

```
# nmcli connection modify Docking_station controller bond0
```

6. 为绑定分配 Wi-Fi 连接的连接配置集：

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

7. 如果您的 Wi-Fi 网络使用 MAC 过滤来只允许列表中的 MAC 地址访问网络，请配置 NetworkManager 将活跃端口的 MAC 地址动态分配给绑定：

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

使用这个设置时，您必须将 Wi-Fi 设备的 MAC 地址设置为 allow 列表，而不是以太网和 Wi-Fi 设备的 MAC 地址。

8. 将与以太网连接关联的设备设置为绑定的主设备：

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

使用这个设置时，如果可用，绑定总是使用以太网连接。

9. 配置当 **bond0** 设备激活时，NetworkManager 会自动激活端口：

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

10. 激活 **bond0** 连接：

```
# nmcli connection up bond0
```

验证

- 显示当前激活的设备，绑定及其端口的状态：

```
# 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
```

其它资源

- [配置以太网连接](#)
- [管理 Wi-Fi 连接](#)
- [配置网络绑定](#)

3.10. 不同的网络绑定模式

Linux 绑定驱动程序提供链路聚合。绑定是并行封装多个网络接口的过程，以提供单个逻辑绑定接口。绑定接口的操作取决于也称为模式的绑定策略。不同的模式提供负载均衡或热待机服务。

存在以下模式：

Balance-rr (Mode 0)

balance-rr 使用循环算法，它按顺序将数据包从第一个可用端口传输到最后一个端口。这个模式提供负载均衡和容错。

这个模式需要切换端口聚合组（也称为 EtherChannel 或类似的端口分组）。EtherChannel 是一个端口链路聚合技术，用于将多个物理以太网链接分组到一个逻辑以太网链接中。

这个模式的缺陷在于它不适用于大量工作负载，以及 TCP 吞吐量或排序数据包交付非常重要。

Active-backup (Mode 1)

Active-backup 使用策略来确定在绑定中只有一个端口活跃。这个模式提供容错功能，不需要任何交换机配置。

如果活动端口失败，则备用端口将变为活动状态。绑定会向网络发送大量地址解析协议 (ARP) 响应。 gratuitous ARP 强制执行 ARP 帧的接收器，以更新它们的转发表。**Active-backup** 模式传输一个 gratuitous ARP，宣布为主机维护连接的新路径。

primary 选项定义绑定接口的首选端口。

Balance-xor (Mode 2)

balance-xor 使用所选传输哈希策略来发送数据包。这个模式提供负载平衡、容错和需要切换配置来设置 Etherchannel 或类似的端口分组。

要改变数据包传输和平衡传输，此模式使用 **xmit_hash_policy** 选项。根据接口上流量的源或目的地，接口需要额外的负载均衡配置。请参阅 [xmit_hash_policy bonding 参数](#)。

Broadcast (Mode 3)

广播使用在所有接口上传输每个数据包的策略。这个模式提供容错，需要交换机配置来设置 EtherChannel 或类似的端口分组。

这个模式的缺陷在于它不适用于大量工作负载，以及 TCP 吞吐量或排序数据包交付非常重要。

802.3ad (Mode 4)

802.3ad 使用同名的 IEEE 标准动态链路聚合策略。此模式提供容错功能。这个模式需要切换配置来设置链路聚合控制协议 (LACP) 端口分组。

这个模式会创建聚合组，它们共享相同的速度和双工设置，并使用活跃聚合器中的所有端口。根据接口上流量的源或目的地，此模式需要额外的负载平衡配置。

默认情况下，传出流量的端口选择取决于传输哈希策略。使用传输哈希策略的 **xmit_hash_policy** 选项更改端口选择和平衡传输。

802.3ad 和 **Balance-xor** 之间的差别是合规性。**802.3ad** 策略在端口聚合组之间协商 LACP。请参阅 [xmit_hash_policy bonding 参数](#)

Balance-tlb (Mode 5)

balance-tlb 使用传输负载均衡策略。这个模式提供容错、负载均衡和建立不需要任何交换机支持的频道绑定。

活动端口接收传入流量。如果活动端口失败，另一个则是接管故障端口的 MAC 地址。要确定哪个接口处理传出流量，请使用以下模式之一：

- 值 **0**：使用哈希分发策略在不进行负载均衡的情况下分发流量
- 值 **1**：利用负载均衡将流量分配到每个端口
使用 bonding 选项 **tlb_dynamic_lb=0**，此绑定模式使用 **xmit_hash_policy bonding** 选项来均衡传输。**primary** 选项定义绑定接口的首选端口。

请参阅 [xmit_hash_policy bonding 参数](#)。

Balance-alb (Mode 6)

balance-alb 使用自适应负载平衡策略。这个模式提供容错、负载平衡，且不需要任何特殊交换机支持。

此模式包含 **balance-trans-mit** 负载平衡 (**balance-tlb**) 和 IPv4 和 IPv6 流量接收负载均衡。绑定会截获本地系统发送的 ARP 回复，并覆盖绑定中某个端口的源硬件地址。ARP 协商管理接收负载平衡。因此，不同的端口为服务器使用不同的硬件地址。

primary 选项定义绑定接口的首选端口。使用 bonding 选项 **tlb_dynamic_lb=0**，此绑定模式使用 **xmit_hash_policy bonding** 选项来均衡传输。请参阅 [xmit_hash_policy bonding 参数](#)。

其他资源

- `/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.rst` 由 `kernel-doc` 软件包提供
- `/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.txt` 由 `kernel-doc` 软件包提供
- [与虚拟机客户机或容器连接的网桥一起使用的绑定模式](#)
- [How are the values for different policies in "xmit_hash_policy" bonding parameter calculated?](#)

3.11. XMIT_HASH_POLICY BONDING 参数

`xmit_hash_policy` 负载均衡参数在 `balance-xor`、`802.3ad`、`balance-alb` 和 `balance-tlb` 模式中选择节点选择的传输散列策略。如果 `tlb_dynamic_lb` 参数为 `0`，则只适用于模式 5 和 6。此参数可能的值是 `layer2`、`layer2+3`、`layer3+4`、`encap2+3`、`encap3+4`，和 `vlan+srcmac`。

详情请查看表：

策略或网络层	Layer2	Layer2+3	Layer3+4	encap2+3	encap3+4	VLAN+src mac
使用	源和目的地 MAC 地址和以太网协议类型的 XOR	源和目标 MAC 地址和 IP 地址的 XOR	源和目标端口和 IP 地址的 XOR	支持的隧道内的目的地 MAC 地址和 IP 地址的 XOR，如虚拟可扩展局域网 (VXLAN)。此模式依赖于 <code>skb_flow_dissect()</code> 函数来获取标头字段	受支持的隧道内的目标端口和 IP 地址的 XOR，如 VXLAN。此模式依赖于 <code>skb_flow_dissect()</code> 函数来获取标头字段	VLAN ID 和源 MAC 厂商和源 MAC 设备的 XOR
流量放置	在同一个底层网络接口上到特定网络对等的所有流量	同一底层网络接口上特定 IP 地址的所有流量	同一底层网络接口上特定 IP 地址和端口的所有流量			

主要选择	如果网络流量在同一广播域中的这个系统和多个其他系统之间	如果此系统和多个其他系统间的网络流量会通过默认网关	如果此系统和其他系统之间的网络流量使用相同的 IP 地址，但会经历多个端口	封装的流量在源系统和多个其它系统中使用多个 IP 地址	封装的流量是源系统和其它使用多个端口号的系统间	如果绑定承载来自多个容器或虚拟机 (VM) 的网络流量，它会将其 MAC 地址直接公开给外部网络，如桥接网络，您无法配置模式 2 或模式 4 的交换机。
辅助选择	如果网络流量主要是此系统和默认网关后面的多个其他系统之间	如果网络流量主要是此系统和另一个系统间的				
Compliant	802.3ad	802.3ad	Not 802.3ad			
默认策略	如果没有提供配置，则这是默认策略	对于非 IP 流量，公式与 L2 传输策略相同	对于非 IP 流量，公式与 L2 传输策略相同			

第 4 章 配置网络团队 (TEAM)

网络组是一种组合或聚合物理和虚拟网络接口的方法，以提供高吞吐量或冗余的逻辑接口。网络团队使用一个小的内核模块来实现数据包流的快速处理和其他任务的用户空间服务。因此，网络团队是一个易扩展的解决方案，来满足负载平衡和冗余的要求。

Red Hat Enterprise Linux 为管理员提供不同的选项来配置团队设备。例如：

- 使用 **nmcli** 使用命令行配置团队连接。
- 使用 RHEL web 控制台使用 Web 浏览器配置组连接。
- 使用 **nm-connection-editor** 应用程序在图形界面中配置组连接。



重要

网络 teaming 在 Red Hat Enterprise Linux 9 中已弃用。考虑使用网络绑定驱动程序作为替代方案。详情请参阅 [配置网络绑定](#)。

4.1. 将网络团队配置迁移到网络绑定

网络 teaming 在 Red Hat Enterprise Linux 9 中已弃用。如果您已经配置了正常工作的网络团队，例如，因为从以前的 RHEL 版本升级，您可以将配置迁移到由 NetworkManager 管理的网络绑定。



重要

team2bond 程序可以将网络团队配置转换为绑定。之后，您必须手动配置绑定的进一步设置，如 IP 地址和 DNS 配置。

先决条件

- **team-team0** NetworkManager 连接配置集被配置并管理 **team0** 设备。
- 已安装 **teamd** 软件包。

步骤

1. 可选：显示 **team-team0** NetworkManager 连接的 IP 配置：

```
# nmcli connection show team-team0 | egrep "^ip"
...
ipv4.method:                manual
ipv4.dns:                    192.0.2.253
ipv4.dns-search:             example.com
ipv4.addresses:              192.0.2.1/24
ipv4.gateway:                192.0.2.254
...
ipv6.method:                 manual
ipv6.dns:                    2001:db8:1::fffd
ipv6.dns-search:             example.com
ipv6.addresses:              2001:db8:1::1/64
ipv6.gateway:                2001:db8:1::fffe
...
```

- 将 **team0** 设备的配置导出到 JSON 文件中：

```
# teamdctl team0 config dump actual > /tmp/team0.json
```

- 删除网络组。例如，如果您在 NetworkManager 中配置了团队，请删除 **team-team0** 连接配置集以及相关端口的配置集：

```
# nmcli connection delete team-team0
# nmcli connection delete team-team0-port1
# nmcli connection delete team-team0-port2
```

- 以空运行模式运行 **team2bond** 程序，显示 **nmcli** 命令，该命令使用类似设置的网络绑定设置为团队设备：

```
# team2bond --config=/tmp/team0.json --rename=bond0
nmcli con add type bond ifname bond0 bond.options "mode=active-
backup,num_grat_arp=1,num_unsol_na=1,resent_igmp=1,miimon=100,miimon=100"
nmcli con add type ethernet ifname enp7s0 controller bond0
nmcli con add type ethernet ifname enp8s0 controller bond0
```

第一个命令包含两个 **miimon** 选项，因为团队配置文件包含两个 **link_watch** 条目。请注意，这不会影响创建绑定。

如果您将服务绑定到团队的设备名称并希望避免更新或破坏这些服务，请省略 **--rename=bond0** 选项。在这种情况下，**team2bond** 为绑定使用与团队相同的接口名称。

- 验证推荐 **team2bond** 工具的绑定选项是否正确。
- 创建绑定。您可以执行建议的 **nmcli** 命令，或使用 **--exec-cmd** 选项重新运行 **team2bond** 命令：

```
# team2bond --config=/tmp/team0.json --rename=bond0 --exec-cmd
Connection 'bond-bond0' (0241a531-0c72-4202-80df-73eadfc126b5) successfully added.
Connection 'bond-port-enp7s0' (38489729-b624-4606-a784-1ccf01e2f6d6) successfully
added.
Connection 'bond-port-enp8s0' (de97ec06-7daa-4298-9a71-9d4c7909daa1) successfully
added.
```

下一步需要绑定连接配置集的名称(**bond-bond0**)。

- 将之前在 **team-team0** 中配置的 IPv4 设置设置为 **bond-bond0** 连接：

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

- 将之前在 **team-team0** 中配置的 IPv6 设置设置为 **bond-bond0** 连接：

```
# nmcli connection modify bond-bond0 ipv6.addresses '2001:db8:1::1/64'
# nmcli connection modify bond-bond0 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify bond-bond0 ipv6.dns '2001:db8:1::fffd'
```

```
# nmcli connection modify bond-bond0 ipv6.dns-search 'example.com'
# nmcli connection modify bond-bond0 ipv6.method manual
```

9. 激活连接：

```
# nmcli connection up bond-bond0
```

验证

1. 显示 **bond-bond0** NetworkManager 连接的 IP 配置：

```
# nmcli connection show bond-bond0 | egrep "^ip"
...
ipv4.method:                manual
ipv4.dns:                    192.0.2.253
ipv4.dns-search:             example.com
ipv4.addresses:              192.0.2.1/24
ipv4.gateway:                192.0.2.254
...
ipv6.method:                 manual
ipv6.dns:                    2001:db8:1::fffd
ipv6.dns-search:             example.com
ipv6.addresses:              2001:db8:1::1/64
ipv6.gateway:                2001:db8:1::fffe
...
```

2. 显示绑定状态：

```
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.13.0-0.rc7.51.el9.x86_64

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: enp7s0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: enp7s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:bf:b1:a9
Slave queue ID: 0

Slave Interface: enp8s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:04:36:0f
Slave queue ID: 0
```


■

在这个示例中，两个端口都是上线的。

3. 验证绑定故障切换是否正常工作：

- a. 从主机中临时删除网络电缆。请注意，无法使用命令行正确测试链路失败事件。
- b. 显示绑定状态：

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

4.2. 了解控制器和端口接口的默认行为

在使用 **NetworkManager** 服务管理或排除团队或绑定端口接口故障排除时，请考虑以下默认行为：

- 启动控制器接口不会自动启动端口接口。
- 启动端口接口总会启动控制器接口。
- 停止控制器接口也会停止端口接口。
- 没有端口的控制器可以启动静态 IP 连接。
- 没有端口的控制器在启动 DHCP 连接时会等待端口。
- 当您添加具有载体的端口时，等待端口且具有 DHCP 连接的控制器会完成。
- 当您添加不具有载体的端口时，等待端口且具有 DHCP 连接的控制器将继续等待。

4.3. 了解 TEAMD 服务、运行程序和 LINK-WATCHERS

团队服务 **teamd** 控制团队驱动程序的一个实例。这个驱动的实例添加硬件设备驱动程序实例组成一个网络接口组。团队驱动程序向内核提供一个网络接口，如 **team0**。

teamd 服务对所有团队方法实现通用逻辑。这些功能对不同的负载共享和备份方法（如循环）是唯一的，并由称为 **runners** 的单独的代码单元来实现。管理员以 JavaScript 对象表示法(JSON)格式指定runners，在创建实例时，JSON 代码被编译到 **teamd** 实例中。另外，在使用 **NetworkManager** 时，您可以在 **team.runner** 参数中设置 runner，**NetworkManager** 会自动创建对应的 JSON 代码。

可用的 runner 如下：

- **broadcast**：转换所有端口上的数据。
- **roundrobin**：依次转换所有端口上的数据。
- **activebackup**：转换一个端口上的数据，而其他端口上的数据则作为备份保留。
- **loadbalance**：转换所有具有活跃的 Tx 负载均衡和基于 Berkeley 数据包过滤器(BPF)的 Tx 端口选择器的端口上的数据。
- **random**：转换随机选择的端口上的数据。
- **lacp**：实现 802.3ad 链路聚合控制协议(LACP)。

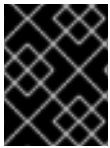
teamd 服务使用链路监视器来监控从属设备的状态。可用的 link-watchers 如下：

- **ethtool** : **libteam** 库使用 **ethtool** 工具来监视链接状态的变化。这是默认的 link-watcher。
- **arp_ping**: **libteam** 库使用 **arp_ping** 工具来监控使用地址解析协议(ARP)的远端硬件地址是否存在。
- **nsna_ping**: 在 IPv6 连接上, **libteam** 库使用来自 IPv6 邻居发现协议的邻居广告和邻居请求功能来监控邻居接口的存在。

每个 runner 都可以使用任何链接监视器, 但 **lACP** 除外。此 runner 只能使用 **ethtool** 链接监视器。

4.4. 使用 NMCLI配置网络团队

要在命令行中配置网络团队, 请使用 **nmcli** 工具。



重要

网络 teaming 在 Red Hat Enterprise Linux 9 中已弃用。考虑使用网络绑定驱动程序作为替代方案。详情请参阅 [配置网络绑定](#)。

前提条件

- 已安装 **teamd** 和 **NetworkManager-team** 软件包。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作组的端口, 必须在服务器中安装物理或者虚拟以太网设备并连接到交换机。
- 要使用 bond、bridge 或 VLAN 设备作为团队的端口, 您可以在创建团队时创建这些设备, 或者预先创建它们, 如下所述 :
 - [使用 nmcli 配置网络绑定](#)
 - [使用 nmcli 配置网桥](#)
 - [使用 nmcli 配置 VLAN 标记](#)

流程

1. 创建团队接口 :

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

此命令创建一个使用 **activebackup** runner、名为 **team0** 的网络团队。

2. 另外, 还可设置链接监视器。例如, 要在 **team0** 连接配置文件中设置 **ethtool** 链接监视器 :

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

链路监视器支持不同的参数。要为链路监视器设置参数, 请在 **name** 属性中以空格分隔的方式来指定它们。请注意, name 属性必须用引号括起来。例如, 要使用 **ethtool** 链接监视器, 并将其 **delay-up** 参数设置为 **2500** 毫秒 (2.5 秒) :

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

要设置多个链路监视器，每个都使用特定的参数，不同的连接监视器以逗号分隔。以下示例使用 **delay-up** 参数设置 **ethtool** 链接监视器，使用 **source-host** 和 **target-host** 参数设置 **arp_ping** 链路监视器：

```
# 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. 显示网络接口，并记录您要添加到团队中的接口名称：

```
# nmcli device status
DEVICE TYPE    STATE    CONNECTION
enp7s0 ethernet disconnected --
enp8s0 ethernet disconnected --
bond0  bond    connected bond0
bond1  bond    connected bond1
...
```

在本例中：

- 没有配置 **enp7s0** 和 **enp8s0**。要将这些设备用作端口，请在下一步中添加连接配置集。请注意，您只能在没有分配给任何连接的团队中使用以太网接口。
- **bond0** 和 **bond1** 已有连接配置文件。要将这些设备用作端口，请在下一步中修改其配置集。

4. 为团队分配端口接口：

- a. 如果没有配置您要分配给团队的接口，为其创建新的连接配置集：

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

这些命令为 **enp7s0** 和 **enp8s0** 创建配置文件，并将它们添加到 **team0** 连接中。

- b. 将现有的连接配置文件分配给团队：

- i. 将这些连接的 **controller** 参数设置为 **team0**：

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

这些命令将名为 **bond0** 和 **bond1** 的现有连接配置文件分配给 **team0** 连接。

- ii. 重新激活连接：

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

5. 配置 IPv4 设置：

- 要将这个团队设备用作其它设备的端口，请输入：

```
# nmcli connection modify team0 ipv4.method disabled
```

- 要使用 DHCP，不需要进行任何操作。
- 要为 **team0** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

6. 配置 IPv6 设置：

- 要将这个团队设备用作其它设备的端口，请输入：

```
# nmcli connection modify team0 ipv6.method disabled
```

- 要使用无状态地址自动配置(SLAAC)，不需要任何操作。
- 要为 **team0** 连接设置静态 IPv6 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

7. 激活连接：

```
# nmcli connection up team0
```

验证

- 显示团队状态：

```
# 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
```

在这个示例中，两个端口都是上线的。

其他资源

- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)
- [了解 teamd 服务、运行程序和 link-watchers](#)
- [nm-settings\(5\) 手册页](#)
- [teamd.conf\(5\) 手册页](#)

4.5. 使用 RHEL WEB 控制台配置网络团队

如果您希望使用基于 Web 浏览器的界面管理网络设置，请使用 RHEL web 控制台来配置网络团队。



重要

网络 teaming 在 Red Hat Enterprise Linux 9 中已弃用。考虑使用网络绑定驱动程序作为替代方案。详情请参阅 [配置网络绑定](#)。

先决条件

- 已安装 **teamd** 和 **NetworkManager-team** 软件包。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作组的端口，必须在服务器中安装物理或者虚拟以太网设备并连接到交换机。
- 要将 bond、bridge 或 VLAN 设备用作团队的端口，请预先创建它们，如下所述：
 - [使用 RHEL web 控制台配置网络绑定](#)
 - [使用 RHEL web 控制台配置网桥](#)
 - [使用 RHEL web 控制台配置 VLAN 标记](#)

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add team**。
3. 输入您要创建的团队设备名称。
4. 选择应该是团队端口的接口。
5. 选择团队的运行程序。
如果您选择 **Load balancing** 或 **802.3ad LACP**，Web 控制台会显示额外的 **Balancer** 字段。
6. 设置链接监视器：
 - 如果您选择 **Ethtool**，请设置链接并关闭延迟。
 - 如果您设置了 **ARP ping** 或 **NSNA ping**，还要设置 ping 间隔并 ping 目标。

Team settings ×

Name

Ports enp7s0
 enp8s0

Runner

Link watch

Link up delay

Link down delay

7. 点应用。
8. 默认情况下，团队使用动态 IP 地址。如果要设置静态 IP 地址：
 - a. 在 **Interfaces** 部分点团队名称。
 - b. 点您要配置的协议旁的 **Edit**。
 - c. 选择 **Addresses** 旁的 **Manual**，并输入 IP 地址、前缀和默认网关。
 - d. 在 **DNS** 部分，点 **+** 按钮，并输入 DNS 服务器的 IP 地址。重复此步骤来设置多个 DNS 服务器。
 - e. 在 **DNS search domains** 部分中，点 **+** 按钮并输入搜索域。
 - f. 如果接口需要静态路由，请在 **Routes** 部分配置它们。

IPv4 settings ×

Addresses Manual ▾ +

Address	Prefix length or netmask	Gateway	-
<input type="text" value="192.0.2.1"/>	<input type="text" value="24"/>	<input type="text" value="192.0.2.254"/>	-

DNS Automatic +

Server -

DNS search domains Automatic +

Search domain -

Routes Automatic +

Apply Cancel

g. 点 应用

验证

1. 在屏幕左侧的导航中选择 **Networking** 选项卡，并检查接口上是否有传入和传出流量。

Interfaces Add bond Add team Add bridge Add VLAN 			
Name	IP address	Sending	Receiving
team0	192.0.2.1/24	1.11 Mbps	61.2 Mbps

2. 显示团队状态：

```
# 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

```

在这个示例中，两个端口都是上线的。

其他资源

- [网络团队运行程序](#)

4.6. 使用 NM-CONNECTION-EDITOR 配置网络团队

如果使用带有图形界面的 Red Hat Enterprise Linux，您可以使用 **nm-connection-editor** 应用程序配置网络团队。

请注意：**nm-connection-editor** 只能向团队添加新端口。要使用现有连接配置文件作为端口，请使用 **nmcli** 工具创建团队，如 [使用 nmcli 配置网络团队](#) 中所述。



重要

网络 teaming 在 Red Hat Enterprise Linux 9 中已弃用。考虑使用网络绑定驱动程序作为替代方案。详情请参阅 [配置网络绑定](#)。

前提条件

- 已安装 **teamd** 和 **NetworkManager-team** 软件包。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作组的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bond 或 VLAN 设备作为团队的端口，请确保这些设备还没有配置。

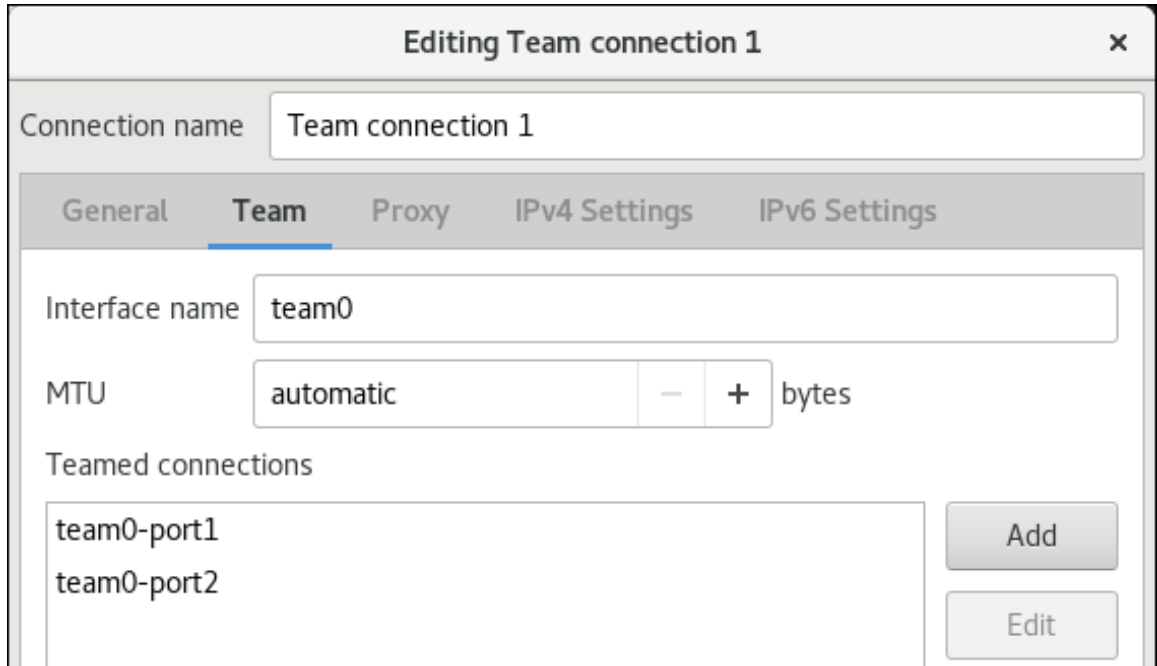
步骤

1. 打开一个终端，输入 **nm-connection-editor**：

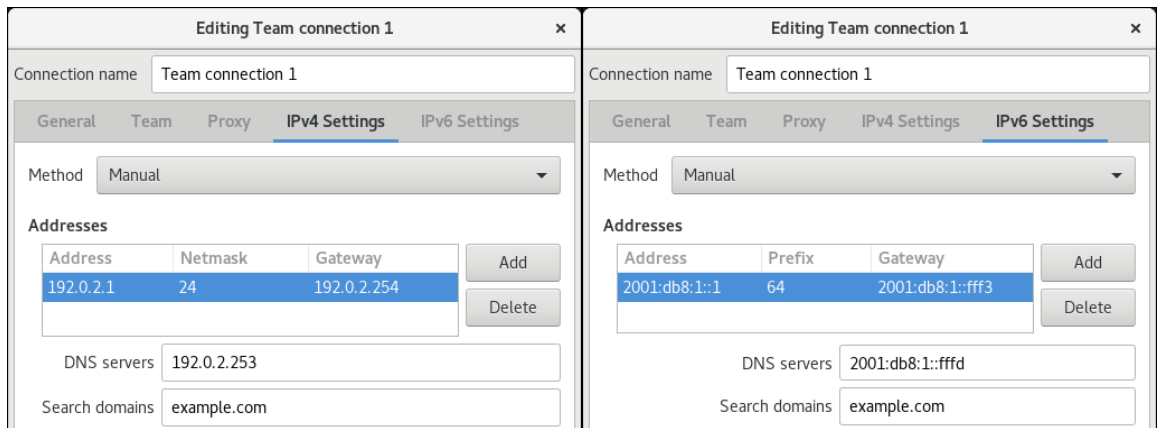
```
$ nm-connection-editor
```

2. 点击 **+** 按钮来添加一个新的连接。
3. 选择 **Team** 连接类型，然后单击 **Create**。
4. 在 **Team** 选项卡中：
 - a. 可选：在 **Interface name** 字段中设置团队接口的名称。
 - b. 点 **Add** 按钮为网络接口添加新连接配置集，并将配置集作为端口添加到团队。

- i. 选择接口的连接类型。例如，为有线连接选择 **Ethernet**。
 - ii. 可选：为端口设置连接名称。
 - iii. 如果为以太网设备创建连接配置文件，请打开 **Ethernet** 选项卡，在 **Device** 字段中选择您要作为端口添加到团队的网络接口。如果您选择了不同的设备类型，请相应地进行配置。请注意，您只能在没有分配给任何连接的团队中使用以太网接口。
 - iv. 点 **Save**。
- c. 对您要添加到团队的每个接口重复前面的步骤。



- d. 点 **Advanced** 按钮将高级选项设置为团队连接。
 - i. 在 **Runner** 选项卡中，选择 runner。
 - ii. 在 **Link Watcher** 选项卡中，设置链接监视器及其可选设置。
 - iii. 点**确定**。
5. 在 **IPv4 Settings** 和 **IPv6 Settings** 标签页中配置 IP 地址设置：
- 要将这个网桥设备用作其他设备的端口，请将 **Method** 字段设置为 **Disabled**。
 - 要使用 DHCP，请将 **Method** 字段保留为默认值 **Automatic (DHCP)**。
 - 要使用静态 IP 设置，请将 **Method** 字段设置为 **Manual**，并相应地填写字段：



6. 点 **Save**。

7. 关闭 **nm-connection-editor**。

验证

- 显示团队状态：

```
# 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
```

其它资源

- [使用 nm-connection-editor 配置网络绑定](#)
- [使用 nm-connection-editor 配置网络团队](#)
- [使用 nm-connection-editor 配置 VLAN 标记](#)
- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)
- [了解 teamd 服务、运行程序和 link-watchers](#)
- [重启 NetworkManager 服务后 NetworkManager 会复制连接](#)

第 5 章 配置 VLAN 标记

Virtual Local Area Network (VLAN) 是物理网络中的一个逻辑网络。当 VLAN 接口通过接口时，VLAN 接口标签带有 VLAN ID 的数据包，并删除返回的数据包的标签。您可以在另一个接口（如以太网、绑定、团队或桥接设备）上创建 VLAN 接口。这些接口称为 **父接口**。

Red Hat Enterprise Linux 为管理员提供不同的选项来配置 VLAN 设备。例如：

- 使用 **nmcli** 使用命令行配置 VLAN 标记。
- 通过 RHEL web 控制台使用 Web 浏览器配置 VLAN 标记。
- 使用 **nmtui** 在基于文本的用户界面中配置 VLAN 标记。
- 使用 **nm-connection-editor** 应用程序在图形界面中配置连接。
- 使用 **nmstatectl** 通过 Nmstate API 配置连接。
- 使用 RHEL 系统角色自动化一个或多个主机上的 VLAN 配置。

5.1. 使用 NMCLI 配置 VLAN 标记

您可以使用 **nmcli** 实用程序在命令行中配置 Virtual Local Area Network (VLAN) 标记。

前提条件

- 您计划用作虚拟 VLAN 接口的父接口支持 VLAN 标签。
- 如果您在绑定接口之上配置 VLAN：
 - 绑定的端口是上线的。
 - 这个绑定没有使用 **fail_over_mac=follow** 选项进行配置。VLAN 虚拟设备无法更改其 MAC 地址以匹配父设备的新 MAC 地址。在这种情况下，流量仍会与不正确的源 MAC 地址一同发送。
 - 这个绑定通常不会预期从 DHCP 服务器或 IPv6 自动配置获取 IP 地址。在创建绑定时通过设置 **ipv4.method=disable** 和 **ipv6.method=ignore** 选项来确保它。否则，如果 DHCP 或 IPv6 自动配置在一段时间后失败，接口可能会关闭。
- 主机连接到的交换机被配置为支持 VLAN 标签。详情请查看您的交换机文档。

流程

1. 显示网络接口：

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

2. 创建 VLAN 接口。例如，要创建一个使用 **enp1s0** 作为其父接口，使用 VLAN ID **10** 标记数据包，名为 **vlan10** 的 VLAN 接口，请输入：

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

请注意，VLAN 必须在范围 0 到 4094 之间。

- 默认情况下，VLAN 连接会继承上级接口的最大传输单元（MTU）。另外，还可设置不同的 MTU 值：

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

- 配置 IPv4 设置：

- 要将这个 VLAN 设备用作其它设备的端口，请输入：

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

- 要使用 DHCP，不需要进行任何操作。
- 要为 **vlan10** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

- 配置 IPv6 设置：

- 要将这个 VLAN 设备用作其它设备的端口，请输入：

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

- 要使用无状态地址自动配置(SLAAC)，不需要任何操作。
- 要为 **vlan10** 连接设置静态 IPv6 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

- 激活连接：

```
# nmcli connection up vlan10
```

验证

- 验证设置：

```
# 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
```

其他资源

- [nm-settings\(5\) 手册页](#)

5.2. 使用 NMCLI 配置嵌套 VLAN

802.1ad 是用于虚拟局域网(VLAN)标记的协议。它也被称为 Q-in-Q 标记。您可以使用此技术在单个以太网框架中创建多个 VLAN 标签，以实现以下优点：

- 通过在 VLAN 中创建多个隔离网络段来提高网络可扩展性。这可让您将大型网络分段并组织成较小的、可管理的单元。
- 通过隔离和控制不同类型的网络流量，改进了流量管理。这可以提高网络性能并减少网络拥塞。
- 通过创建较小的，多个目标网络段，实现高效资源利用率。
- 通过隔离网络流量，降低未经授权访问敏感数据的风险，提高了安全性。

前提条件

- 您计划用作虚拟 VLAN 接口的父接口支持 VLAN 标签。
- 如果您在绑定接口之上配置 VLAN：
 - 绑定的端口是上线的。
 - 这个绑定没有使用 **fail_over_mac=follow** 选项进行配置。VLAN 虚拟设备无法更改其 MAC 地址以匹配父设备的新 MAC 地址。在这种情况下，流量仍会与不正确的源 MAC 地址一同发送。
 - 这个绑定通常不会预期从 DHCP 服务器或 IPv6 自动配置获取 IP 地址。在创建绑定时通过设置 **ipv4.method=disable** 和 **ipv6.method=ignore** 选项来确保它。否则，如果 DHCP 或 IPv6 自动配置在一段时间后失败，接口可能会关闭。
- 主机连接到的交换机被配置为支持 VLAN 标签。详情请查看您的交换机文档。

步骤

1. 显示物理网络设备：

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

2. 创建基础 VLAN 接口。例如，要创建一个名为 **vlan10** 的基本 VLAN 接口，它使用 **enp1s0** 作为其父接口，以及 VLAN ID 为 **10** 的标记数据包，请输入：

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

请注意，VLAN 必须在范围 **0** 到 **4094** 之间。

- 默认情况下，VLAN 连接会继承上级接口的最大传输单元（MTU）。另外，还可设置不同的 MTU 值：

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

- 在基本 VLAN 接口之上创建嵌套的 VLAN 接口：

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

此命令会在父 VLAN 连接 **vlan10** 上创建一个新的 VLAN 连接，名称为 **vlan10.20**，VLAN ID 为 **20**。**dev** 选项指定父网络设备。在本例中，是 **enp1s0.10**。**vlan.protocol** 选项指定 VLAN 封装协议。在本例中，它是 **802.1ad** (Q-in-Q)。

- 配置嵌套 VLAN 接口的 IPv4 设置：

- 要使用 DHCP，不需要进行任何操作。
- 要为 **vlan10.20** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

- 配置嵌套 VLAN 接口的 IPv6 设置：

- 要使用无状态地址自动配置(SLAAC)，不需要任何操作。
- 要为 **vlan10** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

- 激活配置文件：

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

验证

- 验证嵌套 VLAN 接口的配置：

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

其他资源

- [nm-settings\(5\) 手册页](#)

5.3. 使用 RHEL WEB 控制台配置 VLAN 标记

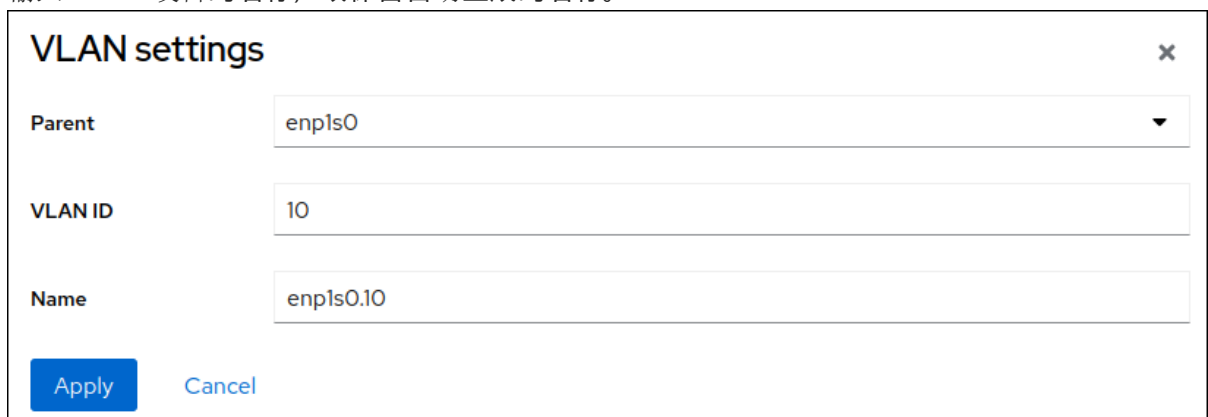
如果您希望使用基于 Web 浏览器的界面管理网络设置，请使用 RHEL web 控制台配置 VLAN 标记。

前提条件

- 您计划用作虚拟 VLAN 接口的父接口支持 VLAN 标签。
- 如果您在绑定接口之上配置 VLAN：
 - 绑定的端口是上线的。
 - 这个绑定没有使用 **fail_over_mac=follow** 选项进行配置。VLAN 虚拟设备无法更改其 MAC 地址以匹配父设备的新 MAC 地址。在这种情况下，流量仍会与不正确的源 MAC 地址一同发送。
 - 这个绑定通常不会预期从 DHCP 服务器或 IPv6 自动配置获取 IP 地址。禁用 IPv4 和 IPv6 协议创建绑定以确保它。否则，如果 DHCP 或 IPv6 自动配置在一段时间后失败，接口可能会关闭。
- 主机连接到的交换机被配置为支持 VLAN 标签。详情请查看您的交换机文档。

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add VLAN**。
3. 选择父设备。
4. 输入 VLAN ID。
5. 输入 VLAN 设备的名称，或保留自动生成的名称。



6. 点应用。
7. 默认情况下，VLAN 设备使用动态 IP 地址。如果要设置静态 IP 地址：
 - a. 点 **Interfaces** 部分中的 VLAN 设备名称。
 - b. 点您要配置的协议旁的 **Edit**。

- c. 选择 **Addresses** 旁的 **Manual**，并输入 IP 地址、前缀和默认网关。
- d. 在 **DNS** 部分，点 **+** 按钮，并输入 DNS 服务器的 IP 地址。重复此步骤来设置多个 DNS 服务器。
- e. 在 **DNS search domains** 部分中，点 **+** 按钮并输入搜索域。
- f. 如果接口需要静态路由，请在 **Routes** 部分配置它们。

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. 点 应用

验证

- 在屏幕左侧的导航中选择 **Networking** 选项卡，并检查接口上是否有传入和传出流量：

Interfaces			
Name	IP address	Sending	Receiving
enp1s0.10	192.0.2.1/24	1.11 Mbps	61.2 Mbps

5.4. 使用 NMTUI 配置 VLAN 标记

nmtui 应用程序为 NetworkManager 提供了一个基于文本的用户界面。您可以使用 **nmtui** 在没有图形界面的主机上配置 VLAN 标签。



注意

在 **nmtui** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

前提条件

- 您计划用作虚拟 VLAN 接口的父接口支持 VLAN 标签。
- 如果您在绑定接口之上配置 VLAN：
 - 绑定的端口是上线的。
 - 这个绑定没有使用 **fail_over_mac=follow** 选项进行配置。VLAN 虚拟设备无法更改其 MAC 地址以匹配父设备的新 MAC 地址。在这种情况下，流量仍会与不正确的源 MAC 地址一同发送。
 - 这个绑定通常不会预期从 DHCP 服务器或 IPv6 自动配置获取 IP 地址。在创建绑定时通过设置 **ipv4.method=disable** 和 **ipv6.method=ignore** 选项来确保它。否则，如果 DHCP 或 IPv6 自动配置在一段时间后失败，接口可能会关闭。
- 主机连接到的交换机被配置为支持 VLAN 标签。详情请查看您的交换机文档。

流程

1. 如果您不知道要在其上配置 VLAN 标签的网络设备名称，请显示可用的设备：

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

2. 启动 **nmtui**：

```
# nmtui
```

3. 选择 **Edit a connection**，然后按 **Enter**。
4. 按 **添加**。
5. 从网络类型列表中选择 **VLAN**，然后按 **Enter** 键。
6. 可选：为要创建的 NetworkManager 配置文件输入一个名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
7. 在 **Device** 字段中输入要创建的 VLAN 设备名称。
8. 在 **Parent** 字段中输入您要在其上配置 VLAN 标记的设备名称。
9. 输入 VLAN ID。ID 必须在 **0** 到 **4094** 之间。

10. 根据您的环境，相应地在 **IPv4 配置**和 **IPv6 配置区**中配置 IP 地址设置。为此，请按这些区域旁边的按钮，并选择：
 - **Disabled**，如果此 VLAN 设备不需要 IP 地址，或者您想要将其用作其它设备的端口。
 - **Automatic**，如果 DHCP 服务器或无状态地址自动配置(SLAAC)为 VLAN 设备动态分配一个 IP 地址。
 - **Manual**，如果网络需要静态 IP 地址设置。在这种情况下，您必须填写更多字段：
 - i. 在您要配置的协议旁边按 **Show** 以显示其他字段。
 - ii. 按 **Addresses** 旁边的 **Add**，并以无类别域间路由(CIDR)格式输入 IP 地址和子网掩码。如果没有指定子网掩码，NetworkManager 会为 IPv4 地址设置 **/32** 子网掩码，为 IPv6 地址设置 **/64** 子网掩码。
 - iii. 输入默认网关的地址。
 - iv. 按 **DNS 服务器** 旁边的 **Add**，并输入 DNS 服务器地址。
 - v. 按 **搜索域** 旁边的 **Add**，并输入 DNS 搜索域。

图 5.1. 具有静态 IP 地址设置的 VLAN 连接示例

Edit Connection

Profile name
Device

VLAN <Hide>

Parent
VLAN id

Cloned MAC address
MTU

IPv4 CONFIGURATION <Manual> <Hide>

Addresses <Remove>

Gateway
DNS servers <Remove>

Search domains

Routing (No custom routes)
 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>

Gateway
DNS servers <Remove>

Search domains

Routing (No custom routes)
 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>

11. 按 **OK** 创建并自动激活新连接。
12. 按 **Back** 返回到主菜单。
13. 选择 **Quit**，然后按 **Enter** 关闭 **nmtui** 应用程序。

验证

- 验证设置：

```
# 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
```

5.5. 使用 NM-CONNECTION-EDITOR 配置 VLAN 标记

您可以使用 **nm-connection-editor** 应用程序在图形界面中配置 Virtual Local Area Network (VLAN) 标记。

前提条件

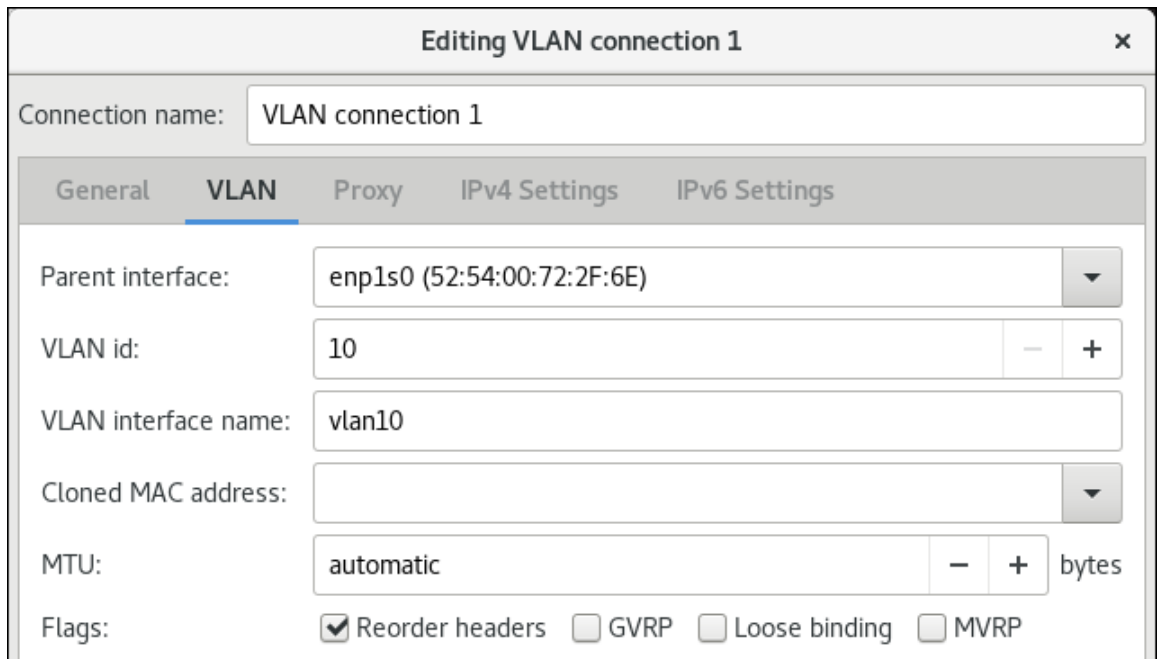
- 您计划用作虚拟 VLAN 接口的父接口支持 VLAN 标签。
- 如果您在绑定接口之上配置 VLAN：
 - 绑定的端口是上线的。
 - 这个绑定没有使用 **fail_over_mac=follow** 选项进行配置。VLAN 虚拟设备无法更改其 MAC 地址以匹配父设备的新 MAC 地址。在这种情况下，流量仍会与不正确的源 MAC 地址一同发送。
- 主机已连接，以支持 VLAN 标签。详情请查看您的交换机文档。

步骤

1. 打开一个终端，输入 **nm-connection-editor**：

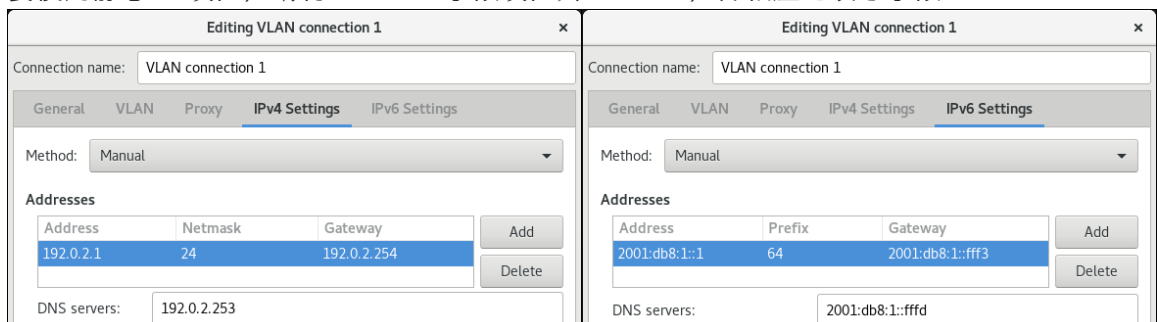
```
$ nm-connection-editor
```

2. 点击 **+** 按钮来添加一个新的连接。
3. 选择 **VLAN** 连接类型，然后单击 **Create**。
4. 在 **VLAN** 选项卡中：
 - a. 选择上级接口。
 - b. 选择 VLAN ID。请注意，VLAN 必须在 **0** 到 **4094** 之间。
 - c. 默认情况下，VLAN 连接会继承上级接口的最大传输单元 (MTU)。另外，还可设置不同的 MTU 值。
 - d. 另外，还可设置 VLAN 接口的名称以及其它特定 VLAN 选项。



5. 在 **IPv4 Settings** 和 **IPv6 Settings** 标签页中配置 IP 地址设置：

- 要将这个网桥设备用作其他设备的端口，请将 **Method** 字段设置为 **Disabled**。
- 要使用 DHCP，请将 **Method** 字段保留为默认值 **Automatic (DHCP)**。
- 要使用静态 IP 设置，请将 **Method** 字段设置为 **Manual**，并相应地填写字段：



6. 点 **Save**。

7. 关闭 **nm-connection-editor**。

验证

1. 验证设置：

```
# 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
```

其他资源

- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)

5.6. 使用 NMSTATECTL 配置 VLAN 标记

使用 **nmstatectl** 工具通过 Nmstate API 配置虚拟局域网 VLAN。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，**nmstatectl** 会自动回滚更改以避免系统处于不正确的状态。

根据您的环境，相应地调整 YAML 文件。例如，要在 VLAN 中使用与以太网适配器不同的设备，请调整您在 VLAN 中使用的端口的 **base-iface** 属性和 **type** 属性。

先决条件

- 要将以太网设备用作 VLAN 中的端口，必须在服务器中安装物理或者虚拟以太网设备。
- **nmstate** 软件包已安装。

步骤

1. 创建包含一个以下内容的 YAML 文件，如 **~/create-vlan.yml**：

```
---
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

```

这些设置定义一个 ID 为 10 的 VLAN，它使用 **enp1s0** 设备。作为子设备，VLAN 连接有以下设置：

- 静态 IPv4 地址 - **192.0.2.1**，子网掩码为 **/24**
- 静态 IPv6 地址 - **2001:db8:1::1**，子网掩码为 **/64**
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**

2. 将设置应用到系统：

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

验证

1. 显示设备和连接的状态：

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

2. 显示连接配置集的所有设置：

```
# 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. 以 YAML 格式显示连接设置：

```
# nmstatectl show vlan0
```

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

5.7. 使用 NETWORK RHEL 系统角色配置 VLAN 标记

您可以使用 **network** RHEL 系统角色配置 VLAN 标记。这个示例在这个以太网连接之上添加了一个以太网连接和 ID 为 **10** 的 VLAN。作为子设备，VLAN 连接包含 IP、默认网关和 DNS 配置。

根据您的环境，相应地进行调整。例如：

- 若要将 VLAN 用作其他连接中的端口（如绑定），请省略 **ip** 属性，并在子配置中设置 IP 配置。
- 若要在 VLAN 中使用 team、bridge 或 bond 设备，请调整您在 VLAN 中使用的端口的 **interface_name** 和 **类型** 属性。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a VLAN that uses an Ethernet connection
      ansible.builtin.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

```

这些设置定义一个 VLAN 以在 **enp1s0** 设备上操作。VLAN 接口有以下设置：

- 静态 IPv4 地址 - **192.0.2.1** 和 /24 子网掩码
 - 静态 IPv6 地址 - **2001:db8:1::1** 和 /64 子网掩码
 - IPv4 默认网关 - **192.0.2.254**
 - IPv6 默认网关 - **2001:db8:1::fffe**
 - IPv4 DNS 服务器 - **192.0.2.200**
 - IPv6 DNS 服务器 - **2001:db8:1::ffbb**
 - DNS 搜索域 - **example.com**
 - VLAN ID - **10**
- VLAN 配置文件中的 **parent** 属性将 VLAN 配置为在 **enp1s0** 设备之上运行。作为子设备，VLAN 连接包含 IP、默认网关和 DNS 配置。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

第 6 章 配置网络桥接

网络桥接是一个链路层设备，它可根据 MAC 地址列表转发网络间的流量。网桥通过侦听网络流量并了解连接到每个网络的主机来构建 MAC 地址表。例如，您可以使用 Red Hat Enterprise Linux 主机上的软件桥接模拟硬件桥接或在虚拟化环境中，将虚拟机(VM)集成到与主机相同的网络中。

桥接需要在桥接应该连接的每个网络中有一个网络设备。当您配置网桥时，网桥被称为 **controller**，其使用的设备为 **ports**。

您可以在不同类型的设备中创建桥接，例如：

- 物理和虚拟以太网设备
- 网络绑定
- 网络团队 (team)
- VLAN 设备

由于 IEEE 802.11 标准指定在 Wi-Fi 中使用 3 个地址帧以便有效地使用随机时间，您无法通过 Ad-Hoc 或者 Infrastructure 模式中的 Wi-Fi 网络配置网桥。

6.1. 使用 NMCLI 配置网络桥接

要在命令行上配置网桥，请使用 **nmcli** 工具。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作网桥的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bond 或 VLAN 设备作为网桥的端口，您可以在创建桥接时创建这些设备，或者预先创建它们，如：
 - [使用 nmcli 配置网络团队](#)
 - [使用 nmcli 配置网络绑定](#)
 - [使用 nmcli 配置 VLAN 标记](#)

步骤

1. 创建网桥接口：

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

这个命令会创建一个名为 **bridge0** 的网桥，输入：

2. 显示网络接口，并记录您要添加到网桥中的接口名称：

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

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

在本例中：

- 没有配置 **enp7s0** 和 **enp8s0**。要将这些设备用作端口，请在下一步中添加连接配置集。
- **bond0** 和 **bond1** 已有连接配置文件。要将这些设备用作端口，请在下一步中修改其配置集。

3. 将接口分配给网桥。

- a. 如果没有配置您要分配给网桥的接口，为其创建新的连接配置集：

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

这些命令为 **enp7s0** 和 **enp8s0** 创建配置文件，并将它们添加到 **bridge0** 连接中。

- b. 如果要将现有的连接配置文件分配给网桥：

- i. 将这些连接的 **controller** 参数设置为 **bridge0**：

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

这些命令将名为 **bond0** 和 **bond1** 的现有连接配置文件分配给 **bridge0** 连接。

- ii. 重新激活连接：

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

4. 配置 IPv4 设置：

- 要将这个网桥设备用作其它设备的端口，请输入：

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

- 要使用 DHCP，不需要进行任何操作。
- 要为 **bridge0** 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器设置，请输入：

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

5. 配置 IPv6 设置：

- 要将这个网桥设备用作其它设备的端口，请输入：

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

- 要使用无状态地址自动配置(SLAAC), 不需要任何操作。
- 要为 **bridge0** 连接设置静态 IPv6 地址、网络掩码、默认网关和 DNS 服务器设置, 请输入 :

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

6. 可选 : 配置网桥的其他属性。例如, 要将 **bridge0** 的生成树协议(STP)优先级设为 **16384**, 请输入 :

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

默认情况下启用 STP。

7. 激活连接 :

```
# nmcli connection up bridge0
```

8. 验证端口是否已连接, 并且 **CONNECTION** 列是否显示端口的连接名称 :

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

当您激活连接的任何端口时, NetworkManager 也会激活网桥, 但不会激活它的其它端口。您可以配置 Red Hat Enterprise Linux 在启用桥接时自动启用所有端口 :

- a. 启用网桥连接的 **connection.autoconnect-ports** 参数 :

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

- b. 重新激活桥接 :

```
# nmcli connection up bridge0
```

验证

- 使用 **ip** 工具来显示作为特定网桥端口的以太网设备的链接状态 :

```
# 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
```

- 使用 **bridge** 工具来显示作为任意网桥设备端口的以太网设备状态 :

```
# 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
...

```

要显示特定以太网设备的状态，请使用 `bridge link show dev <ethernet_device_name>` 命令。

其他资源

- [nm-settings\(5\) 手册页](#)
- [bridge\(8\) 手册页](#)
- [重启 NetworkManager 服务后 NetworkManager 会复制连接](#)
- [如何配置具有 VLAN 信息的网桥？](#)

6.2. 使用 RHEL WEB 控制台配置网桥

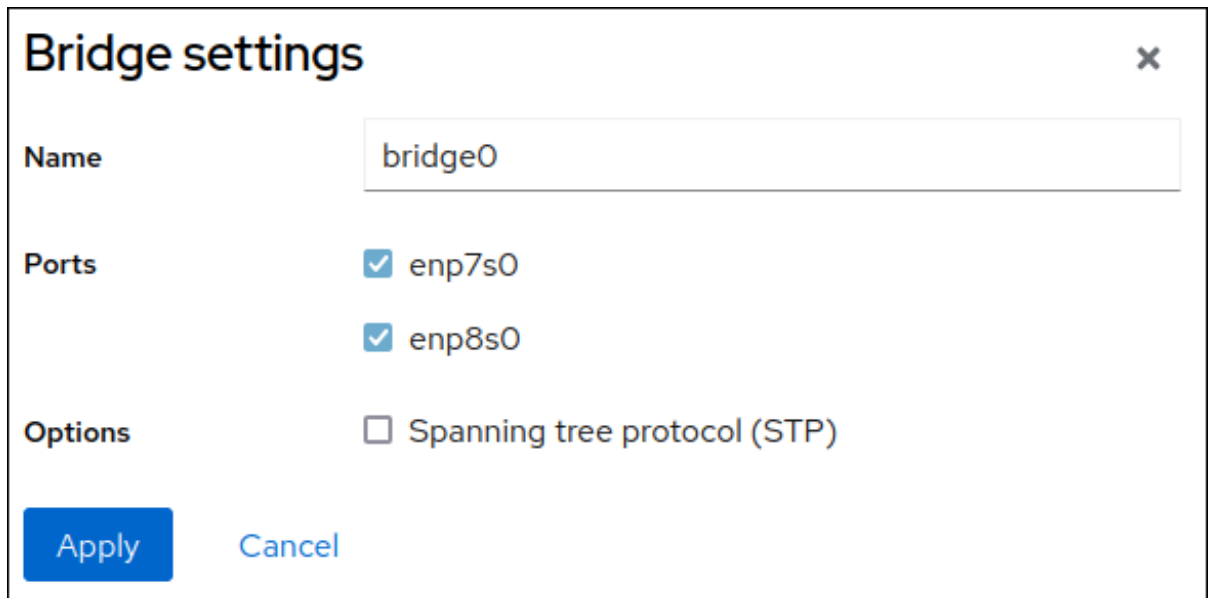
如果您希望使用基于 Web 浏览器的界面管理网络设置，请使用 RHEL web 控制台来配置网桥。

前提条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作网桥的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bond 或 VLAN 设备作为网桥的端口，您可以在创建桥接时创建这些设备，或者预先创建它们，如：
 - [使用 RHEL web 控制台配置网络团队](#)
 - [使用 RHEL web 控制台配置网络绑定](#)
 - [使用 RHEL web 控制台配置 VLAN 标记](#)

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add bridge**。
3. 输入您要创建的网桥设备名称。
4. 选择应该是网桥端口的接口。
5. 可选：启用 **Spanning tree 协议(STP)** 功能，以避免桥接循环和广播。



Bridge settings ×

Name

Ports

- enp7s0
- enp8s0

Options

- Spanning tree protocol (STP)

Apply **Cancel**

6. 点应用。
7. 默认情况下，网桥使用动态 IP 地址。如果要设置静态 IP 地址：
 - a. 在 **Interfaces** 部分，点网桥的名称。
 - b. 点您要配置的协议旁的 **Edit**。
 - c. 选择 **Addresses** 旁的 **Manual**，并输入 IP 地址、前缀和默认网关。
 - d. 在 **DNS** 部分，点 **+** 按钮，并输入 DNS 服务器的 IP 地址。重复此步骤来设置多个 DNS 服务器。
 - e. 在 **DNS search domains** 部分中，点 **+** 按钮并输入搜索域。
 - f. 如果接口需要静态路由，请在 **Routes** 部分配置它们。

IPv4 settings ×

Addresses Manual ▾ +

Address	Prefix length or netmask	Gateway	
<input type="text" value="192.0.2.1"/>	<input type="text" value="24"/>	<input type="text" value="192.0.2.254"/>	-

DNS Automatic +

Server -

DNS search domains Automatic +

Search domain -

Routes Automatic +

Apply Cancel

g. 点 应用

验证

1. 在屏幕左侧的导航中选择 **Networking** 选项卡，并检查接口上是否有传入和传出流量：

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

6.3. 使用 NMTUI 配置网桥

nmtui 应用程序为 NetworkManager 提供了一个基于文本的用户界面。您可以使用 **nmtui** 在没有图形界面的主机上配置网桥。



注意

在 **nmtui** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

前提条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作网桥的端口，必须在服务器中安装物理或者虚拟以太网设备。

步骤

1. 如果您不知道您要在其上配置网桥的网络设备名称，请显示可用的设备：

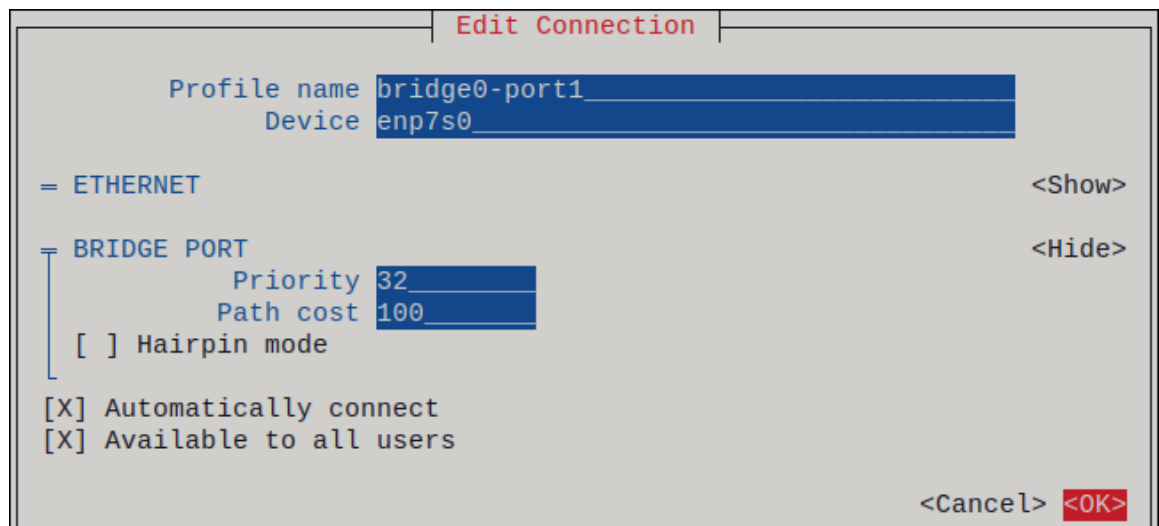
```
# nmcli device status
DEVICE  TYPE    STATE      CONNECTION
enp7s0  ethernet unavailable --
enp8s0  ethernet unavailable --
...
```

2. 启动 **nmtui**：

```
# nmtui
```

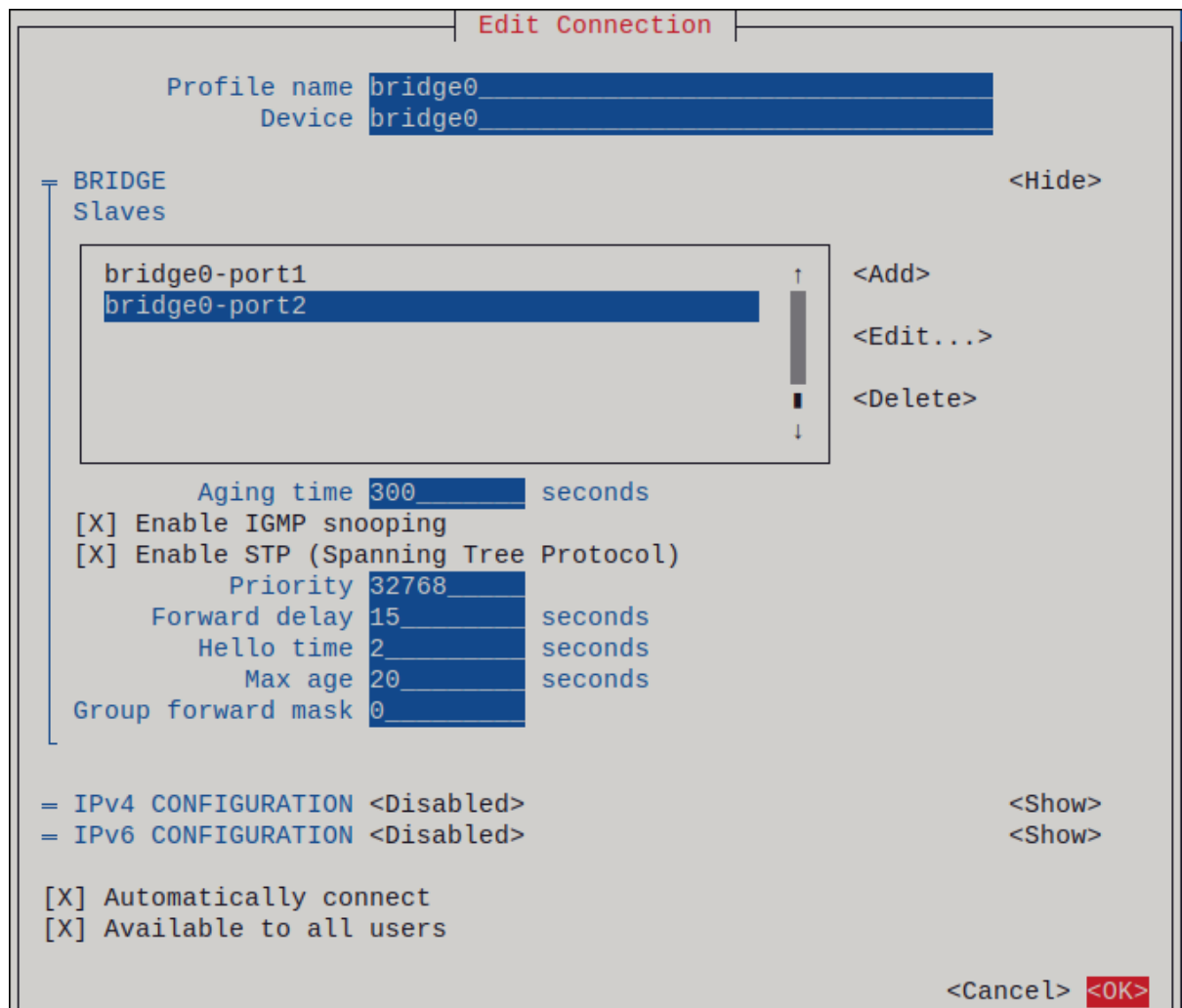
3. 选择 **Edit a connection**，并按 **Enter**。
4. 按 **添加**。
5. 从网络类型列表中选择 **Bridge**，然后按 **Enter**。
6. 可选：为要创建的 NetworkManager 配置文件输入一个名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
7. 在 **Device** 字段中输入要创建的网桥设备名称。
8. 将端口添加到要创建的网桥中：
 - a. 按 **Slaves** 列表旁边的 **Add**。
 - b. 选择您要作为端口添加到网桥的接口类型，例如 **Ethernet**。
 - c. 可选：为这个网桥端口输入要创建的 NetworkManager 配置文件的名称。
 - d. 在 **Device** 字段中输入端口的设备名称。
 - e. 按 **OK** 返回网桥设置窗口。

图 6.1. 将以太网设备作为端口添加到网桥



- f. 重复这些步骤，来向网桥添加更多的端口。
9. 根据您的环境，相应地在 **IPv4 configuration** 和 **IPv6 configuration** 区中配置 IP 地址。为此，请按这些区域旁边的按钮，并选择：
- **Disabled**，如果网桥不需要 IP 地址。
 - **Automatic**，如果 DHCP 服务器或无状态地址自动配置(SLAAC)动态为网桥分配一个 IP 地址。
 - **Manual**，如果网络需要静态 IP 地址设置。在这种情况下，您必须填写更多字段：
 - i. 在您要配置的协议旁边按 **Show** 以显示其他字段。
 - ii. 按 **Addresses** 旁边的 **Add**，并以无类别域间路由(CIDR)格式输入 IP 地址和子网掩码。如果没有指定子网掩码，NetworkManager 会为 IPv4 地址设置 /32 子网掩码，为 IPv6 地址设置 /64 子网掩码。
 - iii. 输入默认网关的地址。
 - iv. 按 **DNS 服务器** 旁边的 **Add**，并输入 DNS 服务器地址。
 - v. 按 **搜索域** 旁边的 **Add**，并输入 DNS 搜索域。

图 6.2. 没有 IP 地址设置的网桥连接的示例



10. 按 **OK** 创建并自动激活新连接。
11. 按 **Back** 返回到主菜单。
12. 选择 **Quit**，然后按 **Enter** 来关闭 **nmtui** 应用程序。

验证

1. 使用 **ip** 工具来显示作为特定网桥端口的以太网设备的链接状态：

```
# 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. 使用 **bridge** 工具来显示作为任意网桥设备端口的以太网设备状态：

```
# 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
...
```

要显示特定以太网设备的状态，请使用 **bridge link show dev < ethernet_device_name >** 命令。

6.4. 使用 NM-CONNECTION-EDITOR 配置网桥

如果使用带有图形界面的 Red Hat Enterprise Linux，您可以使用 **nm-connection-editor** 应用程序配置网桥。

请注意，**nm-connection-editor** 只能向网桥添加新端口。要使用现有连接配置文件作为端口，请使用 **nmcli** 工具创建网桥，如 [使用 nmcli 配置网桥](#) 中所述。

先决条件

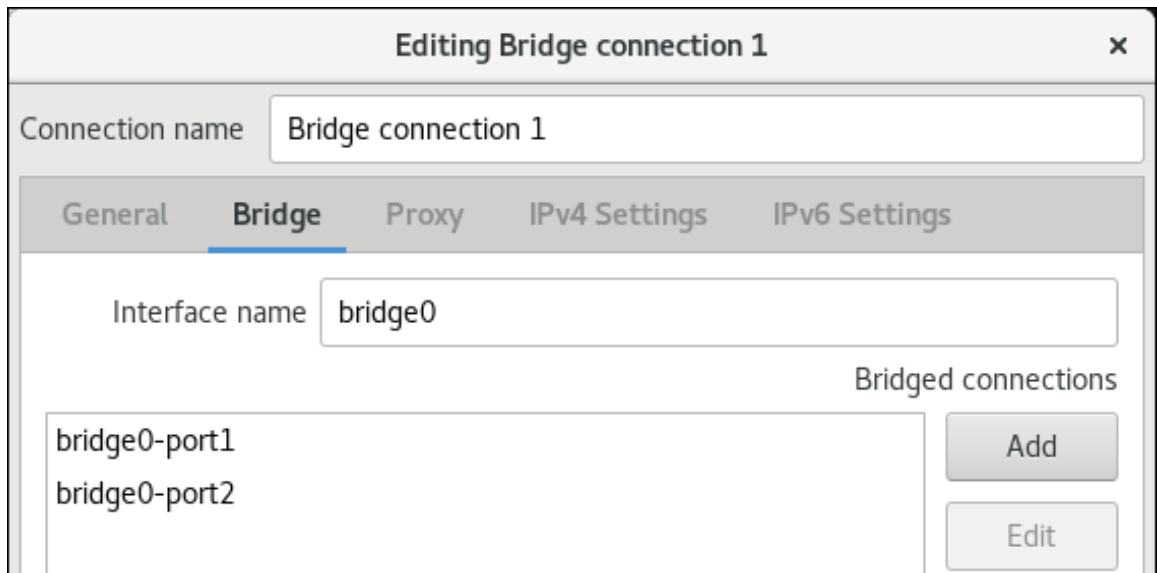
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作网桥的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用 team、bond 或 VLAN 设备作为网桥的端口，请确保这些设备还没有配置。

步骤

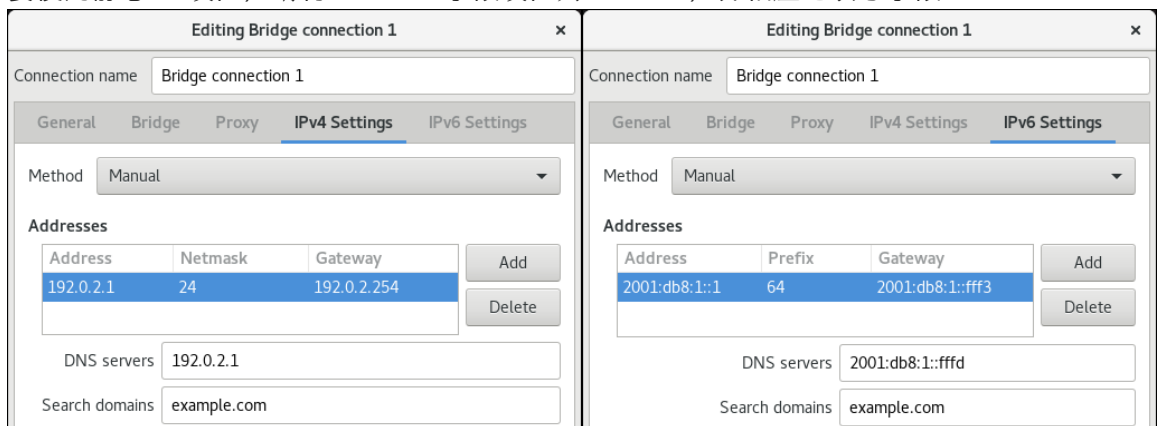
1. 打开一个终端，输入 **nm-connection-editor**：

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

2. 点击 **+** 按钮来添加一个新的连接。
3. 选择 **Bridge** 连接类型，然后单击 **Create**。
4. 在 **Bridge** 选项卡中：
 - a. 可选：在 **Interface name** 字段中设置网桥接口的名称。
 - b. 点 **Add** 按钮为网络接口创建新连接配置集，并将配置集作为端口添加到网桥。
 - i. 选择接口的连接类型。例如，为有线连接选择 **Ethernet**。
 - ii. 另外，还可为端口设备设置连接名称。
 - iii. 如果为以太网设备创建连接配置文件，请打开 **Ethernet** 选项卡，然后在 **Device** 字段中选择您要作为端口添加到网桥的网络接口。如果您选择了不同的设备类型，请相应地进行配置。
 - iv. 点 **Save**。
 - c. 对您要添加到桥接的每个接口重复前面的步骤。



5. 可选：配置其他网桥设置，如生成树协议（STP）选项。
6. 在 **IPv4 Settings** 和 **IPv6 Settings** 标签页中配置 IP 地址设置：
 - 要将这个网桥设备用作其他设备的端口，请将 **Method** 字段设置为 **Disabled**。
 - 要使用 DHCP，请将 **Method** 字段保留为默认值 **Automatic (DHCP)**。
 - 要使用静态 IP 设置，请将 **Method** 字段设置为 **Manual**，并相应地填写字段：



7. 点 **Save**。
8. 关闭 **nm-connection-editor**。

验证

- 使用 **ip** 工具来显示作为特定网桥端口的以太网设备的链接状态。

```
# 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
```

- 使用 **bridge** 工具来显示作为任意网桥设备中端口的以太网设备状态：

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
...
```

要显示特定以太网设备的状态，请使用 **bridge link show dev *ethernet_device_name*** 命令。

其他资源

- [使用 nm-connection-editor 配置网络绑定](#)
- [使用 nm-connection-editor 配置网络团队](#)
- [使用 nm-connection-editor 配置 VLAN 标记](#)
- [配置 NetworkManager 以避免使用特定配置集提供默认网关](#)
- [如何配置具有 VLAN 信息的网桥？](#)

6.5. 使用 NMSTATECTL 配置网络桥接

使用 **nmstatectl** 工具通过 Nmstate API 配置网桥。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，**nmstatectl** 会自动回滚更改以避免系统处于不正确的状态。

根据您的环境，相应地调整 YAML 文件。例如，要在网桥中使用与以太网适配器不同的设备，请调整您在网桥中使用的端口的 **base-iface** 属性和 **type** 属性。

先决条件

- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。
- 要将以太网设备用作网桥中的端口，必须在服务器中安装物理或者虚拟以太网设备。
- 要使用团队、绑定或 VLAN 设备作为网桥中的端口，请在 **port** 列表中设置接口名称，并定义对应的接口。
- **nmstate** 软件包已安装。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 **~/create-bridge.yml** :

```
---
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
```

这些设置使用以下设置定义一个网桥：

- 网桥中的网络接口：**enp1s0** 和 **enp7s0**
- Spanning Tree Protocol (STP): 启用
- 静态 IPv4 地址：**192.0.2.1**，子网掩码为 **/24**
- 静态 IPv6 地址：**2001:db8:1::1**，子网掩码为 **/64**
- IPv4 默认网关：**192.0.2.254**
- IPv6 默认网关：**2001:db8:1::fffe**
- IPv4 DNS 服务器：**192.0.2.200**

- IPv6 DNS 服务器 : **2001:db8:1::ffbb**
- DNS 搜索域 : **example.com**

2. 将设置应用到系统 :

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

验证

1. 显示设备和连接的状态 :

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

2. 显示连接配置集的所有设置 :

```
# 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. 以 YAML 格式显示连接设置 :

```
# nmstatectl show bridge0
```

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)
- [如何配置具有 VLAN 信息的网桥？](#)

6.6. 使用 NETWORK RHEL 系统角色配置网桥

您可以使用 **network** RHEL 系统角色远程配置网桥。

先决条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 `playbook` 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 在服务器中安装两个或者两个以上物理或者虚拟网络设备。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml` :

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bridge that uses two Ethernet ports
      ansible.builtin.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

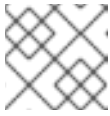
          # Add an Ethernet profile to the bridge
          - name: bridge0-port1
            interface_name: enp7s0
            type: ethernet
            controller: bridge0
            port_type: bridge
            state: up

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

这些设置使用以下设置定义一个网桥：

- 静态 IPv4 地址 - **192.0.2.1** 和 **/24** 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**

- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- 网桥的端口 - **enp7s0** 和 **enp8s0**



注意

在网桥上设置 IP 配置,而不是在 Linux 网桥的端口上设置。

2. 验证 playbook 语法 :

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

请注意, 这个命令只验证语法, 不会防止错误, 但保护有效配置。

3. 运行 playbook :

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

其他资源

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

第 7 章 设置 IPSEC VPN

虚拟专用网络(VPN)是一种通过互联网连接到本地网络的方法。**Libreswan** 提供的 **IPsec** 是创建 VPN 的首选方法。**libreswan** 是 VPN 的用户空间 **IPsec** 实现。VPN 通过在中间网络（如互联网）设置一个隧道来启用 LAN 和另一个远程 LAN 之间的通信。为了安全起见，VPN 隧道总是使用认证和加密。对于加密操作，**Libreswan** 使用 **NSS** 库。

7.1. 使用 CONTROL-CENTER 配置 VPN 连接

如果使用带有图形界面的 Red Hat Enterprise Linux，您可以在 GNOME **control-center** 中配置 VPN 连接。

先决条件

- 已安装 **NetworkManager-libreswan-gnome** 软件包。

步骤

1. 按 **Super** 键，输入 **Settings**，然后按 **Enter** 键打开 **control-center** 应用程序。
2. 选择左侧的 **Network** 条目。
3. 点 + 图标。
4. 选择 **VPN**。
5. 选择 **Identity** 菜单项来查看基本配置选项：

General

Gateway - 远程 VPN 网关的名称或 IP 地址。

认证

类型

- **IKEv2(Certificate)**- 客户端通过证书进行身份验证。它更安全（默认）。
- **IKEv1(XAUTH)** - 客户端通过用户名和密码或预共享密钥(PSK)进行身份验证。
以下配置设置在 **高级** 部分中提供：

图 7.1. VPN 连接的高级选项

IPsec Advanced Options ×

Identification

Domain:

Security

Phase1 Algorithms:

Phase2 Algorithms:

Disable PFS

Phase1 Lifetime:

Phase2 Lifetime:

Disable rekeying

Connectivity

Remote Network:

narrowing

Enable fragmentation

Enable MOBIKE

Apply



警告

当使用 **gnome-control-center** 应用程序配置基于 IPsec 的 VPN 连接时，高级对话框会显示配置，但它不允许任何更改。因此，用户无法更改任何高级 IPsec 选项。使用 **nm-connection-editor** 或 **nmcli** 工具来配置高级属性。

身份识别

- **域** - 如果需要，输入域名。

安全性

- **Phase1 Algorithms** - 对应于 **ike** Libreswan 参数 - 输入用来验证和设置加密频道的算法。
- **Phase2 Algorithms** - 对应于 **esp** Libreswan 参数 - 输入用于 IPsec 协商的算法。
选择 **Disable PFS** 字段来关闭 Perfect Forward Secrecy(PFS)，以确保与不支持 PFS 的旧服务器兼容。
- **Phase1 Lifetime** - 对应于 **ikelifetime** Libreswan 参数 - 用于加密流量的密钥的有效期。
- **Phase2 Lifetime** - 对应于 **salifetime** Libreswan 参数 - 在过期前连接的特定实例应多久。
注意：为了安全起见，加密密钥应该不时地更改。
- **Remote network** - 对应于 **rightsubnet** Libreswan 参数 - 应该通过 VPN 访问的目标专用远程网络。
检查 **缩减** 字段以启用缩小字段。请注意，它只在 IKEv2 协商中有效。
- **Enable fragmentation** - 对应于 **fragmentation** Libreswan 参数 - 是否允许 IKE 分段。有效值为 **yes**（默认）或 **no**。
- **Enable Mobike** - 对应于 **mobike** Libreswan 参数 - 是否允许 Mobility and Multihoming Protocol (MOBIKE、RFC 4555) 启用连接来迁移其端点，而无需从头开始重启连接。这可用于在有线、无线或者移动数据连接之间进行切换的移动设备。值为 **no**（默认）或 **yes**。

6. 选择 IPv4 菜单条目：

IPv4 方法

- **Automatic (DHCP)** - 如果您要连接的网络使用 **DHCP** 服务器来分配动态 **IP** 地址，请选择此选项。
- **Link-Local Only** - 如果您要连接的网络没有 **DHCP** 服务器且您不想手动分配 **IP** 地址，请选择这个选项。随机地址将根据 [RFC 3927](#) 分配，带有前缀 **169.254/16**。
- **手动** - 如果您要手动分配 **IP** 地址，请选择这个选项。
- **Disable** - 在这个连接中禁用 **IPv4**。

DNS

在 **DNS** 部分，当 **Automatic** 为 **ON** 时，将其切换到 **OFF** 以输入您要用逗号分开的 **DNS** 服务器的 **IP** 地址。

Routes

请注意，在 **Routes** 部分，当 **Automatic** 为 **ON** 时，会使用 DHCP 的路由，但您也可以添加额外的静态路由。当 **OFF** 时，只使用静态路由。

- **Address** - 输入远程网络或主机的 IP 地址。
- **Netmask** - 以上输入的 IP 地址的子网掩码或前缀长度。
- **Gateway** - 上面输入的远程网络或主机的网关的 IP 地址。
- **Metric** - 网络成本，赋予此路由的首选值。数值越低，优先级越高。
仅将此连接用于其网络上的资源

选择这个复选框以防止连接成为默认路由。选择这个选项意味着只有特别用于路由的流量才会通过连接自动获得，或者手动输入到连接上。

7. 要在 VPN 连接中配置 IPv6 设置，请选择 **IPv6** 菜单条目：

IPv6 Method

- **Automatic** - 选择这个选项使用 **IPv6 Stateless Address AutoConfiguration (SLAAC)** 根据硬件地址和路由器公告 (RA) 创建自动的、无状态的配置。
- **Automatic, DHCP only** - 选择这个选项以不使用 RA，但从 **DHCPv6** 请求信息以创建有状态的配置。
- **Link-Local Only** - 如果您要连接的网络没有 **DHCP** 服务器且您不想手动分配 IP 地址，请选择这个选项。随机地址将根据 *RFC 4862* 分配，前缀为 **FE80::0**。
- **手动** - 如果您要手动分配 IP 地址，请选择这个选项。
- **Disable** - 在这个连接中禁用 **IPv6**。
请注意，**DNS, Routes, Use this connection only for resources on its network** 项是 **IPv4** 的常规设置。

8. 编辑完 VPN 连接后，点**添加按钮**自定义配置或**应用按钮**为现有配置保存它。

9. 将配置集切换为 **ON** 以激活 VPN 连接。

其他资源

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

7.2. 使用 NM-CONNECTION-EDITOR 配置 VPN 连接

如果使用带有图形界面的 Red Hat Enterprise Linux，您可以在 **nm-connection-editor** 应用程序中配置 VPN 连接。

先决条件

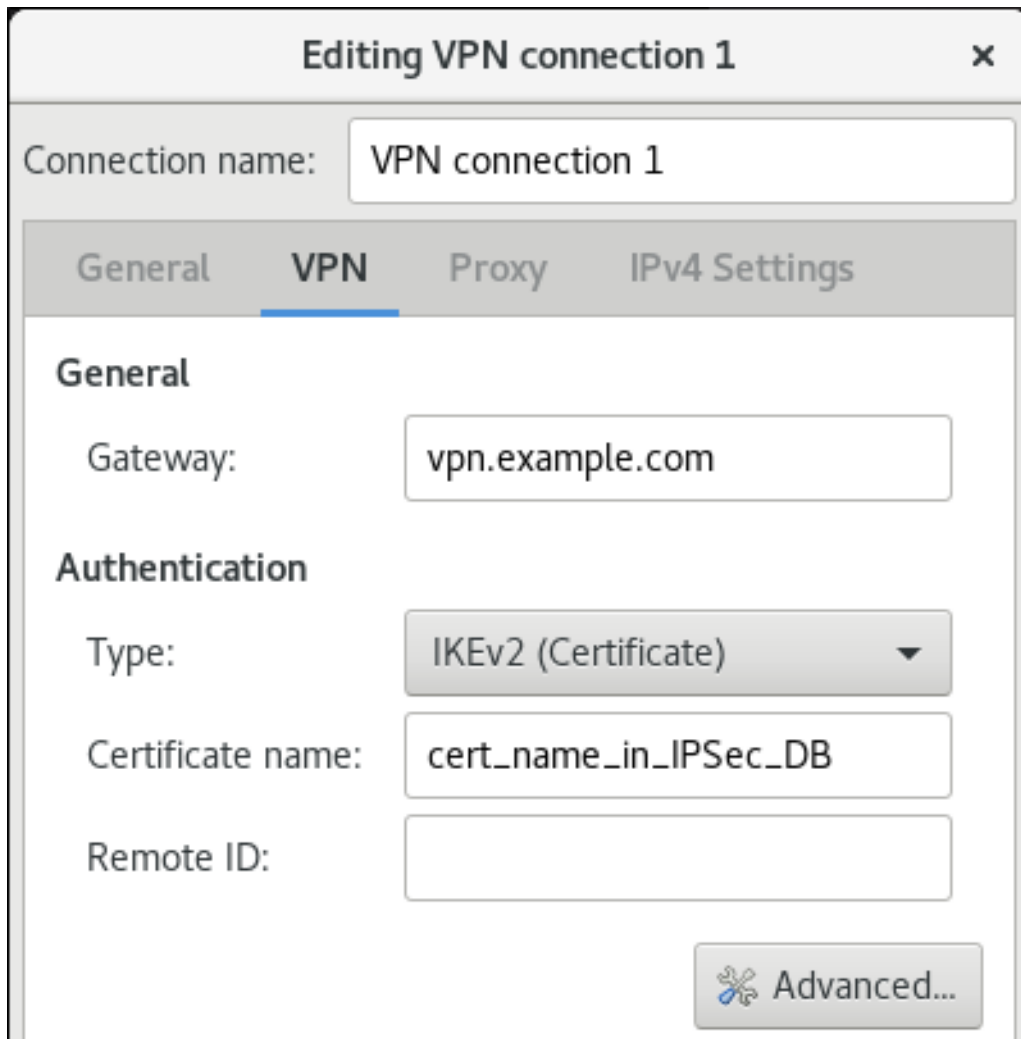
- 已安装 **NetworkManager-libreswan-gnome** 软件包。
- 如果您配置了互联网密钥交换版本 2 (IKEv2) 连接：
 - 证书导入到 IPsec 网络安全服务 (NSS) 数据库中。
 - NSS 数据库中的证书 **nickname** 是已知的。

流程

1. 打开终端窗口，输入：

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

2. 点击 **+** 按钮来添加一个新的连接。
3. 选择 **IPsec based VPN** 连接类型，然后点击 **Create**。
4. 在 **VPN** 选项卡中：
 - a. 在 **Gateway** 字段中输入 VPN 网关的主机名或 IP 地址，然后选择验证类型。根据验证类型，您必须输入不同的额外信息：
 - **IKEv2 (认证)** 使用证书验证客户端，这会更安全。这个设置需要在 IPsec NSS 数据库中指定证书的 nickname
 - **IKEv1(XAUTH)** 使用用户名和密码（预共享密钥）验证用户身份。此设置要求您输入以下值：
 - 用户名
 - 密码
 - 组名称
 - Secret
 - b. 如果远程服务器为 IKE 交换指定了本地标识符，在 **Remote ID** 字段中输入准确的字符串。在运行 Libreswan 的远程服务器中，这个值是在服务器的 **leftid** 参数中设置的。



c. (可选) 点击高级按钮配置附加设置。您可以配置以下设置：

- 身份识别
 - 域 - 如果需要，请输入域名。
- 安全性
 - **Phase1 Algorithms** 对应于 ike **Libreswan** 参数。输入用来验证和设置加密频道的算法。
 - **Phase2 Algorithms** 对应于 **esp** Libreswan 参数。输入用于 IPsec 协商的算法。选择 **Disable PFS** 字段来关闭 Perfect Forward Secrecy(PFS)，以确保与不支持 PFS 的旧服务器兼容。
 - **Phase1 Lifetime** 与 **ikelifetime** Libreswan 参数对应。此参数定义用于加密流量的密钥的有效期。
 - **Phase2 Lifetime** 与 **salifetime** Libreswan 参数对应。这个参数定义安全关联有效期。
- 连接性
 - 远程网络 与 **rightsubnet** Libreswan 参数对应，并定义应通过 VPN 访问的目标专用远程网络。
检查 **缩减** 字段以启用缩小字段。请注意，它只在 IKEv2 协商中有效。

- **Enable fragmentation** 与 **segmentation** Libreswan 参数对应，并定义是否允许 IKE 分段。有效值为 **yes**（默认）或 **no**。
 - **Enable Mobike** 与 **mobike** 参数对应。参数定义是否允许移动和多功能协议（MOBIKE）（RFC 4555）启用连接来迁移其端点而无需从头开始重启连接。这可用于在有线、无线或者移动数据连接之间进行切换的移动设备。值为 **no**（默认）或 **yes**。
5. 在 **IPv4 Settings** 选项卡中，选择 IP 分配方法，并可选择设置额外的静态地址、DNS 服务器、搜索域和路由。

The screenshot shows a window titled "Editing VPN connection 1" with a close button (X) in the top right. The "Connection name" field contains "VPN connection 1". Below this is a tabbed interface with four tabs: "General", "VPN", "Proxy", and "IPv4 Settings", with "IPv4 Settings" being the active tab. Under "Method", a dropdown menu shows "Automatic (VPN)". Below that is a section titled "Additional static addresses" containing a table with three columns: "Address", "Netmask", and "Gateway". To the right of the table are "Add" and "Delete" buttons. Below the table are two input fields: "Additional DNS servers:" and "Additional search domains:". At the bottom right is a "Routes..." button.

6. 保存连接。
7. 关闭 `nm-connection-editor`。



注意

当您点 **+** 按钮添加新连接时，`NetworkManager` 会为那个连接创建新配置文件，然后打开同一个对话框来编辑现有连接。这两个对话框之间的区别在于现有连接配置集有详情菜单条目。

其他资源

- [nm-settings-libreswan\(5\) 手册页](#)

7.3. 配置自动检测和使用 ESP 硬件卸载来加速 IPSEC 连接

卸载硬件的封装安全负载(ESP)来加速以太网上的 IPsec 连接。默认情况下，`Libreswan` 会检测硬件是否支持这个功能，并因此启用 ESP 硬件卸载。如果这个功能被禁用了或明确启用了，您可以切回到自动检测。

前提条件

- 网卡支持 ESP 硬件卸载。
- 网络驱动程序支持 ESP 硬件卸载。
- IPsec 连接已配置且可以正常工作。

流程

1. 编辑应使用 ESP 硬件卸载支持的自动检测连接的 `/etc/ipsec.d/` 目录中的 Libreswan 配置文件。
2. 确保 `nic-offload` 参数没有在连接的设置中设置。
3. 如果您删除了 `nic-offload`，请重启 `ipsec` 服务：

```
# systemctl restart ipsec
```

验证

如果网卡支持 ESP 硬件卸载支持，请按照以下步骤验证结果：

1. 显示 IPsec 连接使用的以太网设备的 `tx_ipsec` 和 `rx_ipsec` 计数器：

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

2. 通过 IPsec 隧道发送流量。例如，ping 远程 IP 地址：

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

3. 再次显示以太网设备的 `tx_ipsec` 和 `rx_ipsec` 计数器：

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

如果计数器值增加了，ESP 硬件卸载正常工作。

其他资源

- [使用 IPsec 配置 VPN](#)

7.4. 在绑定中配置 ESP 硬件卸载以加快 IPSEC 连接

将封装安全负载(ESP)卸载到硬件可加速 IPsec 连接。如果出于故障转移原因而使用网络绑定，配置 ESP 硬件卸载的要求和流程与使用常规以太网设备的要求和流程不同。例如，在这种情况下，您可以对绑定启用卸载支持，内核会将设置应用到绑定的端口。

先决条件

- 绑定中的所有网卡都支持 ESP 硬件卸载。
- 网络驱动程序支持对绑定设备的 ESP 硬件卸载。在 RHEL 中，只有 `ixgbe` 驱动程序支持此功能。

- 绑定已配置且可以正常工作。
- 该绑定使用 **active-backup** 模式。绑定驱动程序不支持此功能的任何其他模式。
- IPsec 连接已配置且可以正常工作。

步骤

1. 对网络绑定启用 ESP 硬件卸载支持：

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

这个命令在对 **bond0** 连接启用 ESP 硬件卸载支持。

2. 重新激活 **bond0** 连接：

```
# nmcli connection up bond0
```

3. 编辑应使用 ESP 硬件卸载的连接的 **/etc/ipsec.d/** 目录中的 Libreswan 配置文件，并将 **nic-offload=yes** 语句附加到连接条目：

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

4. 重启 **ipsec** 服务：

```
# systemctl restart ipsec
```

验证

1. 显示绑定的活动端口：

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

2. 显示活动端口的 **tx_ipsec** 和 **rx_ipsec** 计数器：

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

3. 通过 IPsec 隧道发送流量。例如，ping 远程 IP 地址：

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

4. 再次显示活动端口的 **tx_ipsec** 和 **rx_ipsec** 计数器：

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

如果计数器值增加了，ESP 硬件卸载正常工作。

其他资源

- [配置网络绑定](#)
- 安全网络文档中的[使用 IPsec 配置 VPN](#) 部分

7.5. 使用 NMSTATECTL 配置基于 IPSEC 的 VPN 连接

IPsec（互联网协议安全）是一个安全协议套件，由 **Libreswan** 提供，用于实现 VPN。IPsec 包括在连接建立时启动身份验证的协议，以及在数据传输过程中管理密钥。当应用程序在网络中部署并使用 IP 协议进行通信时，IPsec 可以保护数据通信。

要管理基于 IPsec 的配置来验证 VPN 连接，您可以使用 **nmstatectl** 工具。此实用程序提供对主机网络管理的声明性 API 的命令行访问。以下是 **host-to-subnet** 和 **主机到主机通信模式** 的身份验证类型：

- host-to-subnet PKI 身份验证
- 主机到子网 RSA 身份验证
- 主机到子网 PSK 验证
- 主机到主机的隧道模式身份验证
- 主机到主机的传输模式身份验证

7.5.1. 使用 nmstatectl 配置带有 PKI 身份验证和隧道模式的主机到子网 IPsec VPN

如果要根据 IPsec 中的可信实体身份验证使用加密，公钥基础架构(PKI)通过使用两个主机之间的加密密钥来提供安全通信。两者都通过与可信实体证书颁发机构(CA)共享公钥来生成每个主机维护私钥的私钥和公钥。验证真实性后，CA 会生成数字证书。如果是加密和解密，主机将使用私钥进行加密和公钥解密。

通过使用 Nmstate（用于网络管理的声明性 API），您可以配置基于 PKI 身份验证的 IPsec 连接。设置配置后，Nmstate API 可确保结果与配置文件匹配。如果有任何失败，**nmstate** 会自动回滚更改以避免系统的状态不正确。

要在 **主机到子网** 配置中建立加密通信，远程 IPsec 端通过使用参数 **dhcp: true** 为主机提供另一个 IP。在 nmstate 中定义 **IPsec** 的系统时，**left-named system** 是本地主机，而 **右-named system** 是远程主机。以下流程需要在两个主机上运行。

先决条件

- 通过使用密码，您生成了一个存储证书和密钥的 PKCSzFCP 文件。

步骤

1. 安装所需的软件包：

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. 重启 NetworkManager 服务：

```
# systemctl restart NetworkManager
```

3. 因为 **Libreswan** 已安装，请删除其旧的数据库文件并重新创建它们：

```
# systemctl stop ipsec
# rm /etc/ipsec.d/*db
# ipsec initnss
```

4. 启用并启动 **ipsec** 服务：

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

5. 导入 PKCScriu 文件：

```
# ipsec import node-example.p12
```

导入 PKCSbusybox 文件时，请输入用于创建该文件的密码。

6. 创建包含以下内容的 YAML 文件，如 `~/create-pki-authentication.yml`：

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: 'yes'        2
    left: '192.0.2.250'          3
    leftid: '%fromcert'         4
    leftcert: 'local-host.example.com' 5
    right: '192.0.2.150'        6
    rightid: '%fromcert'        7
    ikev2: 'insist'             8
    ikelifetime: '24h'          9
    salifetime: '24h'          10
```

YAML 文件定义以下设置：

- 1 IPsec 连接名称
- 2 值 **yes** 表示 **libreswan** 创建一个 IPsec **xfrm** 虚拟接口 **ipsec<number>**，并自动找到下一个可用数字
- 3 本地主机的公共网络接口的静态 IPv4 地址
- 4 在本地主机中，**%fromcert** 的值将 ID 设置为从加载的证书获取的可辨识名称(DN)
- 5 本地主机公钥的可辨识名称(DN)
- 6 远程主机的公共网络接口的静态 IPv4 地址
- 7 在远程主机上，**%fromcert** 的值将 ID 设置为从加载的证书获取的可辨识名称(DN)。
- 8 **insist** 值只接受并接收互联网密钥交换(IKEv2)协议
- 9 IKE 协议的持续时间

10 IPsec 安全关联(SA)的持续时间

7. 将设置应用到系统：

```
# nmstatectl apply ~/create-pki-authentication.yml
```

验证

1. 验证 IPsec 状态：

```
# ip xfrm status
```

2. 验证 IPsec 策略：

```
# ip xfrm policy
```

其他资源

- [ipsec.conf\(5\) man page](#)

7.5.2. 使用 nmstatectl 配置带有 RSA 身份验证和隧道模式的主机到子网 IPsec VPN

如果要在 IPsec 中使用基于非对称加密密钥身份验证，则 RSA 算法通过使用私钥和公钥在两个主机之间加密和解密提供安全通信。此方法使用私钥进行加密，以及用于解密的公钥。

通过使用 Nmstate（用于网络管理的声明性 API），您可以配置基于 RSA 的 IPsec 身份验证。设置配置后，Nmstate API 可确保结果与配置文件匹配。如果有任何失败，**nmstate** 会自动回滚更改以避免系统的状态不正确。

要在 **主机到子网** 配置中建立加密通信，远程 IPsec 端通过使用参数 **dhcp: true** 为主机提供另一个 IP。在 nmstate 中定义 **IPsec** 的系统时，**left-named system** 是本地主机，而 **右-named system** 是远程主机。以下流程需要在两个主机上运行。

步骤

1. 安装所需的软件包：

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. 重启 NetworkManager 服务：

```
# systemctl restart NetworkManager
```

3. 如果已安装 **Libreswan**，请删除其旧的数据库文件并重新创建它们：

```
# systemctl stop ipsec
# rm /etc/ipsec.d/*db
# ipsec initnss
```

4. 在每个主机上生成 RSA 密钥对：

```
# ipsec newhostkey --output
```

- 5. 显示公钥：

```
# ipsec showhostkey --list
```

- 6. 上一步返回生成的密钥 **ckaid**。在左侧使用以下命令使用 **ckaid**，例如：

```
# ipsec showhostkey --left --ckaid <0sAwEAAesFfVZqFzRA9F>
```

- 7. 上一命令的输出生成了配置所需的 **leftrsasigkey=** 行。在第二个主机（右）上执行相同的操作：

```
# ipsec showhostkey --right --ckaid <0sAwEAAesFfVZqFzRA9E>
```

- 8. 在引导时启用 **ipsec** 服务：

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

- 9. 创建包含以下内容的 YAML 文件，如 **~/create-rsa-authentication.yml**：

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec                       2
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: '99'          3
    leftrsasigkey: '0sAwEAAesFfVZqFzRA9F'  4
    left: '192.0.2.250'           5
    leftid: 'local-host-rsa.example.com'  6
    right: '192.0.2.150'          7
    rightrsasigkey: '0sAwEAAesFfVZqFzRA9E'  8
    rightid: 'remote-host-rsa.example.com'  9
    ikev2: 'insist'               10
```

YAML 文件定义以下设置：

- 1 IPsec 连接名称
- 2 接口名称
- 3 值 **99** 表示 **libreswan** 创建一个 IPsec **xfrm** 虚拟接口 **ipsec<number>**，并自动找到下一个可用数字
- 4 本地主机的 RSA 公钥
- 5 本地主机的公共网络接口的静态 IPv4 地址
- 6 本地主机的可辨识名称(DN)
- 7 远程主机的 RSA 公钥

- 8 远程主机的公共网络接口的静态 IPv4 地址
- 9 远程主机的可辨识名称(DN)
- 10 **insist** 值只接受并接收互联网密钥交换(IKEv2)协议

10. 将设置应用到系统：

```
# nmstatectl apply ~/create-rsa-authentication.yml
```

验证

1. 显示网络接口的 IP 设置：

```
# ip addr show example_ipsec_conn1
```

2. 验证 IPsec 状态：

```
# ip xfrm status
```

3. 验证 IPsec 策略：

```
# ip xfrm policy
```

其他资源

- [ipsec.conf\(5\) man page](#)

7.5.3. 使用 nmstatectl 配置带有 PSK 身份验证和隧道模式的主机到子网 IPsec VPN

如果要在 IPsec 中使用基于 mutual 身份验证的加密，Pre-Shared Key (PSK)方法通过使用两个主机之间的 secret 密钥提供安全通信。文件存储机密密钥和相同的密钥加密通过隧道流的数据流。

通过使用 Nmstate（用于网络管理的声明性 API），您可以配置基于 PSK 的 IPsec 身份验证。设置配置后，Nmstate API 可确保结果与配置文件匹配。如果有任何失败，**nmstate** 会自动回滚更改以避免系统不正确状态。

要在 **主机到子网**配置中建立加密通信，远程 IPsec 端通过使用参数 **dhcp: true** 为主机提供另一个 IP。在 nmstate 中定义 **IPsec** 的系统时，**left-named system** 是本地主机，而 **右-named system** 是远程主机。以下流程需要在两个主机上运行。



注意

由于此方法使用静态字符串进行身份验证和加密，因此仅用于测试/开发目的。

流程

1. 安装所需的软件包：

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. 重启 NetworkManager 服务：

```
# systemctl restart NetworkManager
```

3. 如果已安装 **Libreswan**，请删除其旧的数据库文件并重新创建它们：

```
# systemctl stop ipsec
# rm /etc/ipsec.d/*db
# ipsec initnss
```

4. 在引导时启用 **ipsec** 服务：

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

5. 创建包含以下内容的 YAML 文件，如 `~/create-pks-authentication.yml`：

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  ipv4:
    enabled: true
    dhcp: true
  libreswan:
    ipsec-interface: 'no'          2
    right: '192.0.2.250'           3
    rightid: 'remote-host.example.org' 4
    left: '192.0.2.150'           5
    leftid: 'local-host.example.org' 6
    psk: "example_password"
    ikev2: 'insist'                7
```

YAML 文件定义以下设置：

- 1 IPsec 连接名称
- 2 设置 **no** value 表示 **libreswan** 只创建 **xfrm** 策略，而不是虚拟 **xfrm** 接口
- 3 远程主机的公共网络接口的静态 IPv4 地址
- 4 远程主机的可辨识名称(DN)
- 5 本地主机的公共网络接口的静态 IPv4 地址
- 6 本地主机的可辨识名称(DN)
- 7 **insist** 值只接受并接受互联网密钥交换(IKEv2)协议

6. 将设置应用到系统：

```
# nmstatectl apply ~/create-pks-authentication.yml
```

验证

1. 显示网络接口的 IP 设置：

```
# ip addr show example_ipsec_conn1
```

2. 验证 IPsec 状态：

```
# ip xfrm status
```

3. 验证 IPsec 策略：

```
# ip xfrm policy
```

7.5.4. 使用 nmstatectl 配置主机到主机的 IPsec VPN，使用 PKI 身份验证和隧道模式

IPsec（互联网协议安全）是一个安全协议套件，用于验证和加密网络和设备中的 IP 通信。**Libreswan** 软件为 VPN 提供 IPsec 实现。

在隧道模式中，通信的源和目标 IP 地址在 IPsec 隧道中加密。外部网络嗅探器只能获取左侧 IP 和右 IP。通常，对于隧道模式，它支持 **host-to-host**、**host-to-subnet** 和 **subnet-to-subnet**。在此模式中，新的 IP 数据包封装现有的数据包及其有效负载和标头。这个模式的封装通过不安全的网络保护 IP 数据、源和目标标头。此模式可用于在子网到子网、远程访问连接和不受信任的网络（如开放公共 Wi-Fi 网络）中传输数据。默认情况下，IPsec 在隧道模式的两个站点之间建立安全频道。使用以下配置，您可以建立 VPN 连接作为 **主机到主机** 架构。

通过使用 Nmstate（用于网络管理的声明性 API），您可以配置 IPsec VPN 连接。设置配置后，Nmstate API 可确保结果与配置文件匹配。如果有任何失败，**nmstate** 会自动回滚更改以避免系统不正确状态。

在 **主机到主机** 配置中，您需要设置 **leftmodecfgclient: no**，以便它无法从服务器接收网络配置，因此该值为 **no**。在 nmstate 中定义 IPsec 的系统时，**left**-named system 是本地主机，而 **右**-named system 是远程主机。以下流程需要在两个主机上运行。

先决条件

- 通过使用密码，您生成了一个存储证书和密钥的 PKCSzFCP 文件。

流程

1. 安装所需的软件包：

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. 重启 NetworkManager 服务：

```
# systemctl restart NetworkManager
```

3. 因为 **Libreswan** 已安装，请删除其旧的数据库文件并重新创建它们：

```
# systemctl stop ipsec
# rm /etc/ipsec.d/*db
# ipsec initnss
```

4. 导入 PKCScriu 文件：

```
# ipsec import node-example.p12
```

在导入 PKCScriu 文件时，输入用于生成该文件的密码。

5. 启用并启动 **ipsec** 服务：

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

6. 创建包含以下内容的 YAML 文件，如 `~/create-p2p-vpn-authentication.yml`：

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  libreswan:
    left: '192.0.2.250'            2
    leftid: 'local-host.example.com' 3
    leftcert: 'local-host.example.com' 4
    leftmodecfgclient: 'no'        5
    right: '192.0.2.150'          6
    rightid: 'remote-host.example.com' 7
    rightsubnet: '192.0.2.150/32' 8
    ikev2: 'insist'               9
```

YAML 文件定义以下设置：

- 1 IPsec 连接名称
- 2 本地主机的公共网络接口的静态 IPv4 地址
- 3 本地主机的可分辨名称(DN)
- 4 在本地主机上安装的证书名称
- 5 不是从远程主机检索客户端配置的值
- 6 远程主机的公共网络接口的静态 IPv4 地址
- 7 远程主机的可区分名称(DN)
- 8 远程主机的子网范围 - **192.0.2.150**，具有 **32** IPv4 地址
- 9 只接受和接收互联网密钥交换(IKEv2)协议的值

7. 将设置应用到系统：

```
# nmstatectl apply ~/create-p2p-vpn-authentication.yml
```

验证

1. 显示创建的 P2P 策略：

```
# ip xfrm policy
```

-
- 2. 验证 IPsec 状态：

```
# ip xfrm status
```

其他资源

- [ipsec.conf\(5\) man page](#)

7.5.5. 使用 nmstatectl 使用 PSK 验证和传输模式配置主机到主机的 IPsec VPN

IPsec（互联网协议安全）是一个安全协议套件，用于验证和加密网络和设备中的 IP 通信。**Libreswan** 工具为 VPN 提供基于 IPsec 的实现。

在传输模式中，加密仅适用于 IP 数据包的有效负载。另外，新的 IPsec 标头也会附加到 IP 数据包中，方法是保留原始 IP 标头。传输模式不会加密通信的源和目标 IP，而是将它们复制到外部 IP 标头中。因此，加密只保护网络中的 IP 数据。此模式可用于在网络的 **主机到主机连接中** 传输数据。此模式通常与 GRE 隧道一起使用，以节省 20 字节(IP 标头)的开销。默认情况下，**IPsec** 实用程序使用隧道模式。要使用传输模式，请为 **主机到主机** 连接数据传输设置 **type: transport**。

通过使用 Nmstate（用于网络管理的声明性 API），您可以配置 IPsec VPN 连接。设置配置后，Nmstate API 可确保结果与配置文件匹配。如果有任何失败，**nmstate** 会自动回滚更改以避免系统不正确状态。要覆盖默认的 **隧道** 模式，请指定 **传输模式**。

在 nmstate 中定义 **IPsec** 的系统时，**left-named system** 是本地主机，而 **右-named system** 是远程主机。以下流程需要在两个主机上运行。

先决条件

- 通过使用密码，您生成了一个存储证书和密钥的 PKCSzFCP 文件。

步骤

1. 安装所需的软件包：

```
# dnf install nmstate libreswan NetworkManager-libreswan
```

2. 重启 NetworkManager 服务：

```
# systemctl restart NetworkManager
```

3. 因为 **Libreswan** 已安装，请删除其旧的数据库文件并重新创建它们：

```
# systemctl stop ipsec
# rm /etc/ipsec.d/*db
# ipsec initnss
```

4. 导入 PKCScriu 文件：

```
# ipsec import node-example.p12
```

导入 PKCSbusybox 文件时，请输入用于创建该文件的密码。

5. 启用并启动 **ipsec** 服务：

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

6. 创建包含以下内容的 YAML 文件，如 `~/create-p2p-transport-authentication.yml`：

```
---
interfaces:
- name: 'example_ipsec_conn1'      1
  type: ipsec
  libreswan:
    type: 'transport'              2
    ipsec-interface: '99'          3
    left: '192.0.2.250'            4
    leftid: '%fromcert'            5
    leftcert: 'local-host.example.org' 6
    right: '192.0.2.150'           7
    prefix-length: '32'            8
    rightid: '%fromcert'           9
    ikev2: 'insist'                10
    ikelifetime: '24h'             11
    salifetime: '24h'              12
```

YAML 文件定义以下设置：

- 1 IPsec 连接名称
- 2 IPsec 模式
- 3 值 **99** 表示 **libreswan** 创建一个 IPsec **xfrm** 虚拟接口 `ipsec<number>`，并自动找到一个可用数字
- 4 本地主机的公共网络接口的静态 IPv4 地址
- 5 在本地主机中，**%fromcert** 的值设置为从加载的证书获取的可辨识名称(DN)
- 6 本地主机公钥的可辨识名称(DN)
- 7 远程主机的公共网络接口的静态 IPv4 地址
- 8 本地主机静态 IPv4 地址的子网掩码
- 9 在远程主机上，**%fromcert** 的值将 ID 设置为从加载的证书获取的可辨识名称(DN)
- 10 只接受和接收互联网密钥交换(IKEv2)协议的值
- 11 IKE 协议的持续时间
- 12 IPsec 安全关联(SA)的持续时间

7. 将设置应用到系统：

```
# nmstatectl apply ~/create-p2p-transport-authentication.yml
```

验证

1. 验证 IPsec 状态：

```
# ip xfrm status
```

2. 验证 IPsec 策略：

```
# ip xfrm policy
```

其他资源

- [ipsec.conf\(5\)](#) man page

第 8 章 设置 WIREGUARD VPN

WireGuard 是一个在 Linux 内核中运行的高性能 VPN 解决方案。它使用现代加密机制，并且比许多其他 VPN 解决方案更容易配置。另外，WireGuard 的小代码库降低了安全攻击的攻击面，因此提高安全性。对于身份验证和加密，WireGuard 使用类似于 SSH 的键。



重要

WireGuard 只作为技术预览提供。红帽产品服务级别协议 (SLA) 不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些预览可让用户早期访问将来的产品功能，让用户在开发过程中测试并提供反馈意见。

如需有关 [技术预览功能支持范围](#) 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

请注意，参与 WireGuard VPN 的所有主机都是同级的。本文档使用术语 **客户端** 来描述建立连接的主机，使用 **服务器** 来描述固定主机名或客户端连接的 IP 地址的主机，并可选通过这个服务器路由所有流量。

要设置 WireGuard VPN，您必须完成以下步骤。您可以使用不同的选项执行大多数步骤：

1. 为 VPN 中的每个主机创建公钥和私钥。
2. 使用 [nmcli](#)、[nmtui](#)、[RHEL web 控制台](#)、[nm-connection-editor](#) 或 [wg-quick](#) 服务来配置 WireGuard 服务器。
3. 使用 [命令行](#)、[RHEL web 控制台](#) 或 [图形界面](#) 在 WireGuard 服务器上配置 `firewalld`。
4. 使用 [nmcli](#)、[nmtui](#)、[RHEL web 控制台](#)、[nm-connection-editor](#) 或 [wg-quick](#) 服务来配置 WireGuard 客户端。

WireGuard 在网络层（层 3）上运行。因此，您无法使用 DHCP，且必须为服务器和客户端上的隧道设备分配静态 IP 地址或 IPv6 本地链接地址。



重要

只有在禁用 RHEL 中的 Federal Information Processing Standard(FIPS)模式时，才能使用 WireGuard。

8.1. WIREGUARD 使用的协议和原语

WireGuard 使用以下协议和原语：

- ChaCha20 用于通过 Poly1305 进行身份验证，使用带有关联数据(AEAD)的 Authenticated Encryption，如 [RFC7539](#) 所述
- Curve25519 用于 Elliptic-curve Diffie-Hellman(ECDH)密钥交换
- 用于哈希和密钥哈希的 BLAKE2s，如 [RFC7693](#) 所述
- 用于哈希表键的 SipHash24
- 用于密钥派生的 HKDF，如 [RFC5869](#) 所述

8.2. WIREGUARD 如何使用隧道 IP 地址、公钥和远程端点

当 WireGuard 将网络数据包发送到对等点时：

1. WireGuard 从数据包读取目标 IP，并将其与本地配置中允许的 IP 地址列表进行比较。如果未找到 peer，WireGuard 会丢弃数据包。
2. 如果 peer 有效，WireGuard 使用对等的公钥对数据包进行加密。
3. 发送主机查找主机的最新互联网 IP 地址，并将加密数据包发送到此地址。

当 WireGuard 接收数据包时：

1. WireGuard 使用远程主机的私钥解密数据包。
2. WireGuard 从数据包读取内部源地址，并在本地主机上对等点的设置中查询 IP 地址是否配置。如果源 IP 位于允许列表中，WireGuard 会接受数据包。如果 IP 地址不在列表中，WireGuard 会丢弃数据包。

公钥和允许的 IP 地址的关联称为 **加密密钥路由表**。这意味着，当发送数据包时，IP 地址列表的行为与路由表相似，在接收数据包时作为一种访问控制列表。

8.3. 使用 NAT 和防火墙后面的 WIREGUARD 客户端

WireGuard 使用 UDP 协议，只有在对等点发送数据包时才会传输数据。路由器上的有状态防火墙和网络地址转换(NAT)可跟踪连接，以启用 NAT 或防火墙接收数据包的对等点。

为了保持连接处于活动状态，WireGuard 支持 **持久性 keepalives**。这意味着您可以设置一个间隔，其中 WireGuard 发送 keepalive 数据包。默认情况下，禁用持久的 keep-alive 功能来减少网络流量。如果您在带有 NAT 的网络中使用客户端，或者防火墙在一定时间不活动状态后关闭连接，在客户端上启用此功能。



注意

请注意，您无法使用 RHEL web 控制台在 WireGuard 连接中配置 keepalive 数据包。要配置此功能，请使用 **nmcli** 工具编辑连接配置集。

8.4. 创建在 WIREGUARD 连接中使用的私钥和公钥

WireGuard 使用 base64 编码的私钥和公钥来互相验证主机。因此，您必须在参与 WireGuard VPN 的每个主机上创建密钥。



重要

对于安全连接，请为每个主机创建不同的密钥，并确保只使用远程 WireGuard 主机共享公钥。不要使用本文档中使用的示例键。

如果您计划使用 RHEL web 控制台创建 WireGuard VPN 连接，您可以在 web 控制台中生成公钥和私钥对。

步骤

1. 安装 **wireguard-tools** 软件包：

```
# dnf install wireguard-tools
```

2. 为主机创建私钥和对应的公钥：

```
# wg genkey | tee /etc/wireguard/$HOSTNAME.private.key | wg pubkey > /etc/wireguard/$HOSTNAME.public.key
```

您需要密钥文件的内容，而不是文件本身。但是，红帽建议在将来需要记住密钥时保留文件。

3. 在密钥文件中设置安全权限：

```
# chmod 600 /etc/wireguard/$HOSTNAME.private.key /etc/wireguard/$HOSTNAME.public.key
```

4. 显示私钥：

```
# cat /etc/wireguard/$HOSTNAME.private.key  
YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=
```

您需要私钥在本地主机上配置 WireGuard 连接。不要共享私钥。

5. 显示公钥：

```
# cat /etc/wireguard/$HOSTNAME.public.key  
UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
```

您需要公钥在远程主机上配置 WireGuard 连接。

其他资源

- **wg(8)** man page

8.5. 使用 NMCLI 配置 WIREGUARD 服务器

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 服务器。使用此方法让 NetworkManager 管理 WireGuard 连接。

此流程假设以下设置：

- server：
 - 私钥: **YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
 - 隧道 IPv4 地址：**192.0.2.1/24**
 - 频道 IPv6 地址：**2001:db8:1::1/32**
- Client:
 - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
 - 隧道 IPv4 地址：**192.0.2.2/24**
 - 频道 IPv6 地址：**2001:db8:1::2/32**

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 服务器的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 客户端的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 添加 NetworkManager WireGuard 连接配置集：

```
# nmcli connection add type wireguard con-name server-wg0 ifname wg0 autoconnect
no
```

此命令创建一个名为 **server-wg0** 的配置文件，并将虚拟接口 **wg0** 分配给它。要防止连接在添加后而没有最终配置的情况下自动启动，请禁用 **autoconnect** 参数。

2. 设置服务器的隧道 IPv4 地址和子网掩码：

```
# nmcli connection modify server-wg0 ipv4.method manual ipv4.addresses
192.0.2.1/24
```

3. 设置服务器的隧道 IPv6 地址和子网掩码：

```
# nmcli connection modify server-wg0 ipv6.method manual ipv6.addresses
2001:db8:1::1/32
```

4. 将服务器的私钥添加到连接配置集中：

```
# nmcli connection modify server-wg0 wireguard.private-key
"YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg="
```

5. 为传入的 WireGuard 连接设定端口：

```
# nmcli connection modify server-wg0 wireguard.listen-port 51820
```

在主机上始终设置固定端口号，接收传入的 WireGuard 连接。如果您没有设置端口，WireGuard 会在每次激活 **wg0** 接口时使用随机的空闲端口。

6. 为您要允许与此服务器通信的每个客户端添加对等配置。您必须手动添加这些设置，因为 **nmcli** 工具不支持设置相应的连接属性。
 - a. 编辑 `/etc/NetworkManager/system-connections/server-wg0.nmconnection` 文件，并追加：

```
[wireguard-peer.bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=]
allowed-ips=192.0.2.2;2001:db8:1::2;
```

- `[wireguard-peer.<public_key_of_the_client>]` 条目定义客户端对等的部分，以及包含客户端公钥的部分的名称。

- **allowed-ips** 参数设置允许向这个服务器发送数据的客户端的隧道 IP 地址。为每个客户端添加一个部分。

b. 重新载入 **server-wg0** 连接配置文件：

```
# nmcli connection load /etc/NetworkManager/system-connections/server-wg0.nmconnection
```

7. 可选：将连接配置为自动启动，请输入：

```
# nmcli connection modify server-wg0 autoconnect yes
```

8. 重新激活 **server-wg0** 连接：

```
# nmcli connection up server-wg0
```

后续步骤

- [在 WireGuard 服务器上配置 firewalld 服务。](#)

验证

1. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

2. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
  link/none
  inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
    valid_lft forever preferred_lft forever
  inet6 2001:db8:1::1/32 scope global noprefixroute
    valid_lft forever preferred_lft forever
  inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

其他资源

- **wg(8)** man page
- **nm-settings (5)** 手册页中的 **WireGuard** 设置 部分

8.6. 使用 NMTUI 配置 WIREGUARD 服务器

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 服务器。使用此方法让 NetworkManager 管理 WireGuard 连接。

此流程假设以下设置：

- server :
 - 私钥: **YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
 - 隧道 IPv4 地址 : **192.0.2.1/24**
 - 频道 IPv6 地址 : **2001:db8:1::1/32**
- Client:
 - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
 - 隧道 IPv4 地址 : **192.0.2.2/24**
 - 频道 IPv6 地址 : **2001:db8:1::2/32**

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 服务器的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 客户端的公钥
 - 服务器的静态隧道 IP 地址和子网掩码
- 已安装 **NetworkManager-tui** 软件包。

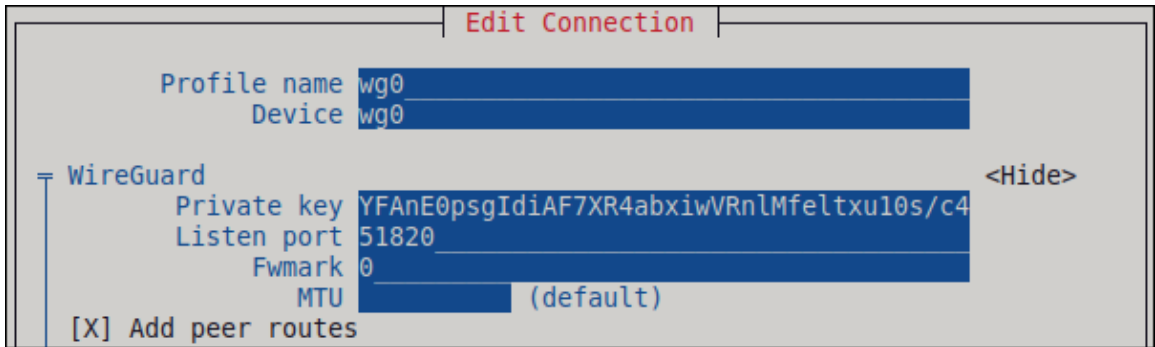
流程

1. 启动 **nmtui** 应用程序：

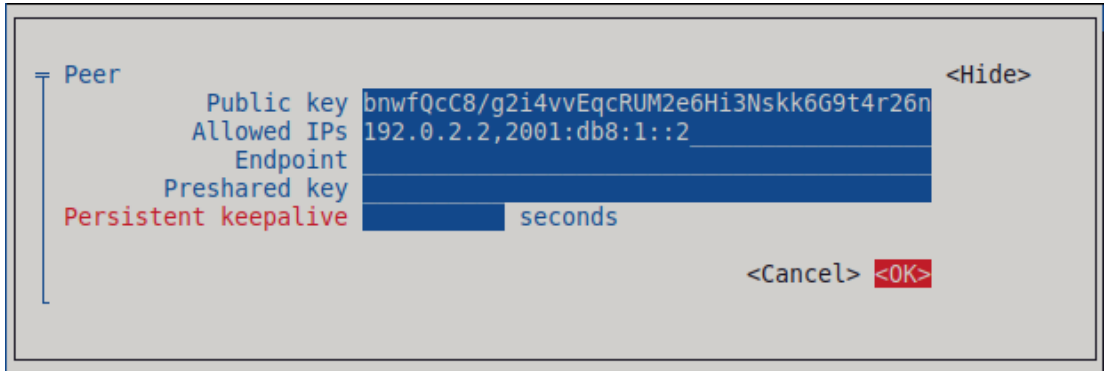
```
# nmtui
```

2. 选择 **Edit a connection**，点 **Enter**。
3. 选择 **添加**，然后按 **Enter** 键。
4. 选择列表中的 **WireGuard** 连接类型，然后按 **Enter** 键。
5. 在 **Edit connection** 窗口中：
 - a. 输入连接名称和虚拟接口，如 **wg0**，以便 NetworkManager 应分配给连接。
 - b. 输入服务器的私钥。

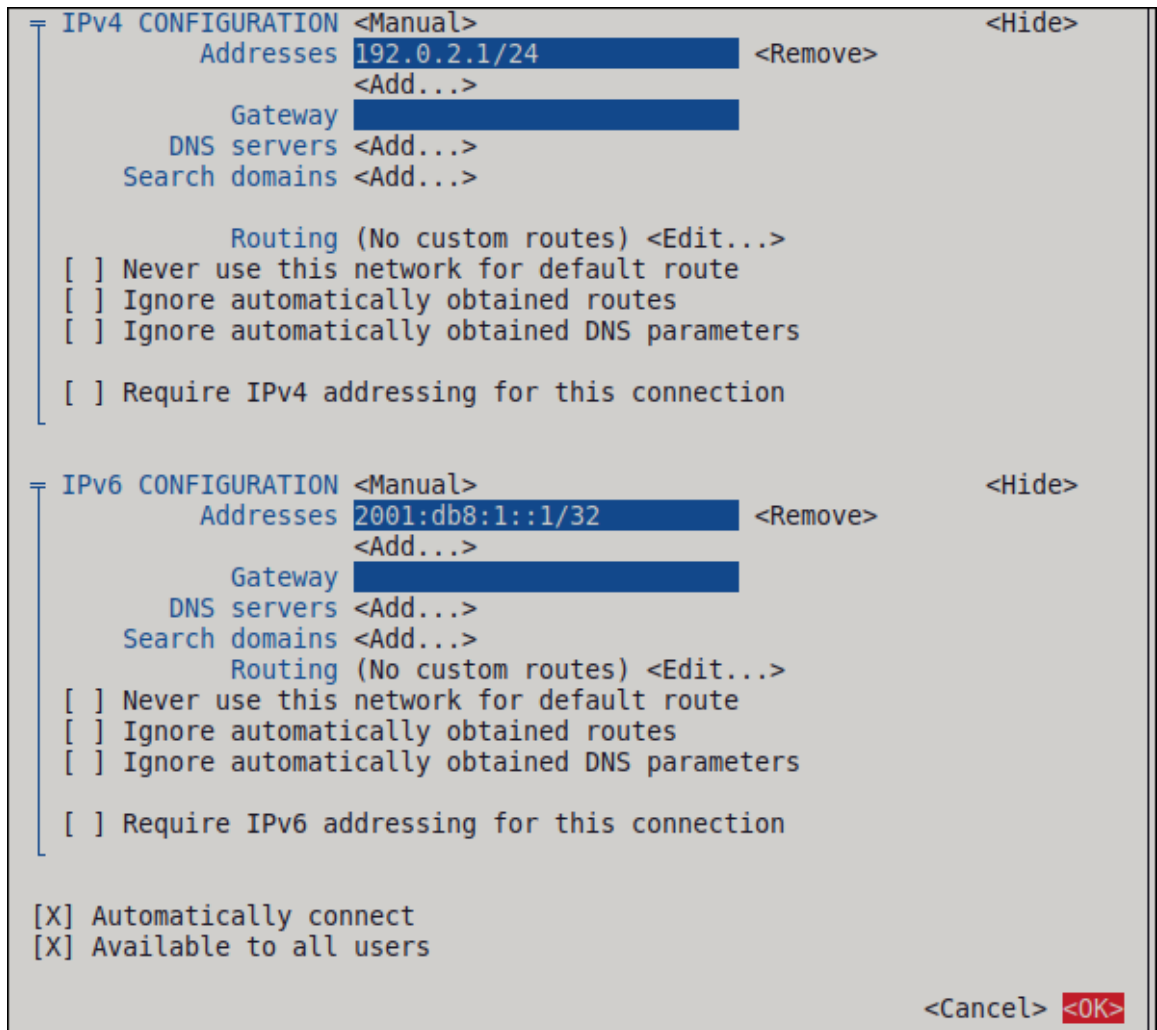
- c. 为传入的 WireGuard 连接设置侦听端口号，如 **51820**。
在主机上始终设置固定端口号，接收传入的 WireGuard 连接。如果您没有设置端口号，WireGuard 会在每次激活接口时都使用一个随机的空闲端口。



- d. 点 **Peers** 窗格旁的 **Add s**:
- i. 输入客户端的公钥。
 - ii. 将 **Allowed IP** 字段设置为允许向这个服务器发送数据的客户端的隧道 IP 地址。
 - iii. 选择 **确定**，然后按 **Enter** 键。



- e. 选择 **IPv4 Configuration** 旁边的 **显示**，然后按 **Enter** 键。
- i. 选择 IPv4 配置方法 **Manual**。
 - ii. 输入隧道 IPv4 地址和子网掩码。将 **Gateway** 字段留空。
- f. 选择 **IPv6 Configuration** 旁边的 **显示**，然后按 **Enter** 键。
- i. 选择 IPv6 配置方法 **Manual**。
 - ii. 输入隧道 IPv6 地址和子网掩码。将 **Gateway** 字段留空。
- g. 选择 **确定**，然后按 **Enter** 键



6. 在带有连接列表的窗口中，选择 **Back**，然后按 **Enter** 键。
7. 在 **NetworkManager TUI** 主窗口中，选择 **Quit**，然后按 **Enter** 键。

后续步骤

- 在 [WireGuard 服务器上配置 firewalld 服务](#)。

验证

1. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

2. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
```

```

20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

其他资源

- **wg(8)** man page

8.7. 使用 RHEL WEB 控制台配置 WIREGUARD 服务器

您可以使用基于浏览器的 RHEL web 控制台配置 WireGuard 服务器。使用此方法让 NetworkManager 管理 WireGuard 连接。

先决条件

- 已登陆到 RHEL web 控制台。
- 您知道以下信息：
 - 服务器和客户端的静态隧道 IP 地址和子网掩码
 - 客户端的公钥

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add VPN**。
3. 如果没有安装 **wireguard-tools** 和 **systemd-resolved** 软件包，Web 控制台会显示对应的通知。点 **Install** 安装这些软件包。
4. 输入您要创建的 WireGuard 设备名称。
5. 配置此主机的密钥对：
 - 如果要使用 web 控制台创建的密钥：
 - i. 在 **私钥** 区域中保留预先选择的 **Generated** 选项。
 - ii. 请注意 **公钥** 值。配置客户端时需要此信息。
 - 如果要使用现有的私钥：
 - i. 在 **私钥区域**中选择 **粘贴现有密钥**。
 - ii. 将私钥粘贴到文本字段中。Web 控制台自动计算对应的公钥。
6. 为传入的 WireGuard 连接设置侦听端口号，如 **51820**。

在主机上始终设置固定端口号，接收传入的 WireGuard 连接。如果您没有设置端口，WireGuard 会在每次激活接口时都使用一个随机的空闲端口。

7. 设置服务器的隧道 IPv4 地址和子网掩码。
要同时设置 IPv6 地址，您必须在创建后编辑连接。
8. 为您要允许与此服务器通信的每个客户端添加对等配置：
 - a. 单击 **Add peer**。
 - b. 输入客户端的公钥。
 - c. **Endpoint** 字段留空。
 - d. 将 **Allowed IP** 字段设置为允许向这个服务器发送数据的客户端的隧道 IP 地址。

Add WireGuard VPN ×

Name

Private key Generated Paste existing key

 📄

Listen port

IPv4 addresses
 Multiple addresses can be specified using commas or spaces as delimiters.

Peers Add peer

Public key	Endpoint	Allowed IPs
<input type="text" value="bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t ..."/>	<input type="text"/>	<input type="text" value="192.0.2.2"/> 🗑️

9. 点 **Add** 创建 WireGuard 连接。
10. 如果您还想设置隧道 IPv6 地址：
 - a. 在 **Interfaces** 部分点 WireGuard 连接的名称。
 - b. 点 **IPv6** 旁边的 **编辑**。
 - c. 将 **Addresses** 字段设置为 **Manual**，并输入服务器的隧道 IPv6 地址和前缀。
 - d. 点击 **Save**。

后续步骤

- 在 WireGuard 服务器上配置 firewalld 服务。

验证

1. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

2. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
  link/none
  inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
    valid_lft forever preferred_lft forever
  inet6 2001:db8:1::1/32 scope global noprefixroute
    valid_lft forever preferred_lft forever
  inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

8.8. 使用 NM-CONNECTION-EDITOR 配置 WIREGUARD 服务器

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 服务器。使用此方法让 NetworkManager 管理 WireGuard 连接。

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 服务器的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 客户端的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 打开终端窗口，输入：

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


2. 点 **+** 按钮添加新连接。
3. 选择 **WireGuard** 连接类型，然后点 **Create**。
4. 可选：更新连接名称。
5. 在 **General** 选项卡中，选择 **Connect automatically with priority**。另外，还可设置优先级值。
6. 在 **WireGuard** 选项卡中：
 - a. 输入虚拟接口的名称，如 **wg0**，这是 NetworkManager 应分配给连接的名称。
 - b. 输入服务器的私钥。
 - c. 为传入的 WireGuard 连接设置侦听端口号，如 **51820**。
在主机上始终设置固定端口号，接收传入的 WireGuard 连接。如果您没有设置端口，WireGuard 会在每次激活接口时都使用一个随机的空闲端口。
 - d. 点 **Add** 添加对等点：
 - i. 输入客户端的公钥。
 - ii. 将 **Allowed IP** 字段设置为允许向这个服务器发送数据的客户端的隧道 IP 地址。
 - iii. 点 **应用**。
7. 在 **IPv4 设置** 标签页中：
 - a. 在 **Method** 列表中选择 **Manual**。
 - b. 单击 **Add** 来输入隧道 IPv4 地址和子网掩码。将 **Gateway** 字段留空。
8. 在 **IPv6 设置** 标签页中：
 - a. 在 **Method** 列表中选择 **Manual**。
 - b. 单击 **Add** 来输入隧道 IPv6 地址和子网掩码。将 **Gateway** 字段留空。
9. 点 **Save** 存储连接配置集。

后续步骤

- 在 [WireGuard 服务器上配置 firewalld 服务](#)。

验证

1. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

2. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

其他资源

- **wg(8)** man page

8.9. 使用 **WG-QUICK** 服务配置 **WIREGUARD** 服务器

您可以通过在 **/etc/wireguard/** 目录中创建配置文件来配置 WireGuard 服务器。使用此方法独立于 NetworkManager 配置服务。

此流程假设以下设置：

- server :
 - 私钥: **YFAnE0psglDiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
 - 隧道 IPv4 地址：**192.0.2.1/24**
 - 频道 IPv6 地址：**2001:db8:1::1/32**
- Client:
 - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
 - 隧道 IPv4 地址：**192.0.2.2/24**
 - 频道 IPv6 地址：**2001:db8:1::2/32**

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 服务器的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 客户端的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 安装 `wireguard-tools` 软件包：

```
# dnf install wireguard-tools
```

2. 使用以下内容创建 `/etc/wireguard/wg0.conf` 文件：

```
[Interface]
Address = 192.0.2.1/24, 2001:db8:1::1/32
ListenPort = 51820
PrivateKey = YFAnE0psgldiAF7XR4abxiwVRnIMfeltxu10s/c4JXg=

[Peer]
PublicKey = bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
AllowedIPs = 192.0.2.2, 2001:db8:1::2
```

- **[Interface]** 部分描述了服务器上接口的 WireGuard 设置：
 - **Address**：逗号分隔的服务器的隧道 IP 地址的列表。
 - **PrivateKey**：服务器的私钥。
 - **ListenPort**：WireGuard 在其上侦听传入的 UDP 连接的端口。在主机上始终设置固定端口号，接收传入的 WireGuard 连接。如果您没有设置端口，WireGuard 会在每次激活 **wg0** 接口时使用随机的空闲端口。
 - 每个 **[Peer]** 部分描述了一个客户端的设置：
 - **PublicKey**：客户端的公钥。
 - **AllowedIPs**：允许向这个服务器发送数据的客户端的隧道 IP 地址。
3. 启用并启动 WireGuard 连接：

```
# systemctl enable --now wg-quick@wg0
```

`systemd` 实例名称必须与 `/etc/wireguard/` 目录中不带 `.conf` 后缀的配置文件的名称匹配。该服务还会将这个名称用于虚拟网络接口。

后续步骤

- 在 [WireGuard 服务器上配置 firewalld 服务](#)。

验证

1. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

要在输出中显示私钥，请使用 `WG_HIDE_KEYS=never wg show wg0` 命令。

2. 显示 `wg0` 设备的 IP 配置：

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 scope global wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global
        valid_lft forever preferred_lft forever
```

其他资源

- `wg(8)` man page
- `wg-quick(8)` 手册页

8.10. 使用命令行在 WIREGUARD 服务器上配置 FIREWALLD

您必须在 WireGuard 服务器上配置 `firewalld` 服务，以允许来自客户端的传入的连接。另外，如果客户端能够使用 WireGuard 服务器作为默认网关，并通过隧道路由所有流量，则必须启用伪装。

步骤

1. 为 `firewalld` 服务中的传入连接打开 WireGuard 端口：

```
# firewall-cmd --permanent --add-port=51820/udp --zone=public
```

2. 如果客户端应该使用 WireGuard 服务器作为默认网关来通过隧道路由所有流量，请为 `public` 区域启用伪装：

```
# firewall-cmd --permanent --zone=public --add-masquerade
```

3. 重新加载 `firewalld` 规则。

```
# firewall-cmd --reload
```

验证

- 显示 `public` 区域的配置：

```
# firewall-cmd --list-all
public (active)
...
ports: 51820/udp
masquerade: yes
...
```

其他资源

- [firewall-cmd \(1\) 手册页](#)

8.11. 使用 RHEL WEB 控制台在 WIREGUARD 服务器上配置 FIREWALLD

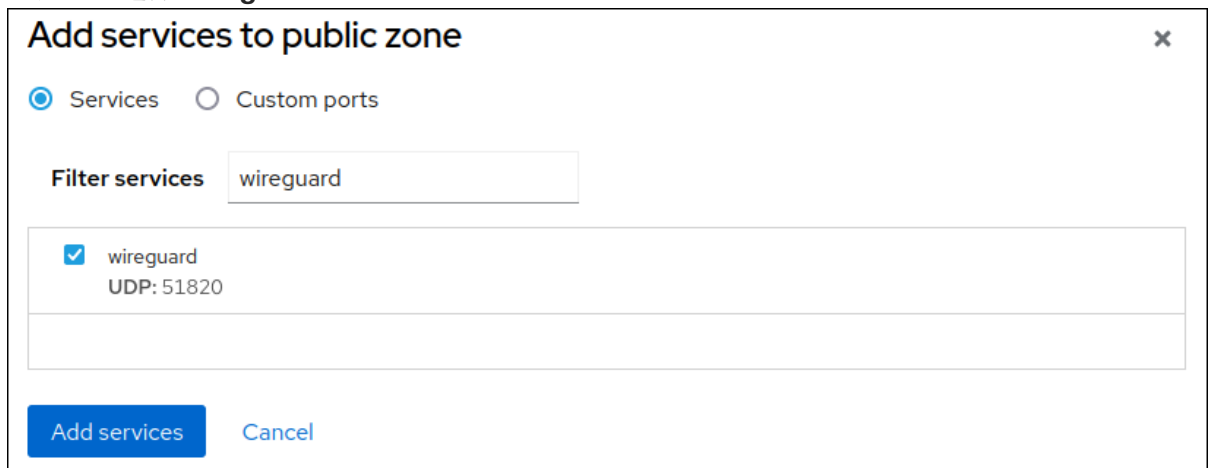
您必须在 WireGuard 服务器上配置 `firewalld` 服务，以允许来自客户端的进入连接。另外，如果客户端能够使用 WireGuard 服务器作为默认网关，并通过隧道路由所有流量，则必须启用伪装。

先决条件

- 已登陆到 RHEL web 控制台。

流程

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 点 Firewall 部分中的 **Edit rules and zones**。
3. 在 **Filter services** 字段中输入 **wireguard**。
4. 从列表中选择 **wireguard** 条目。



5. 点 **Add services**。
6. 如果客户端应该使用 WireGuard 服务器作为默认网关来通过隧道路由所有流量，请为 **public** 区域启用伪装：

```
# firewall-cmd --permanent --zone=public --add-masquerade
# firewall-cmd --reload
```

请注意，您无法在 web 控制台的 `firewalld` 区域中启用伪装。

验证

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 点 Firewall 部分中的 **Edit rules and zones**。
3. 该列表包含 **wireguard** 服务的条目，并显示您在 WireGuard 连接配置文件中配置的 UDP 端口。
4. 要验证 `firewalld` 公共区 中是否启用了伪装，请输入：

```
# firewall-cmd --list-all --zone=public
```

```
public (active)
...
ports: 51820/udp
masquerade: yes
...
```

8.12. 使用图形界面在 WIREGUARD 服务器上配置 FIREWALLD

您必须在 WireGuard 服务器上配置 **firewalld** 服务，以允许来自客户端的传入的连接。另外，如果客户端能够使用 WireGuard 服务器作为默认网关，并通过隧道路由所有流量，则必须启用伪装。

步骤

1. 按 **Super** 键，输入 **firewall**，然后从结果中选择 **Firewall** 应用程序。
2. 在 **Configuration** 列表中选择 **Permanent**。
3. 选择 **public** 区域。
4. 允许到 WireGuard 端口的传入连接：
 - a. 在 **Ports** 选项卡上，单击 **Add**。
 - b. 输入您为传入 WireGuard 连接设置的端口号：
 - c. 从 **Protocol** 列表中选择 **udp**。
 - d. 点**确定**。
5. 如果客户端应该通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关：
 - a. 导航到 **public** 区域的 **Masquerading** 选项卡。
 - b. 选择 **Masquerade zone**。
6. 选择 **Options** → **Reload Firewalld**。

验证

- 显示 **public** 区域的配置：

```
# firewall-cmd --list-all
public (active)
...
ports: 51820/udp
masquerade: yes
...
```

8.13. 使用 NMCLI 配置 WIREGUARD 客户端

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 客户端。使用此方法让 NetworkManager 管理 WireGuard 连接。

此流程假设以下设置：

- Client:
 - Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
 - 隧道 IPv4 地址 : **192.0.2.2/24**
 - 隧道 IPv6 地址 : **2001:db8:1::2/32**
- server :
 - 公钥 : **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
 - 隧道 IPv4 地址 : **192.0.2.1/24**
 - 隧道 IPv6 地址 : **2001:db8:1::1/32**

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息 :
 - 客户端的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 服务器的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 添加 NetworkManager WireGuard 连接配置集 :

```
# nmcli connection add type wireguard con-name client-wg0 ifname wg0 autoconnect
no
```

此命令创建一个名为 **client-wg0** 的配置文件，并将虚拟接口 **wg0** 分配给它。要防止连接在添加后而没有最终配置的情况下自动启动，请禁用 **autoconnect** 参数。

2. 可选：配置 NetworkManager，使其不会自动启动 **client-wg** 连接：

```
# nmcli connection modify client-wg0 autoconnect no
```

3. 设置客户端的隧道 IPv4 地址和子网掩码：

```
# nmcli connection modify client-wg0 ipv4.method manual ipv4.addresses 192.0.2.2/24
```

4. 设置客户端的隧道 IPv6 地址和子网掩码：

```
# nmcli connection modify client-wg0 ipv6.method manual ipv6.addresses
2001:db8:1::2/32
```

5. 如果要通过隧道路由所有流量，请将服务器的隧道 IP 地址设置为默认网关：

```
# nmcli connection modify client-wg0 ipv4.gateway 192.0.2.1 ipv6.gateway
2001:db8:1::1
```

通过隧道路由所有流量要求您在后续步骤中将此客户端上的 **allowed-ips** 设置为 **0.0.0.0/0:::0**。

请注意，通过隧道路由所有流量可能会影响到其他主机的连接，具体取决于服务器路由和防火墙配置。

6. 将客户端的私钥添加到连接配置文件中：

```
# nmcli connection modify client-wg0 wireguard.private-key
"aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A="
```

7. 为您要允许与此客户端通信的每台服务器添加对等配置。您必须手动添加这些设置，因为 **nmcli** 工具不支持设置相应的连接属性。
 - a. 编辑 `/etc/NetworkManager/system-connections/client-wg0.nmconnection` 文件，并追加：

```
[wireguard-peer.UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=]
endpoint=server.example.com:51820
allowed-ips=192.0.2.1;2001:db8:1::1;
persistent-keepalive=20
```

- `[wireguard-peer.<public_key_of_the_server>]` 条目定义了服务器的对等部分，部分名称有服务器的公钥。
- **endpoint** 参数设置服务器的主机名或 IP 地址以及端口。客户端使用此信息来建立连接。
- **allowed-ips** 参数设置可向这个客户端发送数据的 IP 地址列表。例如，将参数设置为：
 - 服务器隧道 IP 地址，以仅允许服务器与此客户端通信。上例中的值可配置这种情况。
 - **0.0.0.0/0:::0**；允许任何远程 IPv4 和 IPv6 地址与此客户端进行通信。使用此设置通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关。
- 可选的 **persistent-keepalive** 参数定义一个 WireGuard 向服务器发送一个实时数据包的间隔（以秒为单位）。如果您在网络中使用具有网络地址转换(NAT)的客户端，或者防火墙在一段时间不活跃后关闭 UDP 连接，则设置此参数。

- b. 重新载入 **client-wg0** 连接配置文件：

```
# nmcli connection load /etc/NetworkManager/system-connections/client-wg0.nmconnection
```

8. 重新激活 **client-wg0** 连接：

```
# nmcli connection up client-wg0
```

验证

1. Ping 服务器的 IP 地址：


```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

请注意，如果您已经通过 VPN 隧道发送流量，则输出只有 **latest handshake** 和 **transfer** 条目。

3. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
  link/none
  inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
    valid_lft forever preferred_lft forever
  inet6 2001:db8:1::2/32 scope global noprefixroute
    valid_lft forever preferred_lft forever
  inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

其他资源

- **wg(8)** man page
- **nm-settings (5)** 手册页中的 **WireGuard** 设置 部分

8.14. 使用 NMTUI 配置 WIREGUARD 客户端

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 客户端。使用此方法让 NetworkManager 管理 WireGuard 连接。

此流程假设以下设置：

- Client:
 - Private key: **aPUcp5vHz8yMLrzK8SsDyYnV33lhE/k20e52iKJFV0A=**
 - 隧道 IPv4 地址：**192.0.2.2/24**
 - 频道 IPv6 地址：**2001:db8:1::2/32**

- server :
 - 公钥 : `UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=`
 - 隧道 IPv4 地址 : `192.0.2.1/24`
 - 频道 IPv6 地址 : `2001:db8:1::1/32`

先决条件

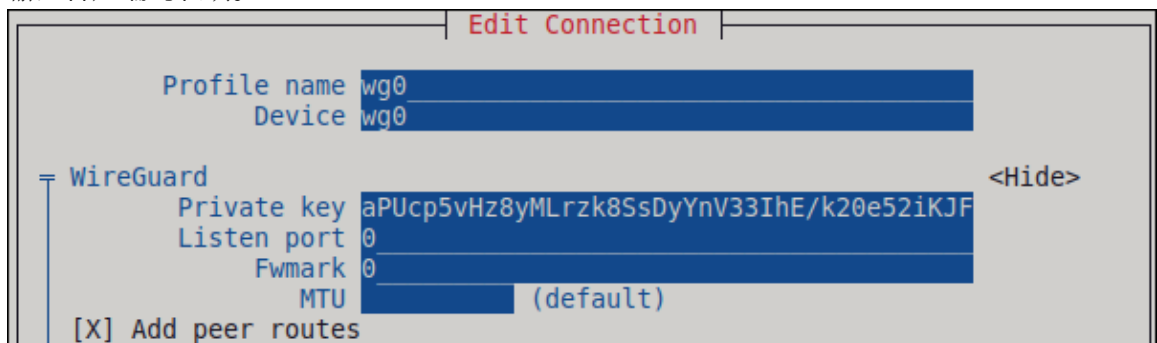
- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息 :
 - 客户端的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 服务器的公钥
 - 服务器的静态隧道 IP 地址和子网掩码
- 已安装 `NetworkManager-tui` 软件包

流程

1. 启动 `nmtui` 应用程序 :

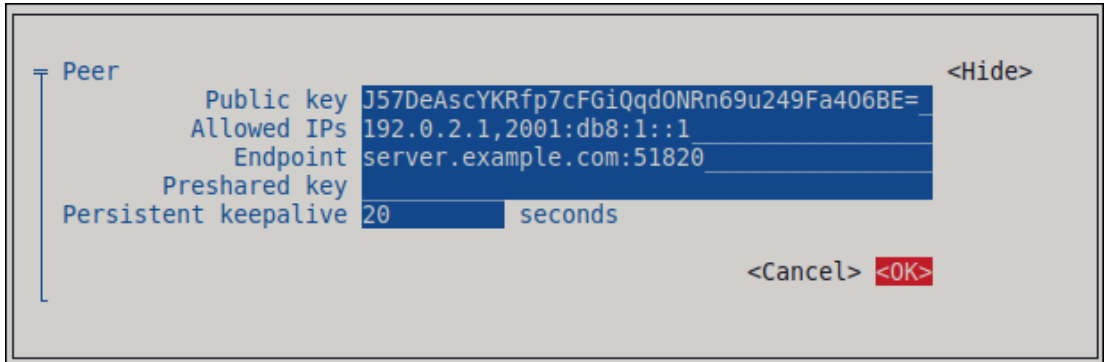
```
# nmtui
```

2. 选择 **Edit a connection**, 点 **Enter**。
3. 选择 **添加**, 然后按 **Enter** 键。
4. 选择列表中的 **WireGuard** 连接类型, 然后按 **Enter** 键。
5. 在 **Edit connection** 窗口中 :
 - a. 输入连接名称和虚拟接口, 如 `wg0`, 以便 NetworkManager 应分配给连接。
 - b. 输入客户端的私钥。

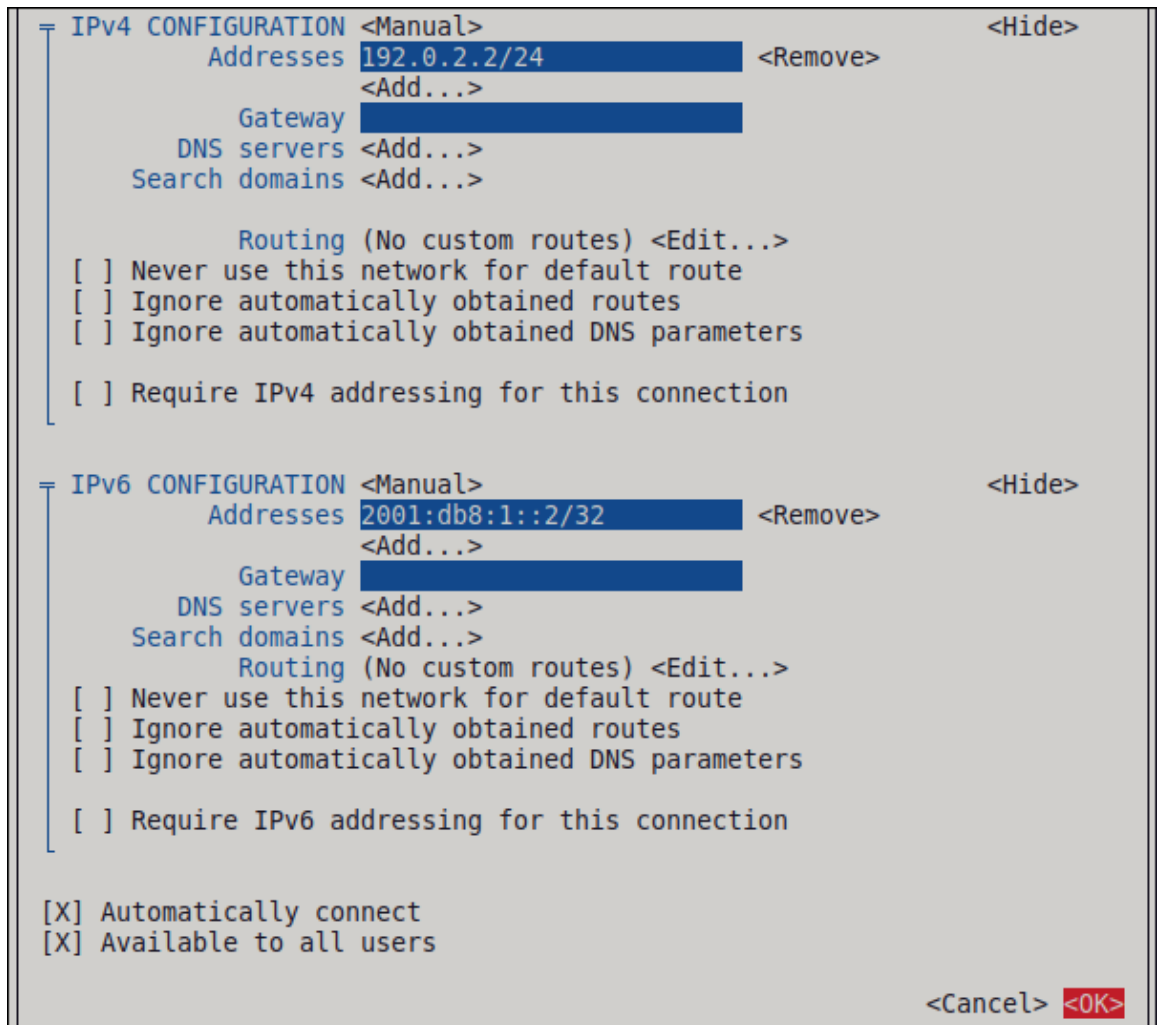


- c. 点 **Peers** 窗格旁的 **Add s**:
 - i. 输入服务器的公钥。
 - ii. 设置 **Allowed IPs** 字段。例如, 将其设置为 :

- 服务器隧道 IP 地址，以仅允许服务器与此客户端通信。
 - **0.0.0.0/0,::/0** 以允许任何远程 IPv4 和 IPv6 地址与此客户端通信。使用此设置通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关。
- iii. 在 **Endpoint** 字段中输入 WireGuard 服务器的主机名或 IP 地址和端口。使用以下格式：
hostname_or_IP:port_number
- iv. 可选：如果您在带有网络地址转换(NAT)的网络中使用客户端，或者防火墙在一定时间不活动状态后关闭 UDP 连接，则设置持久的间隔（以秒为单位）。在这个间隔中，客户端向服务器发送一个保留数据包。
- v. 选择 **确定**，然后按 **Enter** 键。



- d. 选择 **IPv4 Configuration** 旁边的 **显示**，然后按 **Enter** 键。
- i. 选择 IPv4 配置方法 **Manual**。
 - ii. 输入隧道 IPv4 地址和子网掩码。将 **Gateway** 字段留空。
- e. 选择 **IPv6 Configuration** 旁边的 **显示**，然后按 **Enter** 键。
- i. 选择 IPv6 配置方法 **Manual**。
 - ii. 输入隧道 IPv6 地址和子网掩码。将 **Gateway** 字段留空。
- f. 可选：选择 **Automatically connect**。
- g. 选择 **确定**，然后按 **Enter** 键



6. 在带有连接列表的窗口中，选择 **Back**，然后按 **Enter** 键。
7. 在 **NetworkManager TUI** 主窗口中，选择 **Quit**，然后按 **Enter** 键。

验证

1. Ping 服务器的 IP 地址：

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

要在输出中显示私钥，请使用 `WG_HIDE_KEYS=never wg show wg0` 命令。

请注意，如果已经通过 VPN 隧道发送流量，输出只会包含 `latest handshake` 和 `transfer` 条目。

3. 显示 `wg0` 设备的 IP 配置：

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

其它资源

- [wg\(8\) 手册页](#)

8.15. 使用 RHEL WEB 控制台配置 WIREGUARD 客户端

您可以使用基于浏览器的 RHEL web 控制台配置 WireGuard 客户端。使用此方法让 NetworkManager 管理 WireGuard 连接。

先决条件

- 已登陆到 RHEL web 控制台。
- 您知道以下信息：
 - 服务器和客户端的静态隧道 IP 地址和子网掩码
 - 服务器的公钥

步骤

1. 在屏幕左侧的导航中选择 **Networking** 选项卡。
2. 在 **Interfaces** 部分点 **Add VPN**。
3. 如果没有安装 **wireguard-tools** 和 **systemd-resolved** 软件包，Web 控制台会显示对应的通知。点 **Install** 安装这些软件包。
4. 输入您要创建的 WireGuard 设备名称。
5. 配置此主机的密钥对：
 - 如果要使用 web 控制台创建的密钥：
 - i. 在 **私钥** 区域中保留预先选择的 **Generated** 选项。
 - ii. 请注意 **公钥** 值。配置客户端时需要此信息。
 - 如果要使用现有的私钥：

- i. 在 **私钥区域**中选择 **粘贴现有密钥**。
 - ii. 将私钥粘贴到文本字段中。Web 控制台自动计算对应的公钥。
6. 在 **Listen port** 字段中保留 **0** 值。
 7. 设置客户端的隧道 IPv4 地址和子网掩码。
要同时设置 IPv6 地址，您必须在创建后编辑连接。
 8. 为您要允许与此客户端通信的服务器添加对等配置：
 - a. 单击 **Add peer**。
 - b. 输入服务器的公钥。
 - c. 将 **Endpoint** 字段设置为主机名或 IP 地址以及服务器的端口，如 **server.example.com:51820**。客户端使用此信息来建立连接。
 - d. 将 **Allowed IP** 字段设置为允许向这个服务器发送数据的客户端的隧道 IP 地址。例如，将字段设置为以下之一：
 - 服务器隧道 IP 地址，以仅允许服务器与此客户端通信。下面的屏幕截图中的值可配置这种情况。
 - **0.0.0.0/0** 以允许任何远程 IPv4 地址与此客户端通信。使用此设置通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关。

Add WireGuard VPN ✕

Name

Private key Generated Paste existing key

Public key

Listen port Will be set to "Automatic"

IPv4 addresses
Multiple addresses can be specified using commas or spaces as delimiters.

Peers ? Add peer

Public key	Endpoint	Allowed IPs	
<input type="text" value="UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u24 ..."/>	<input type="text" value="server.example.com ..."/>	<input type="text" value="192.0.2.1/24"/>	

Add Cancel

9. 点 **Add** 创建 WireGuard 连接。

10. 如果您还想设置隧道 IPv6 地址：
 - a. 在 **Interfaces** 部分点 WireGuard 连接的名称。
 - b. 点 **IPv6** 旁边的 **编辑**。
 - c. 将 **Addresses** 字段设置为 **Manual**，并输入客户端的隧道 IPv6 地址和前缀。
 - d. 点击 **Save**。

验证

1. Ping 服务器的 IP 地址：

```
# ping 192.0.2.1
```

当您尝试通过隧道发送流量时，WireGuard 会建立连接。

2. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 45513

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

请注意，如果您已经通过 VPN 隧道发送流量，则输出只有 **latest handshake** 和 **transfer** 条目。

3. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
  link/none
  inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
    valid_lft forever preferred_lft forever
  inet6 2001:db8:1::2/32 scope global noprefixroute
    valid_lft forever preferred_lft forever
  inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

8.16. 使用 NM-CONNECTION-EDITOR 配置 WIREGUARD 客户端

您可以通过在 NetworkManager 中创建连接配置集来配置 WireGuard 客户端。使用此方法让 NetworkManager 管理 WireGuard 连接。

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 客户端的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 服务器的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 打开终端窗口，输入：

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

2. 点 **+** 按钮添加新连接。
3. 选择 **WireGuard** 连接类型，然后点 **Create**。
4. 可选：更新连接名称。
5. 可选：在 **General** 选项卡中，选择 **Connect automatically with priority**。
6. 在 **WireGuard** 选项卡中：
 - a. 输入虚拟接口的名称，如 **wg0**，这是 NetworkManager 应分配给连接的名称。
 - b. 输入客户端的私钥。
 - c. 点 **Add** 添加对等点：
 - i. 输入服务器的公钥。
 - ii. 设置 **Allowed IPs** 字段。例如，将其设置为：
 - 服务器隧道 IP 地址，以仅允许服务器与此客户端通信。
 - **0.0.0.0/0:::0**；允许任何远程 IPv4 和 IPv6 地址与此客户端进行通信。使用此设置通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关。
请注意，通过隧道路由所有流量可能会影响到其他主机的连接，具体取决于服务器路由和防火墙配置。
 - iii. 在 **Endpoint** 字段中输入 WireGuard 服务器的主机名或 IP 地址以及端口。使用以下格式：**hostname_or_IP:port_number**
 - iv. 可选：如果您在带有网络地址转换(NAT)的网络中使用客户端，或者防火墙在一定时间不活动状态后关闭 UDP 连接，则设置持久的间隔（以秒为单位）。在这个间隔，客户端会向服务器发送一个实时数据包。
 - v. 点应用。
7. 在 **IPv4 设置** 标签页中：

- a. 在 **Method** 列表中选择 **Manual**。
 - b. 单击 **Add** 来输入隧道 IPv4 地址和子网掩码。
 - c. 如果要通过隧道路由所有流量，请在 **Gateway** 字段中设置服务器的隧道 IPv4 地址。否则，将字段留空。
通过隧道路由所有 IPv4 流量需要在这个客户端的 **Allowed IP** 字段中包含 **0.0.0.0/0**。
8. 在 **IPv6 设置** 标签页中：
- a. 在 **Method** 列表中选择 **Manual**。
 - b. 单击 **Add** 来输入隧道 IPv6 地址和子网掩码。
 - c. 如果要通过隧道路由所有流量，请在 **Gateway** 字段中设置服务器的隧道 IPv6 地址。否则，将字段留空。
通过隧道路由所有 IPv4 流量需要在这个客户端的 **Allowed IP** 字段中包含 **::/0**。
9. 点 **Save** 存储连接配置集。

验证

1. Ping 服务器的 IP 地址：

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. 显示 **wg0** 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

请注意，如果您已经通过 VPN 隧道发送流量，则输出只有 **latest handshake** 和 **transfer** 条目。

3. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
  link/none
  inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
    valid_lft forever preferred_lft forever
  inet6 2001:db8:1::2/32 scope global noprefixroute
```

```
valid_lft forever preferred_lft forever
inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
valid_lft forever preferred_lft forever
```

其他资源

- [wg\(8\) 手册页](#)

8.17. 使用 WG-QUICK 服务配置 WIREGUARD 客户端

您可以通过在 `/etc/wireguard/` 目录中创建配置文件来配置 WireGuard 客户端。使用此方法独立于 NetworkManager 配置服务。

此流程假设以下设置：

- Client:
 - Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
 - 隧道 IPv4 地址：**192.0.2.2/24**
 - 频道 IPv6 地址：**2001:db8:1::2/32**
- server :
 - 公钥：**UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
 - 隧道 IPv4 地址：**192.0.2.1/24**
 - 频道 IPv6 地址：**2001:db8:1::1/32**

先决条件

- 您已为服务器和客户端生成了公钥和私钥。
- 您知道以下信息：
 - 客户端的私钥
 - 客户端的静态隧道 IP 地址和子网掩码
 - 服务器的公钥
 - 服务器的静态隧道 IP 地址和子网掩码

步骤

1. 安装 `wireguard-tools` 软件包：

```
# dnf install wireguard-tools
```

2. 使用以下内容创建 `/etc/wireguard/wg0.conf` 文件：

```
[Interface]
Address = 192.0.2.2/24, 2001:db8:1::2/32
```

```
PrivateKey = aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=
```

```
[Peer]
```

```
PublicKey = UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
```

```
AllowedIPs = 192.0.2.1, 2001:db8:1::1
```

```
Endpoint = server.example.com:51820
```

```
PersistentKeepalive = 20
```

- **[Interface]** 部分描述了客户端上接口的 WireGuard 设置：
 - **Address**：逗号分隔的客户端隧道 IP 地址的列表。
 - **PrivateKey**：客户端的私钥。
- **[Peer]** 部分描述了服务器的设置：
 - **PublicKey**：服务器的公钥。
 - **AllowedIPs**：允许向此客户端发送数据的 IP 地址。例如，将参数设置为：
 - 服务器隧道 IP 地址，以仅允许服务器与此客户端通信。上例中的值可配置这种情况。
 - **0.0.0.0/0, ::/0** 允许任何远程 IPv4 和 IPv6 地址与此客户端进行通信。使用此设置通过隧道路由所有流量，并使用 WireGuard 服务器作为默认网关。
 - **Endpoint**：设置服务器的主机名或 IP 地址以及端口。客户端使用此信息来建立连接。
 - 可选的 **persistent-keepalive** 参数定义 WireGuard 将 keepalive 数据包发送给服务器的间隔（以秒为单位）。如果您在网络中使用具有网络地址转换(NAT)的客户端，或者防火墙在一段时间不活跃后关闭 UDP 连接，则设置此参数。

3. 启用并启动 WireGuard 连接：

```
# systemctl enable --now wg-quick@wg0
```

systemd 实例名称必须与 `/etc/wireguard/` 目录中不带 `.conf` 后缀的配置文件的名称匹配。该服务还会将这个名称用于虚拟网络接口。

验证

1. Ping 服务器的 IP 地址：

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. 显示 wg0 设备的接口配置：

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
```

```
allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
latest handshake: 1 minute, 41 seconds ago
transfer: 824 B received, 1.01 KiB sent
persistent keepalive: every 20 seconds
```

要在输出中显示私钥，请使用 **WG_HIDE_KEYS=never wg show wg0** 命令。

请注意，如果已经通过 VPN 隧道发送流量，输出只会包含 **latest handshake** 和 **transfer** 条目。

3. 显示 **wg0** 设备的 IP 配置：

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 scope global wg0
        valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32__ scope global
        valid_lft forever preferred_lft forever
```

其他资源

- [wg\(8\) 手册页](#)
- [wg-quick \(8\) 手册页](#)

第 9 章 配置 IP 隧道

与 VPN 类似，IP 隧道通过第三个网络（如互联网）直接连接两个网络。然而，不是所有的隧道协议都支持加密。

两个建立隧道网络的路由器至少需要两个接口：

- 一个连接到本地网络的接口
- 一个连接到建立隧道的网络的接口。

要建立隧道，您可以在两个路由器中使用来自远程子网的 IP 地址创建一个虚拟接口。

NetworkManager 支持以下 IP 隧道：

- 通用路由封装（GRE）
- IPv6 上的通用路由封装（IP6GRE）
- 通用路由封装终端接入点（GRETAP）
- 通用路由登录在 IPv6（IP6GRETAP）上
- IPv4 over IPv4（IPIP）
- IPv4 over IPv6（IPIP6）
- IPv6 over IPv6（IP6IP6）
- 简单的互联网转换（SIT）

根据类型，这些通道在 Open Systems Interconnection（OSI）的第 2 层或 3 层动作。

9.1. 使用 nmcli 配置 IPIP 隧道来封装 IPV4 数据包中的 IPV4 流量

IP over IP（IPIP）隧道在 OSI 层 3 上运行，并封装 IPv4 数据包中的 IPv4 流量，如 [RFC 2003 所述](#)。

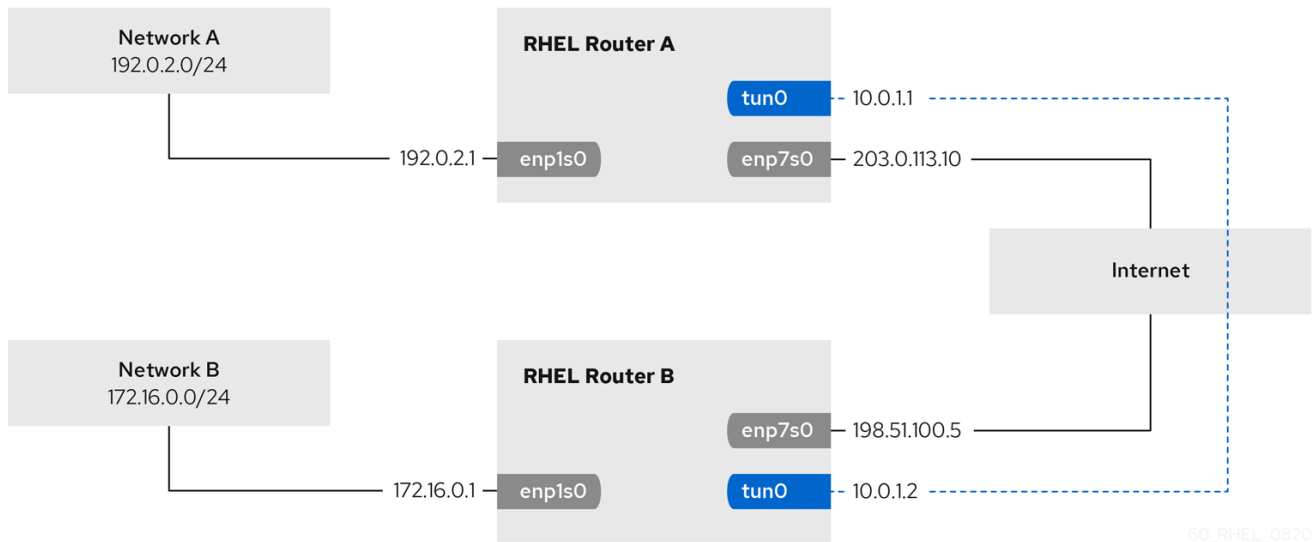


重要

通过 IPIP 隧道发送的数据没有加密。出于安全考虑，只在已经加密的数据中使用隧道，比如 HTTPS。

请注意，IPIP 隧道只支持单播数据包。如果您需要支持多播的 IPv4 隧道，请参阅 [使用 nmcli 配置 GRE 隧道来封装 IPv4 数据包中的第 3 层流量](#)。

例如，您可以在两个 RHEL 路由器之间创建一个 IPIP 隧道来通过互联网连接两个内部子网，如下图所示：



60_RHEL_0820

先决条件

- 每个 RHEL 路由器都有一个网络接口，它连接到其本地子网。
- 每个 RHEL 路由器都有一个连接到互联网的网络接口。
- 您需要通过隧道发送的流量是 IPv4 单播。

流程

1. 在网络 A 的 RHEL 路由器上：

a. 创建名为 **tun0** 的 IPIP 隧道接口：

```
# 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
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

b. 将 IPv4 地址设为 **tun0** 设备：

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.1/30'
```

请注意，具有两个可用 IP 地址的 /30 子网足以满足隧道的需要。

c. 将 **tun0** 连接配置为使用手动 IPv4 配置：

```
# nmcli connection modify tun0 ipv4.method manual
```

d. 添加一个静态路由，其将到 **172.16.0.0/24** 网络的流量路由到路由器 B 上的隧道 IP：

```
# nmcli connection modify tun0 +ipv4.routes "172.16.0.0/24 10.0.1.2"
```

e. 启用 **tun0** 连接。

```
# nmcli connection up tun0
```

f. 启用数据包转发：

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. 在网络 B 中的 RHEL 路由器中：

a. 创建名为 **tun0** 的 IPIP 隧道接口：

```
# 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
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

b. 将 IPv4 地址设为 **tun0** 设备：

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.2/30'
```

c. 将 **tun0** 连接配置为使用手动 IPv4 配置：

```
# nmcli connection modify tun0 ipv4.method manual
```

d. 添加一个静态路由，其将路由到 **192.0.2.0/24** 网络的流量路由到路由器 A 上的隧道 IP：

```
# nmcli connection modify tun0 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

e. 启用 **tun0** 连接。

```
# nmcli connection up tun0
```

f. 启用数据包转发：

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

验证

- 从每个 RHEL 路由器中，ping 路由器的内部接口的 IP 地址：

a. 在路由器 A 上，ping **172.16.0.1**：

```
# ping 172.16.0.1
```

b. 在路由器 B 上，ping **192.0.2.1**：

```
# ping 192.0.2.1
```

其他资源

- [nmcli\(1\) 手册页](#)
- [nm-settings\(5\) 手册页](#)

9.2. 使用 NMCLI 配置 GRE 隧道来封装 IPV4 数据包中的第 3 层流量

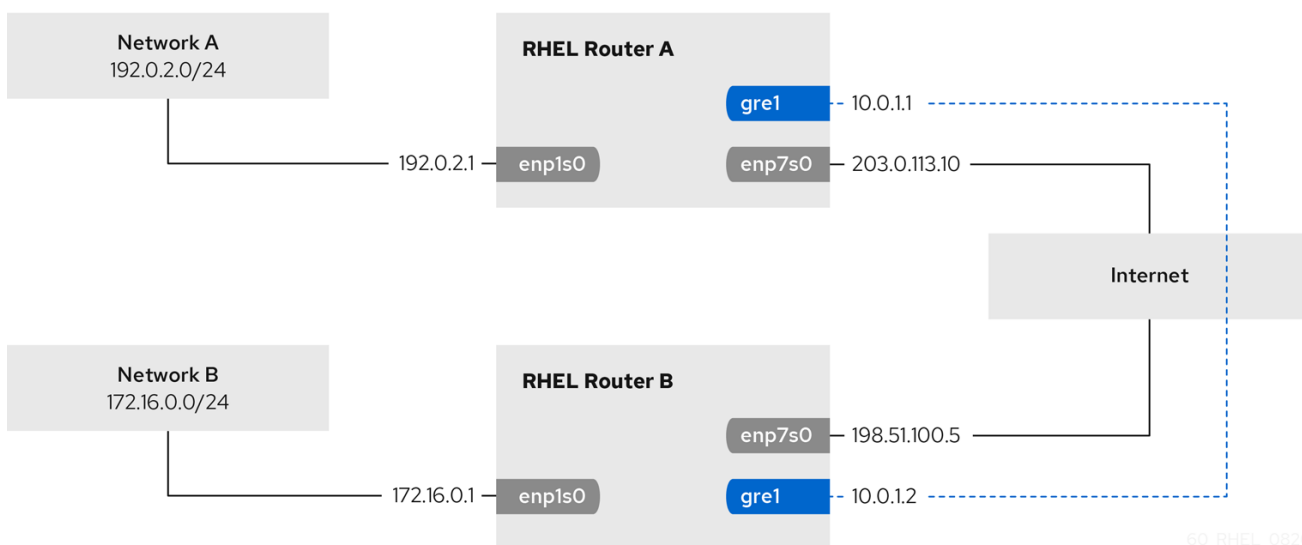
Generic Routing Encapsulation (GRE) 隧道封装 IPv4 数据包中的第 3 层流量，如 RFC 2784 所述。GRE 隧道可以使用有效的以太网类型封装任何第 3 层协议。



重要

通过 GRE 隧道发送的数据没有加密。出于安全考虑，只在已经加密的数据中使用隧道，比如 HTTPS。

例如，您可以在两个 RHEL 路由器之间创建一个 GRE 隧道来通过互联网连接两个内部子网，如下图所示：



注意

保留 **gre0** 设备名称。对该设备使用 **gre1** 或者不同的名称。

先决条件

- 每个 RHEL 路由器都有一个网络接口，它连接到其本地子网。
- 每个 RHEL 路由器都有一个连接到互联网的网络接口。

流程

1. 在网络 A 的 RHEL 路由器上：
 - a. 创建名为 **gre1** 的 GRE 隧道接口：

```
# 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
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

- b. 将 IPv4 地址设为 **gre1** 设备：


```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.1/30'
```

请注意，具有两个可用 IP 地址的 /30 子网足以满足隧道的需要。

- c. 将 **gre1** 连接配置为使用手动 IPv4 配置：

```
# nmcli connection modify gre1 ipv4.method manual
```

- d. 添加一个静态路由，其将到 **172.16.0.0/24** 网络的流量路由到路由器 B 上的隧道 IP：

```
# nmcli connection modify gre1 +ipv4.routes "172.16.0.0/24 10.0.1.2"
```

- e. 启用 **gre1** 连接。

```
# nmcli connection up gre1
```

- f. 启用数据包转发：

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. 在网络 B 中的 RHEL 路由器中：

- a. 创建名为 **gre1** 的 GRE 隧道接口：

```
# 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
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

- b. 将 IPv4 地址设为 **gre1** 设备：

```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.2/30'
```

- c. 将 **gre1** 连接配置为使用手动 IPv4 配置：

```
# nmcli connection modify gre1 ipv4.method manual
```

- d. 添加一个静态路由，其将路由到 **192.0.2.0/24** 网络的流量路由到路由器 A 上的隧道 IP：

```
# nmcli connection modify gre1 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

- e. 启用 **gre1** 连接。

```
# nmcli connection up gre1
```

- f. 启用数据包转发：

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

1. 从每个 RHEL 路由器中，ping 路由器的内部接口的 IP 地址：

a. 在路由器 A 上，ping **172.16.0.1**：

```
# ping 172.16.0.1
```

b. 在路由器 B 上，ping **192.0.2.1**：

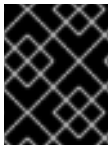
```
# ping 192.0.2.1
```

其他资源

- [nmcli\(1\) 手册页](#)
- [nm-settings\(5\) 手册页](#)

9.3. 配置 GRE-TAP 隧道来通过 IPV4 传输以太网帧

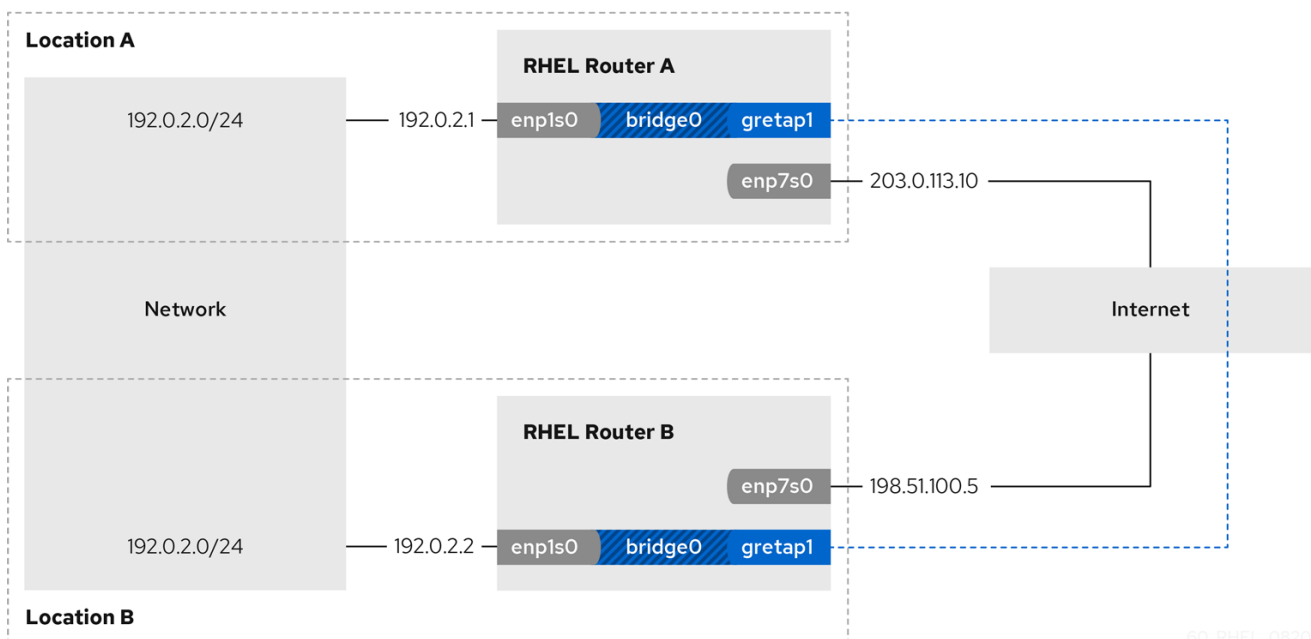
通用路由封装终端接入点(GRE-TAP)隧道在 OSI 级别 2 上运行，并封装 IPv4 数据包中的以太网流量，如 [RFC 2784](#) 所述。



重要

通过 GRE-TAP 隧道发送的数据没有加密。出于安全考虑，通过 VPN 或不同的加密连接建立隧道。

例如，您可以在两个 RHEL 路由器之间创建一个 GRE-TAP 隧道，以使用网桥连接两个网络，如下图所示：



60_RHEL_0820



注意

保留 **gretap0** 设备名称。对该设备使用 **gretap1** 或者不同的名称。

先决条件

- 每个 RHEL 路由器都有一个网络接口，它连接到其本地网络，接口没有分配 IP 配置。
- 每个 RHEL 路由器都有一个连接到互联网的网络接口。

流程

1. 在网络 A 的 RHEL 路由器上：

- a. 创建名为 **bridge0** 的网桥接口：

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

- b. 配置网桥的 IP 设置：

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify bridge0 ipv4.method manual
```

- c. 为连接到本地网络的接口添加新连接配置集到网桥：

```
# nmcli connection add type ethernet port-type bridge con-name bridge0-port1
ifname enp1s0 controller bridge0
```

- d. 为网桥添加 GRE-TAP 隧道接口的新连接配置集：

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap port-type bridge con-
name bridge0-port2 ifname gretap1 remote 198.51.100.5 local 203.0.113.10
controller bridge0
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

- e. 可选：如果您不需要，STP（Spanning Tree Protocol）：

```
# nmcli connection modify bridge0 bridge.stp no
```

默认情况下，STP 被启用并导致在使用连接前出现延迟。

- f. 配置激活 **bridge0** 连接会自动激活网桥端口：

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

- g. 激活 **bridge0** 连接：

```
# nmcli connection up bridge0
```

2. 在网络 B 中的 RHEL 路由器中：

- a. 创建名为 **bridge0** 的网桥接口：

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

- b. 配置网桥的 IP 设置：

■

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.2/24'
# nmcli connection modify bridge0 ipv4.method manual
```

- c. 为连接到本地网络的接口添加新连接配置集到网桥：

```
# nmcli connection add type ethernet port-type bridge con-name bridge0-port1
ifname enp1s0 controller bridge0
```

- d. 为网桥添加 GREYAP 隧道接口的新连接配置集：

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap port-type bridge con-
name bridge0-port2 ifname gretap1 remote 203.0.113.10 local 198.51.100.5
controller bridge0
```

remote 和 **local** 参数设置远程和本地路由器的公共 IP 地址。

- e. 可选：如果您不需要，STP（Spanning Tree Protocol）：

```
# nmcli connection modify bridge0 bridge.stp no
```

- f. 配置激活 **bridge0** 连接会自动激活网桥端口：

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

- g. 激活 **bridge0** 连接：

```
# nmcli connection up bridge0
```

验证

1. 在两个路由器上，验证 **enp1s0** 和 **gretap1** 连接是否已连接，并且 **CONNECTION** 列是否显示端口的连接名称：

```
# nmcli device
nmcli device
DEVICE TYPE STATE CONNECTION
...
bridge0 bridge connected bridge0
enp1s0 ethernet connected bridge0-port1
gretap1 iptunnel connected bridge0-port2
```

2. 从每个 RHEL 路由器中，ping 路由器的内部接口的 IP 地址：

- a. 在路由器 A 上，ping **192.0.2.2**：

```
# ping 192.0.2.2
```

- b. 在路由器 B 上，ping **192.0.2.1**：

```
# ping 192.0.2.1
```

其他资源

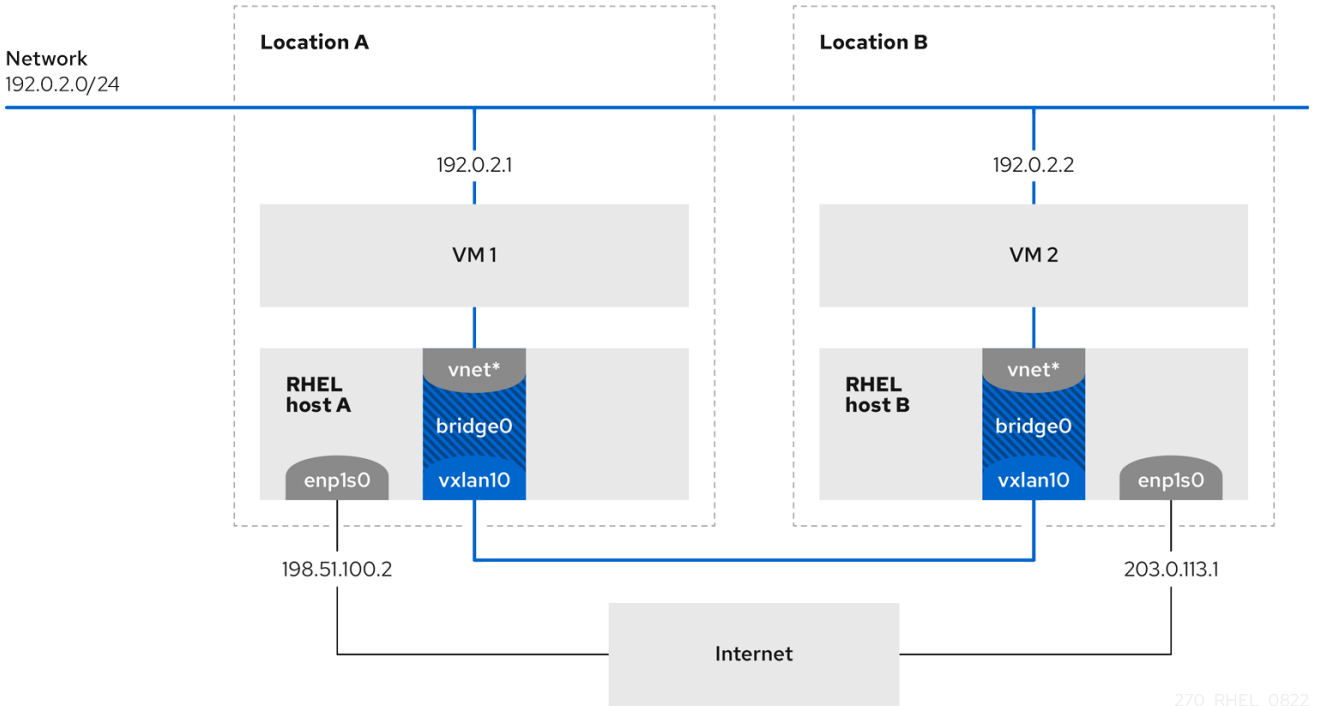
- [nmcli\(1\) 手册页](#)
- [nm-settings\(5\) 手册页](#)

9.4. 其他资源

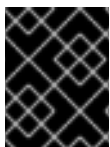
- [ip-link\(8\) 手册页](#)

第 10 章 使用 VXLAN 为虚拟机创建虚拟第 2 层域

虚拟可扩展局域网(VXLAN)是一种网络协议，它使用 UDP 协议在 IP 网络上对第 2 层流量进行隧道化。例如，在不同主机上运行的某些虚拟机(VM)可以通过 VXLAN 隧道进行通信。主机可以位于不同的子网中，甚至位于世界各地的不同数据中心。从虚拟机的角度来看，同一 VXLAN 中的其他虚拟机都在同一第 2 层域中：



在本例中，RHEL-host-A 和 RHEL-host-B 使用网桥 **br0** 来在每台具有 VXLAN 为 **vxlan10** 的主机上连接虚拟机的虚拟网络。由于此配置，VXLAN 对虚拟机不可见，虚拟机不需要任何特殊的配置。如果您稍后将更多的虚拟机连接到同一虚拟网络，则虚拟机将自动成为同一虚拟第 2 层域的成员。



重要

就像正常的第 2 层流量一样，VXLAN 中的数据是不加密的。出于安全考虑，在 VPN 或其他类型的加密连接上使用 VXLAN。

10.1. VXLAN 的优点

虚拟可扩展局域网(VXLAN)提供了以下主要优点：

- VXLAN 使用 24 位 ID。因此，您可以创建高达 16,777,216 个隔离网络。例如，虚拟 LAN(VLAN) 只支持 4,096 个隔离网络。
- VXLAN 使用 IP 协议。这可让您路由流量，并在同一第 2 层域中的不同网络和位置虚拟运行系统。
- 与大多数隧道协议不同，VXLAN 不仅仅是一个点对点的网络。VXLAN 可以动态了解其他端点的 IP 地址，也可以使用静态配置的转发条目。
- 某些网卡支持 UDP 隧道相关的卸载功能。

其他资源

- `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vxlan.rst` 由 `kernel-doc` 软件包提供

10.2. 在主机上配置以太网接口

要将 RHEL 虚拟机主机连接到以太网，请创建一个网络连接配置文件，配置 IP 设置，并激活配置文件。

在 RHEL 主机上运行此过程，并相应地调整 IP 地址配置。

先决条件

- 主机连接到以太网。

步骤

1. 在 NetworkManager 中添加新的以太网连接配置文件：

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

2. 配置 IPv4 设置：

```
# 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
```

如果网络使用 DHCP，请跳过这一步。

3. 激活 `Example` 连接：

```
# nmcli connection up Example
```

验证

1. 显示设备和连接的状态：

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

2. 在远程网络中 ping 主机以验证 IP 设置：

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

请注意，在该主机上配置网络前，您无法 ping 其他虚拟机主机。

其他资源

- `nm-settings(5)` 手册页

10.3. 创建附加了 VXLAN 的网桥

要使虚拟可扩展局域网(VXLAN)对虚拟机(VM)不可见，请在主机上创建一个网桥，并将 VXLAN 附加给网桥。使用 NetworkManager 创建网桥和 VXLAN。您不能将虚拟机的任何流量访问点(TAP)设备（通常在主机上称为 **vnet ***）添加到网桥。在虚拟机启动时，**libvirtd** 服务会动态添加它们。

在 RHEL 主机上运行此过程，并相应地调整 IP 地址。

步骤

1. 创建网桥 **br0** :

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

此命令在网桥设备上设置没有 IPv4 和 IPv6 地址，因为此网桥在第 2 层工作。

2. 创建 VXLAN 接口，并将其附加到 **br0** :

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

这个命令使用以下设置：

- **ID 10** : 设置 VXLAN 标识符。
- **local 198.51.100.2** : 设置传出数据包的源 IP 地址。
- **remote 203.0.113.1** : 当目的地链路层地址在 VXLAN 设备转发数据库中未知时，设置要在传出数据包中使用的单播或多播 IP 地址。
- **控制器 br0** : 将要创建的此 VXLAN 连接设置为 **br0** 连接中的端口。
- **ipv4.method disabled** 和 **ipv6.method disabled**: 在网桥上禁用 IPv4 和 IPv6。

默认情况下，NetworkManager 使用 **8472** 作为目的地端口。如果目的地端口不同，还要将 **destination-port <port_number>** 选项传给命令。

3. 激活 **br0** 连接配置文件：

```
# nmcli connection up br0
```

4. 在本地防火墙中为传入的 UDP 连接打开端口 **8472** :

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

验证

- 显示转发表：

```
# 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  
...
```


其他资源

- [nm-settings\(5\) 手册页](#)

10.4. 在带有现有网桥的 LIBVIRT 中创建一个虚拟网络

要使虚拟机(VM)使用带有附加虚拟可扩展局域网(VXLAN)的 **br0** 网桥，首先将虚拟网络添加到使用此网桥的 **libvirtd** 服务中。

先决条件

- 您已安装了 **libvirt** 软件包。
- 您已启动并启用了 **libvirtd** 服务。
- 您已在 RHEL 上配置了带有 VXLAN 的 **br0** 设备。

步骤

1. 使用以下内容创建 `~/vxlan10-bridge.xml` 文件：

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

2. 使用 `~/vxlan10-bridge.xml` 文件来在 **libvirt** 中创建一个新的虚拟网络：

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

3. 删除 `~/vxlan10-bridge.xml` 文件：

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

4. 启动 **vxlan10-bridge** 虚拟网络：

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

5. 将 **vxlan10-bridge** 虚拟网络配置为在 **libvirtd** 服务启动时自动启动：

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

验证

- 显示虚拟网络列表：

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

其他资源

- [virsh\(1\) 手册页](#)

10.5. 配置虚拟机以使用 VXLAN

要在主机上将虚拟机配置为使用带有附加虚拟可扩展 LAN(VXLAN)的网桥设备，请创建一个使用 `vxlan10-bridge` 虚拟网络的新虚拟机或更新现有虚拟机的设置以使用这个网络。

在 RHEL 主机上执行此流程。

先决条件

- 您在 `libvirtd` 中配置了 `vxlan10-bridge` 虚拟网络。

步骤

- 要创建新的虚拟机，并将其配置为使用 `vxlan10-bridge` 网络，请在创建虚拟机时将 `--network network:vxlan10-bridge` 选项传给 `virt-install` 命令：

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

- 要更改现有虚拟机的网络设置：

- 将虚拟机的网络接口连接到 `vxlan10-bridge` 虚拟网络：

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

- 关闭虚拟机，并重新启动它：

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

验证

- 显示主机上虚拟机的虚拟网络接口：

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

- 显示连接到 `vxlan10-bridge` 网桥的接口：

```
# 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
```

请注意，**libvirtd** 服务会动态更新网桥的配置。当您启动使用了 **vlan10-bridge** 网络的虚拟机时，主机上对应的 **vnet*** 设备会显示为网桥的端口。

3. 使用地址解析协议(ARP)请求来验证虚拟机是否在同一 VXLAN 中：
 - a. 启动同一 VXLAN 中的两个或多个虚拟机。
 - b. 将 ARP 请求从一个虚拟机发送到另一个虚拟机：

```
# 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))
```

如果命令显示回复，则虚拟机位于同一第 2 层域中，在这种情况下，是在同一 VXLAN 中。

安装 **iputils** 软件包以使用 **arping** 工具。

其他资源

- [virt-install\(1\) 手册页](#)
- [virt-xml\(1\) 手册页](#)
- [virsh\(1\) 手册页](#)
- [arping\(8\) 手册页](#)

第 11 章 管理 WIFI 连接

RHEL 提供多个实用程序和应用程序来配置和连接到 wifi 网络，例如：

- 使用 **nmcli** 工具，使用命令行配置连接。
- 使用 **nmtui** 应用程序在基于文本的用户界面中配置连接。
- 使用 GNOME 系统菜单快速连接到不需要任何配置的 wifi 网络。
- 使用 **GNOME Settings** 应用程序，使用 GNOME 应用程序配置连接。
- 使用 **nm-connection-editor** 应用程序在图形用户界面中配置连接。
- 使用 **network** RHEL 系统角色自动化一个或多个主机上连接的配置。

11.1. 支持的 WIFI 安全类型

根据安全类型网络支持，您可以更安全地传输数据。



警告

不要连接到不使用加密的网络，或者只支持不安全的 WEP 或 WPA 标准。

Red Hat Enterprise Linux 9 支持以下 wifi 安全类型：

- **None**：加密被禁用，数据在网络中以纯文本形式传输。
- **Enhanced Open**：使用投机无线加密（OWE），设备会协商唯一对主密钥 (PMK) 以在无线网络中加密连接，而无需身份验证。
- **LEAP**：由 Cisco 开发的轻量级可扩展验证协议，是可扩展身份验证协议 (EAP) 的专有版本。
- **WPA & WPA2 Personal**：在个人模式中，WPA) 和 Wi-Fi Protected Access 2 (WPA2) 验证方法使用预共享密钥。
- **WPA & WPA2 Enterprise**：在企业模式中，WPA 和 WPA2 使用 EAP 框架，并对用户进行身份验证以远程身份验证服务 (RADIUS) 服务器。
- **WPA3 Personal** - Wi-Fi Protected Access 3(WPA3) Personal 使用 Simultaneous Authentication of Equals (SAE) 而不是预共享密钥 (PSK) 来防止字典攻击。WPA3 使用完美转发保密 (PFS)。

11.2. 使用 NMCLI 连接到 WIFI 网络

您可以使用 **nmcli** 实用程序连接到 wifi 网络。当您第一次尝试连接到网络时，实用程序会自动为其创建一个 NetworkManager 连接配置集。如果网络需要额外的设置，如静态 IP 地址，您可以在它自动创建后修改配置集。

先决条件

- 在主机上安装了 wifi 设备。
- 如果存在硬件交换机，启用 wifi 设备。

流程

1. 如果网络管理器 (NetworkManager) 中禁用了 wifi radio，请启用此功能：

```
# nmcli radio wifi on
```

2. 可选：显示可用的 wifi 网络：

```
# 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
```

服务设置标识符 (**SSID**) 列包含网络的名称。如果列显示 --，则此网络的接入点不会广播 SSID。

3. 连接到 wifi 网络：

```
# nmcli device wifi connect Office --ask
Password: wifi-password
```

如果您要在命令中设置密码而不是以交互方式输入密码，请在命令中使用 **password *wifi-password*** 选项而不是 **--ask**：

```
# nmcli device wifi connect Office wifi-password
```

请注意，如果网络需要静态 IP 地址，NetworkManager 无法在此时激活连接。您可以在后续步骤中配置 IP 地址。

4. 如果网络需要静态 IP 地址：

- a. 配置 IPv4 地址设置，例如：

```
# 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. 配置 IPv6 地址设置，例如：

```
# 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. 重新激活连接：

```
# nmcli connection up Office
```

验证

1. 显示活跃连接：

```
# nmcli connection show --active
```

```
NAME ID TYPE DEVICE
Office 2501eb7e-7b16-4dc6-97ef-7cc460139a58 wifi wlp0s20f3
```

如果输出列出了您创建的 wifi 连接，则连接会活跃。

2. Ping 主机名或 IP 地址：

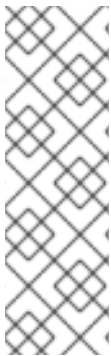
```
# ping -c 3 example.com
```

其他资源

- [nm-settings-nmcli\(5\) 手册页](#)

11.3. 使用 GNOME 系统菜单连接到 WIFI 网络

您可以使用 GNOME 系统菜单连接到 wifi 网络。当您第一次连接到网络时，GNOME 会为它创建一个 NetworkManager 连接配置集。如果您将连接配置集配置为不自动连接，也可以使用 GNOME 系统菜单使用现有 NetworkManager 连接配置集手动连接到 wifi 网络。



注意

第一次使用 GNOME 系统菜单建立与 wifi 网络的连接存在某种限制。例如，您无法配置 IP 地址设置。在这种情况下，首先配置连接：

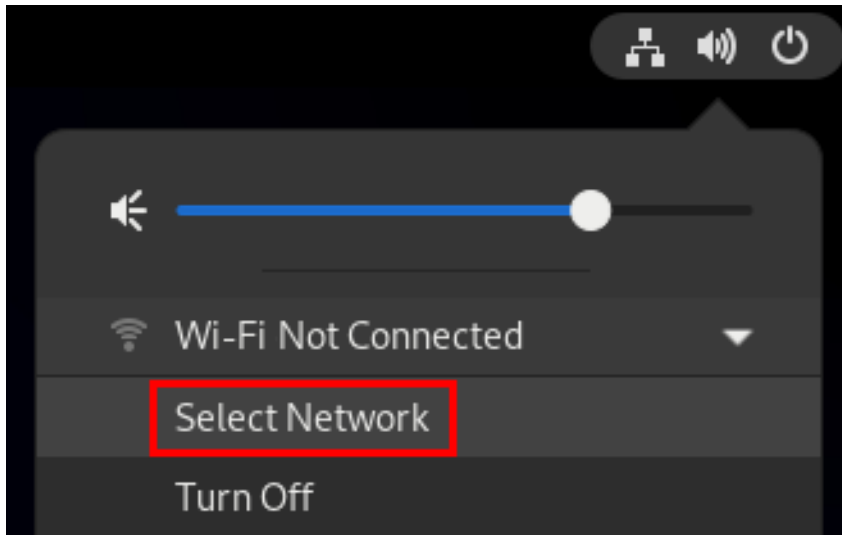
- 在 [GNOME settings](#) 应用程序中
- 在 [nm-connection-editor](#) 应用程序中
- 使用 `nmcli` 命令

前提条件

- 在主机上安装了 wifi 设备。
- 如果存在硬件交换机，启用 wifi 设备。

流程

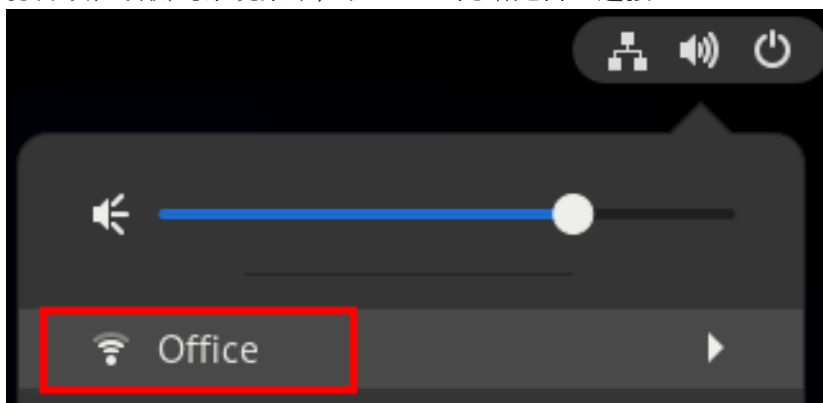
1. 打开顶栏右侧的系统菜单。
2. 展开 **Wi-Fi Not Connected** 条目。
3. 点 **Select Network:**



4. 选择您要连接到的 wifi 网络。
5. 点 **连接**。
6. 如果这是您第一次连接到这个网络，请输入网络的密码，然后点 **Connect**。

验证

1. 打开顶栏右侧的系统菜单，验证 wifi 网络是否已连接：



如果网络出现在列表中，它已被连接。

2. Ping 主机名或 IP 地址：

```
# ping -c 3 example.com
```

11.4. 使用 GNOME 设置应用程序连接到 WIFI 网络

您可以使用名为 **gnome-control-center** 的 **GNOME** 设置应用程序连接到 wifi 网络并配置连接。当您第一次连接到网络时，GNOME 会为它创建一个 NetworkManager 连接配置集。

在 **GNOME** 设置中，您可以为 RHEL 支持的所有 wifi 网络安全类型配置 wifi 连接。

前提条件

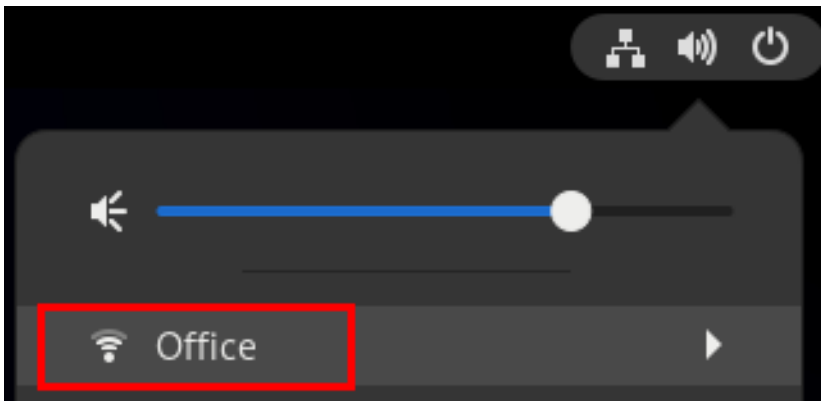
- 在主机上安装了 wifi 设备。
- 如果存在硬件交换机，启用 wifi 设备。

流程

1. 按 **Super** 键，键入 **Wi-Fi**，然后按 **Enter** 键。
2. 点您要连接的 wifi 网络的名称。
3. 输入网络的密码，点 **Connect**。
4. 如果网络需要额外的设置，如静态 IP 地址或 WPA2 个人以外的安全类型：
 - a. 点网络名称旁边的齿轮图标。
 - b. 可选：在 **Details** 标签页中配置网络配置集无法自动连接。
如果停用这个功能，则您必须总是手动连接到网络，例如使用 **GNOME settings** 或 GNOME 系统菜单。
 - c. 在 IPv4 选项卡上配置 **IPv4** 设置，并在 IPv6 选项卡上配置 **IPv6** 设置。
 - d. 在 **Security** 选项卡中，选择网络验证，如 **WPA3 Personal**，然后输入密码。
根据所选安全性，应用程序会显示其他字段。相应地填充它们。详情请参阅 wifi 网络管理员。
 - e. 点应用。

验证

1. 打开顶栏右侧的系统菜单，验证 wifi 网络是否已连接：



如果网络出现在列表中，它已被连接。

2. Ping 主机名或 IP 地址：

```
# ping -c 3 example.com
```

11.5. 使用 NMTUI 配置 WIFI 连接

nmtui 应用程序为 NetworkManager 提供了一个基于文本的用户界面。您可以使用 **nmtui** 连接到 wifi 网络。



注意

在 **nmtui** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

步骤

1. 如果您不知道要在连接中使用的网络设备名称，显示可用的设备：

```
# nmcli device status
DEVICE  TYPE   STATE      CONNECTION
wlp2s0  wifi   unavailable --
...
```

2. 启动 **nmtui**：

```
# nmtui
```

3. 选择 **Edit a connection**，点 **Enter**。
4. 按 **Add** 按钮。
5. 从网络类型列表中选择 **Wi-Fi**，然后按 **Enter**。
6. 可选：为要创建的 NetworkManager 配置文件输入一个名称。
在具有多个配置文件的主机上，有意义的名称可以更容易识别配置文件的用途。
7. 在 **Device** 字段中输入网络设备名称。
8. 在 **SSID** 字段中输入 Wi-Fi 网络的名称，即服务集标识符(SSID)。
9. 将 **Mode** 字段设为默认值，**Client**。
10. 选择 **Security** 字段，按 **Enter**，然后从列表中设置网络的验证类型。
根据您选择的验证类型，**nmtui** 会显示不同的字段。
11. 填写与验证类型相关的字段。
12. 如果 Wi-Fi 网络需要静态 IP 地址：
 - a. 按协议旁边的 **Automatic** 按钮，然后从显示的列表中选择 **Manual**。
 - b. 按您要配置的协议旁边的 **Show** 按钮，来显示其他字段并填写它们。
13. 按 **OK** 按钮来创建并自动激活新连接。

Edit Connection

Profile name
 Device

WI-FI <Hide>

SSID
 Mode

Security
 Password
 Show password

BSSID
 Cloned MAC address
 MTU

IPv4 CONFIGURATION <Show>
 IPv6 CONFIGURATION <Show>

Automatically connect
 Available to all users

<Cancel>

14. 按 **Back** 按钮返回到主菜单。
15. 选择 **Quit**，然后按 **Enter** 来关闭 `nmcli` 应用程序。

验证

1. 显示活跃连接：

```
# nmcli connection show --active
NAME ID TYPE DEVICE
Office 2501eb7e-7b16-4dc6-97ef-7cc460139a58 wifi wlp0s20f3
```

如果输出列出了您创建的 wifi 连接，则连接会活跃。

2. Ping 主机名或 IP 地址：

```
# ping -c 3 example.com
```

11.6. 使用 NM-CONNECTION-EDITOR 配置 WIFI 连接

您可以使用 `nm-connection-editor` 应用程序为无线网络创建连接配置集。在此应用程序中，您可以配置 RHEL 支持的所有 wifi 网络验证类型。

默认情况下，NetworkManager 为连接配置集启用自动连接功能，并在有可用时自动连接到保存的网络。

先决条件

- 在主机上安装了 wifi 设备。
- 如果存在硬件交换机，启用 wifi 设备。

步骤

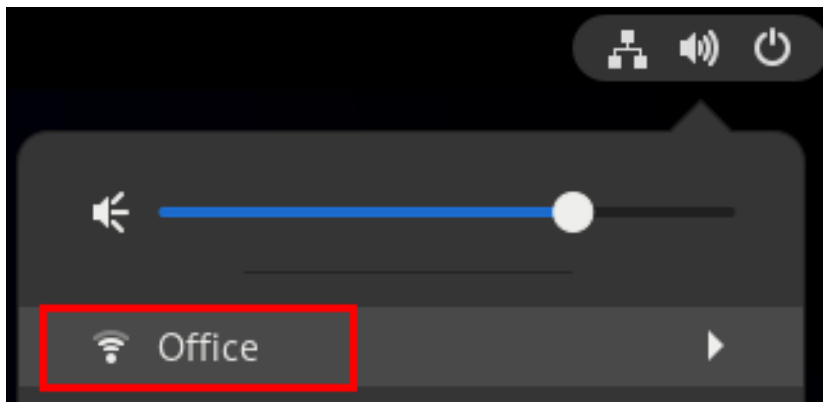
1. 打开终端窗口并输入：

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

2. 点击 **+** 按钮来添加一个新的连接。
3. 选择 **Wi-Fi** 连接类型，再点 **Create**。
4. 可选：为连接配置集设置名称。
5. 可选：在 **General** 选项卡中配置网络配置集无法自动连接。
如果停用这个功能，则您必须总是手动连接到网络，例如使用 **GNOME settings** 或 GNOME 系统菜单。
6. 在 **Wi-Fi** 选项卡中，在 **SSID** 字段中输入服务设置标识符 (SSID)。
7. 在 **Wi-Fi Security** 选项卡中，为网络选择身份验证类型，如 **WPA3 Personal**，然后输入密码。
根据所选安全性，应用程序会显示其他字段。相应地填充它们。详情请参阅 wifi 网络管理员。
8. 在 IPv4 选项卡上配置 **IPv4** 设置，并在 IPv6 选项卡上配置 **IPv6** 设置。
9. 点击 **Save**。
10. 关闭 **Network Connections** 窗口。

验证

1. 打开顶栏右侧的系统菜单，验证 wifi 网络是否已连接：



如果网络出现在列表中，它已被连接。

2. Ping 主机名或 IP 地址：

```
# ping -c 3 example.com
```

11.7. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的 WIFI 连接

使用 RHEL 系统角色，您可以自动化 wifi 连接的创建。例如，您可以使用 Ansible Playbook 为 **wlp1s0** 接口远程添加无线连接配置文件。创建的配置集使用 802.1X 标准将客户端验证到 Wi-Fi 网络。该 playbook 将连接配置集配置为使用 DHCP。要配置静态 IP 设置，相应地调整 **ip** 字典中的参数。

前提条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 网络支持 802.1X 网络身份验证。
- 您已在受管节点上安装了 **wpa_supplicant** 软件包。
- DHCP 位于受管节点的网络中。
- control 节点上存在 TLS 身份验证所需的以下文件：
 - 客户端密钥存储在 **/srv/data/client.key** 文件中。
 - 客户端证书存储在 **/srv/data/client.crt** 文件中。
 - CA 证书存储在 **/srv/data/ca.crt** 文件中。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure a wifi connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400

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

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

    - block:
      - ansible.builtin.import_role:
          name: rhel-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"

```

这些设置为 **wlp1s0** 接口定义一个 wifi 连接配置文件。该配置文件使用 802.1X 标准将客户端向 wifi 网络进行身份验证。连接从 DHCP 服务器和 IPv6 无状态地址自动配置(SLAAC)检索 IPv4 地址、IPv6 地址、默认网关、路由、DNS 服务器和搜索域。

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

11.8. 使用 NMCLI在现有配置集中配置带有 802.1X 网络身份验证的 WIFI 连接

使用 **nmcli** 工具，您可以将客户端配置为向网络进行身份验证。例如，您可以在名为 **wlp1s0** 的 NetworkManager wifi 连接配置文件中 使用 Microsoft Challenge-Handshake Authentication Protocol 版本 2 (MSCHAPv2)配置 Protected Extensible Authentication Protocol(PEAP)身份验证。

先决条件

- 网络必须具有 802.1X 网络身份验证。
- Wi-Fi 连接配置集存在于 NetworkManager 中，且具有有效的 IP 配置。
- 如果需要客户端验证验证方的证书，则证书颁发机构(CA)证书必须存储在 `/etc/pki/ca-trust/source/anchors/` 目录中。
- **wpa_supplicant** 软件包已安装。

流程

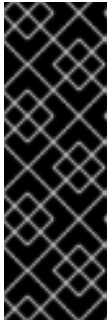
1. 将 Wi-Fi 安全模式设为 **wpa-eap**、将可扩展验证协议(EAP)设为 **peap**，将内部验证协议设为 **mschapv2**，和用户名：

```
# nmcli connection modify wlp1s0 wireless-security.key-mgmt wpa-eap 802-1x.eap
peap 802-1x.phase2-auth mschapv2 802-1x.identity user_name
```

请注意，您必须在单个命令中设置 **wireless-security.key-mgmt**、**802-1x.eap**、**802-1x.phase2-auth** 和 **802-1x.identity** 参数。

2. 另外，还可将该密码存储在配置中：

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



重要

默认情况下，NetworkManager 将密码以纯文本形式存储在 **/etc/sysconfig/network-scripts/keys-connection_name** 文件中，该文件只对 **root** 用户可读。但是，配置文件中的纯文本密码可能会造成安全风险。

要提高安全性，请将 **802-1x.password-flags** 参数设为 **0x1**。有了这个设置，在具有 GNOME 桌面环境或运行 **nm-applet** 的服务器上，NetworkManager 从这些服务中检索密码。在其他情况下，NetworkManager 会提示输入密码。

3. 如果客户端需要验证验证方的证书，请将连接配置文件中的 **802-1x.ca-cert** 参数设置为 CA 证书的路径：

```
# nmcli connection modify wlp1s0 802-1x.ca-cert /etc/pki/ca-trust/source/anchors/ca.crt
```



注意

为安全起见，客户端应验证验证器的证书。

4. 激活连接配置集：

```
# nmcli connection up wlp1s0
```

验证

- 访问需要网络身份验证的网络上的资源。

其他资源

- [管理 wifi 连接](#)
- [nm-settings\(5\) 手册页](#)
- [nmcli\(1\) 手册页](#)

11.9. 手动设置无线规范域

在 RHEL 上，**udev** 规则执行 **setregdomain** 工具来设置无线规范域。然后，实用程序为内核提供此信息。

默认情况下，**setregdomain** 会尝试自动决定国家代码。如果此操作失败，则无线规范域设置可能会出错。要临时解决这个问题，您可以手动设置国家代码。



重要

手动设置规范域将禁用自动检测。因此，如果您稍后在不同的国家/地区使用计算机，之前配置的设置可能不再正确。在这种情况下，删除 **/etc/sysconfig/regdomain** 文件以切回到自动检测，或使用此流程再次更新规范域设置。

流程

1. 可选：显示当前规范域设置：

```
# iw reg get
global
country US: DFS-FCC
...
```

2. 使用以下内容创建 **/etc/sysconfig/regdomain** 文件：

```
COUNTRY=<country_code>
```

将 **COUNTRY** 变量设置为一个 ISO 3166-1 alpha2 国家代码，如 **DE** 代表德国，**US** 代表美国。

3. 设置规范域：

```
# setregdomain
```

验证

- 显示规范域设置：

```
# iw reg get
global
country DE: DFS-ETSI
...
```

其他资源

- [setregdomain\(1\) 手册页](#)
- [iw\(8\) 手册页](#)
- [Specification.bin\(5\) 手册页](#)
- [ISO 3166 国家代码](#)

第 12 章 将 RHEL 配置为 WPA2 或 WPA3 个人访问令牌

在具有 wifi 设备的主机上，您可以使用 NetworkManager 将这个主机配置为接入点。Wi-Fi Protected Access 2 (WPA2)和 Wi-Fi Protected Access 3 (WPA3)提供安全验证方法，无线客户端可以使用预共享密钥(PSK)连接到访问点，并使用 RHEL 主机上和网络中的服务。

当您配置接入点时，NetworkManager 会自动：

- 配置 **dnsmasq** 服务来为客户端提供 DHCP 和 DNS 服务
- 启用 IP 转发
- 添加 **nftables** 防火墙规则来伪装来自 wifi 设备的流量并配置 IP 转发

前提条件

- wifi 设备支持在接入点模式下运行。
- wifi 设备没有使用。
- 主机可以访问互联网。

流程

1. 列出 wifi 设备来识别应提供访问点的 wifi 设备：

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

2. 验证该设备是否支持访问点模式：

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

要使用 wifi 设备作为接入点，该设备必须支持此功能。

3. 安装 **dnsmasq** 和 **NetworkManager-wifi** 软件包：

```
# dnf install dnsmasq NetworkManager-wifi
```

NetworkManager 使用 **dnsmasq** 服务为访问点的客户端提供 DHCP 和 DNS 服务。

4. 创建初始访问点配置：

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

此命令在 **wlp0s20f3** 设备（提供 WPA2 和 WPA3 个人访问令牌）上创建一个连接配置集。无线网络名称，Service Set Identifier (SSID) 是 **Example-Hotspot**，并使用预共享密钥 **密码**。

5. 可选：将访问点配置为只支持 WPA3:

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


6. 默认情况下，NetworkManager 使用 IP 地址 **10.42.0.1** 作为 wifi 设备，并将剩余的 **10.42.0.0/24** 子网的 IP 地址分配给客户端。要配置不同的子网和 IP 地址，请输入：

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

您设置的 IP 地址（本例中为 **192.0.2.254**）是 NetworkManager 分配给 wifi 设备的 IP 地址。客户端将此 IP 地址用作默认网关和 DNS 服务器。

7. 激活连接配置集：

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

验证

1. 在服务器中：

- a. 验证 NetworkManager 是否启动了 **dnsmasq** 服务，并且服务侦听端口 67 (DHCP) 和 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. 显示 **nftables** 规则集，以确保 NetworkManager 为来自 **10.42.0.0/24** 子网的流量启用转发和伪装：

```
# 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. 在带有 wifi 适配器的客户端中：

- a. 显示可用网络列表：

```
# 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. 连接到 **Example-Hotspot** 无线网络。请参阅[管理 Wi-Fi 连接](#)。
- c. 对远程网络或互联网中的主机发出 ping 以验证连接是否正常工作：

```
# ping -c 3 www.redhat.com
```

其他资源

- [nm-settings\(5\)](#) 手册页

第 13 章 使用 MACSEC 加密同一物理网络中的第 2 层流量

您可以使用 MACsec 来保护两个设备（点到点）之间的通信。例如，您的分支办公室通过城际以太网与中心办公室连接，您可以在连接办公室的两个主机上配置 MACsec，以提高安全性。

介质访问控制安全(MACsec)是一种第 2 层的协议，用于保护以太网链路上的不同的流量类型，包括：

- 动态主机配置协议(DHCP)
- 地址解析协议(ARP)
- 互联网协议版本 4 / 6(IPv4 / IPv6)以及
- 任何使用 IP 的流量（如 TCP 或 UDP）

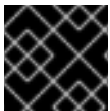
MACsec 默认使用 GCM-AES-128 算法加密并验证 LAN 中的所有流量，并使用预共享密钥在参与的主机之间建立连接。如果要更改预共享密钥，您需要更新网络中使用 MACsec 的所有主机上的 NM 配置。

MACsec 连接将以太网设备（如以太网网卡、VLAN 或隧道设备）用作父设备。您可以只在 MACsec 设备上设置 IP 配置，以便只使用加密连接与其他主机进行通信，也可以在父设备上设置 IP 配置。在后者的情况下，您可以使用父设备使用未加密连接和 MACsec 设备加密连接与其他主机通信。

macsec 不需要任何特殊硬件。例如，您可以使用任何交换机，除非您只想在主机和交换机之间加密流量。在这种情况下，交换机还必须支持 MACsec。

换句话说，有 2 种常用的方法来配置 MACsec：

- 主机到主机，和
- 主机到交换机，然后交换机到其他主机



重要

您只能在相同（物理或虚拟）LAN 的主机间使用 MACsec。

13.1. 使用 NMCLI 配置 MACSEC 连接

您可以使用 **nmcli** 工具将以太网接口配置为使用 MACsec。例如，您可以在通过以太网连接的两个主机之间创建一个 MACsec 连接。

步骤

1. 在配置 MACsec 的第一个主机上：

- 为预共享密钥创建连接关联密钥(CAK)和连接关联密钥名称(CKN)：
 - a. 创建一个 16 字节的十六进制 CAK：

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

- b. 创建一个 32 字节的十六进制 CKN：

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. 在您要通过 MACsec 连接连接的两个主机上：

3. 创建 MACsec 连接：

```
# 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
```

在 `macsec.mka-cak` 和 `macsec.mka-ckn` 参数中使用上一步生成的 CAK 和 CKN。在 MACsec-protected 网络的每个主机上，这些值必须相同。

4. 配置 MACsec 连接中的 IP 设置。

a. 配置 IPv4 设置。例如，要为 `macsec0` 连接设置静态 IPv4 地址、网络掩码、默认网关和 DNS 服务器，请输入：

```
# 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. 配置 IPv6 设置。例如，要为 `macsec0` 连接设置静态 IPv6 地址、网络掩码、默认网关和 DNS 服务器，请输入：

```
# 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. 激活连接：

```
# nmcli connection up macsec0
```

验证

1. 验证流量是否加密：

```
# tcpdump -nn -i enp1s0
```

2. 可选：显示未加密的流量：

```
# tcpdump -nn -i macsec0
```

3. 显示 MACsec 统计信息：

```
# ip macsec show
```

4. 显示每种保护类型的单独的计数器：仅完整性（关闭加密）和加密（打开加密）

```
# ip -s macsec show
```

13.2. 使用 NMSTATECTL 配置 MACSEC 连接

您可以通过 `nmstatectl` 工具，以声明的方式将以太网接口配置为使用 MACsec。例如，在 YAML 文件中，您可以描述网络所需的状态，假设其在通过以太网连接的两个主机之间有一个 MACsec 连接。`nmstatectl` 工具解释 YAML 文件，并在主机之间部署持久和一致的网络配置。

使用 MACsec 安全标准保护链路层的通信，也称为 Open Systems Interconnection (OSI) 模型的第 2 层，提供以下显著优点：

- 第 2 层的加密消除了在第 7 层加密单个服务的需要。这减少了管理与每个主机上每个端点的大量证书关联的开销。
- 直接连接的网络设备（如路由器和交换机）之间的点对点安全性。
- 不需要对应用程序和高层协议进行更改。

先决条件

- 服务器配置中存在一个物理或虚拟以太网网络接口控制器(NIC)。
- `nmstate` 软件包已安装。

步骤

1. 在您配置 MACsec 的第一个主机上，为预共享密钥创建连接关联密钥(CAK)和连接关联密钥名称(CKN)：

- a. 创建一个 16 字节的十六进制 CAK：

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

- b. 创建一个 32 字节的十六进制 CKN：

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. 在您要通过 MACsec 连接连接的两个主机上，完成以下步骤：

- a. 使用以下设置创建一个 YAML 文件，如 `create-macsec-connection.yml`：

```
---
routes:
  config:
    - destination: 0.0.0.0/0
      next-hop-interface: macsec0
      next-hop-address: 192.0.2.2
      table-id: 254
    - destination: 192.0.2.2/32
      next-hop-interface: macsec0
      next-hop-address: 0.0.0.0
      table-id: 254
dns-resolver:
  config:
    search:
      - example.com
    server:
      - 192.0.2.200
```

```

- 2001:db8:1::ffbb
interfaces:
- name: macsec0
  type: macsec
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 32
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
  macsec:
    encrypt: true
    base-iface: enp0s1
    mka-cak: 50b71a8ef0bd5751ea76de6d6c98c03a
    mka-ckn: f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fc1c4550
    port: 0
    validation: strict
    send-sci: true

```

- b. 在 **mka-cak** 和 **mka-ckn** 参数中使用上一步中生成的 CAK 和 CKN。在 MACsec-protected 网络的每个主机上，这些值必须相同。
- c. 可选：在同一 YAML 配置文件中，您还可以配置以下设置：
 - 静态 IPv4 地址 - **192.0.2.1**，子网掩码为 **/32**
 - 静态 IPv6 地址 - **2001:db8:1::1**，子网掩码为 **/64**
 - IPv4 默认网关 - **192.0.2.2**
 - IPv4 DNS 服务器 - **192.0.2.200**
 - IPv6 DNS 服务器 - **2001:db8:1::ffbb**
 - DNS 搜索域 - **example.com**

3. 将设置应用到系统：

```
# nmstatectl apply create-macsec-connection.yml
```

验证

1. 以 YAML 格式显示当前状态：

```
# **nmstatectl show macsec0
```

2. 验证流量是否加密：

```
# tcpdump -nn -i enp0s1
```

3. 可选：显示未加密的流量：

```
# tcpdump -nn -i macsec0
```

4. 显示 MACsec 统计信息：

```
# ip macsec show
```

5. 显示每种保护类型的单独的计数器：仅完整性（关闭加密）和加密（打开加密）

```
# ip -s macsec show
```

其他资源

- [MACsec：加密网络流量的一个不同的解决方案](#)

13.3. 其他资源

- [MACsec：加密网络流量的另一种解决方案](#) 博客。

第 14 章 开始使用 IPVLAN

IPVLAN 是虚拟网络设备的驱动程序，可在容器环境中用于访问主机网络。IPVLAN 会将一个 MAC 地址公开给外部网络，而不管主机网络中所创建的 IPVLAN 设备的数量。这意味着，用户可以在多个容器中有多个 IPVLAN 设备，相应的交换机会读取单个 MAC 地址。当本地交换机对它可管理的 MAC 地址的总数施加约束时，IPVLAN 驱动程序很有用。

14.1. IPVLAN 模式

IPVLAN 有以下模式可用：

- L2 模式**
 在 IPVLAN **L2 模式** 中，虚拟设备接收并响应地址解析协议(ARP)请求。**netfilter** 框架仅在拥有虚拟设备的容器中运行。容器化流量的默认命名空间中不会执行 **netfilter** 链。使用 **L2 模式** 会提供良好的性能，但对网络流量的控制要小。
- L3 模式**
 在 **L3 模式** 中，虚拟设备只处理 **L3** 以上的流量。虚拟设备不响应 ARP 请求，用户必须手动为相关点上的 IPVLAN IP 地址配置邻居条目。相关容器的出口流量位于默认命名空间中的 **netfilter** POSTROUTING 和 OUTPUT 链上，而入口流量以与 **L2 模式** 相同的方式被线程化。使用 **L3 模式** 会提供很好的控制，但可能会降低网络流量性能。
- L3S 模式**
 在 **L3S 模式** 中，虚拟设备处理方式与 **L3 模式** 中的处理方式相同，但相关容器的出口和入口流量都位于默认命名空间中的 **netfilter** 链上。**L3S 模式** 的行为方式和 **L3 模式** 相似，但提供了对网络的更大控制。



注意

对于 **L3** 和 **L3S 模式**，IPVLAN 虚拟设备不接收广播和多播流量。

14.2. IPVLAN 和 MACVLAN 的比较

下表显示了 MACVLAN 和 IPVLAN 之间的主要区别：

MACVLAN	IPVLAN
为每个 MACVLAN 设备使用 MAC 地址。 请注意，如果交换机达到其可以在 MAC 表中存储的最大 MAC 地址数，则连接会丢失。	使用不限制 IPVLAN 设备数的单个 MAC 地址。
全局命名空间的 Netfilter 规则不会影响到子命名空间中到或来自 MACVLAN 设备的流量。	可以在 L3 mode 和 L3S mode 下控制到或来自 IPVLAN 设备的流量。

IPVLAN 和 MACVLAN 不需要任何级别的封装。

14.3. 使用 IPROUTE2 创建和配置 IPVLAN 设备

此流程演示了如何使用 **iproute2** 设置 IPVLAN 设备。

步骤

1. 要创建 IPVLAN 设备，请输入以下命令：

```
# ip link add link real_NIC_device name IPVLAN_device type ipvlan mode l2
```

请注意：网络接口控制器（NIC）是将计算机连接到网络的一个硬件组件。

例 14.1. 创建 IPVLAN 设备

```
# ip link add link enp0s31f6 name my_ipvlan type ipvlan mode l2
# 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. 要给接口分配 IPv4 或 IPv6 地址，请输入以下命令：

```
# ip addr add dev IPVLAN_device IP_address/subnet_mask_prefix
```

3. 如果在 L3 模式或 L3S 模式中配置 IPVLAN 设备，请进行以下设置：

- a. 在远程主机上为远程 peer 配置邻居设置：

```
# ip neigh add dev peer_device IPVLAN_device_IP_address lladdr MAC_address
```

其中 *MAC_address* 是 IPVLAN 设备所基于的实际网卡的 MAC 地址。

- b. 使用以下命令为 L3 模式配置 IPVLAN 设备：

```
# ip route add dev <real_NIC_device> <peer_IP_address/32>
```

对于 L3S 模式：

```
# ip route add dev real_NIC_device peer_IP_address/32
```

其中 IP-address 代表远程 peer 的地址。

4. 要设置活跃的 IPVLAN 设备，请输入以下命令：

```
# ip link set dev IPVLAN_device up
```

5. 要检查 IPVLAN 设备是否活跃，请在远程主机中执行以下命令：

```
# ping IP_address
```

其中 *IP_address* 使用 IPVLAN 设备的 IP 地址。

第 15 章 配置 NETWORKMANAGER 以忽略某些设备

默认情况下，NetworkManager 管理所有设备。要忽略某些设备，您可以通过将 NetworkManager 设置为 **unmanaged**。

15.1. 使用 NMCLI 配置回环接口

默认情况下，NetworkManager 不管理回环(lo)接口。为 lo 接口创建连接配置文件后，您可以使用 NetworkManager 配置这个设备。一些示例如下：

- 为 lo 接口分配额外的 IP 地址
- 定义 DNS 地址
- 更改 lo 接口的最大传输单元(MTU)大小

步骤

1. 创建一个新的类型 **loopback**：

```
# nmcli connection add con-name example-loopback type loopback
```

2. 配置自定义连接设置，例如：

- a. 要为接口分配额外的 IP 地址，请输入：

```
# nmcli connection modify example-loopback +ipv4.addresses 192.0.2.1/24
```



注意

NetworkManager 通过始终分配重启后仍保留的 IP 地址 **127.0.0.1** 和 **::1** 来管理 lo 接口。您不能覆盖 **127.0.0.1** 和 **::1**。但是，您可以为接口分配额外的 IP 地址。

- b. 要设置自定义最大传输单元(MTU)，请输入：

```
# nmcli con mod example-loopback loopback.mtu 16384
```

- c. 要为您的 DNS 服务器设置 IP 地址，请输入：

```
# nmcli connection modify example-loopback ipv4.dns 192.0.2.0
```

如果您在 loopback 连接配置文件中设置了一个 DNS 服务器，则此条目总是在 **/etc/resolv.conf** 文件中。DNS 服务器条目保持独立，无论主机是否在不同网络之间漫游。

3. 激活连接：

```
# nmcli connection up example-loopback
```

验证

1. 显示 lo 接口的设置：

ip address show lo

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16384 qdisc noqueue state UNKNOWN group
default qlen 1000
```

```
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft
forever preferred_lft forever inet 192.0.2.1/24 brd 192.0.2.255 scope global lo valid_lft forever
preferred_lft forever
```

```
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
```

2. 验证 DNS 地址：

cat /etc/resolv.conf

```
...
nameserver 192.0.2.0
...
```

15.2. 永久将设备配置为网络管理器（NETWORKMANAGER）中非受管设备

您可以根据几个标准,如接口名称、MAC 地址或设备类型,将设备永久配置为 **unmanaged**。

要临时将网络设备配置为 **unmanaged**，请参阅 [在 NetworkManager 中临时将设备配置为非受管](#)。

步骤

1. 可选：显示要识别设备的设备列表或您要设置为 **unmanaged** 的设备的 MAC 地址：**# 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. 使用以下内容创建 **/etc/NetworkManager/conf.d/99-unmanaged-devices.conf** 文件：

- 要将特定接口配置为 unmanaged，请添加：

```
[keyfile]
unmanaged-devices=interface-name:enp1s0
```

- 要将具有特定 MAC 地址的设备配置为 unmanaged，请添加：

```
[keyfile]
unmanaged-devices=mac:52:54:00:74:79:56
```

- 要将特定类型的所有设备配置为 unmanaged，请添加：

```
[keyfile]
unmanaged-devices=type:ethernet
```

- 要将多个设备设置为非受管设备，请在 `unmanaged-devices` 参数中使用分号来分隔条目，例如：

```
[keyfile]
unmanaged-devices=interface-name:enp1s0;interface-name:enp7s0
```

3. 重新载入 `NetworkManager` 服务：

```
# systemctl reload NetworkManager
```

验证

- 显示设备列表：

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

`enp1s0` 设备旁边的 `unmanaged` 状态表示 `NetworkManager` 没有管理该设备。

故障排除

- 如果该设备没有显示为 `unmanaged`，则显示 `NetworkManager` 配置：

```
# NetworkManager --print-config
...
[keyfile]
unmanaged-devices=interface-name:enp1s0
...
```

如果输出与您配置的设置不匹配，请确保您的设置没有被优先级较高的配置文件所覆盖。有关 `NetworkManager` 如何合并多个配置文件的详情，请查看 [NetworkManager.conf \(5\)](#) 手册页。

15.3. 将设备临时配置为在 NETWORKMANAGER 中不被管理

您可以临时将设备配置为 `unmanaged`。

可以使用这个方法用于特定目的，如测试。要永久将网络设备配置为 `unmanaged`，请参阅 [在 NetworkManager 中永久将设备配置为非受管](#)。

步骤

1. 可选：显示设备列表，以便识别您要将其设置为 `unmanaged` 的设备：

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

2. 将 `enp1s0` 设备设置为 `unmanaged` 状态：

```
# nmcli device set enp1s0 managed no
```

验证

- 显示设备列表：

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

enp1s0 设备旁边的 **unmanaged** 状态表示 NetworkManager 没有管理该设备。

其他资源

- [NetworkManager.conf\(5\)](#) man page

第 16 章 创建 DUMMY 接口

作为 Red Hat Enterprise Linux 用户，您可以创建并使用 dummy 网络接口进行调试和测试。dummy 接口提供了一个设备来路由数据包而无需实际传送数据包。它可让您创建使用网络管理器（NetworkManager）管理的其他回送设备，使不活跃 SLIP（Serial Line Internet Protocol）地址类似本地程序的地址。

16.1. 使用 nmcli 使用 IPV4 和 IPV6 地址创建 DUMMY 接口

您可以创建一个具有各种设置的虚拟接口，如 IPv4 和 IPv6 地址。创建接口后，NetworkManager 会自动将其分配给默认的 **public firewalld** 区域。

流程

- 创建一个具有静态 IPv4 和 IPv6 地址的名为 **dummy0** 的伪接口：

```
# 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
```



注意

要配置没有 IPv4 和 IPv6 地址的伪接口，请将 **ipv4.method** 和 **ipv6.method** 参数设为 **disabled**。否则，IP 自动配置会失败，NetworkManager 会停用连接，并删除该设备。

验证

- 列出连接配置文件：

```
# nmcli connection show
NAME          UUID                                TYPE  DEVICE
dummy-dummy0  aaf6eb56-73e5-4746-9037-eed42caa8a65  dummy  dummy0
```

其它资源

- [nm-settings\(5\) 手册页](#)

第 17 章 使用 NETWORKMANAGER 为特定连接禁用 IPV6

在使用 NetworkManager 来管理网络接口的系统上，如果网络只使用 IPv4，您可以禁用 IPv6 协议。如果您禁用了 **IPv6**，NetworkManager 会自动在内核中设置相应的 **sysctl** 值。



注意

如果使用内核可调参数或内核引导参数禁用 IPv6，则必须额外考虑系统配置。如需更多信息，请参阅文章 [如何在 RHEL 中禁用或启用 IPv6 协议？](#)

17.1. 使用 NMCLI 在连接中禁用 IPV6

您可以使用 **nmcli** 工具在命令行上禁用 **IPv6** 协议。

先决条件

- 系统使用 NetworkManager 来管理网络接口。

步骤

1. 另外，还可显示网络连接列表：

```
# nmcli connection show
NAME UUID TYPE DEVICE
Example 7a7e0151-9c18-4e6f-89ee-65bb2d64d365 ethernet enp1s0
...
```

2. 将连接的 **ipv6.method** 参数设为 **disabled**：

```
# nmcli connection modify Example ipv6.method "disabled"
```

3. 重启网络连接：

```
# nmcli connection up Example
```

验证

1. 显示设备的 IP 设置：

```
# 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
```

如果没有显示 **inet6** 条目，则 **IPv6** 在该设备上被禁用。

2. 验证 `/proc/sys/net/ipv6/conf/enp1s0/disable_ipv6` 文件现在是否包含值 **1**：

```
# cat /proc/sys/net/ipv6/conf/enp1s0/disable_ipv6
1
```

值 **1** 表示针对该设备禁用 **IPv6**。

第 18 章 更改主机名

系统的主机名是系统本身的名称。您可在安装 RHEL 时设置名称，之后可以更改它。

18.1. 使用 NMCLI 更改主机名

您可以使用 **nmcli** 工具更新系统主机名。请注意，其他工具可能会使用不同的术语，如静态或持久主机名。

步骤

1. 可选：显示当前主机名设置：

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

2. 设置新主机名：

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

3. NetworkManager 自动重启 **systemd-hostnamed** 来激活新主机名。要使更改生效，请重启主机：

```
# reboot
```

或者，如果您知道哪个服务使用主机名：

- a. 重启在服务启动时仅读取主机名的所有服务：

```
# systemctl restart <service_name>
```

- b. 活跃的 shell 用户必须重新登录才能使更改生效。

验证

- 显示主机名：

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

18.2. 使用 HOSTNAMECTL 更改主机名

您可以使用 **hostnamectl** 工具更新主机名。默认情况下，这个工具设置以下主机名类型：

- 静态主机名：存储在 **/etc/hostname** 文件中。通常，服务使用此名称作为主机名。
- 用户友善的主机名：一个描述性名称，如 **数据中心 A 中的代理服务器**。
- 临时主机名：通常从网络配置接收的回退值。

步骤

1. 可选：显示当前主机名设置：

```
# hostnamectl status --static  
old-hostname.example.com
```

2. 设置新主机名：

```
# hostnamectl set-hostname new-hostname.example.com
```

这个命令将静态、用户友善和临时主机名设为新值。要只设置一个特定类型，请将 **--static**、**--pretty** 或 **--transient** 选项传给命令。

3. **hostnamectl** 工具自动重启 **systemd-hostnamed** 来激活新主机名。要使更改生效，请重启主机：

```
# reboot
```

或者，如果您知道哪个服务使用主机名：

- a. 重启在服务启动时仅读取主机名的所有服务：

```
# systemctl restart <service_name>
```

- b. 活跃的 shell 用户必须重新登录才能使更改生效。

验证

- 显示主机名：

```
# hostnamectl status --static  
new-hostname.example.com
```

其他资源

- [hostnamectl\(1\)](#)
- [systemd-hostnamed.service\(8\)](#)

第 19 章 配置 NETWORKMANAGER DHCP 设置

NetworkManager 提供与 DHCP 相关的不同配置选项。例如，您可以将 NetworkManager 配置为使用内置 DHCP 客户端（默认）或外部客户端，您可以影响单个配置文件的 DHCP 设置。

19.1. 更改 NETWORKMANAGER 的 DHCP 客户端

默认情况下，NetworkManager 使用其内部的 DHCP 客户端。但是，如果您需要不提供内置客户端的 DHCP 客户端，您也可以将 NetworkManager 配置为使用 **dhclient**。

请注意，RHEL 不提供 **dhcpcd**，因此 NetworkManager 无法使用这个客户端。

步骤

1. 使用以下内容创建 **/etc/NetworkManager/conf.d/dhcp-client.conf** 文件：

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

您可以对 **internal**（默认）或 **dhclient** 设置 **dhcp** 参数。

2. 如果对 **dhclient** 设置 **dhcp** 参数，请安装 **dhcp-client** 软件包：

```
# dnf install dhcp-client
```

3. 重启 NetworkManager：

```
# systemctl restart NetworkManager
```

请注意，重启会临时中断所有网络连接。

验证

- 在 **/var/log/messages** 日志文件中搜索类似于如下的条目：

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

此日志条目确认 NetworkManager 使用 **dhclient** 作为 DHCP 客户端。

其他资源

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

19.2. 配置 NETWORKMANAGER 连接的 DHCP 行为

DHCP 客户端在每次连接到网络时都从 DHCP 服务器请求动态 IP 地址和对应配置信息。

当您连接配置为从 DHCP 服务器检索 IP 地址时，网络管理器（NetworkManager）从 DHCP 服务器请求 IP 地址。默认情况下，客户端会等待 45 秒时间完成此请求。当 **DHCP** 连接启动时，**dhcp** 客户端会从 **DHCP** 服务器请求 IP 地址。

先决条件

- 在主机上配置了使用 DHCP 的连接。

流程

1. 设置 `ipv4.dhcp-timeout` 和 `ipv6.dhcp-timeout` 属性。例如，要将这两个选项都设为 30 秒，请输入：

```
# nmcli connection modify <connection_name> ipv4.dhcp-timeout 30 ipv6.dhcp-timeout 30
```

另外，将参数设置为 **infinity** 以配置网络管理器(NetworkManager)不会停止尝试请求和续订 IP 地址，直到成功为止。

2. 可选：配置如果网络管理器（NetworkManager）在超时前没有接收 IPv4 地址时的行为：

```
# nmcli connection modify <connection_name> ipv4.may-fail <value>
```

如果将 `ipv4.may-fail` 选项设为：

- **yes**，连接的状态取决于 IPv6 配置：
 - 如果启用了 IPv6 配置并成功，NetworkManager 会激活 IPv6 连接，不再尝试激活 IPv4 连接。
 - 如果禁用或未配置 IPv6 配置，连接会失败。
 - **no**，连接会被停止。在这种情况下：
 - 如果启用了连接的 **autoconnect** 属性，NetworkManager 会根据 **autoconnect-retries** 属性中设置的值尝试多次激活连接。默认值为 **4**。
 - 如果连接仍然无法获得 DHCP 地址，则自动激活会失败。请注意，5 分钟后，自动连接过程会再次启动，从 DHCP 服务器获取 IP 地址。
3. 可选：配置如果网络管理器（NetworkManager）在超时前没有接收 IPv6 地址时的行为：

```
# nmcli connection modify <connection_name> ipv6.may-fail <value>
```

其它资源

- `nm-settings(5)` 手册页

第 20 章 使用 NETWORKMANAGER 的一个分配程序脚本运行 HCLIENT EXIT HOOKS

您可以使用 NetworkManager 分配程序脚本来执行 **dhclient** exit hooks。

20.1. NETWORKMANAGER 分配程序脚本的概念

在发生网络事件时，**NetworkManager-dispatcher** 服务会按字母顺序执行用户提供的脚本。这些脚本通常是 shell 脚本，但可以是任何可执行的脚本或应用程序。例如，您可以使用分配程序脚本来调整您无法使用 NetworkManager 进行管理的与网络相关的设置。

您可以在以下目录中存储分配程序脚本：

- **/etc/NetworkManager/dispatcher.d/**：root 用户可以编辑的分配程序脚本的通用位置。
- **/usr/lib/NetworkManager/dispatcher.d/**：用于预先部署的不可变分配程序脚本。

为了安全起见，**NetworkManager-dispatcher** 服务只有在满足以下条件时才执行脚本：

- 脚本归 **root** 用户所有。
- 该脚本仅可由 **root** 读写。
- 脚本上没有设置 **setuid** 位。

NetworkManager-dispatcher 服务使用两个参数运行每个脚本：

1. 操作所在的设备的接口名称。
2. 当接口被激活时，如 **up** 操作。

NetworkManager(8) 手册页中的 **分配程序脚本** 部分提供了在脚本中可以使用的操作和环境变量的概述。

NetworkManager-dispatcher 服务一次运行一个脚本，但与主 NetworkManager 进程异步运行。请注意，如果脚本已排队，服务将始终运行它，即使后续事件使其过时。但是，**NetworkManager-dispatcher** 服务运行脚本，它们是指向 **/etc/NetworkManager/dispatcher.d/no-wait.d/** 中的文件的符号链接，而无需等待之前脚本的终止，且并行运行。

其他资源

- **NetworkManager(8)** 手册页

20.2. 创建运行 DHCLIENT EXIT HOOKS 的 NETWORKMANAGER 分配程序脚本

当 DHCP 服务器分配或更新 IPv4 地址时，NetworkManager 可以运行存储在 **/etc/dhcp/dhclient-exit-hooks.d/** 目录中的分配程序脚本。然后，该分配程序脚本可以运行 **dhclient** 退出钩子。

先决条件

- **dhclient** exit hooks 存储在 **/etc/dhcp/dhclient-exit-hooks.d/** 目录下。

步骤

1. 使用以下内容创建 `/etc/NetworkManager/dispatcher.d/12-dhclient-down` 文件：

```
#!/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. 将 `root` 用户设为文件的所有者：

```
# chown root:root /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

3. 设置权限，以便只有 `root` 用户才能执行它：

```
# chmod 0700 /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

4. 恢复 SELinux 上下文：

```
# restorecon /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

其他资源

- [NetworkManager\(8\) 手册页](#)

第 21 章 手动配置 /ETC/RESOLV.CONF 文件

默认情况下，NetworkManager 使用活跃的 NetworkManager 连接配置文件中的 DNS 设置动态更新 `/etc/resolv.conf` 文件。但是，您可以在 `/etc/resolv.conf` 中禁用此行为，并手动配置 DNS 设置。



注意

或者，如果您在 `/etc/resolv.conf` 中需要特定的 DNS 服务器顺序，请参阅 [配置 DNS 服务器的顺序](#)。

21.1. 在 NETWORKMANAGER 配置中禁用 DNS 处理

默认情况下，NetworkManager 在 `/etc/resolv.conf` 文件中管理 DNS 设置，您可以配置 DNS 服务器的顺序。或者，如果您喜欢在 `/etc/resolv.conf` 中手动配置 DNS 设置，您可以在 NetworkManager 中禁用 DNS 处理。

流程

1. 以 root 用户身份，使用文本编辑器创建包含以下内容的 `/etc/NetworkManager/conf.d/90-dns-none.conf` 文件：

```
[main]
dns=none
```

2. 重新载入 NetworkManager 服务：

```
# systemctl reload NetworkManager
```



注意

重新加载服务后，NetworkManager 不再更新 `/etc/resolv.conf` 文件。但是该文件的最后内容将被保留。

3. (可选) 从 `/etc/resolv.conf` 中删除 NetworkManager 生成的注释，以避免混淆。

验证

1. 编辑 `/etc/resolv.conf` 文件并手动更新配置。
2. 重新载入 NetworkManager 服务：

```
# systemctl reload NetworkManager
```

3. 显示 `/etc/resolv.conf` 文件：

```
# cat /etc/resolv.conf
```

如果您成功禁用了 DNS 处理，NetworkManager 不会覆盖手动配置的设置。

故障排除

- 显示 NetworkManager 配置，以确保没有具有高优先级的配置文件覆盖设置：

```
# NetworkManager --print-config
...
dns=none
...
```

其他资源

- [NetworkManager.conf\(5\) man page](#)
- [使用 NetworkManager 配置 DNS 服务器的顺序](#)

21.2. 使用符号链接替换 /ETC/RESOLV.CONF 来手动配置 DNS 设置

默认情况下，NetworkManager 在 `/etc/resolv.conf` 文件中管理 DNS 设置，您可以配置 DNS 服务器的顺序。或者，如果您喜欢在 `/etc/resolv.conf` 中手动配置 DNS 设置，您可以在 NetworkManager 中禁用 DNS 处理。例如，如果 `/etc/resolv.conf` 是一个符号链接，则 NetworkManager 不会自动更新 DNS 配置。

先决条件

- NetworkManager `rc-manager` 配置选项没有设置为 `file`。要验证，请使用 `NetworkManager --print-config` 命令。

流程

1. 创建一个文件，如 `/etc/resolv.conf.manually-configured`，并将您环境的 DNS 配置添加到其中。使用与原始 `/etc/resolv.conf` 中一样的参数和语法。
2. 删除 `/etc/resolv.conf` 文件：

```
# rm /etc/resolv.conf
```

3. 创建名为 `/etc/resolv.conf` 的符号链接，该链接指向 `/etc/resolv.conf.manually-configured`：

```
# ln -s /etc/resolv.conf.manually-configured /etc/resolv.conf
```

其他资源

- [resolv.conf\(5\) 手册页](#)
- [NetworkManager.conf\(5\) man page](#)
- [使用 NetworkManager 配置 DNS 服务器的顺序](#)

第 22 章 配置 DNS 服务器顺序

大多数应用程序使用 **glibc** 库的 **getaddrinfo()** 函数来解决 DNS 请求。默认情况下，**glibc** 将所有 DNS 请求发送到 **/etc/resolv.conf** 文件中指定的第一个 DNS 服务器。如果这个服务器没有回复，RHEL 会使用这个文件中的下一个服务器。使用 NetworkManager 可让您影响 **etc/resolv.conf** 中的 DNS 服务器顺序。

22.1. NETWORKMANAGER 如何在 /ETC/RESOLV.CONF 中对 DNS 服务器进行排序

NetworkManager 根据以下规则对 **/etc/resolv.conf** 文件中的 DNS 服务器进行排序：

- 如果只有一个连接配置集，NetworkManager 将使用那个连接中指定的 IPv4 和 IPv6 DNS 服务器顺序。
- 如果激活多个连接配置集，NetworkManager 会根据 DNS 优先级值对 DNS 服务器进行排序。如果您设置了 DNS 优先级，NetworkManager 的行为取决于 **dns** 参数中设置的值。您可以在 **/etc/NetworkManager/NetworkManager.conf** 文件的 **[main]** 部分中设置此参数：
 - **dns=default** 或者如果 **dns** 参数没有设置：

NetworkManager 根据每个连接中的 **ipv4.dns-priority** 和 **ipv6.dns-priority** 参数对不同的连接的 DNS 服务器进行排序。

如果没有设置值，或者您将 **ipv4.dns-priority** 和 **ipv6.dns-priority** 设为 **0**，则 NetworkManager 将使用全局默认值。请参阅 [DNS 优先级参数的默认值](#)。

- **dns=dnsmasq** 或 **dns=systemd-resolved**：

当您使用这些设置中的一个时，NetworkManager 将 **dnsmasq** 的 **127.0.0.1** 或 **127.0.0.53** 设为 **/etc/resolv.conf** 文件中的 **nameserver** 条目。

dnsmasq 和 **systemd-resolved** 服务将对 NetworkManager 连接中设置的搜索域的查询转发到该连接中指定的 DNS 服务器，并将对其他域的查询转发到带有默认路由的连接。当多个连接有相同的搜索域集时，**dnsmasq** 和 **systemd-resolved** 将这个域的查询转发到具有最低优先级值的连接中设置的 DNS 服务器。

DNS 优先级参数的默认值

NetworkManager 对连接使用以下默认值：

- **50** 用于 VPN 连接
- **100** 用于其他连接

有效的 DNS 优先级值：

您可以将全局默认值和特定于连接的 **ipv4.dns-priority** 和 **ipv6.dns-priority** 参数设为 **-2147483647** 和 **2147483647** 之间的值。

- 低的值具有更高的优先级。
- 负值具有一个特殊的效果，它会排除其他带有更大值的配置。例如，如果至少有一个连接具有负优先级值，NetworkManager 只使用在连接配置集中指定的具有最低优先级的 DNS 服务器。
- 如果多个连接具有相同的 DNS 优先级，NetworkManager 会按照以下顺序排列 DNS 的优先顺序：
 - a. VPN 连接

- b. 带有活跃的默认路由的连接。活跃的默认路由是具有最低指标的默认路由。

其他资源

- [nm-settings\(5\) 手册页](#)
- [在不同域中使用不同的 DNS 服务器](#)

22.2. 设置 NETWORKMANAGER 范围默认 DNS 服务器优先级值

NetworkManager 为连接使用以下 DNS 优先级默认值：

- **50** 用于 VPN 连接
- **100** 用于其他连接

您可以对 IPv4 和 IPv6 连接使用自定义的默认值覆盖这些系统范围的默认值。

步骤

1. 编辑 `/etc/NetworkManager/NetworkManager.conf` 文件：

- a. 添加 `[connection]` 部分（如果其不存在）：

```
[connection]
```

- b. 将自定义默认值添加到 `[connection]` 部分。例如，要将 IPv4 和 IPv6 的新默认值设为 **200**，请添加：

```
ipv4.dns-priority=200  
ipv6.dns-priority=200
```

您可以将参数设为 **-2147483647** 和 **2147483647** 之间的值。请注意，将参数设置为 **0** 将启用内置的默认值（对于 VPN 连接为 **50**，对于其他连接为 **100**）。

2. 重新载入 `NetworkManager` 服务：

```
# systemctl reload NetworkManager
```

其他资源

- [NetworkManager.conf\(5\) man page](#)

22.3. 设置网络管理器连接的 DNS 优先级

如果您需要 DNS 服务器的特定顺序，您可以在连接配置文件中设置优先级值。NetworkManager 使用这些值来在服务创建或更新 `/etc/resolv.conf` 文件时对服务器进行排序。

请注意，只有在您配置了多个与不同 DNS 服务器的连接时，设置 DNS 优先级才有意义。如果您只有一个与多个 DNS 服务器的连接，请在连接配置集中按首选顺序手动设置 DNS 服务器。

先决条件

- 系统配置了多个网络管理器连接。

- 系统在 `/etc/NetworkManager/NetworkManager.conf` 文件中未设置 `dns` 参数，或者该参数被设为了 `default`。

步骤

1. 另外，还可显示可用的连接：

```
# 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. 设置 `ipv4.dns-priority` 和 `ipv6.dns-priority` 参数。例如，要将这两个参数都设置为 `10`，请输入：

```
# nmcli connection modify <connection_name> ipv4.dns-priority 10 ipv6.dns-priority 10
```

3. 另外，还可为其他连接重复前面的步骤。
4. 重新激活您更新的连接：

```
# nmcli connection up <connection_name>
```

验证

- 显示 `/etc/resolv.conf` 文件的内容以验证 DNS 服务器顺序是否正确：

```
# cat /etc/resolv.conf
```

第 23 章 在不同域中使用不同的 DNS 服务器

默认情况下，Red Hat Enterprise Linux (RHEL)将所有 DNS 请求发送到 `/etc/resolv.conf` 文件中指定的第一个 DNS 服务器。如果这个服务器没有回复，RHEL 会使用这个文件中的下一个服务器。在一个 DNS 服务器无法解析所有域的环境中，管理员可将 RHEL 配置为将特定域的 DNS 请求发送到所选 DNS 服务器。

例如，您要将服务器连接到虚拟专用网络(VPN)，VPN 中的主机将使用 `example.com` 域。在这种情况下，您可以以下方式配置 RHEL，以处理 DNS 查询：

- 仅将 `example.com` 的 DNS 请求发送到 VPN 网络中的 DNS 服务器。
- 将所有其他请求发送到使用默认网关在连接配置文件中配置的 DNS 服务器。

23.1. 在 NETWORKMANAGER 中使用 DNSMASQ 将特定域的 DNS 请求发送到所选的 DNS 服务器

您可以将 NetworkManager 配置为启动一个 `dnsmasq` 实例。然后，此 DNS 缓存服务器侦听 `loopback` 设备上的端口 `53`。因此，该服务只能从本地系统访问，而不可从网络访问。

使用这个配置，NetworkManager 将 `nameserver 127.0.0.1` 条目添加到 `/etc/resolv.conf` 文件中，`dnsmasq` 会动态将 DNS 请求路由到 NetworkManager 连接配置文件中指定的对应 DNS 服务器。

先决条件

- 系统配置了多个网络管理器连接。
- 在负责解析特定域的 NetworkManager 连接配置文件中配置 DNS 服务器和搜索域。
例如，要确保 VPN 连接中指定的 DNS 服务器解析 `example.com` 域的查询，VPN 连接配置文件必须包含以下设置：
 - 可以解析 `example.com` 的 DNS 服务器
 - 在 `ipv4.dns-search` 和 `ipv6.dns-search` 参数中将搜索域设置为 `example.com`
- `dnsmasq` 服务没有运行或配置为侦听除 `localhost` 之外的不同的接口。

步骤

1. 安装 `dnsmasq` 软件包：

```
# dnf install dnsmasq
```

2. 编辑 `/etc/NetworkManager/NetworkManager.conf` 文件，并在 `[main]` 部分中设置以下条目：

```
dns=dnsmasq
```

3. 重新载入 `NetworkManager` 服务：

```
# systemctl reload NetworkManager
```

验证

1. 在 **NetworkManager** 单元的 **systemd** 日志中搜索服务使用不同的 DNS 服务器的域：

```
# journalctl -xeu NetworkManager
...
Jun 02 13:30:17 <client_hostname>_dnsmasq[5298]: using nameserver 198.51.100.7#53
for domain example.com
...
```

2. 使用 **tcpdump** 数据包嗅探器验证 DNS 请求的正确路由：

- a. 安装 **tcpdump** 软件包：

```
# dnf install tcpdump
```

- b. 在一个终端上，启动 **tcpdump** 以捕获所有接口上的 DNS 流量：

```
# tcpdump -i any port 53
```

- c. 在另一个终端上，解析存在异常的域的主机名，以及另一个域，例如：

```
# host -t A www.example.com
# host -t A www.redhat.com
```

- d. 在 **tcpdump** 输出中验证 Red Hat Enterprise Linux 只向指定的 DNS 服务器并通过对应的接口发送对 **example.com** 域的 DNS 查询：

```
...
13:52:42.234533 tun0 Out IP server.43534 > 198.51.100.7.domain: 50121+ A?
www.example.com. (33)
...
13:52:57.753235 enp1s0 Out IP server.40864 > 192.0.2.1.domain: 6906+ A?
www.redhat.com. (33)
...
```

Red Hat Enterprise Linux 将 **www.example.com** 的 DNS 查询发送到 **198.51.100.7** 上的 DNS 服务器，并将 **www.redhat.com** 的查询发送到 **192.0.2.1**。

故障排除

1. 验证 **/etc/resolv.conf** 文件中的 **nameserver** 条目是否指向 **127.0.0.1**：

```
# cat /etc/resolv.conf
nameserver 127.0.0.1
```

如果缺少该条目，请检查 **/etc/NetworkManager/NetworkManager.conf** 文件中的 **dns** 参数。

2. 验证 **dnsmasq** 服务是否侦听 **loopback** 设备上的端口 **53**：

```
# ss -tulpn | grep "127.0.0.1:53"
udp UNCONN 0 0 127.0.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=7340,fd=18))
tcp LISTEN 0 32 127.0.0.1:53 0.0.0.0:* users:(("dnsmasq",pid=7340,fd=19))
```

如果服务没有侦听 **127.0.0.1:53**，请检查 **NetworkManager** 单元的日志条目：

■

```
# journalctl -u NetworkManager
```

23.2. 使用 NETWORKMANAGER 中的 SYSTEMD-RESOLVED 将特定域的 DNS 请求发送到所选 DNS 服务器

您可以将 NetworkManager 配置为启动 **systemd-resolved** 的一个实例。然后，这个 DNS stub 解析器监听 IP 地址 **127.0.0.53** 上的端口 **53**。因此，这个 stub 解析器只能从本地系统访问，而不可从网络访问。

使用这个配置，NetworkManager 将 **nameserver 127.0.0.53** 条目添加到 **/etc/resolv.conf** 文件中，**systemd-resolved** 将 DNS 请求动态路由到 NetworkManager 连接配置文件中指定的对应的 DNS 服务器。

重要

systemd-resolved 服务仅作为技术预览提供。红帽产品服务级别协议（SLA）不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些预览可让用户早期访问将来的产品功能，让用户在开发过程中测试并提供反馈意见。

如需有关 [技术预览功能支持范围](#) 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

有关支持的解决方案，请参阅 [在 NetworkManager 中使用 dnsmasq 将特定域的 DNS 请求发送到所选 DNS 服务器](#)。

先决条件

- 系统配置了多个网络管理器连接。
- 在负责解析特定域的 NetworkManager 连接配置文件中配置 DNS 服务器和搜索域。例如，要确保 VPN 连接中指定的 DNS 服务器解析 **example.com** 域的查询，VPN 连接配置文件必须包含以下设置：
 - 可以解析 **example.com** 的 DNS 服务器
 - 在 **ipv4.dns-search** 和 **ipv6.dns-search** 参数中将搜索域设置为 **example.com**

流程

1. 启用并启动 **systemd-resolved** 服务：

```
# systemctl --now enable systemd-resolved
```

2. 编辑 **/etc/NetworkManager/NetworkManager.conf** 文件，并在 **[main]** 部分中设置以下条目：

```
dns=systemd-resolved
```

3. 重新载入 **NetworkManager** 服务：

```
# systemctl reload NetworkManager
```

验证

1. 显示 DNS 服务器 **systemd-resolved** 使用，以及服务为哪个域使用不同的 DNS 服务器：

```
# resolvectl
...
Link 2 (enp1s0)
  Current Scopes: DNS
    Protocols: +DefaultRoute ...
  Current DNS Server: 192.0.2.1
    DNS Servers: 192.0.2.1

Link 3 (tun0)
  Current Scopes: DNS
    Protocols: -DefaultRoute ...
  Current DNS Server: 198.51.100.7
    DNS Servers: 198.51.100.7 203.0.113.19
    DNS Domain: example.com
```

输出确认 **systemd-resolved** 为 **example.com** 域使用不同的 DNS 服务器。

2. 使用 **tcpdump** 数据包嗅探器验证 DNS 请求的正确路由：

a. 安装 **tcpdump** 软件包：

```
# dnf install tcpdump
```

b. 在一个终端上，启动 **tcpdump** 以捕获所有接口上的 DNS 流量：

```
# tcpdump -i any port 53
```

c. 在另一个终端上，解析存在异常的域的主机名，以及另一个域，例如：

```
# host -t A www.example.com
# host -t A www.redhat.com
```

d. 在 **tcpdump** 输出中验证 Red Hat Enterprise Linux 只向指定的 DNS 服务器并通过对应的接口发送对 **example.com** 域的 DNS 查询：

```
...
13:52:42.234533 tun0 Out IP server.43534 > 198.51.100.7.domain: 50121+ A?
www.example.com. (33)
...
13:52:57.753235 enp1s0 Out IP server.40864 > 192.0.2.1.domain: 6906+ A?
www.redhat.com. (33)
...
```

Red Hat Enterprise Linux 将 **www.example.com** 的 DNS 查询发送到 **198.51.100.7** 上的 DNS 服务器，并将 **www.redhat.com** 的查询发送到 **192.0.2.1**。

故障排除

1. 验证 **/etc/resolv.conf** 文件中的 **nameserver** 条目是否指向 **127.0.0.53**：

```
# cat /etc/resolv.conf
nameserver 127.0.0.53
```

如果缺少该条目，请检查 **/etc/NetworkManager/NetworkManager.conf** 文件中的 **dns** 参数。

2. 验证 **systemd-resolved** 服务是否监听本地 IP 地址 **127.0.0.53** 上的端口 **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))
```

如果服务没有侦听 **127.0.0.53:53**, 请检查 **systemd-resolved** 服务是否在运行。

第 24 章 管理默认网关设置

默认网关是在没有其他路由与数据包的目的地址匹配时转发网络数据包的路由器。在本地网络中，默认网关通常是与距离互联网有一跳的主机。

24.1. 使用 nmcli 在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 `nmcli` 工具对之前创建的连接设置或更新默认网关设置。

先决条件

- 至少需要在设置默认网关的连接上配置一个静态 IP 地址。
- 如果用户在物理控制台中登录，用户权限就足够了。否则，用户必须具有 `root` 权限。

步骤

1. 设置默认网关的 IP 地址：
要设置 IPv4 默认网关，请输入：

```
# nmcli connection modify <connection_name> ipv4.gateway  
"<IPv4_gateway_address>"
```

要设置 IPv6 默认网关，请输入：

```
# nmcli connection modify <connection_name> ipv6.gateway  
"<IPv6_gateway_address>"
```

2. 重启网络连接以使更改生效：

```
# nmcli connection up <connection_name>
```



警告

所有目前使用这个网络连接的连接在重启过程中暂时中断。

验证

- 验证路由是否处于活跃状态：
 - a. 要显示 IPv4 默认网关，请输入：

```
# ip -4 route  
default via 192.0.2.1 dev example proto static metric 100
```

- b. 要显示 IPv6 默认网关，请输入：

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

24.2. 使用 NMCLI 交互模式在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 **nmcli** 工具的交互模式对之前创建的连接设置或更新默认网关设置。

先决条件

- 至少需要在设置默认网关的连接上配置一个静态 IP 地址。
- 如果用户在物理控制台中登录，用户权限就足够了。否则，该用户必须具有 **root** 权限。

步骤

1. 为所需连接打开 **nmcli** 交互模式：

```
# nmcli connection edit <connection_name>
```

2. 设置默认网关

要设置 IPv4 默认网关，请输入：

```
nmcli> set ipv4.gateway "<IPv4_gateway_address>"
```

要设置 IPv6 默认网关，请输入：

```
nmcli> set ipv6.gateway "<IPv6_gateway_address>"
```

3. 另外，还可验证默认网关是否正确设置：

```
nmcli> print
...
ipv4.gateway:      <IPv4_gateway_address>
...
ipv6.gateway:      <IPv6_gateway_address>
...
```

4. 保存配置：

```
nmcli> save persistent
```

5. 重启网络连接以使更改生效：

```
nmcli> activate <connection_name>
```

**警告**

所有目前使用这个网络连接的连接在重启过程中暂时中断。

6. 保留 **nmcli** 交互模式：

```
nmcli> quit
```

验证

- 验证路由是否处于活跃状态：
 - a. 要显示 IPv4 默认网关，请输入：

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

- b. 要显示 IPv6 默认网关，请输入：

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

24.3. 使用 **NM-CONNECTION-EDITOR** 在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 **nm-connection-editor** 应用程序对之前创建的连接设置或更新默认网关设置。

先决条件

- 至少需要在设置默认网关的连接上配置一个静态 IP 地址。

步骤

1. 打开一个终端，输入 **nm-connection-editor**：

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

2. 选择要修改的连接，并点击 gear wheel 图标编辑现有连接。
3. 设置 IPv4 默认网关。例如，要将连接上的默认网关的 IPv4 地址设为 **192.0.2.1**：
 - a. 打开 **IPv4 Settings** 选项卡。
 - b. 在网关地址所在的 IP 范围旁边的 **gateway** 字段中输入地址：

Addresses		
Address	Netmask	Gateway
192.0.2.123	24	192.0.2.1

4. 设置 IPv6 默认网关。例如，要将连接上默认网关的 IPv6 地址设为 **2001:db8:1::1** :
 - a. 打开 **IPv6** 选项卡。
 - b. 在网关地址所在的 IP 范围旁边的 **gateway** 字段中输入地址 :

Addresses		
Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

5. 点击 **确定**。
6. 点击 **Save**。
7. 重启网络连接以使更改生效。例如，要使用命令行重启 **example** 连接 :

```
# nmcli connection up example
```



警告

所有目前使用这个网络连接的连接在重启过程中暂时中断。

8. (可选) 验证路由是否活跃。
显示 IPv4 默认网关 :

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

显示 IPv6 默认网关 :

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

其他资源

- [使用 nm-connection-editor 配置以太网连接](#)

24.4. 使用 CONTROL-CENTER 在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 **control-center** 应用程序对之前创建的连接设置或更新默认网关设置。

先决条件

- 至少需要在设置默认网关的连接上配置一个静态 IP 地址。
- 连接的网络配置在 **control-center** 应用程序中打开。

步骤

1. 设置 IPv4 默认网关。例如，要将连接上的默认网关的 IPv4 地址设为 **192.0.2.1**：

- a. 打开 **IPv4** 选项卡。
- b. 在网关地址所在的 IP 范围旁边的 **gateway** 字段中输入地址：

Addresses		
Address	Netmask	Gateway
192.0.2.123	255.255.255.0	192.0.2.1

2. 设置 IPv6 默认网关。例如，要将连接上默认网关的 IPv6 地址设为 **2001:db8:1::1**：

- a. 打开 **IPv6** 选项卡。
- b. 在网关地址所在的 IP 范围旁边的 **gateway** 字段中输入地址：

Addresses		
Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

3. 点应用。
4. 返回到 **Network** 窗口，通过将连接的按钮切换为 **Off**，然后返回到 **On** 来禁用并重新启用连接，以使更改生效。



警告

所有目前使用这个网络连接的连接在重启过程中暂时中断。

5. (可选) 验证路由是否活跃。

显示 IPv4 默认网关：

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

显示 IPv6 默认网关：

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

其他资源

- [使用 control-center 配置以太网连接](#)

24.5. 使用 NMSTATECTL 在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 **nmstatectl** 工具对之前创建的连接设置或更新默认网关设置。

使用 **nmstatectl** 工具通过 Nmstate API 设置默认网关。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，**nmstatectl** 会自动回滚更改以避免系统处于不正确的状态。

先决条件

- 至少需要在设置默认网关的连接上配置一个静态 IP 地址。
- **enp1s0** 接口已配置，默认网关的 IP 地址在此接口的 IP 配置子网内。
- **nmstate** 软件包已安装。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 **~/set-default-gateway.yml**：

```
---
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.1
    next-hop-interface: enp1s0
```

这些设置将 **192.0.2.1** 定义为默认网关，可通过 **enp1s0** 接口到达默认网关。

2. 将设置应用到系统：

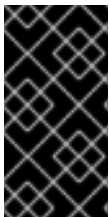
```
# nmstatectl apply ~/set-default-gateway.yml
```

其他资源

- **nmstatectl(8)** 手册页
- **/usr/share/doc/nmstate/examples/** 目录

24.6. 使用 NETWORK RHEL 系统角色在现有连接上设置默认网关

在大多数情况下，管理员在创建连接时设置默认网关。但是，您也可以使用 **network** RHEL 系统角色在之前创建的连接上设置或更新默认网关设置，以设置默认网关。



重要

当您运行一个使用 **network** RHEL 系统角色的 play 时，如果设置值与 play 中指定的值不匹配，则角色会使用相同的名称覆盖现有的连接配置文件。要防止将这些值重置为其默认值，请始终在 play 中指定网络连接配置文件的整个配置，即使配置（如 IP 配置）已存在。

根据它是否已存在，流程使用以下设置创建或更新 **enp1s0** 连接配置文件：

- 静态 IPv4 地址 - **198.51.100.20**，子网掩码为 **/24**
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码

- IPv4 默认网关 - **198.51.100.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **198.51.100.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```

---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP and default gateway
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 198.51.100.20/24
                - 2001:db8:1::1/64
              gateway4: 198.51.100.254
              gateway6: 2001:db8:1::fffe
            dns:
              - 198.51.100.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            state: up
  
```

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

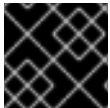
- `/usr/share/ansible/roles/rhel-system-roles.network/README.md` 文件
- `/usr/share/doc/rhel-system-roles/network/` directory

24.7. NETWORKMANAGER 如何管理多个默认网关

在某些情况下，您可能需要在主机上设置多个默认网关。但是，为了避免异步路由问题，同一协议的每个默认网关都需要单独的指标值。请注意，RHEL 只使用到设置成最低指标的默认网关的连接。

您可以使用以下命令为连接的 IPv4 和 IPv6 网关设置指标：

```
# nmcli connection modify <connection_name> ipv4.route-metric <value> ipv6.route-metric <value>
```



重要

不要为多个连接配置集中的同一协议设置相同的指标值以避免路由问题。

如果您设置了没有指标值的默认网关，则 NetworkManager 会自动根据接口类型设置指标值。为此，NetworkManager 将这个网络类型的默认值分配给激活的第一个连接，并根据激活的顺序为同一类型的每一个其他连接设置递增值。例如，如果带有默认网关的两个以太网连接存在，则 NetworkManager 会将路由上的 **100** 指标设置为您首先激活的连接的默认网关。对于第二个连接，NetworkManager 会设为 **101**。

以下是经常使用的网络类型及其默认指标的概述：

连接类型	默认指标值
VPN	50
Ethernet	100
MACsec	125
InfiniBand	150
Bond	300
Team	350
VLAN	400
Bridge	425
TUN	450

连接类型	默认指标值
Wi-Fi	600
IP tunnel	675

其他资源

- [配置基于策略的路由以定义其他路由](#)

24.8. 配置 NETWORKMANAGER 以避免使用特定配置集提供默认网关

您可以配置 NetworkManager 从不使用特定的配置文件来提供默认网关。对于没有连接到默认网关的连接配置集，请按照以下步骤操作。

先决条件

- 没有连接到默认网关的连接的 NetworkManager 连接配置文件存在。

流程

1. 如果连接使用动态 IP 配置，请配置 NetworkManager 不使用该连接作为 IPv4 和 IPv6 连接的默认路由：

```
# nmcli connection modify <connection_name> ipv4.never-default yes ipv6.never-default yes
```

请注意，将 **ipv4.never-default** 和 **ipv6.never-default** 设为 **yes**，会自动从连接配置文件中删除相应协议默认网关的 IP 地址。

2. 激活连接：

```
# nmcli connection up <connection_name>
```

验证

- 使用 **ip -4** 路由和 **ip -6 route** 命令，来验证 RHEL 是否未对 IPv4 和 IPv6 协议的默认路由使用网络接口。

24.9. 修复因为多个默认网关导致的意外路由行为

只有一些情况，比如使用多路径 TCP 时，您需要在主机上有多个默认网关。在大多数情况下，您只配置一个默认网关，来避免意外的路由行为或异步路由问题。



注意

要将流量路由到不同的互联网提供商，请使用基于策略的路由，而不是多个默认网关。

先决条件

- 主机使用 NetworkManager 管理网络连接，这是默认设置。
- 主机有多个网络接口。
- 主机配置了多个默认网关。

流程

1. 显示路由表：

- 对于 IPv4，请输入：

```
# 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
...
```

- 对于 IPv6，请输入：

```
# 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
...
```

以 **default** 开头的条目表示默认路由。注意 **dev** 旁边显示的这些条目的接口名称。

2. 使用以下命令显示您使用在上一步中识别的接口的 NetworkManager 连接：

```
# nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp1s0
GENERAL.CONNECTION: Corporate-LAN
IP4.GATEWAY: 192.0.2.1
IP6.GATEWAY: 2001:db8:1::1

# nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp7s0
GENERAL.CONNECTION: Internet-Provider
IP4.GATEWAY: 198.51.100.1
IP6.GATEWAY: 2001:db8:2::1
```

在这些示例中，名为 **Corporate-LAN** 和 **Internet-Provider** 的配置文件设置了默认网关。因为在本地网络中，默认网关通常是距离互联网一跳的主机，所以此流程的剩下部分假设 **Corporate-LAN** 中的默认网关是不正确的。

3. 配置 NetworkManager 不使用 **Corporate-LAN** 连接作为 IPv4 和 IPv6 连接的默认路由：

```
# nmcli connection modify Corporate-LAN ipv4.never-default yes ipv6.never-default yes
```

请注意，将 **ipv4.never-default** 和 **ipv6.never-default** 设为 **yes**，会自动从连接配置文件中删除相应协议默认网关的 IP 地址。

4. 激活 **Corporate-LAN** 连接：

```
# nmcli connection up Corporate-LAN
```

验证

- 显示 IPv4 和 IPv6 路由表，并确认每个协议都只有一个默认网关：

- 对于 IPv4，请输入：

```
# ip -4 route
default via 192.0.2.1 dev enp1s0 proto static metric 101
...
```

- 对于 IPv6，请输入：

```
# ip -6 route
default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
...
```

其他资源

- [配置基于策略的路由以定义其他路由](#)

第 25 章 配置静态路由

路由可确保您可以在相互连接的网络间发送和接收流量。在较大环境中，管理员通常配置服务以便路由器可以动态地了解其他路由器。在较小的环境中，管理员通常会配置静态路由，以确保流量可以从一个网络到下一个网络访问。

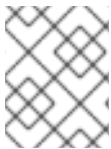
如果适用所有这些条件，您需要静态路由以在多个网络间获得正常运行的通信：

- 流量必须通过多个网络。
- 通过默认网关的独占流量流不足。

[需要静态路由部分的网络示例](#) 描述了在没有配置静态路由时不同网络间的流量流。

25.1. 需要静态路由的网络示例

您需要在这个示例中的静态路由，因为并非所有 IP 网络都通过一个路由器直接连接。如果没有静态路由，一些网络无法相互通信。此外，某些网络流的流量仅有一个方向。



注意

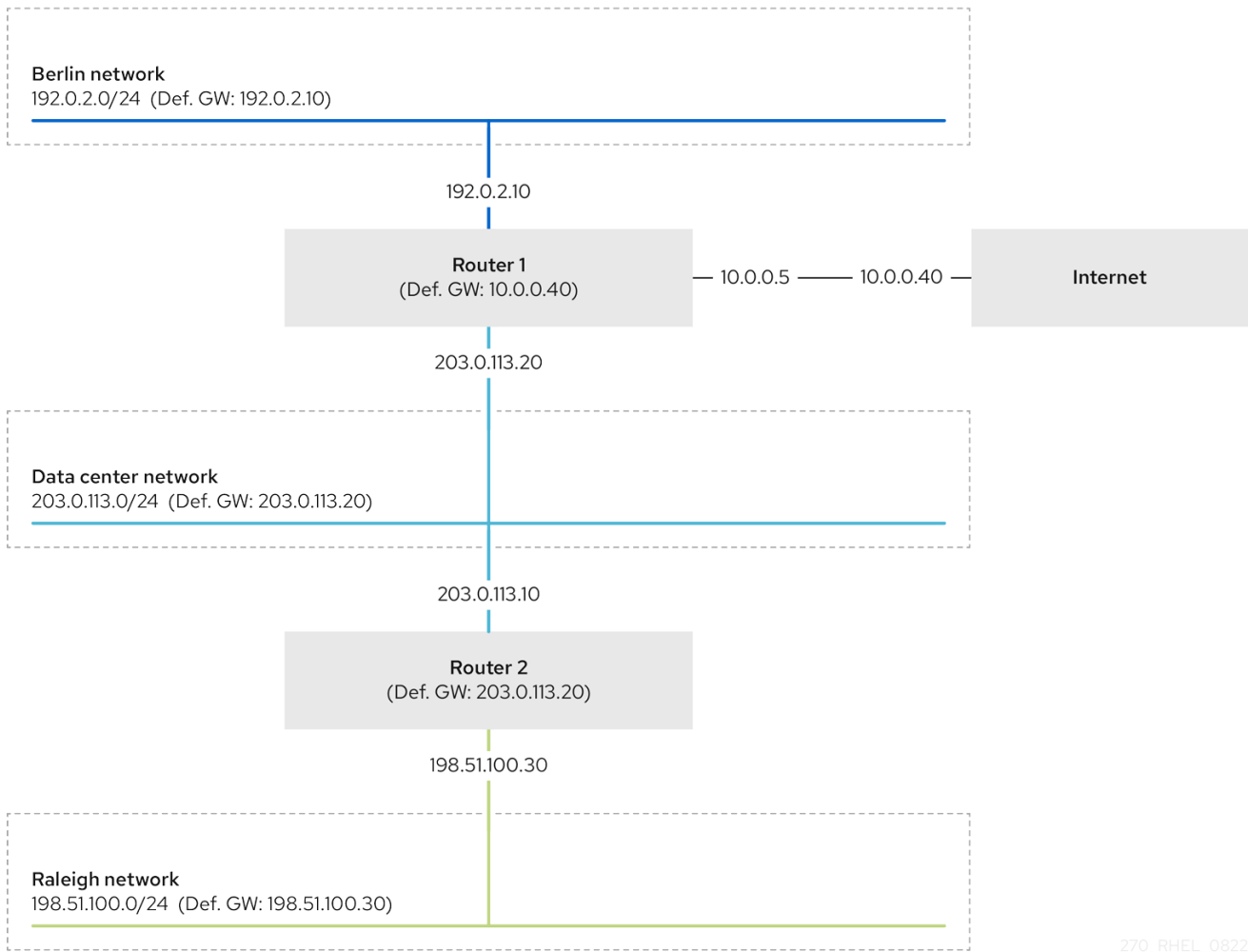
本例中的网络拓扑是假设的，仅用于解释静态路由的概念。在生产环境中并不推荐使用这个拓扑。

对于本示例中所有网络中的一个正常运行的通信，请将静态路由配置为 Raleigh (**198.51.100.0/24**)，下一个越点 (hop) 路由器 2 (**203.0.113.10**)。下一个跃点的 IP 地址是数据中心网络中的一个路由器 2 (**203.0.113.0/24**)。

您可以配置静态路由，如下所示：

- 对于简化的配置，仅在路由器 1 上设置此静态路由。但是，这会增加路由器 1 上的流量，因为来自数据中心 (**203.0.113.0/24**) 的主机将流量发送到 Raleigh (**198.51.100.0/24**)，始终通过路由器 1 到路由器 2。
- 对于更复杂的配置，请在数据中心的所有主机上配置此静态路由 (**203.0.113.0/24**)。然后，此子网中的所有主机直接向路由器 2 (**203.0.113.10**) 发送更接近 Raleigh 的主机 (**198.51.100.0/24**)。

有关网络流量流或不符的更多详情，请参见以下示意图中的说明。

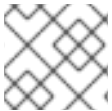


270_RHEL_0822

在所需的静态路由没有配置的情况下，如下是通信可以正常工作和不能正常工作的情况：

- Berlin 网络中的主机 (**192.0.2.0/24**) :
 - 可以与同一子网中的其他主机通信，因为它们是直接连接的。
 - 可以与互联网通信，因为路由器 1 位于 Berlin 网络(**192.0.2.0/24**)中，并且有一个通向互联网的默认网关。
 - 可以与数据中心网络 (**203.0.113.0/24**)通信，因为路由器 1 在 Berlin (**192.0.2.0/24**) 和数据中心(**203.0.113.0/24**) 网络中都有接口。
 - 无法与 Raleigh 网络 (**198.51.100.0/24**) 通信，因为路由器 1 在此网络中没有接口。因此，路由器 1 将流量发送到自己的默认网关(互联网)。
- 数据中心网络中的主机 (**203.0.113.0/24**):
 - 可以与同一子网中的其他主机通信，因为它们是直接连接的。
 - 可以与互联网通信，因为它们的默认网关设为 Router 1，Router 1 在数据中心 (**203.0.113.0/24**)和互联网两个网络中有接口。
 - 可以与 Berlin 网络 (**192.0.2.0/24**) 通信，因为它们的默认网关设置为路由器 1，并且路由器 1 在数据中心 (**203.0.113.0/24**) 和 Berlin (**192.0.2.0/24**) 网络中都存在接口。

- 无法与 Raleigh 网络 (198.51.100.0/24) 通信，因为该网络中没有接口。因此，数据中心中的主机 (203.0.113.0/24) 将流量发送到其默认网关 (路由器 1)。Router 1 在 Raleigh 网络 (198.51.100.0/24) 中没有接口，因此 Router 1 会将此流量发送到自己的默认网关 (互联网)。
- Raleigh 网络中的主机 (198.51.100.0/24) :
 - 可以与同一子网中的其他主机通信，因为它们是直接连接的。
 - 无法与互联网上的主机进行通信。因为默认的网关设置，路由器 2 将流量发送到路由器 1。路由器 1 的实际行为取决于反向路径过滤器 (`rp_filter`) 系统控制 (`sysctl`) 设置。默认情况下，在 RHEL 上，Router 1 会丢弃传出的流量，而不是将其路由到互联网。但是，无论配置的行为如何，都无法在没有静态路由的情况下进行通信。
 - 无法与数据中心网络通信 (203.0.113.0/24)。由于默认网关设置，传出流量通过路由器 2 到达目的地。但是，对数据包的回复不会到达发送者，因为数据中心网络中的主机 (203.0.113.0/24) 将回复发送到其默认网关 (Router 1)。然后，Router 1 将流量发送到互联网。
 - 无法与 Berlin 网络通信 (192.0.2.0/24)。因为默认的网关设置，路由器 2 将流量发送到路由器 1。路由器 1 的实际行为取决于 `rp_filter sysctl` 设置。默认情况下，在 RHEL 中，路由器 1 会丢弃传出流量，而不是将其发送到 Berlin 网络 (192.0.2.0/24)。但是，无论配置的行为如何，都无法在没有静态路由的情况下进行通信。



注意

除了配置静态路由外，还必须在两个路由器上启用 IP 转发。

其他资源

- [如果在服务器上设置了 `net.ipv4.conf.all.rp_filter`，为什么无法 ping 服务器？](#)
- [启用 IP 转发](#)

25.2. 如何使用 NMCLI 工具配置静态路由

要配置静态路由，请使用具有以下语法的 `nmcli` 工具：

```
$ nmcli connection modify connection_name ipv4.routes "ip[/prefix] [next_hop] [metric] [attribute=value] [attribute=value] ..."
```

该命令支持以下路由属性：

- `cnwnd=n`：设置拥塞窗口 (CWND) 大小，以数据包数量定义。
- `lock-cwnd=true|false`：定义内核是否可以更新 CWND 值。
- `lock-mtu=true|false`：定义内核是否可以将 MTU 更新为路径 MTU 发现。
- `lock-window=true|false`：定义内核是否可更新 TCP 数据包的最大窗口大小。
- `mtu=<mtu_value>`：设置要与目的地的路径一起使用的最大传输单元 (MTU)。
- `onlink=true|false`：定义下一个跃点是否直接附加到此链接，即使它与任何接口前缀都不匹配。
- `scope=<scope>`：对于 IPv4 路由，此属性设置路由前缀所涵盖的目的地的范围。将值设为整数 (0-255)。

- **src= <source_address >** : 在将流量发送到路由前缀所涵盖的目的地时, 将源地址设置为首选。
- **table= <table_id >** : 设置应将路由添加到的表的 ID。如果省略此参数, NetworkManager 将使用 **main** 表。
- **ToS= <type_of_service_key >** : 设置服务类型(TOS)密钥。将值设为整数(0-255)。
- **type= <route_type >** : 设置路由类型。NetworkManager 支持 **unicast**、**local**、**blackhole**、**unreachable**、**prohibit** 和 **throw** 路由类型。默认为 **unicast**。
- **window= <window_size >** : 设置要公告到这些目的地的最大窗口大小, 以字节为单位。

重要

如果您使用没有前面的 + 符号的 **ipv4.routes** 选项, **nmcli** 会覆盖这个参数的所有当前设置。

- 要创建额外路由, 请输入 :

```
$ nmcli connection modify connection_name +ipv4.routes "<route>"
```

- 要删除特定的路由, 请输入 :

```
$ nmcli connection modify connection_name -ipv4.routes "<route>"
```

25.3. 使用 NMCLI 配置静态路由

您可以使用 **nmcli connection modify** 命令将静态路由添加到现有 NetworkManager 连接配置集。

以下流程配置以下路由 :

- 到远程 **198.51.100.0/24** 网络的 IPv4 路由。IP 地址为 **192.0.2.10** 的对应网关可以通过 **LAN** 连接配置文件访问。
- 到远程 **2001:db8:2::/64** 网络的 IPv6 路由。IP 地址为 **2001:db8:1::10** 的对应网关可以通过 **LAN** 连接配置文件访问。

前提条件

- **LAN** 连接配置文件存在, 它将此主机配置为与网关位于同一个 IP 子网中。

流程

1. 将静态 IPv4 路由添加到 **LAN** 连接配置文件 :

```
# nmcli connection modify LAN +ipv4.routes "198.51.100.0/24 192.0.2.10"
```

要在一个步骤中设置多个路由, 请将单独的路由以逗号分隔的传递给命令 :

```
# nmcli connection modify <connection_profile> +ipv4.routes
"<remote_network_1>/<subnet_mask_1> <gateway_1>,
<remote_network_n>/<subnet_mask_n> <gateway_n>, ..."
```

2. 将静态 IPv6 路由添加到 **LAN** 连接配置文件：

```
# nmcli connection modify LAN +ipv6.routes "2001:db8:2::/64 2001:db8:1::10"
```

3. 重新激活连接：

```
# nmcli connection up LAN
```

验证

1. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

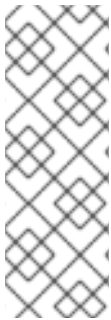
2. 显示 IPv6 路由：

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

25.4. 使用 NMTUI 配置静态路由

nmtui 应用程序为 NetworkManager 提供了一个基于文本的用户界面。您可以使用 **nmtui** 在没有图形界面的主机上配置静态路由。

例如，以下流程将路由添加到 **192.0.2.0/24** 网络，该网络使用运行在 **198.51.100.1** 上的网关，该网络可通过现有的连接配置文件访问。



注意

在 **nmtui** 中：

- 使用光标键导航。
- 选择一个按钮并按 **Enter** 键。
- 使用空格选择和 **清除复选框**。

先决条件

- 网络已配置。
- 静态路由的网关必须在接口上直接访问。
- 如果用户在物理控制台中登录，用户权限就足够了。否则，命令需要 **root** 权限。

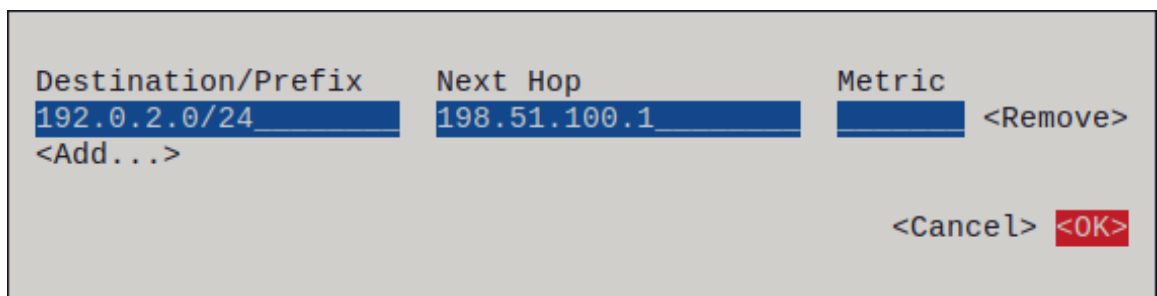
步骤

1. 启动 **nmtui**：

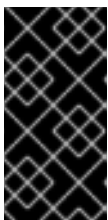
```
# nmtui
```


2. 选择 **Edit a connection**, 然后按 **Enter**。
3. 选择您可通过其到达目的地网络的下一跳的连接配置文件, 然后按 **Enter**。
4. 根据它是 IPv4 还是 IPv6 路由, 按协议配置区域旁边的 **Show** 按钮。
5. 按 **Routing** 旁边的 **Edit** 按钮。这将打开一个新窗口, 您可在其中配置静态路由 :
 - a. 按 **Add** 按钮并填写 :
 - 目的网络, 包括无类别域间路由(CIDR)格式的前缀
 - 下一跳的 IP 地址
 - 指标值, 如果您向同一网络添加多个路由, 并且希望根据效率对路由进行优先排序
 - b. 对您要添加的每个路由重复上一步, 且通过此连接配置文件可达。
 - c. 按 **OK** 按钮返回到连接设置的窗口。

图 25.1. 没有指标的静态路由的示例



6. 按 **OK** 按钮返回到 **nmtui** 主菜单。
7. 选择 **Activate a connection**, 然后按 **Enter** 键。
8. 选择您编辑的连接配置文件, 然后按 **Enter** 两次来停用并再次激活它。



重要

如果您通过使用您要重新激活的连接配置文件的远程连接（如 SSH）来运行 **nmtui**, 请跳过这一步。在这种情况下, 如果您在 **nmtui** 中停用了它, 连接将被终止, 因此您无法再次激活它。要避免这个问题, 请使用 **nmcli connection <it>connection_profile > up** 命令在上述场景中重新激活连接。

9. 按 **Back** 按钮返回到主菜单。
10. 选择 **Quit**, 然后按 **Enter** 关闭 **nmtui** 应用程序。

验证

- 验证路由是否处于活跃状态 :

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

25.5. 使用 CONTROL-CENTER 配置静态路由

您可以在 GNOME 中使用 **control-center**，来将静态路由添加到网络连接配置中。

以下流程配置以下路由：

- 到远程 **198.51.100.0/24** 网络的 IPv4 路由。对应的网关的 IP 地址 **192.0.2.10**。
- 到远程 **2001:db8:2::/64** 网络的 IPv6 路由。对应的网关具有 IP 地址 **2001:db8:1::10**。

前提条件

- 网络已配置。
- 此主机与网关位于同一个 IP 子网中。
- 连接的网络配置在 **control-center** 应用程序中打开。请参阅 [使用 nm-connection-editor 配置以太网连接](#)。

流程

1. 在 IPv4 标签页中：

- 可选：点 **IPv4** 选项卡的 **Routes** 部分中的 **On** 按钮来禁用自动路由，使其只使用静态路由。如果启用了自动路由，Red Hat Enterprise Linux 将使用静态路由和从 DHCP 服务器接收的路由。
- 输入 IPv4 路由的地址、子网掩码、网关和可选的指标值：

Routes				Automatic <input checked="" type="checkbox"/>
Address	Netmask	Gateway	Metric	
198.51.100.0	24	192.0.2.10		<input type="checkbox"/>

2. 在 IPv6 标签页中：

- 可选：点 **IPv4** 选项卡的 **Routes** 部分中的 **On** 按钮来禁用自动路由，使其只使用静态路由。
- 输入 IPv6 路由的地址、子网掩码、网关和可选的指标值：

Routes				Automatic <input checked="" type="checkbox"/>
Address	Prefix	Gateway	Metric	
2001:db8:2::	64	2001:db8:1::10		<input type="checkbox"/>

3. 点应用。

- 返回到 **Network** 窗口，通过将连接的按钮切换为 **Off**，然后返回到 **On** 来禁用并重新启用连接，以使更改生效。



警告

重启连接会破坏那个接口的连接。

验证

1. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. 显示 IPv6 路由：

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

25.6. 使用 NM-CONNECTION-EDITOR 配置静态路由

您可以使用 **nm-connection-editor** 应用程序将静态路由添加到网络连接配置中。

以下流程配置以下路由：

- 到远程 **198.51.100.0/24** 网络的 IPv4 路由。IP 地址为 **192.0.2.10** 的对应网关可以通过 **example** 连接访问。
- 到远程 **2001:db8:2::/64** 网络的 IPv6 路由。IP 地址为 **2001:db8:1::10** 的对应网关可以通过 **example** 连接访问。

前提条件

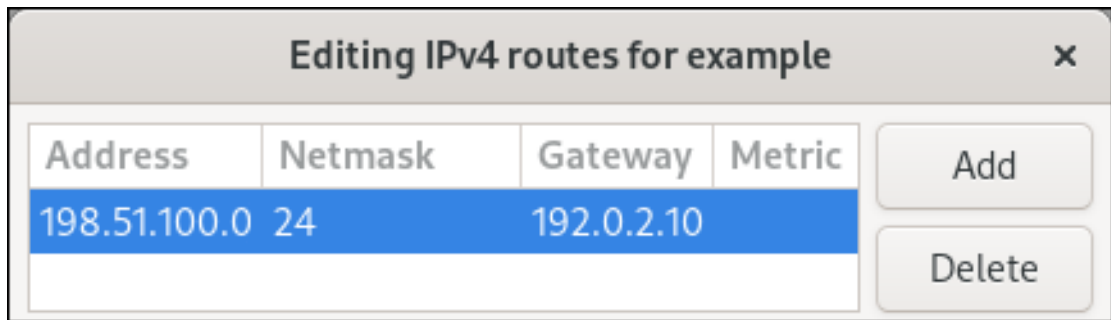
- 网络已配置。
- 此主机与网关位于同一个 IP 子网中。

流程

1. 打开一个终端，输入 **nm-connection-editor**：

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

2. 选择 **example** 连接配置集，并点齿轮图标编辑现有连接。
3. 在 **IPv4 设置** 标签页中：
 - a. 点击 **路由** 按钮。
 - b. 点击 **添加** 按钮并输入地址、子网掩码、网关以及可选的指标值。

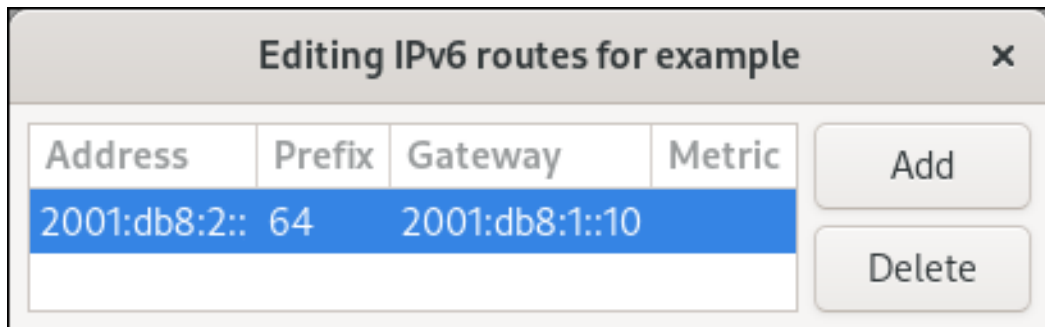


c. 点确定。

4. 在 **IPv6 设置** 标签页中：

a. 点击 **路由** 按钮。

b. 点击 **添加** 按钮并输入地址、子网掩码、网关以及可选的指标值。



c. 点确定。

5. 点击 **Save**。

6. 重启网络连接以使更改生效。例如，要使用命令行重启 **example** 连接：

```
# nmcli connection up example
```

验证

1. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. 显示 IPv6 路由：

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

25.7. 使用 NMCLI 交互模式配置静态路由

您可以使用 **nmcli** 工具的交互模式，将静态路由添加到网络连接配置中。

以下流程配置以下路由：

- 到远程 **198.51.100.0/24** 网络的 IPv4 路由。IP 地址为 **192.0.2.10** 的对应网关可以通过 **example** 连接访问。
- 到远程 **2001:db8:2::/64** 网络的 IPv6 路由。IP 地址为 **2001:db8:1::10** 的对应网关可以通过 **example** 连接访问。

前提条件

- **example** 连接配置集存在，它将此主机配置为与网关在同一 IP 子网中。

流程

1. 为 **example** 连接打开 **nmcli** 交互模式：

```
# nmcli connection edit example
```

2. 添加静态 IPv4 路由：

```
nmcli> set ipv4.routes 198.51.100.0/24 192.0.2.10
```

3. 添加静态 IPv6 路由：

```
nmcli> set ipv6.routes 2001:db8:2::/64 2001:db8:1::10
```

4. (可选) 验证路由是否已正确添加到配置中：

```
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 }
...
```

ip 属性显示要路由的网络，**nh** 属性显示网关（下一跳）。

5. 保存配置：

```
nmcli> save persistent
```

6. 重启网络连接：

```
nmcli> activate example
```

7. 保留 **nmcli** 交互模式：

```
nmcli> quit
```

验证

1. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. 显示 IPv6 路由：

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

其他资源

- [nmcli\(1\) 手册页](#)
- [nm-settings-nmcli\(5\) 手册页](#)

25.8. 使用 NMSTATECTL 配置静态路由

使用 `nmstatectl` 工具通过 Nmstate API 配置一个静态路由。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，`nmstatectl` 会自动回滚更改以避免系统处于不正确的状态。

前提条件

- `enp1s0` 网络接口已配置，且与网关位于同一个 IP 子网。
- `nmstate` 软件包已安装。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 `~/add-static-route-to-enp1s0.yml`：

```
---
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
```

这些设置定义以下静态路由：

- 到远程 `198.51.100.0/24` 网络的 IPv4 路由。IP 地址为 `192.0.2.10` 的对应网关可以通过 `enp1s0` 接口访问。
 - 到远程 `2001:db8:2::/64` 网络的 IPv6 路由。IP 地址为 `2001:db8:1::10` 的对应网关可以通过 `enp1s0` 接口访问。
2. 将设置应用到系统：

```
# nmstatectl apply ~/add-static-route-to-enp1s0.yml
```

短址

1. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. 显示 IPv6 路由：

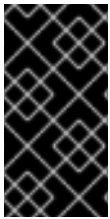
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

25.9. 使用 NETWORK RHEL 系统角色配置静态路由

您可以使用 **network** RHEL 系统角色配置静态路由。



重要

当您运行一个使用 **network** RHEL 系统角色的 play 时，如果设置值与 play 中指定的值不匹配，则角色会使用相同的名称覆盖现有的连接配置文件。要防止将这些值重置为其默认值，请始终在 play 中指定网络连接配置文件的整个配置，即使配置（如 IP 配置）已存在。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。

流程

1. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml`：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with static IP and additional routes
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp7s0
            type: ethernet
```

```
autoconnect: yes
ip:
  address:
    - 192.0.2.1/24
    - 2001:db8:1::1/64
  gateway4: 192.0.2.254
  gateway6: 2001:db8:1::fffe
  dns:
    - 192.0.2.200
    - 2001:db8:1::ffbb
  dns_search:
    - example.com
  route:
    - network: 198.51.100.0
      prefix: 24
      gateway: 192.0.2.10
    - network: 2001:db8:2::
      prefix: 64
      gateway: 2001:db8:1::10
state: up
```

根据它是否已存在，流程使用以下设置创建或更新 **enp7s0** 连接配置文件：

- 静态 IPv4 地址 - **192.0.2.1** 和 /24 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 /64 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- 静态路由：
 - **198.51.100.0/24**，网关为 **192.0.2.10**
 - **2001:db8:2::/64**，网关为 **2001:db8:1::10**

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

验证

1. 在受管节点上：

a. 显示 IPv4 路由：

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp7s0
```

b. 显示 IPv6 路由：

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

其他资源

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

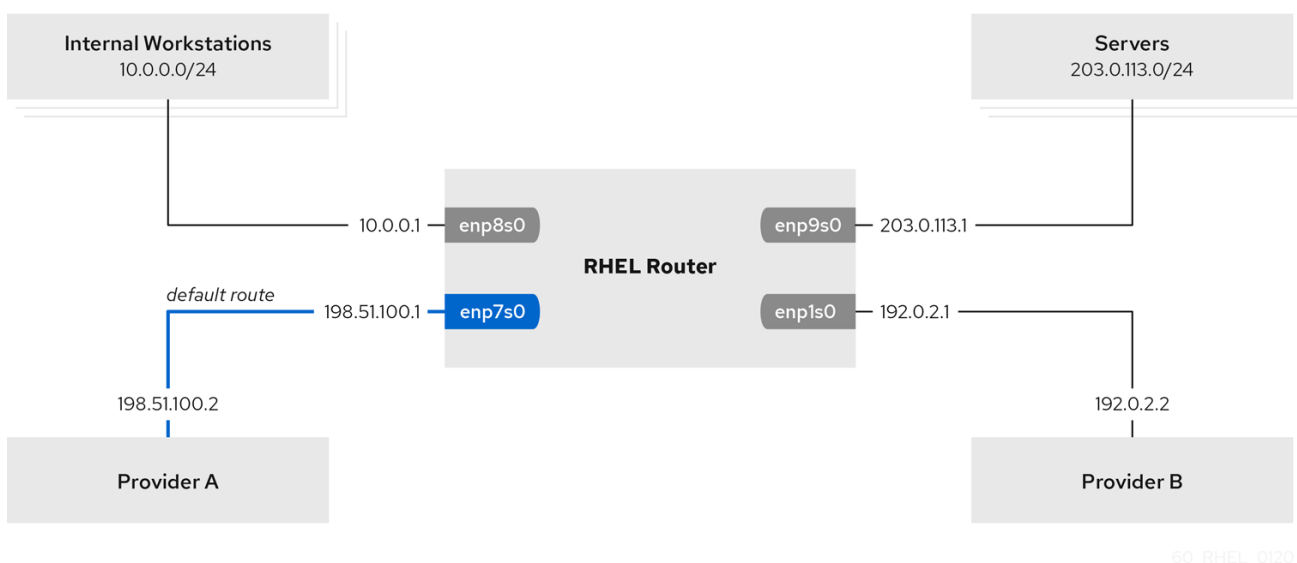
第 26 章 配置基于策略的路由以定义其他路由

默认情况下，RHEL 中的内核决定使用路由表根据目标地址转发网络数据包。基于策略的路由允许您配置复杂的路由场景。例如，您可以根据各种条件来路由数据包，如源地址、数据包元数据或协议。

26.1. 使用 NMCLI 将特定子网的流量路由到不同的默认网关

您可以使用基于策略的路由为来自特定子网的流量配置不同的默认网关。例如，您可以将 RHEL 配置为默认路由，使用默认路由将所有流量路由到互联网提供商 A。但是，从内部工作站子网接收的流量路由到供应商 B。

该流程假设以下网络拓扑：



60_RHEL_0120

先决条件

- 系统使用 **NetworkManager** 来配置网络，这是默认设置。
- 要在流程中设置的 RHEL 路由器有四个网络接口：
 - **enp7s0** 接口已连接到提供商 A 的网络。提供商网络中的网关 IP 为 **198.51.100.2**，网络使用 **/30** 网络掩码。
 - **enp1s0** 接口连接到提供商 B 的网络。提供商网络中的网关 IP 为 **192.0.2.2**，网络使用 **/30** 网络掩码。
 - **enp8s0** 接口已与连有内部工作站的 **10.0.0.0/24** 子网相连。
 - **enp9s0** 接口已与连有公司服务器的 **203.0.113.0/24** 子网相连。
- 内部工作站子网中的主机使用 **10.0.0.1** 作为默认网关。在此流程中，您可以将这个 IP 地址分配给路由器的 **enp8s0** 网络接口。
- 服务器子网中的主机使用 **203.0.113.1** 作为默认网关。在此流程中，您可以将这个 IP 地址分配给路由器的 **enp9s0** 网络接口。
- **firewalld** 服务已启用，并处于活动状态。

1. 将网络接口配置为供应商 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
```

`nmcli connection add` 命令创建 NetworkManager 连接配置文件。该命令使用以下选项：

- **type ethernet** : 定义连接类型为以太网。
 - **con-name <connection_name>** : 设置配置集的名称。使用有意义的名称以避免混淆。
 - **ifname <network_device>**: 设置网络接口。
 - **ipv4.method manual**: 允许配置静态 IP 地址。
 - **ipv4.addresses <IP_address> / <subnet_mask>**: 设置 IPv4 地址和子网掩码。
 - **ipv4.gateway <IP_address>** : 设置默认网关地址。
 - **ipv4.dns <IP_of_DNS_server>** : 设置 DNS 服务器的 IPv4 地址。
 - **connection.zone <firewalld_zone>** : 将网络接口分配给定义的 `firewalld` 区。请注意, `firewalld` 会为分配给 `external` 区域的接口自动启用伪装。
2. 将网络接口配置为供应商 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
```

此命令使用 `ipv4.routes` 参数而不是 `ipv4.gateway` 来设置默认网关。这需要将这个连接的默认网关分配给不同于默认的路由表(5000)。当连接被激活时, NetworkManager 会自动创建这个新的路由表。

3. 将网络接口配置为内部工作站子网 :

```
# 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
```

此命令使用 `ipv4.routes` 参数将静态路由添加到 ID 为 5000 的路由表中。10.0.0.0/24 子网的这个静态路由使用到供应商 B 的本地网络接口的 IP 地址(192.0.2.1)来作为下一跳。

另外, 命令使用 `ipv4.routing-rules` 参数来添加优先级为 5 的路由规则, 该规则将来自 10.0.0.0/24 子网的流量路由到表 5000。低的值具有更高的优先级。

请注意, `ipv4.routing-rules` 参数的语法与 `ip rule add` 命令中的语法相同, 但 `ipv4.routing-rules` 总是需要指定优先级。

4. 将网络接口配置为服务器子网 :

```
# nmcli connection add type ethernet con-name Servers ifname enp9s0 ipv4.method
  manual ipv4.addresses 203.0.113.1/24 connection.zone trusted
```

验证

1. 在内部工作站子网的 RHEL 主机上：

- a. 安装 **traceroute** 软件包：

```
# dnf install traceroute
```

- b. 使用 **traceroute** 工具显示到互联网上主机的路由：

```
# 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
 ...
```

命令的输出显示路由器通过 **192.0.2.1**，即提供商 B 的网络来发送数据包。

2. 在服务器子网的 RHEL 主机上：

- a. 安装 **traceroute** 软件包：

```
# dnf install traceroute
```

- b. 使用 **traceroute** 工具显示到互联网上主机的路由：

```
# 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
 ...
```

命令的输出显示路由器通过 **198.51.100.2**，即供应商 A 的网络来发送数据包。

故障排除步骤

在 RHEL 路由器中：

1. 显示规则列表：

```
# 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
```

默认情况下，RHEL 包含表 **local**、**main** 和 **default** 的规则。

2. 显示表 **5000** 中的路由：

```
# 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. 显示接口和防火墙区：

-

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

4. 验证 **external** 区是否启用了伪装：

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

其他资源

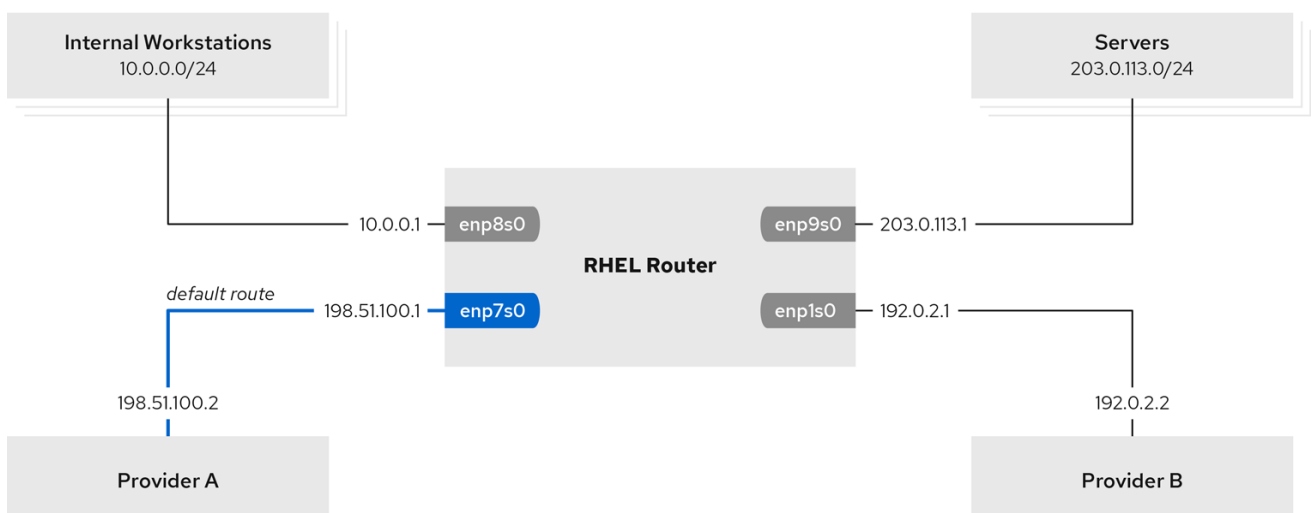
- [nm-settings\(5\) 手册页](#)
- [nmcli\(1\) 手册页](#)

26.2. 使用 NETWORK RHEL 系统角色将特定子网的流量路由到不同的默认网关

您可以使用基于策略的路由为来自特定子网的流量配置不同的默认网关。例如，您可以将 RHEL 配置为默认路由，使用默认路由将所有流量路由到互联网提供商 A。但是，从内部工作站子网接收的流量路由到供应商 B。

要远程和在多个节点上配置基于策略的路由，您可以使用 **network** RHEL 系统角色。

此流程假设以下网络拓扑：



60_RHEL_0120

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 受管节点使用 **NetworkManager** 和 **firewalld** 服务。
- 您要配置的受管节点有 4 个网络接口：
 - **enp7s0** 接口已连接到提供商 A 的网络。提供商网络中的网关 IP 为 **198.51.100.2**，网络使用 **/30** 网络掩码。
 - **enp1s0** 接口连接到提供商 B 的网络。提供商网络中的网关 IP 为 **192.0.2.2**，网络使用 **/30** 网络掩码。
 - **enp8s0** 接口已与连有内部工作站的 **10.0.0.0/24** 子网相连。
 - **enp9s0** 接口已与连有公司服务器的 **203.0.113.0/24** 子网相连。
- 内部工作站子网中的主机使用 **10.0.0.1** 作为默认网关。在此流程中，您可以将这个 IP 地址分配给路由器的 **enp8s0** 网络接口。
- 服务器子网中的主机使用 **203.0.113.1** 作为默认网关。在此流程中，您可以将这个 IP 地址分配给路由器的 **enp9s0** 网络接口。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configuring policy-based routing
  hosts: managed-node-01.example.com
  tasks:
    - name: Routing traffic from a specific subnet to a different default gateway
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: Provider-A
            interface_name: enp7s0
            type: ethernet
            autoconnect: True
            ip:
              address:
                - 198.51.100.1/30
              gateway4: 198.51.100.2
            dns:
              - 198.51.100.200
            state: up
            zone: external

          - name: Provider-B
            interface_name: enp1s0
            type: ethernet
```

```

autoconnect: True
ip:
  address:
    - 192.0.2.1/30
  route:
    - network: 0.0.0.0
      prefix: 0
      gateway: 192.0.2.2
      table: 5000
state: up
zone: external

- name: Internal-Workstations
  interface_name: enp8s0
  type: ethernet
  autoconnect: True
  ip:
    address:
      - 10.0.0.1/24
    route:
      - network: 10.0.0.0
        prefix: 24
        table: 5000
    routing_rule:
      - priority: 5
        from: 10.0.0.0/24
        table: 5000
  state: up
  zone: trusted

- name: Servers
  interface_name: enp9s0
  type: ethernet
  autoconnect: True
  ip:
    address:
      - 203.0.113.1/24
  state: up
  zone: trusted

```

2. 验证 playbook 语法 :

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook :

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

验证

1. 在内部工作站子网的 RHEL 主机上 :
 - a. 安装 **traceroute** 软件包 :

```
# dnf install traceroute
```

- b. 使用 **traceroute** 工具显示到互联网上主机的路由：

```
# 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
 ...
```

命令的输出显示路由器通过 **192.0.2.1**，即提供商 B 的网络来发送数据包。

2. 在服务器子网的 RHEL 主机上：

- a. 安装 **traceroute** 软件包：

```
# dnf install traceroute
```

- b. 使用 **traceroute** 工具显示到互联网上主机的路由：

```
# 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
 ...
```

命令的输出显示路由器通过 **198.51.100.2**，即供应商 A 的网络来发送数据包。

3. 在使用 RHEL 系统角色配置的 RHEL 路由器上：

- a. 显示规则列表：

```
# 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
```

默认情况下，RHEL 包含表 **local**、**main** 和 **default** 的规则。

- b. 显示表 **5000** 中的路由：

```
# 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. 显示接口和防火墙区：

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```


d. 验证 **external** 区是否启用了伪装：

```
# firewall-cmd --info-zone=external
external (active)
target: default
icmp-block-inversion: no
interfaces: enp1s0 enp7s0
sources:
services: ssh
ports:
protocols:
masquerade: yes
...
```

其他资源

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

第 27 章 在不同的接口上重复使用相同的 IP 地址

使用虚拟路由和转发(VRF)，管理员可以在同一主机上同时使用多个路由表。为此，VRF 将网络在第 3 层进行分区。这可让管理员使用每个 VRF 域的独立路由表隔离流量。这个技术与虚拟 LAN（虚拟 LAN）类似，后者在第二层对网络进行分区，其中操作系统使用不同的 VLAN 标签来隔离共享相同物理介质的流量。

与第二层上的分区相比，VRF 的一个优点是，考虑到所涉及的对等路由的数量，路由可以更好地扩展。

Red Hat Enterprise Linux 为每个 VRF 域使用虚拟 **vrt** 设备，并通过向 VRF 设备添加现有网络设备来向 VRF 域添加路由。之前附加到原始设备的地址和路由将在 VRF 域中移动。

请注意，每个 VRF 域间都是相互隔离的。

27.1. 在不同接口上永久重复使用相同的 IP 地址

您可以使用虚拟路由和转发(VRF)功能来对一个服务器的不同接口永久使用同样的 IP 地址。



重要

要在重新使用相同的 IP 地址时让远程对等两个 VRF 接口都联系，网络接口必须属于不同的广播域。网络中的广播域是一组节点，它们接收其中任何一个节点发送的广播流量。在大多数配置中，所有连接到同一交换机的节点都属于相同的域。

先决条件

- 以 **root** 用户身份登录。
- 没有配置网络接口。

流程

1. 创建并配置第一个 VRF 设备：

- 为 VRF 设备创建连接并将其分配到路由表中。例如，要创建一个分配给 **1001** 路由表、名为 **vrf0** 的 VRF 设备：

```
# nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1001 ipv4.method disabled ipv6.method disabled
```

- 启用 **vrf0** 设备：

```
# nmcli connection up vrf0
```

- 为刚刚创建的 VRF 分配网络设备。例如，要向 **vrf0** VRF 设备添加 **enp1s0** 以太网设备，并向 **enp1s0** 分配 IP 地址和子网掩码，请输入：

```
# nmcli connection add type ethernet con-name vrf.enp1s0 ifname enp1s0 controller vrf0 ipv4.method manual ipv4.address 192.0.2.1/24
```

- 激活 **vrf.enp1s0** 连接：

```
# nmcli connection up vrf.enp1s0
```

2. 创建并配置下一个 VRF 设备：

- a. 创建 VRF 设备并将其分配到路由表中。例如，要创建一个分配给 **1002** 路由表、名为 **vrf1** 的 VRF 设备，请输入：

```
# nmcli connection add type vrf ifname vrf1 con-name vrf1 table 1002 ipv4.method disabled ipv6.method disabled
```

- b. 激活 **vrf1** 设备：

```
# nmcli connection up vrf1
```

- c. 为刚刚创建的 VRF 分配网络设备。例如，要向 **vrf1** VRF 设备添加 **enp7s0** 以太网设备，并给 **enp7s0** 分配 IP 地址和子网掩码，请输入：

```
# nmcli connection add type ethernet con-name vrf.enp7s0 ifname enp7s0 controller vrf1 ipv4.method manual ipv4.address 192.0.2.1/24
```

- d. 激活 **vrf.enp7s0** 设备：

```
# nmcli connection up vrf.enp7s0
```

27.2. 在不同接口中临时重复使用相同的 IP 地址

您可以使用虚拟路由和转发(VRF)功能来对一个服务器的不同接口临时使用同样的 IP 地址。这个过程仅用于测试目的，因为配置是临时的并在重启系统后会丢失。



重要

要在重新使用相同的 IP 地址时让远程对等两个 VRF 接口都联系，网络接口必须属于不同的广播域。广播域是一组节点，它们接收被其中任何一个发送的广播流量。在大多数配置中，所有连接到同一交换机的节点都属于相同的域。

先决条件

- 以 **root** 用户身份登录。
- 没有配置网络接口。

流程

1. 创建并配置第一个 VRF 设备：

- a. 创建 VRF 设备并将其分配到路由表中。例如，要创建一个分配给 **1001** 路由表、名为 **blue** 的 VRF 设备：

```
# ip link add dev blue type vrf table 1001
```

- b. 启用 **blue** 设备：

```
# ip link set dev blue up
```

- c. 为 VRF 设备分配网络设备。例如，要向 **blue** VRF 设备添加 **enp1s0** 以太网设备：

```
# ip link set dev enp1s0 master blue
```

- d. 启用 **enp1s0** 设备：

```
# ip link set dev enp1s0 up
```

- e. 向 **enp1s0** 设备分配 IP 地址和子网掩码。例如，将其设为 **192.0.2.1/24**：

```
# ip addr add dev enp1s0 192.0.2.1/24
```

2. 创建并配置下一个 VRF 设备：

- a. 创建 VRF 设备并将其分配到路由表中。例如，要创建一个分配给 **1002** 路由表、名为 **red** 的 VRF 设备：

```
# ip link add dev red type vrf table 1002
```

- b. 启用 **red** 设备：

```
# ip link set dev red up
```

- c. 为 VRF 设备分配网络设备。例如，要向 **red** VRF 设备添加 **enp7s0** 以太网设备：

```
# ip link set dev enp7s0 master red
```

- d. 启用 **enp7s0** 设备：

```
# ip link set dev enp7s0 up
```

- e. 为 **enp7s0** 设备分配与 **blue** VRF 域中 **enp1s0** 设备所使用的相同的 IP 地址和子网掩码：

```
# ip addr add dev enp7s0 192.0.2.1/24
```

3. 另外，还可按照上述步骤创建更多 VRF 设备。

27.3. 其他资源

- **kernel-doc** 软件包的 `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vrf.txt`

第 28 章 在隔离的 VRF 网络内启动服务

使用虚拟路由和转发(VRF)，您可以使用与操作系统主路由表不同的路由表创建隔离网络。然后，您可以启动服务和应用程序，以便它们只能访问该路由表中定义的网络。

28.1. 配置 VRF 设备

要使用虚拟路由和转发(VRF)，您可以创建一个 VRF 设备，并将物理或虚拟网络接口和路由信息附加给它。



警告

要防止您将自己远程锁定，请在本地控制台中或通过您不想分配给 VRF 设备的网络接口远程执行此流程。

先决条件

- 您已在本地登录或使用与您要分配给 VRF 设备不同的网络接口。

步骤

1. 使用同名的虚拟设备创建 **vrf0** 连接，并将其附加到路由表 **1000**：

```
# nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1000 ipv4.method disabled ipv6.method disabled
```

2. 向 **vrf0** 连接添加 **enp1s0** 设备，并配置 IP 设置：

```
# nmcli connection add type ethernet con-name enp1s0 ifname enp1s0 controller vrf0 ipv4.method manual ipv4.address 192.0.2.1/24 ipv4.gateway 192.0.2.254
```

此命令会创建 **enp1s0** 连接，来作为 **vrf0** 连接的端口。由于此配置，路由信息会自动分配给与 **vrf0** 设备关联的路由表 **1000**。

3. 如果您在隔离网络中需要静态路由：

- a. 添加静态路由：

```
# nmcli connection modify enp1s0 +ipv4.routes "198.51.100.0/24 192.0.2.2"
```

这向 **198.51.100.0/24** 网络添加了一个路由，该网络使用 **192.0.2.2** 作为路由器。

- b. 激活连接：

```
# nmcli connection up enp1s0
```

验证

1. 显示与 **vrf0** 关联的设备的 IP 设置：

```
# ip -br addr show vrf vrf0
enp1s0 UP 192.0.2.1/24
```

2. 显示 VRF 设备及其关联的路由表：

```
# ip vrf show
Name          Table
-----
vrf0          1000
```

3. 显示主路由表：

```
# ip route show
default via 203.0.113.0/24 dev enp7s0 proto static metric 100
```

主路由表没有提到任何与设备 **enp1s0** 或 **192.0.2.1/24** 子网关联的路由。

4. 显示路由表 **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
```

default 条目表示使用此路由表的服务，将 **192.0.2.254** 用作其默认网关，而不是主路由表中的默认网关。

5. 在与 **vrf0** 关联的网络中执行 **traceroute** 工具，以验证工具是否使用表 **1000** 的路由：

```
# 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
 ...
```

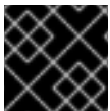
第一跳是分配给路由表 **1000** 的默认网关，而不是系统的主路由表中的默认网关。

其他资源

- [ip-vrf\(8\) 手册页](#)

28.2. 在隔离的 VRF 网络内启动服务

您可以将服务（如 Apache HTTP 服务器）配置为在隔离的虚拟路由和转发(VRF)网络中启动。



重要

服务只能绑定到同一 VRF 网络中的本地 IP 地址。

先决条件

- 您已配置了 **vrf0** 设备。
- 您已将 Apache HTTP 服务器配置为仅侦听分配给与 **vrf0** 设备关联的接口的 IP 地址。

步骤

1. 显示 **httpd** systemd 服务的内容：

```
# systemctl cat httpd
...
[Service]
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
...
```

在后续步骤中需要 **ExecStart** 参数的内容，以在隔离的 VRF 网络中运行相同的命令。

2. 创建 **/etc/systemd/system/httpd.service.d/** 目录：

```
# mkdir /etc/systemd/system/httpd.service.d/
```

3. 使用以下内容创建 **/etc/systemd/system/httpd.service.d/override.conf** 文件：

```
[Service]
ExecStart=
ExecStart=/usr/sbin/ip vrf exec vrf0 /usr/sbin/httpd $OPTIONS -DFOREGROUND
```

要覆盖 **ExecStart** 参数，您首先需要对其取消设置，然后将其设为所示的新值。

4. 重新加载 **systemd**。

```
# systemctl daemon-reload
```

5. 重新启动 **httpd** 服务。

```
# systemctl restart httpd
```

验证

1. 显示 **httpd** 进程的进程 ID(PID)：

```
# pidof -c httpd
1904 ...
```

2. 显示 PID 的 VRF 关联，例如：

```
# ip vrf identify 1904
vrf0
```

3. 显示与 **vrf0** 设备关联的所有 PID：

```
# ip vrf pids vrf0
1904 httpd
...
```

其他资源

- [ip-vrf\(8\) 手册页](#)

第 29 章 在 NETWORKMANAGER 连接配置文件中配置 ETHTOOL 设置

NetworkManager 可以永久配置某些网络驱动程序和硬件设置。与使用 **ethtool** 工具管理这些设置相比，这有重启后不会丢失设置的好处。

您可以在 NetworkManager 连接配置文件中设置以下 **ethtool** 设置：

卸载功能

网络接口控制器可以使用 TCP 卸载引擎(TOE)将处理某些操作卸载到网络接口控制器。这提高了网络吞吐量。

中断合并设置

通过使用中断合并，系统收集网络数据包，并为多个数据包生成一个中断。这会增加使用一个硬件中断发送到内核的数据量，从而减少中断负载，并最大化吞吐量。

环缓冲区

这些缓冲区存储传入和传出的网络数据包。您可以增加环缓冲的大小，来减少高数据包丢弃率。

频道设置

网络接口管理其关联的频道数以及硬件设置和网络驱动程序。与网络接口关联的所有设备都通过中断请求(IRQ)相互通信。每个设备队列保存待处理的 IRQ，并通过称为频道的数据行相互通信。队列类型与特定频道类型相关联。这些频道类型包括：

- 用于接收队列的 **rx**
- 用于传输队列的 **tx**
- **其他** 用于链路中断或单一根输入/输出虚拟化(SR-IOV)协调
- **组合** 用于基于硬件容量的多用途频道

29.1. 使用 NMCLI 配置 ETHTOOL OFFLOAD 功能

您可以使用 NetworkManager 来在连接配置文件中启用和禁用 **ethtool** 卸载功能。

步骤

1. 例如：要启用 RX 卸载特性，并在 **enp1s0** 连接配置文件中禁用 TX 卸载，请输入：

```
# nmcli con modify enp1s0 ethtool.feature-rx on ethtool.feature-tx off
```

这个命令明确启用 RX 卸载并禁用 TX 卸载功能。

2. 要删除之前启用或禁用的卸载功能的设置，请将功能的参数设置为 null 值。例如，要删除 TX 卸载的配置，请输入：

```
# nmcli con modify enp1s0 ethtool.feature-tx ""
```

3. 重新激活网络配置集：

```
# nmcli connection up enp1s0
```

短址

- 使用 **ethtool -k** 命令显示网络设备的当前卸载特性：

```
# ethtool -k network_device
```

其他资源

- [nm-settings-nmcli\(5\) 手册页](#)

29.2. 使用 NETWORK RHEL 系统角色配置 ETHTOOL OFFLOAD 功能

您可以使用 **network** RHEL 系统角色配置 NetworkManager 连接的 **ethtool** 功能。



重要

当您运行一个使用 **network** RHEL 系统角色的 play 时，如果设置值与 play 中指定的值不匹配，则角色会使用相同的名称覆盖现有的连接配置文件。要防止将这些值重置为其默认值，请始终在 play 中指定网络连接配置文件的整个配置，即使配置（如 IP 配置）已存在。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool features
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 198.51.100.20/24
                - 2001:db8:1::1/64
            gateway4: 198.51.100.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 198.51.100.200
              - 2001:db8:1::ffbb
            dns_search:
```

```

- example.com
ethtool:
  features:
    gro: "no"
    gso: "yes"
    tx_sctp_segmentation: "no"
state: up

```

此 playbook 使用以下设置创建 **enp1s0** 连接配置文件，或者如果配置文件已存在就更新它：

- 静态 IPv4 地址 - **198.51.100.20**，子网掩码为 **/24**
- 静态 IPv6 地址 - **2001:db8:1::1**，子网掩码为 **/64**
- IPv4 默认网关 - **198.51.100.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **198.51.100.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- **ethtool** 功能：
 - 通用接收卸载(GRO)：禁用
 - 通用分段卸载(GSO)：启用
 - TX 流控制传输协议(SCTP)分段：禁用

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

29.3. 使用 NMCLI 配置 ETHTOOL COALESCE 设置

您可以使用 NetworkManager 来在连接配置文件中设置 **ethtool** 合并设置。

步骤

1. 例如，要在 **enp1s0** 连接配置文件中将接收的数据包的最大数量设置为延迟到 **128**，请输入：

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames 128
```

- 要删除合并设置，请将其设置为 null 值。例如，要删除 **ethtool.coalesce-rx-frames** 设置，请输入：

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames ""
```

- 重新激活网络配置集：

```
# nmcli connection up enp1s0
```

验证

- 使用 **ethtool -c** 命令显示网络设备的当前卸载特性：

```
# ethtool -c network_device
```

其他资源

- [nm-settings-nmcli\(5\) 手册页](#)

29.4. 使用 NETWORK RHEL 系统角色配置 ETHTOOL COALESCE 设置

您可以使用 **network** RHEL 系统角色配置 NetworkManager 连接的 **ethtool** coalesce 设置。



重要

当您运行一个使用 **network** RHEL 系统角色的 play 时，如果设置值与 play 中指定的值不匹配，则角色会使用相同的名称覆盖现有的连接配置文件。要防止将这些值重置为其默认值，请始终在 play 中指定网络连接配置文件的整个配置，即使配置（如 IP 配置）已存在。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。

步骤

- 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with ethtool coalesce settings
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
```

```

network_connections:
- name: enp1s0
  type: ethernet
  autoconnect: yes
  ip:
    address:
      - 198.51.100.20/24
      - 2001:db8:1::1/64
    gateway4: 198.51.100.254
    gateway6: 2001:db8:1::fffe
  dns:
    - 198.51.100.200
    - 2001:db8:1::ffbb
  dns_search:
    - example.com
  ethtool:
    coalesce:
      rx_frames: 128
      tx_frames: 128
  state: up

```

此 playbook 使用以下设置创建 **enp1s0** 连接配置文件，或者如果配置文件已存在就更新它：

- 静态 IPv4 地址 - **198.51.100.20**，子网掩码为 **/24**
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码
- IPv4 默认网关 - **198.51.100.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **198.51.100.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- **ethtool** coalesce 设置：
 - RX 帧： **128**
 - TX 帧： **128**

2. 验证 playbook 语法：

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook：

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

其他资源

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

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

29.5. 使用 NMCLI 增加环缓冲的大小，以减少数据包丢弃率

如果数据包丢失率导致应用程序报告数据丢失、超时或其他问题，则可以增加以太网设备的环缓冲区的大小。

接收环缓冲在设备驱动程序和网络接口控制器(NIC)之间共享。该卡分配一个传输(TX)和接收(RX)环缓冲。名称意味着，环缓冲是循环缓冲区，溢出会覆盖现有数据。可以通过两种方法将数据从 NIC 移至内核，硬件中断和软件中断也称为 SoftIRQ。

内核使用 RX 环缓冲区来存储传入的数据包，直到设备驱动程序可以处理它们。设备驱动程序排空 RX 环，通常使用 SoftIRQ，其将传入的数据包置于名为 `sk_buff` 或 `skb` 的内核数据结构中，以通过内核直至拥有相关套接字的应用程序开始其过程。

内核使用 TX 环缓冲区来存放应发送到网络的传出数据包。这些环缓冲位于堆栈的底部，是可能会发生数据包丢弃的关键点，进而会对网络性能造成负面影响。

步骤

1. 显示接口的数据包丢失统计信息：

```
# ethtool -S enp1s0
...
rx_queue_0_drops: 97326
rx_queue_1_drops: 63783
...
```

请注意，命令的输出取决于网卡和驱动程序。

discard 或 **drop** 计数器中的高值表示可用缓冲区的填满速度快于内核可以处理数据包的速度。增加环缓冲有助于避免此类丢失。

2. 显示最大环缓冲大小：

```
# ethtool -g enp1s0
Ring 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
```

如果 **Pre-set maximums** 部分中的值大于 **Current hardware settings** 部分中的值，您可以在下一步中更改设置。

3. 确定使用接口的 NetworkManager 连接配置文件：

```
# nmcli connection show
```

```
NAME          UUID                               TYPE    DEVICE
Example-Connection a5eb6490-cc20-3668-81f8-0314a27f3f75 ethernet enp1s0
```

4. 更新连接配置文件，并增加环缓冲区：

- 要增加 RX 环缓冲区，请输入：

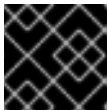
```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

- 要增加 TX 环缓冲区，请输入：

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

5. 重新载入 NetworkManager 连接：

```
# nmcli connection up Example-Connection
```



重要

根据 NIC 使用的驱动程序，环缓冲区中的更改可能会很快中断网络连接。

其他资源

- [ifconfig 和 ip 命令报告数据包丢弃](#)
- [我是否应该关注 0.05% 的数据包丢弃率？](#)
- [ethtool\(8\) 手册页](#)

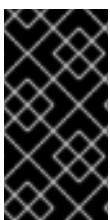
29.6. 使用 NETWORK RHEL 系统角色增加环缓冲的大小，以减少数据包丢弃率

如果数据包丢失率导致应用程序报告数据丢失、超时或其他问题，则可以增加以太网设备的环缓冲区的大小。

环缓冲区是循环缓冲区，溢出会覆盖现有数据。网卡分配一个传输(TX)和接收(RX)环缓冲区。接收环缓冲在设备驱动程序和网络接口控制器(NIC)之间共享。数据可以通过硬件中断或软件中断（也称为 SoftIRQ）从 NIC 移到内核。

内核使用 RX 环缓冲区来存储传入的数据包，直到设备驱动程序可以处理它们。设备驱动程序排空 RX 环，通常使用 SoftIRQ，其将传入的数据包置于名为 `sk_buff` 或 `skb` 的内核数据结构中，以通过内核直至拥有相关套接字的应用程序开始其过程。

内核使用 TX 环缓冲区来存放应发送到网络的传出数据包。这些环缓冲位于堆栈的底部，是可能会发生数据包丢弃的关键点，进而会对网络性能造成负面影响。



重要

当您运行一个使用 `network` RHEL 系统角色的 play 时，如果设置值与 play 中指定的值不匹配，则角色会使用相同的名称覆盖现有的连接配置文件。要防止将这些值重置为其默认值，请始终在 play 中指定网络连接配置文件的整个配置，即使配置（如 IP 配置）已存在。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 您知道设备支持的最大环缓冲区大小。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml** :

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure an Ethernet connection with increased ring buffer sizes
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 198.51.100.20/24
                - 2001:db8:1::1/64
              gateway4: 198.51.100.254
              gateway6: 2001:db8:1::fffe
            dns:
              - 198.51.100.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
            ethtool:
              ring:
                rx: 4096
                tx: 4096
            state: up
```

此 playbook 使用以下设置创建 **enp1s0** 连接配置文件，或者如果配置文件已存在就更新它：

- 静态 IPv4 地址 - **198.51.100.20**，子网掩码为 **/24**
- 静态 IPv6 地址 - **2001:db8:1::1**，子网掩码为 **/64**
- IPv4 默认网关 - **198.51.100.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **198.51.100.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**

- DNS 搜索域 - **example.com**
 - 环缓冲区条目的最大数量 :
 - 接收(RX) : 4096
 - 传输(TX) : 4096
2. 验证 playbook 语法 :

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

请注意, 这个命令只验证语法, 不会防止错误, 但保护有效配置。

3. 运行 playbook :

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

其他资源

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

29.7. 使用 NMCLI 配置 ETHTOOL 频道设置

通过使用 NetworkManager, 您可以管理网络设备和连接。**ethtool** 工具管理网络接口卡的链接速度和相关设置。**ethtool** 处理与相关设备的 IRQ 通信, 以管理连接配置文件中的相关频道设置。

步骤

1. 显示与网络设备关联的频道 :

```
# ethtool --show-channels enp1s0
Channel parameters for enp1s0:
Pre-set maximums:
RX: 4
TX: 3
Other: 10
Combined: 63

Current hardware settings:
RX: 1
TX: 1
Other: 1
Combined: 1
```

2. 更新网络接口的频道设置 :

```
# nmcli connection modify enp1s0 ethtool.channels-rx 4 ethtool.channels-tx 3
ethtools.channels-other 9 ethtool.channels-combined 50
```

3. 重新激活网络配置集 :

```
# nmcli connection up enp1s0
```

验证

- 检查与网络设备关联的更新频道设置：

```
# ethtool --show-channels enp1s0  
Channel parameters for enp1s0:  
Pre-set maximums:  
RX: 4  
TX: 3  
Other: 10  
Combined: 63  
  
Current hardware settings:  
RX: 4  
TX: 3  
Other: 9  
Combined: 50
```

其他资源

- [nmcli \(5\) 手册页](#)

第 30 章 网络管理器调试介绍

增加所有或某些域的日志级别有助于记录 NetworkManager 所执行的操作的更多详情。您可以使用这些信息排除问题。NetworkManager 提供不同的级别和域来生成日志信息。`/etc/NetworkManager/NetworkManager.conf` 文件是 NetworkManager 的主配置文件。日志存储在日志中。

30.1. NETWORKMANAGER 重新应用方法的简介

NetworkManager 服务使用配置文件来管理设备的连接设置。桌面总线(D-Bus) API 可以创建、修改和删除这些连接设置。对于配置文件中的任何更改，D-Bus API 会将现有的设置克隆到连接的修改设置中。虽然克隆，但更改不会应用到修改后的设置。要使它生效，请重新激活连接的现有设置或使用 **reapply ()** 方法。

reapply () 方法具有以下功能：

1. 在不停用或重启网络接口的情况下更新修改后的连接设置。
2. 从修改后的连接设置中删除待处理的更改。因为 **NetworkManager** 不会恢复手动更改，因此您可以重新配置该设备并恢复外部或手动参数。
3. 创建与现有连接设置不同的修改的连接设置。

另外，**reapply ()** 方法支持以下属性：

- **bridge.ageing-time**
- **bridge.forward-delay**
- **bridge.group-address**
- **bridge.group-forward-mask**
- **bridge.hello-time**
- **bridge.max-age**
- **bridge.multicast-hash-max**
- **bridge.multicast-last-member-count**
- **bridge.multicast-last-member-interval**
- **bridge.multicast-membership-interval**
- **bridge.multicast-querier**
- **bridge.multicast-querier-interval**
- **bridge.multicast-query-interval**
- **bridge.multicast-query-response-interval**
- **bridge.multicast-query-use-ifaddr**
- **bridge.multicast-router**

- **bridge.multicast-snooping**
- **bridge.multicast-startup-query-count**
- **bridge.multicast-startup-query-interval**
- **bridge.priority**
- **bridge.stp**
- **bridge.VLAN-filtering**
- **bridge.VLAN-protocol**
- **bridge.VLANs**
- **802-3-ethernet.accept-all-mac-addresses**
- **802-3-ethernet.cloned-mac-address**
- **IPv4.addresses**
- **IPv4.dhcp-client-id**
- **IPv4.dhcp-iaid**
- **IPv4.dhcp-timeout**
- **IPv4.DNS**
- **IPv4.DNS-priority**
- **IPv4.DNS-search**
- **IPv4.gateway**
- **IPv4.ignore-auto-DNS**
- **IPv4.ignore-auto-routes**
- **IPv4.may-fail**
- **IPv4.method**
- **IPv4.never-default**
- **IPv4.route-table**
- **IPv4.routes**
- **IPv4.routing-rules**
- **IPv6.addr-gen-mode**
- **IPv6.addresses**
- **IPv6.dhcp-duid**

- IPv6.dhcp-iaid
- IPv6.dhcp-timeout
- IPv6.DNS
- IPv6.DNS-priority
- IPv6.DNS-search
- IPv6.gateway
- IPv6.ignore-auto-DNS
- IPv6.may-fail
- IPv6.method
- IPv6.never-default
- IPv6.ra-timeout
- IPv6.route-metric
- IPv6.route-table
- IPv6.routes
- IPv6.routing-rules

其他资源

- [nm-settings-nmcli\(5\) 手册页](#)

30.2. 设置 NETWORKMANAGER 日志级别

默认情况下，所有日志域都设为记录 **INFO** 日志级别。在收集调试日志前禁用速率限制。通过速率限制，如果短时间内有太多信息，**systemd-journald** 会丢弃它们。当日志级别为 **TRACE** 时，可能会发生这种情况。

此流程禁用速率限制，并为所有（ALL）域启用记录调试日志。

步骤

1. 要禁用速率限制，请编辑 `/etc/systemd/journald.conf` 文件，取消 **[Journal]** 部分中的 **RateLimitBurst** 参数的注释，并将其值设为 **0**：

```
RateLimitBurst=0
```

2. 重启 **systemd-journald** 服务。

```
# systemctl restart systemd-journald
```

3. 使用以下内容创建 `/etc/NetworkManager/conf.d/95-nm-debug.conf` 文件：

```
[logging]
domains=ALL:TRACE
```

domains 参数可以包含多个用逗号分隔的 **domain:level** 对。

4. 重启 NetworkManager 服务。

```
# systemctl restart NetworkManager
```

验证

- 查询 **systemd** 日志以显示 **NetworkManager** 单元的日志条目：

```
# 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
...
```

30.3. 使用 nmcli 在运行时临时设置日志级别

您可以使用 **nmcli** 在运行时更改日志级别。

步骤

1. 可选：显示当前的日志设置：

```
# 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. 要修改日志级别和域，请使用以下选项：

- 要将所有域的日志级别都设为同样的 **LEVEL**，请输入：

```
# nmcli general logging level LEVEL domains ALL
```

- 要更改特定域的级别，请输入：

```
# nmcli general logging level LEVEL domains DOMAINS
```

请注意，使用这个命令更新日志级别会禁用所有其他域的日志功能。

- 要更改特定域的级别并保持其它域的级别，请输入：

```
# nmcli general logging level KEEP domains DOMAIN:LEVEL,DOMAIN:LEVEL
```

30.4. 查看 NETWORKMANAGER 日志

您可以查看 NetworkManager 日志进行故障排除。

流程

- 要查看日志，请输入：

```
# journalctl -u NetworkManager -b
```

其他资源

- [NetworkManager.conf\(5\) man page](#)
- [journalctl\(1\) 手册页](#)

30.5. 调试级别和域

您可以使用 **levels** 和 **domains** 参数来管理 NetworkManager 的调试。levels 定义了详细程度，而 domains 定义了消息的类别，以记录给定的严重程度（级别）的日志。

日志级别	描述
OFF	不记录任何有关 NetworkManager 的信息
ERR	仅记录严重错误
WARN	记录可以反映操作的警告信息
INFO	记录各种有助于跟踪状态和操作的信息
DEBUG	为调试启用详细日志记录
TRACE	启用比 DEBUG 级别更详细的日志

请注意，后续的级别记录来自以前级别的所有信息。例如，将日志级别设为 **INFO** 也会记录包含在 **ERR** 和 **WARN** 日志级别中的消息。

其他资源

- [NetworkManager.conf\(5\) man page](#)

第 31 章 使用 LLDP 来调试网络配置问题

您可以使用链路层发现协议(LLDP)来调试拓扑中的网络配置问题。这意味着 LLDP 可以报告与其他主机或路由器以及交换机的配置不一致。

31.1. 使用 LLDP 信息调试不正确的 VLAN 配置

如果您将交换机端口配置为使用特定的 VLAN，而主机没有收到这些 VLAN 数据包，则您可以使用链路层发现协议(LLDP)来调试问题。在没有收到数据包的主机上执行这个流程。

先决条件

- **nmstate** 软件包已安装。
- 交换机支持 LLDP。
- LLDP 在邻居设备上已启用。

流程

1. 使用以下内容创建 `~/enable-LLDP-enp1s0.yml` 文件：

```
interfaces:
  - name: enp1s0
    type: ethernet
    lldp:
      enabled: true
```

2. 使用 `~/enable-LLDP-enp1s0.yml` 文件来在接口 **enp1s0** 上启用 LLDP：

```
# nmstatectl apply ~/enable-LLDP-enp1s0.yml
```

3. 显示 LLDP 信息：

```
# 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

```

- 验证输出，以确保设置与您预期的配置匹配。例如，连接到交换机的接口的 LLDP 信息显示此主机连接的交换机端口使用 VLAN ID **448**：

```

- type: 127
  ieee-802-1-vlans:
  - name: v2-0488-03-0505
    vid: 488

```

如果 **enp1s0** 接口的网络配置使用不同的 VLAN ID，请相应地进行修改。

其他资源

[配置 VLAN 标记](#)

第 32 章 LINUX 流量控制

Linux 提供管理和操作数据包传输的工具。Linux 流量控制 (TC) 子系统帮助进行策略、分类、控制以及调度网络流量。TC 还可以通过使用过滤器和动作在分类过程中利用数据包内容分栏。TC 子系统通过使用排队规则(**qdisc**)这个 TC 架构的基本元素来达到此目的。

调度机制在进入或退出不同的队列前确定或者重新安排数据包。最常见的调度程序是先入先出 (FIFO) 调度程序。您可以使用 **tc** 工具临时执行 **qdiscs** 操作，也可以使用 NetworkManager 永久执行操作。

在 Red Hat Enterprise Linux 中，您可以使用各种方法配置默认的排队规则来管理网络接口上的流量。

32.1. 排队规则概述

排队规则(**qdiscs**)帮助排队，之后通过网络接口调度流量传输。**qdisc** 有两个操作：

- 排队请求，以便在以后传输时对数据包进行排队
- 出队请求，以便可以选择其中一个排队的数据包进行即时传输。

每个 **qdisc** 都有一个 16 位十六进制标识数字，称为 **句柄**，带有一个附加的冒号，如 **1:** 或 **abcd:** 这个数字被称为 **qdisc** 主号码。如果 **qdisc** 有类，则标识符是由两个数字组成的对，主号码在次要号码之前，即 **<major>:<minor>**，例如 **abcd:1**。次要号码的编号方案取决于 **qdisc** 类型。有时，编号是系统化的，其中第一类的 ID 为 **<major>:1**，第二类的 ID 为 **<major>:2**，等等。一些 **qdiscs** 允许用户在创建类时随机设置类的次要号码。

类 **qdiscs**

存在不同类型的 **qdiscs**，帮助向和从网络接口传输数据包。您可以使用根、父或子类配置 **qdiscs**。子对象可以被附加的位置称为类。**qdisc** 中的类是灵活的，始终包含多个子类或一个子类 **qdisc**。禁止包含类 **qdisc** 本身的类，这有助于实现复杂的流量控制场景。

类 **qdiscs** 本身不存储任何数据包。相反，它们根据特定于 **qdisc** 的标准，将排队和出队请求传到它们其中的一个子类。最后，这个递归数据包传递最终结束保存数据包的位置（在出现排队时从中提取）。

无类别 **qdiscs**

有些 **qdiscs** 不包含子类，它们称为无类别 **qdiscs**。与类 **qdiscs** 相比，无类别 **qdiscs** 需要较少的自定义。通常情况下，将它们附加到接口就足够了。

其他资源

- **tc(8)** 手册页
- **tc-actions(8)** 手册页

32.2. 连接跟踪简介

在防火墙中，**Netfilter** 框架过滤来自外部网络的数据包。数据包到达后，**Netfilter** 分配一个连接跟踪条目。连接跟踪是逻辑网络的一个 Linux 内核网络功能，其跟踪这些连接中的连接，并识别数据包流。此功能过滤并分析每个数据包，设置连接跟踪表以存储连接状态，并根据识别的数据包更新连接状态。例如，如果是 FTP 连接，**Netfilter** 会分配一个连接跟踪条目，以确保 FTP 连接的所有数据包都以同样的方式工作。连接跟踪条目存储一个 **Netfilter** 标记，并在内存表中跟踪连接状态信息，其中新数据包元组与现有条目进行映射。如果数据包元组没有与现有条目进行映射，则数据包会添加一个新的连接跟踪条目，该条目将对同一连接的数据包进行分组。

您可以控制和分析网络接口上的流量。**tc** 流量控制器工具使用 **qdisc** 规程配置网络中的数据包调度程序。**qdisc** 内核配置的排队规程将数据包排队到接口。通过使用 **qdisc**，内核在网络接口传输流量前捕获所有流量。另外，要限制属于同一连接的数据包的带宽率，请使用 **tc qdisc** 命令。

要把连接跟踪标记中的数据检索到各个字段，请使用带有 **ctinfo** 模块的 **tc** 工具以及 **connmark** 功能。对于存储数据包标记信息，**ctinfo** 模块将 **Netfilter** 标记和连接状态信息复制到套接字缓冲区(**skb**)标记元数据字段中。

通过物理介质传输数据包会删除数据包的所有元数据。在数据包丢失其元数据之前，**ctinfo** 模块会将 **Netfilter** 标记值映射并复制到数据包 **IP** 字段中的 Diffserv 代码点(DSCP)的特定值。

其他资源

- **tc(8)** 和 **tc-ctinfo (8)** 手册页

32.3. 使用 TC 工具检查网络接口的 QDISCS

默认情况下，Red Hat Enterprise Linux 系统使用 **fq_codel qdisc**。您可以使用 **tc** 工具检查 **qdisc** 计数器。

步骤

1. 可选：查看您当前的 **qdisc**：

```
# tc qdisc show dev enp0s1
```

2. 检查当前的 **qdisc** 计数器：

```
# 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** - 由于所有队列已满而丢弃数据包的次数
- **overlimits** - 配置的链路容量已满的次数
- **sent** - 出队的数量

32.4. 更新默认的 QDISC

如果使用当前的 **qdisc** 观察网络数据包丢失情况，您可以根据您的网络要求更改 **qdisc**。

步骤

1. 查看当前的默认的 **qdisc**：

```
# sysctl -a | grep qdisc
net.core.default_qdisc = fq_codel
```

2. 查看当前以太网连接的 **qdisc**：

```
# 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
```

- 更新现有的 **qdisc** :

```
# sysctl -w net.core.default_qdisc=pfifo_fast
```

- 要应用更改，请重新加载网络驱动程序：

```
# modprobe -r NETWORKDRIVERNAME
# modprobe NETWORKDRIVERNAME
```

- 启动网络接口：

```
# ip link set enp0s1 up
```

验证

- 查看以太网连接的 **qdisc** :

```
# tc -s qdisc show dev enp0s1
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
Sent 373186 bytes 5333 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
....
```

其他资源

- 如何在 [Red Hat Enterprise Linux](#) 上设置 **sysctl** 变量

32.5. 使用 TC 工具临时设置网络接口的当前 QDISC

您可以更新当前的 **qdisc** 而不更改默认的 **qdisc**。

步骤

- 可选：查看当前的 **qdisc** :

```
# tc -s qdisc show dev enp0s1
```

- 更新当前的 **qdisc** :

```
# tc qdisc replace dev enp0s1 root htb
```

验证

- 查看更新后的当前 **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
```

32.6. 使用 NETWORKMANAGER 永久设置网络接口的当前 QDISC

您可以更新 NetworkManager 连接当前的 **qdisc** 值。

步骤

1. 可选：查看当前的 **qdisc**：

```
# tc qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2
```

2. 更新当前的 **qdisc**：

```
# nmcli connection modify enp0s1 tc.qdiscs 'root pfifo_fast'
```

3. 可选：要在现有的 **qdisc** 上添加另一个 **qdisc**，请使用 **+tc.qdisc** 选项：

```
# nmcli connection modify enp0s1 +tc.qdisc 'ingress handle ffff'
```

4. 激活更改：

```
# nmcli connection up enp0s1
```

验证

- 查看网络接口当前的 **qdisc**：

```
# 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 -----
```

其他资源

- [nm-settings\(5\) 手册页](#)

32.7. 使用 TC-CTINFO 工具配置数据包的速率限制

您可以使用速率限制来限制网络流量，并防止网络中资源耗尽。通过速率限制，您还可以通过限制特定时间段内重复的数据包请求来减少服务器上的负载。另外，您可以通过使用 **tc-ctinfo** 工具在内核中配置流量控制来管理带宽率。

连接跟踪条目存储 **Netfilter** 标记和连接信息。当路由器转发来自防火墙的数据包时，路由器会删除或修改数据包的连接跟踪条目。连接跟踪信息(**ctinfo**)模块将数据从连接跟踪标记检索到各个字段。此内核模块通过将 **Netfilter** 标记复制到套接字缓冲区(**skb**)标记元数据字段来保留 **Netfilter** 标记。

先决条件

- **iperf3** 工具安装在服务器和客户端上。

步骤

1. 在服务器上执行以下步骤：

- a. 在网络接口中添加一个虚拟链接：

```
# ip link add name ifb4eth0 numtxqueues 48 numrxqueues 48 type ifb
```

这个命令有以下参数：

name ifb4eth0

设置新的虚拟设备接口。

numtxqueues 48

设置传输队列的数量。

numrxqueues 48

设置接收队列的数量。

type ifb

设置新设备的类型。

- b. 更改接口的状态：

```
# ip link set dev ifb4eth0 up
```

- c. 在物理网络接口上添加 **qdisc** 属性，并将其应用到传入流量：

```
# tc qdisc add dev enp1s0 handle ffff: ingress
```

在 **handle ffff:** 选项中，**handle** 参数将主号 **ffff:** 作为默认值分配给 **enp1s0** 物理网络接口上的类 **qdisc**，其中 **qdisc** 是一个分析流量控制的排队规则参数。

- d. 在 **ip** 协议的物理接口上添加一个过滤器，以对数据包进行分类：

```
# tc filter add dev enp1s0 parent ffff: protocol ip u32 match u32 0 0 action ctinfo cpmark 100 action mirrored egress redirect dev ifb4eth0
```

这个命令有以下属性：

parent ffff:

为父 **qdisc** 设置主号 **ffff:**。

u32 match u32 0 0

设置 **u32** 过滤器，以 **匹配 u32** 模式的 IP 标头。第一个 **0** 表示 IP 标头的第二个字节，另一个 **0** 用于掩码匹配，告知过滤器要匹配哪个位。

action ctinfo

设置操作，以将连接跟踪标记的数据检索到各个字段。

cpmark 100

将连接跟踪标记(connmark) **100** 复制到数据包 IP 标头字段中。

action mirrored egress redirect dev ifb4eth0

设置 **action mirrored**，来将接收的数据包重定向到 **ifb4eth0** 目的接口。

- e. 向接口中添加类 **qdisc**：

```
# tc qdisc add dev ifb4eth0 root handle 1: htb default 1000
```

此命令将主号 **1** 设置为 root **qdisc**，并将 **htb** 层次结构令牌存储桶与 **minor-id 1000** 的类 **qdisc** 一起使用。

- f. 将接口上的流量限制为 1 Mbit/s，上限为 2 Mbit/s：

```
# tc class add dev ifb4eth0 parent 1:1 classid 1:100 htb ceil 2mbit rate 1mbit prio 100
```

这个命令有以下参数：

parent 1:1

将 **parent** 设为 **classid** 为 **1**，将 **root** 设为 **1**。

classid 1:100

将 **classid** 设置为 **1:100**，其中 **1** 是父 **qdisc** 的数量，**100** 是父 **qdisc** 的类的数量。

htb ceil 2mbit

htb 类 **qdisc** 允许上限带宽 **2 Mbit/s** 作为 **ceil** 速率限制。

- g. 将无类别 **qdisc** 的 Stochastic Fairness Queuing (**sfq**)应用到 时间间隔为 **60** 秒的接口，以减少队列算法的影响：

```
# tc qdisc add dev ifb4eth0 parent 1:100 sfq perturb 60
```

- h. 在接口中添加防火墙标记(**fw**)过滤器：

```
# tc filter add dev ifb4eth0 parent 1:0 protocol ip prio 100 handle 100 fw classid 1:100
```

- i. 从连接标记(**CONNMARK**)恢复数据包元标记：

```
# nft add rule ip mangle PREROUTING counter meta mark set ct mark
```

在这个命令中，**nft** 工具有一个带有 **PREROUTING** 链规则规范的 **mangle** 表，该表在路由之前更改传入的数据包，来将数据包标记替换为 **CONNMARK**。

- j. 如果没有 **nft** 表和链存在，请创建一个表并添加一个链规则：

```
# nft add table ip mangle
# nft add chain ip mangle PREROUTING {type filter hook prerouting priority mangle \;}
```

- k. 在指定目标地址 **192.0.2.3** 上接收的 **tcp** 数据包上设置 **meta** 标记：

```
# nft add rule ip mangle PREROUTING ip daddr 192.0.2.3 counter meta mark set 0x64
```

- l. 将数据包标记保存到连接标记中：

```
# nft add rule ip mangle PREROUTING counter ct mark set mark
```

- m. 使用 **-s** 参数在系统上运行 **iperf3** 工具来作为服务器，然后服务器等待客户端连接的响应：

```
# iperf3 -s
```

2. 在客户端上，将 **iperf3** 作为客户端运行，并连接到在 IP 地址 **192.0.2.3** 上侦听定期的 HTTP 请求响应时间戳的服务器：

```
# iperf3 -c 192.0.2.3 -t TCP_STREAM | tee rate
```

192.0.2.3 是服务器的 IP 地址，而 **192.0.2.4** 是客户端的 IP 地址。

3. 按 **Ctrl+C** 终止服务器上的 **iperf3** 工具：

```
Accepted connection from 192.0.2.4, port 52128
[5] local 192.0.2.3 port 5201 connected to 192.0.2.4 port 52130
[ID] Interval      Transfer Bitrate
[5] 0.00-1.00 sec  119 KBytes 973 Kbits/sec
[5] 1.00-2.00 sec  116 KBytes 950 Kbits/sec
...
[ID] Interval      Transfer Bitrate
[5] 0.00-14.81 sec  1.51 MBytes 853 Kbits/sec receiver

iperf3: interrupt - the server has terminated
```

4. 按 **Ctrl+C** 终止客户端上的 **iperf3** 工具：

```
Connecting to host 192.0.2.3, port 5201
[5] local 192.0.2.4 port 52130 connected to 192.0.2.3 port 5201
[ID] Interval      Transfer Bitrate  Retr Cwnd
[5] 0.00-1.00 sec  481 KBytes 3.94 Mb/s 0 76.4 KBytes
[5] 1.00-2.00 sec  223 KBytes 1.83 Mb/s 0 82.0 KBytes
...
[ID] Interval      Transfer Bitrate  Retr
[5] 0.00-14.00 sec  3.92 MBytes 2.35 Mb/s 32 sender
[5] 0.00-14.00 sec  0.00 Bytes 0.00 bits/sec receiver

iperf3: error - the server has terminated
```

验证

1. 显示接口上 **htb** 和 **sfq** 类的数据包计数的统计信息：

```
# tc -s qdisc show dev ifb4eth0

qdisc htb 1: root
...
Sent 26611455 bytes 3054 pkt (dropped 76, overlimits 4887 requeues 0)
...
qdisc sfq 8001: parent
...
Sent 26535030 bytes 2296 pkt (dropped 76, overlimits 0 requeues 0)
...
```


2. 显示 **mirred** 和 **ctinfo** 操作的数据包计数的统计信息：

```
# tc -s filter show dev enp1s0 ingress
filter parent ffff: protocol ip pref 49152 u32 chain 0
filter parent ffff: protocol ip pref 49152 u32 chain 0 fh 800: ht divisor 1
filter parent ffff: protocol ip pref 49152 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0
terminal flowid not_in_hw (rule hit 8075 success 8075)
  match 00000000/00000000 at 0 (success 8075 )
  action order 1: ctinfo zone 0 pipe
    index 1 ref 1 bind 1 cpmark 0x00000064 installed 3105 sec firstused 3105 sec DSCP set
0 error 0
  CPMARK set 7712
  Action statistics:
  Sent 25891504 bytes 3137 pkt (dropped 0, overlimits 0 requeues 0)
  backlog 0b 0p requeues 0

  action order 2: mirred (Egress Redirect to device ifb4eth0) stolen
    index 1 ref 1 bind 1 installed 3105 sec firstused 3105 sec
  Action statistics:
  Sent 25891504 bytes 3137 pkt (dropped 0, overlimits 61 requeues 0)
  backlog 0b 0p requeues 0
```

3. 显示 **htb** 速率限制器及其配置的统计信息：

```
# tc -s class show dev ifb4eth0
class htb 1:100 root leaf 8001: prio 7 rate 1Mbit ceil 2Mbit burst 1600b cburst 1600b
Sent 26541716 bytes 2373 pkt (dropped 61, overlimits 4887 requeues 0)
backlog 0b 0p requeues 0
lended: 7248 borrowed: 0 giants: 0
tokens: 187250 ctokens: 93625
```

其他资源

- **tc(8)** 和 **tc-ctinfo(8)** 手册页
- **nft(8)** 手册页

32.8. RHEL 中可用的 QDISCS

每个 **qdisc** 解决唯一的与网络相关的问题。以下是 RHEL 中可用的 **qdiscs** 列表。您可以使用以下任何一个 **qdisc** 来根据您的网络要求来塑造网络流量。

表 32.1. RHEL 中的可用调度程序

qdisc 名称	包含在	卸载支持
异步传输模式(ATM)	kernel-modules-extra	
基于类的队列	kernel-modules-extra	
基于信用的塑造程序	kernel-modules-extra	是

qdisc 名称	包含在	卸载支持
CHOOSE 和 Keep 用于有响应的流量，CHOOSE 和 Kill 用于没有响应的流量 (CHOKe)	kernel-modules-extra	
受控的延迟 (CoDel)	kernel-core	
轮循(DRR)	kernel-modules-extra	
Differentiated Services marker (DSMARK)	kernel-modules-extra	
Enhanced Transmission Selection (ETS)	kernel-modules-extra	是
Fair Queue (FQ)	kernel-core	
Fair Queuing Controlled Delay (FQ_CODEL)	kernel-core	
Generalized Random Early Detection (GRED)	kernel-modules-extra	
Hierarchical Fair Service Curve (HSFC)	kernel-core	
Heavy-Hitter Filter (HHF)	kernel-core	
Hierarchy Token Bucket (HTB)	kernel-core	
INGRESS	kernel-core	是
Multi Queue Priority (MQPRIO)	kernel-modules-extra	是
Multiqueue (MULTIQ)	kernel-modules-extra	是
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	是

qdisc 名称	包含在	卸载支持
Stochastic Fair Blue (SFB)	kernel-modules-extra	
Stochastic Fairness Queueing (SFQ)	kernel-core	
Token Bucket Filter (TBF)	kernel-core	是
Trivial Link Equalizer (TEQL)	kernel-modules-extra	



重要

qdisc 卸载需要对 NIC 的硬件和驱动程序的支持。

其他资源

- [tc\(8\) 手册页](#)

第 33 章 使用带有存储在文件系统上证书的 802.1X 标准将 RHEL 客户端认证到网络

管理员通常使用基于 IEEE 802.1X 标准的基于端口的网络访问控制 (NAC) 来保护网络不受未经授权 LAN 和 Wi-Fi 客户端的影响。要让客户端能够连接到这样的网络，您必须在此客户端上配置 802.1X 身份验证。

33.1. 使用 nmcli 在现有以太网连接中配置 802.1X 网络身份验证

您可以使用 `nmcli` 工具在命令行上配置带有 802.1X 网络身份验证的以太网连接。

先决条件

- 网络支持 802.1X 网络身份验证。
- 以太网连接配置集存在于 NetworkManager 中，且具有有效的 IP 配置。
- 客户端上存在 TLS 身份验证所需的以下文件：
 - 存储的客户端密钥位于 `/etc/pki/tls/private/client.key` 文件中，该文件归 `root` 用户所有，且只对 `root` 可读。
 - 客户端证书存储在 `/etc/pki/tls/certs/client.crt` 文件中。
 - 证书颁发机构(CA)证书存储在 `/etc/pki/tls/certs/ca.crt` 文件中。
- `wpa_supplicant` 软件包已安装。

步骤

1. 将扩展验证协议(EAP)设置为 `tls`，将路径设置为客户端证书和密钥文件：

```
# 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
```

请注意，您必须在一个命令中设置 `802-1x.eap`、`802-1x.client-cert` 和 `802-1x.private-key` 参数。

2. 设置 CA 证书的路径：

```
# nmcli connection modify enp1s0 802-1x.ca-cert /etc/pki/tls/certs/ca.crt
```

3. 设置证书中使用的用户的身份：

```
# nmcli connection modify enp1s0 802-1x.identity user@example.com
```

4. 另外，还可将该密码存储在配置中：

```
# nmcli connection modify enp1s0 802-1x.private-key-password password
```



重要

默认情况下，NetworkManager 在 `/etc/sysconfig/network-scripts/keys-connection_name` 文件中以明文形式保存密码，该文件只对 `root` 用户可读。但是，在配置文件中清除文本密码会有安全隐患。

要提高安全性，请将 `802-1x.password-flags` 参数设为 `0x1`。有了这个设置，在具有 GNOME 桌面环境或运行 `nm-applet` 的服务器上，NetworkManager 从这些服务中检索密码。在其他情况下，NetworkManager 会提示输入密码。

5. 激活连接配置集：

```
# nmcli connection up enp1s0
```

验证

- 访问需要网络身份验证的网络上的资源。

其他资源

- [配置以太网连接](#)
- [nm-settings\(5\) 手册页](#)
- [nmcli\(1\) 手册页](#)

33.2. 使用 NMSTATECTL 配置带有 802.1X 网络身份验证的静态以太网连接

使用 `nmstatectl` 工具通过 Nmstate API 配置带有 802.1X 网络身份验证的以太网连接。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，`nmstatectl` 会自动回滚更改以避免系统处于不正确的状态。



注意

`nmstate` 库只支持 TLS 可扩展身份验证协议(EAP)方法。

先决条件

- 网络支持 802.1X 网络身份验证。
- 受管节点使用 NetworkManager。
- 客户端上存在 TLS 身份验证所需的以下文件：
 - 存储的客户端密钥位于 `/etc/pki/tls/private/client.key` 文件中，该文件归 `root` 用户所有，且只对 `root` 可读。
 - 客户端证书存储在 `/etc/pki/tls/certs/client.crt` 文件中。
 - 证书颁发机构(CA)证书存储在 `/etc/pki/tls/certs/ca.crt` 文件中。

步骤

1. 创建一个包含以下内容的 YAML 文件，如 `~/create-ethernet-profile.yml`：

```
---
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
```

这些设置使用以下设置为 **enp1s0** 设备定义一个以太网连接配置文件：

- 静态 IPv4 地址 - **192.0.2.1** 和 **/24** 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**

- DNS 搜索域 - **example.com**
 - 使用 **TLS** EAP 协议的 802.1x 网络身份验证
2. 将设置应用到系统：

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

验证

- 访问需要网络身份验证的网络上的资源。

33.3. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的静态以太网连接

您可以使用 **network** RHEL 系统角色远程配置具有 802.1X 网络身份验证的以太网连接。

前提条件

- [您已准备好控制节点和受管节点](#)
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 **sudo** 权限。
- 网络支持 802.1X 网络身份验证。
- 受管节点使用 NetworkManager。
- control 节点上存在 TLS 身份验证所需的以下文件：
 - 客户端密钥存储在 **/srv/data/client.key** 文件中。
 - 客户端证书存储在 **/srv/data/client.crt** 文件中。
 - 证书颁发机构(CA)证书存储在 **/srv/data/ca.crt** 文件中。

流程

1. 创建一个包含以下内容的 playbook 文件，如 **~/playbook.yml**：

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

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

```

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

- name: Configure connection
  ansible.builtin.include_role:
    name: rhel-system-roles.network
  vars:
    network_connections:
      - name: enp1s0
        type: ethernet
        autoconnect: yes
        ip:
          address:
            - 192.0.2.1/24
            - 2001:db8:1::1/64
          gateway4: 192.0.2.254
          gateway6: 2001:db8:1::fffe
          dns:
            - 192.0.2.200
            - 2001:db8:1::ffbb
          dns_search:
            - example.com
        ieee802_1x:
          identity: user_name
          eap: tls
          private_key: "/etc/pki/tls/private/client.key"
          private_key_password: "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

```

这些设置使用以下设置为 **enp1s0** 设备定义一个以太网连接配置文件：

- 静态 IPv4 地址 - **192.0.2.1** 和 **/24** 子网掩码
- 静态 IPv6 地址 - **2001:db8:1::1** 和 **/64** 子网掩码
- IPv4 默认网关 - **192.0.2.254**
- IPv6 默认网关 - **2001:db8:1::fffe**
- IPv4 DNS 服务器 - **192.0.2.200**
- IPv6 DNS 服务器 - **2001:db8:1::ffbb**
- DNS 搜索域 - **example.com**
- 使用 **TLS** 可扩展身份验证协议(EAP)进行 802.1X 网络身份验证。

2. 验证 playbook 语法：

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


请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook :

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

其他资源

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

33.4. 使用 NETWORK RHEL 系统角色配置带有 802.1X 网络身份验证的 WIFI 连接

使用 RHEL 系统角色，您可以自动化 wifi 连接的创建。例如，您可以使用 Ansible Playbook 为 `wlp1s0` 接口远程添加无线连接配置文件。创建的配置集使用 802.1X 标准将客户端验证到 Wi-Fi 网络。该 playbook 将连接配置集配置为使用 DHCP。要配置静态 IP 设置，相应地调整 `ip` 字典中的参数。

先决条件

- 您已准备好控制节点和受管节点
- 以可在受管主机上运行 playbook 的用户登录到控制节点。
- 用于连接到受管节点的帐户对其具有 `sudo` 权限。
- 网络支持 802.1X 网络身份验证。
- 您已在受管节点上安装了 `wpa_supplicant` 软件包。
- DHCP 位于受管节点的网络中。
- control 节点上存在 TLS 身份验证所需的以下文件：
 - 客户端密钥存储在 `/srv/data/client.key` 文件中。
 - 客户端证书存储在 `/srv/data/client.crt` 文件中。
 - CA 证书存储在 `/srv/data/ca.crt` 文件中。

步骤

1. 创建一个包含以下内容的 playbook 文件，如 `~/playbook.yml` :

```
---
- name: Configure a wifi connection with 802.1X authentication
  hosts: managed-node-01.example.com
  tasks:
    - name: Copy client key for 802.1X authentication
      ansible.builtin.copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0400
```

```

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

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

- block:
  - ansible.builtin.import_role:
      name: rhel-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"

```

这些设置为 **wlp1s0** 接口定义一个 wifi 连接配置文件。该配置文件使用 802.1X 标准将客户端向 wifi 网络进行身份验证。连接从 DHCP 服务器和 IPv6 无状态地址自动配置(SLAAC)检索 IPv4 地址、IPv6 地址、默认网关、路由、DNS 服务器和搜索域。

2. 验证 playbook 语法 :

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

请注意，这个命令只验证语法，不会防止错误，但保护有效配置。

3. 运行 playbook :

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

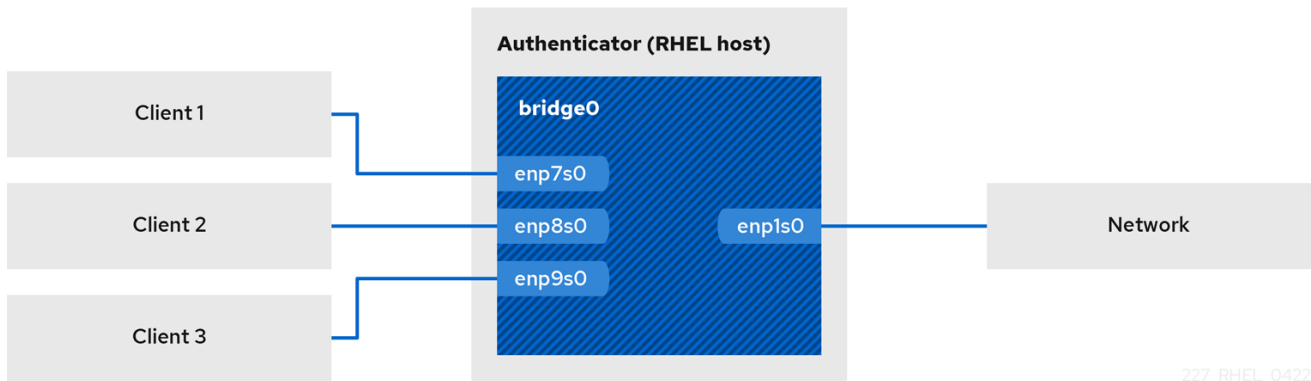
其他资源

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

第 34 章 使用带有 FREERADIUS 后端的 HOSTAPD 为 LAN 客户端设置 802.1X 网络身份验证服务

IEEE 802.1X 标准定义了安全身份验证和授权方法，以保护网络不接受未授权的客户端。通过使用 **hostapd** 服务和 FreeRADIUS，您可以在您的网络中提供网络访问控制(NAC)。

在本文中，RHEL 主机充当一个网桥，以使用现有的网络连接不同的客户端。但是，RHEL 主机只授权认证的客户端可以访问网络。



34.1. 先决条件

- **freeradius** 软件包的全新安装。
如果已经安装了该软件包，请删除 `/etc/raddb/` 目录，卸载，然后再次安装该软件包。不要使用 `dnf reinstall` 命令重新安装软件包，因为 `/etc/raddb/` 目录中的权限和符号链接会不同。

34.2. 在 AUTHENTICATOR 中设置桥接

网桥是一个链路层设备，它根据 MAC 地址表在主机和网络之间转发流量。如果将 RHEL 设置为 802.1X 验证器，请将要在其上执行身份验证的接口和 LAN 接口添加到网桥。

先决条件

- 服务器有多个以太网接口。

步骤

1. 创建网桥接口：

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

2. 将以太网接口分配给网桥：

```
# nmcli connection add type ethernet port-type bridge con-name br0-port1 ifname enp1s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port2 ifname enp7s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port3 ifname enp8s0 controller br0
# nmcli connection add type ethernet port-type bridge con-name br0-port4 ifname enp9s0 controller br0
```

3. 启用网桥以转发 LAN(EAPOL)数据包上的可扩展验证协议：

```
# nmcli connection modify br0 group-forward-mask 8
```

4. 配置连接以自动激活端口：

```
# nmcli connection modify br0 connection.autoconnect-ports 1
```

5. 激活连接：

```
# nmcli connection up br0
```

验证

1. 显示作为特定网桥端口的以太网设备的链接状态：

```
# 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. 验证 **br0** 设备上是否启用了 EAPOL 数据包的转发：

```
# cat /sys/class/net/br0/bridge/group_fwd_mask
0x8
```

如果命令返回 **0x8**，则启用了转发。

其他资源

- [nm-settings\(5\) 手册页](#)

34.3. FREERADIUS 的证书要求

对于安全的 FreeRADIUS 服务，您需要 TLS 证书来满足不同的目的：

- 用于加密连接到服务器的 TLS 服务器证书。使用信任的证书颁发机构(CA)来发布证书。服务器证书要求将扩展密钥使用(EKU)字段设为 **TLS Web Server Authentication**。
- 由同一 CA 为扩展身份验证协议传输层安全(EAP-TLS)发布的客户端证书。EAP-TLS 提供基于证书的身份验证，并默认启用。客户端证书需要其 EKU 字段设为 **TLS Web Client Authentication**。



警告

要安全连接，请使用您公司的 CA 或创建自己的 CA 来为 FreeRADIUS 发布证书。如果使用公共 CA，则您允许其验证用户，并为 EAP-TLS 发布客户端证书。

34.4. 在 FREERADIUS 服务器上，出于测试目的创建一组证书

出于测试目的，**freeradius** 软件包会在 `/etc/raddb/certs/` 目录中安装脚本和配置文件，以创建自己的证书颁发机构(CA)并发布证书。



重要

如果您使用默认配置，这些脚本生成的证书会在 60 天后过期，密钥使用不安全的密码 ("whatever")。但是，您可以自定义 CA、服务器和客户端配置。

执行此流程后，会创建本文档稍后所需要的以下文件：

- `/etc/raddb/certs/ca.pem`: CA 证书
- `/etc/raddb/certs/server.key`: 服务器证书的私钥
- `/etc/raddb/certs/server.pem`: 服务器证书
- `/etc/raddb/certs/client.key`: 客户端证书的私钥
- `/etc/raddb/certs/client.pem`: 客户端证书

先决条件

- **freeradius** 软件包已安装。

步骤

1. 进到 `/etc/raddb/certs/` 目录：

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

2. 可选：在 `/etc/raddb/certs/ca.cnf` 文件中自定义 CA 配置：

```
...
[ 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. 可选：在 `/etc/raddb/certs/server.cnf` 文件中自定义服务器配置：

```

...
[ 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. 可选：在 `/etc/raddb/certs/client.cnf` 文件中自定义客户端配置：

```

...
[ 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. 创建证书：

```
# make all
```

6. 将 `/etc/raddb/certs/server.pem` 文件中的组更改为 `radiusd`：

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

其他资源

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

34.5. 配置 FREERADIUS 以验证使用 EAP 保护的客户端

FreeRADIUS 支持通过不同的扩展验证协议(EAP)的方法。但是，对于安全网络，请配置 FreeRADIUS 以仅支持以下安全 EAP 身份验证方法：

- EAP-TLS（传输层安全）使用安全 TLS 连接来使用证书验证客户端。要使用 EAP-TLS，需要每个网络客户端的 TLS 客户端证书，以及服务器的服务器证书。请注意，同样的证书颁发机构(CA)必须已发布了证书。始终使用您自己的 CA 来创建证书，因为您使用的 CA 发布的所有客户端证书都可以向 FreeRADIUS 服务器进行身份验证。
- EAP-TTLS（隧道传输层安全）使用安全 TLS 连接作为外部身份验证协议来设置隧道。然后，内部身份验证使用密码身份验证协议(PAP)或质询握手身份验证协议(CHAP)。要使用 EAP-TTLS，您需要一个 TLS 服务器证书。
- EAP-PEAP（受保护的验证协议）使用安全 TLS 连接作为外部身份验证协议来设置隧道。验证器验证 RADIUS 服务器的证书。之后，请求方通过使用 Microsoft 挑战握手身份验证协议版本 2 (MS-CHAPv2)或其他方法，通过加密的隧道来进行身份验证。



注意

默认的 FreeRADIUS 配置文件充当文档，并描述了所有参数和指令。如果要禁用某些特性，请注释掉它们，而不是删除配置文件中的相应部分。这可让您保留配置文件和包含的文档的结构。

先决条件

- **freeradius** 软件包已安装。
- `/etc/raddb/` 目录中的配置文件保持不变，就像 **freeradius** 软件包提供的那样。
- 服务器上存在以下文件：
 - FreeRADIUS 主机的 TLS 私钥：`/etc/raddb/certs/server.key`
 - FreeRADIUS 主机的 TLS 服务器证书：`/etc/raddb/certs/server.pem`
 - TLS CA 证书：`/etc/raddb/certs/ca.pem`

如果您将文件存储在不同的位置或者它们有不同的名称，请在 `/etc/raddb/mods-available/eap` 文件中相应地设置 `private_key_file`、`certificate_file` 和 `ca_file` 参数。

步骤

1. 如果带有 Diffie-Hellman(DH)参数的 `/etc/raddb/certs/dh` 不存在，就创建一个。例如，要创建带有 2048 位素数的 DH 文件，请输入：

```
# openssl dhparam -out /etc/raddb/certs/dh 2048
```

为了安全起见，请不要使用小于 2048 位素数的 DH 文件。根据位数，文件的创建可能需要几分钟。

2. 使用 DH 参数对 TLS 私钥、服务器证书、CA 证书和文件设置安全权限：

```
# 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. 编辑 `/etc/raddb/mods-available/eap` 文件：

- a. 在 `private_key_password` 参数中设置私钥的密码：

```
eap {
  ...
  tls-config tls-common {
    ...
    private_key_password = key_password
    ...
  }
}
```

- b. 根据您的环境，将 `eap` 指令中的 `default_eap_type` 参数设为您使用的主要 EAP 类型：

```
eap {
  ...
  default_eap_type = ttls
  ...
}
```

对于安全环境，请仅使用 `ttls`、`tls` 或 `peap`。

- c. 注释掉 `md5` 指令，以禁用不安全的 EAP-MD5 身份验证方法：

```
eap {
  ...
  # md5 {
  # }
  ...
}
```

请注意，在默认的配置文件中，其他不安全的 EAP 身份验证方法默认被注释掉了。

4. 编辑 `/etc/raddb/sites-available/default` 文件，然后注释掉 `eap` 以外的所有身份验证方法：

```
authenticate {
  ...
  # Auth-Type PAP {
  #   pap
  # }

  # Auth-Type CHAP {
  #   chap
  # }

  # Auth-Type MS-CHAP {
  #   mschap
  # }
```



```
# mschap
# digest
...
}
```

这将只为外部身份验证启用 EAP，并禁用纯文本身身份验证方法。

5. 编辑 `/etc/raddb/clients.conf` 文件：

- a. 在 `localhost` 和 `localhost_ipv6` 客户端指令中设置安全密码：

```
client localhost {
    ipaddr = 127.0.0.1
    ...
    secret = localhost_client_password
    ...
}

client localhost_ipv6 {
    ipv6addr = ::1
    secret = localhost_client_password
}
```

- b. 为网络验证器添加客户端指令：

```
client hostapd.example.org {
    ipaddr = 192.0.2.2/32
    secret = hostapd_client_password
}
```

- c. 可选：如果其他主机也能够访问 FreeRADIUS 服务，还要为它们添加客户端指令，例如：

```
client <hostname_or_description> {
    ipaddr = <IP_address_or_range>
    secret = <client_password>
}
```

`ipaddr` 参数接受 IPv4 和 IPv6 地址，您可以使用可选的无类别域间路由(CIDR)表示法来指定范围。但是，在这个参数中您只能设置一个值。例如，要授予 IPv4 和 IPv6 地址的访问权限，您必须添加两个客户端指令。

为客户端指令使用一个描述性名称，如主机名或一个描述 IP 范围在哪里使用的词语。

6. 如果要使用 EAP-TTLS 或 EAP-PEAP，请将用户添加到 `/etc/raddb/users` 文件中：

```
example_user    Cleartext-Password := "user_password"
```

对于应使用基于证书的身份验证(EAP-TLS)的用户，不要添加任何条目。

7. 验证配置文件：

```
# radiusd -XC
...
Configuration appears to be OK
```

- 在 **firewalld** 服务中打开 RADIUS 端口：

```
# firewall-cmd --permanent --add-service=radius
# firewall-cmd --reload
```

- 启用并启动 **radiusd** 服务：

```
# systemctl enable --now radiusd
```

验证

- 针对 FreeRADIUS 服务器或验证器测试 EAP-TTLS 身份验证
- 针对 FreeRADIUS 服务器或验证器测试 EAP-TLS 身份验证

故障排除

- 停止 **radiusd** 服务：

```
# systemctl stop radiusd
```

- 以 debug 模式启动该服务：

```
# radiusd -X
...
Ready to process requests
```

- 在 FreeRADIUS 主机上执行验证测试，如 [验证](#) 部分中所述。

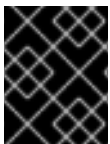
后续步骤

- 禁用不再需要的身份验证方法，以及您不使用的其他功能。

34.6. 在有线网络中将 HOSTAPD 配置为验证器

主机访问点守护进程 (**hostapd**) 服务可在有线网络中充当验证器，来提供 802.1X 身份验证。为此，**hostapd** 服务需要一个用来验证客户端的 RADIUS 服务器。

hostapd 服务提供集成的 RADIUS 服务器。但是，使用集成的 RADIUS 服务器只用于测试目的。对于生产环境，请使用 FreeRADIUS 服务器，它支持其他特性，如不同的身份验证方法和访问控制。



重要

hostapd 服务不与流量平面交互。该服务仅充当身份验证器。例如，使用脚本或服务，该脚本或服务使用 **hostapd** 控制接口、根据身份验证事件的结果来允许或拒绝流量。

先决条件

- hostapd** 软件包已安装。
- FreeRADIUS 服务器已配置，它已准备好对客户端进行身份验证。

步骤

1. 使用以下内容创建 `/etc/hostapd/hostapd.conf` 文件：

```
# General settings of hostapd
# =====

# Control interface settings
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel

# Enable logging for all modules
logger_syslog=-1
logger_stdout=-1

# Log level
logger_syslog_level=2
logger_stdout_level=2

# Wired 802.1X authentication
# =====

# Driver interface type
driver=wired

# Enable IEEE 802.1X authorization
ieee8021x=1

# Use port access entry (PAE) group address
# (01:80:c2:00:00:03) when sending EAPOL frames
use_pae_group_addr=1

# Network interface for authentication requests
interface=br0

# RADIUS client configuration
# =====

# Local IP address used as NAS-IP-Address
own_ip_addr=192.0.2.2

# Unique NAS-Identifier within scope of RADIUS server
nas_identifier=hostapd.example.org

# RADIUS authentication server
auth_server_addr=192.0.2.1
auth_server_port=1812
auth_server_shared_secret=hostapd_client_password

# RADIUS accounting server
acct_server_addr=192.0.2.1
acct_server_port=1813
acct_server_shared_secret=hostapd_client_password
```

有关此配置中使用的参数的详情，请查看 `/usr/share/doc/hostapd/hostapd/hostapd.conf` 示例配置文件中的描述。

2. 启用并启动 **hostapd** 服务：

```
# systemctl enable --now hostapd
```

验证

- 请参阅：
 - [针对 FreeRADIUS 服务器或验证器测试 EAP-TTLS 身份验证](#)
 - [针对 FreeRADIUS 服务器或验证器测试 EAP-TLS 身份验证](#)

故障排除

1. 停止 **hostapd** 服务：

```
# systemctl stop hostapd
```

2. 以 debug 模式启动该服务：

```
# hostapd -d /etc/hostapd/hostapd.conf
```

3. 在 FreeRADIUS 主机上执行验证测试，如 [验证](#) 部分中所述。

其他资源

- [hostapd.conf\(5\) 手册页](#)
- `/usr/share/doc/hostapd/hostapd.conf` 文件

34.7. 针对 FREERADIUS 服务器或验证器测试 EAP-TTLS 身份验证

要测试在隧道传输层安全(EAP-TTLS)上使用可扩展的身份验证协议(EAP-TTLS)的身份验证是否按预期工作，请运行此流程：

- 设置 FreeRADIUS 服务器后
- 将 **hostapd** 服务设为 802.1X 网络身份验证验证器后。

此流程中使用的测试工具的输出提供有关 EAP 通信的其他信息，并帮助您调试问题。

先决条件

- 当您要验证：
 - FreeRADIUS 服务器：
 - **hostapd** 软件包提供的 `eapol_test` 工具已安装。
 - 您在其上运行此流程的客户端已在 FreeRADIUS 服务器的客户端数据库中被授权。
 - 由同命软件包提供的验证器 `wpa_supplicant` 工具已安装。

- 您在 `/etc/pki/tls/certs/ca.pem` 文件中存储了证书颁发机构(CA)证书。

步骤

1. 使用以下内容创建 `/etc/wpa_supplicant/wpa_supplicant-TTLS.conf` 文件：

```
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. 要向以下进行身份验证：

- FreeRADIUS 服务器，请输入：

```
# 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
```

`-a` 选项定义了 FreeRADIUS 服务器的 IP 地址，而 `-s` 选项指定您要在其上运行 FreeRADIUS 服务器的客户端配置中命令的主机的密码。

- 验证器，请输入：

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -D wired -i
enp0s31f6
...
enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
```

`-i` 选项指定 `wpa_supplicant` 通过 LAN(EAPOL)数据包发送扩展验证协议的网络接口名称。

若要了解更多调试信息，请将 `-d` 选项传给命令。

其他资源

- `/usr/share/doc/wpa_supplicant/wpa_supplicant.conf` file

34.8. 针对 FREERADIUS 服务器或验证器测试 EAP-TLS 身份验证

要测试使用可扩展的身份验证协议(EAP)传输层安全(EAP-TLS)的身份验证是否按预期工作，请运行此流程：

- 设置 FreeRADIUS 服务器后
- 将 `hostapd` 服务设为 802.1X 网络身份验证验证器后。

此流程中使用的测试工具的输出提供有关 EAP 通信的其他信息，并帮助您调试问题。

先决条件

- 当您要验证：
 - FreeRADIUS 服务器：
 - `hostapd` 软件包提供的 `eapol_test` 工具已安装。
 - 您在其上运行此流程的客户端已在 FreeRADIUS 服务器的客户端数据库中被授权。
 - 由同命软件包提供的验证器 `wpa_supplicant` 工具已安装。
- 您在 `/etc/pki/tls/certs/ca.pem` 文件中存储了证书颁发机构(CA)证书。
- 发布客户端证书的 CA 与发布 FreeRADIUS 服务器的服务器证书的 CA 是同一个。
- 您将客户端证书存储在 `/etc/pki/tls/certs/client.pem` 文件中。
- 将客户端的私钥存储在 `/etc/pki/tls/private/client.key` 中

流程

1. 使用以下内容创建 `/etc/wpa_supplicant/wpa_supplicant-TLS.conf` 文件：

```
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. 要向以下进行身份验证：
 - FreeRADIUS 服务器，请输入：

```
# 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
```

-a 选项定义了 FreeRADIUS 服务器的 IP 地址，而 **-s** 选项指定您要在其上运行 FreeRADIUS 服务器的客户端配置中命令的主机的密码。

- 验证器，请输入：

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TLS.conf -D wired -i
enp0s31f6
...
enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
```

-i 选项指定 **wpa_supplicant** 通过 LAN(EAPOL)数据包发送扩展验证协议的网络接口名称。

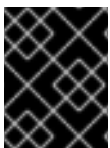
若要了解更多调试信息，请将 **-d** 选项传给命令。

其他资源

- `/usr/share/doc/wpa_supplicant/wpa_supplicant.conf` file

34.9. 根据 HOSTAPD 身份验证事件阻止和允许流量

hostapd 服务不与流量平面交互。该服务仅充当身份验证器。但是，您可以编写一个脚本，根据身份验证事件的结果来允许或拒绝流量。



重要

此流程不受支持，没有企业级的解决方案。它只演示如何通过评估由 **hostapd_cli** 检索的事件来阻止或允许流量。

当 **802-1x-tr-mgmt** systemd 服务启动时，RHEL 会阻止 **hostapd** 监听端口上的所有流量，但 LAN(EAPOL)数据包上可扩展验证协议除外，并使用 **hostapd_cli** 工具连接到 **hostapd** 控制接口。`/usr/local/bin/802-1x-tr-mgmt` 脚本随后评估事件。根据 **hostapd_cli** 收到的不同事件，该脚本允许或阻止 MAC 地址的流量。请注意，当 **802-1x-tr-mgmt** 服务停止时，所有流量会自动允许。

在 **hostapd** 服务器上执行此流程。

先决条件

- **hostapd** 服务已配置，服务已准备好对客户端进行身份验证。

流程

1. 使用以下内容创建 `/usr/local/bin/802-1x-tr-mgmt` 文件：

```
#!/bin/sh

if [ "x$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 ibname "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 [ "x$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. 使用以下内容创建 `/etc/systemd/system/802-1x-tr-mgmt@.service` systemd 服务文件：

```
[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. 重新载入 systemd :

```
# systemctl daemon-reload
```

4. 启用并启动接口名称 **hostapd** 正在侦听的 **802-1x-tr-mgmt** 服务 :

```
# systemctl enable --now 802-1x-tr-mgmt@br0.service
```

验证

- 通过客户端向网络进行身份验证。请参阅 :
 - [针对 FreeRADIUS 服务器或验证器测试 EAP-TTLS 身份验证](#)
 - [针对 FreeRADIUS 服务器或验证器测试 EAP-TLS 身份验证](#)

其他资源

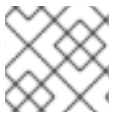
- **systemd.service(5)** 手册页

第 35 章 多路径 TCP 入门

传输控制协议(TCP)确保通过互联网的可靠的数据传输，并自动调整其带宽以响应网络负载。多路径 TCP (MPTCP) 是原始 TCP 协议 (single-path) 的扩展。MPTCP 支持传输连接同时在多个路径中运行，并为用户端点设备带来网络连接冗余。

35.1. 了解 MPTCP

多路径 TCP (MPTCP) 协议允许在连接端点间同时使用多个路径。协议设计提高了连接稳定性，与单一路径 TCP 相比也带来了其他好处。



注意

在 MPTCP 术语中，链接被视为路径。

以下是使用 MPTCP 的一些优点：

- 它允许一个连接同时使用多个网络接口。
- 如果连接绑定到链路速度，则使用多个链接可能会增加连接的吞吐量。请注意，如果连接绑定到 CPU，则使用多个链路会导致连接性能下降。
- 它提高对链接故障的恢复能力。

有关 MPTCP 的详情，请查看 [附加资源](#)。

其他资源

- [了解多路径 TCP：端点和高可用性以及未来的网络](#)
- [RFC8684：通过多个地址进行多路径操作的 TCP 扩展](#)

35.2. 准备 RHEL 启用 MPTCP 支持

默认情况下，RHEL 中禁用 MPTCP 支持。启用 MPTCP，以便支持此特性的应用程序可以使用它。另外，如果应用程序默认有 TCP 套接字，则必须配置用户空间应用程序来强制使用 MPTCP 套接字。

先决条件

安装以下软件包：

- **iperf3**
- **mptcpd**
- **systemtap**

流程

1. 在内核中启用 MPTCP 套接字：

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. 启动 **iperf3** 服务器，并强制它创建 MPTCP 套接字而不是 TCP 套接字：

```
# mptcpize run iperf3 -s
Server listening on 5201
```

3. 将客户端连接到服务器，并强制它创建 MPTCP 套接字而不是 TCP 套接字：

```
# mptcpize iperf3 -c 127.0.0.1 -t 3
```

4. 建立连接后，验证 **ss** 输出以查看特定于子流的状态：

```
# 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
```

5. 验证 MPTCP 计数器：

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX      2      0.0
MPTcpExtMPCapableSYNTAX    2      0.0
MPTcpExtMPCapableSYNACKRX  2      0.0
MPTcpExtMPCapableACKRX     2      0.0
```

其他资源

- [TCP\(7\) man page](#)
- [mptcpize\(8\) 手册页](#)

35.3. 使用 IPROUTE2 为 MPTCP 应用程序临时配置和启用多个路径

每个 MPTCP 连接都使用一个类似于纯 TCP 的单个子流。要获得 MPTCP 优点，请为每个 MPTCP 连接的最大子流数指定更高的限值。然后配置额外的端点以创建这些子流。



重要

在重启机器后，这个过程配置不会保留。

请注意，MPTCP 尚不支持为同一套接字混合 IPv6 和 IPv4 端点。使用属于同一地址系列的端点。

前提条件

- 已安装 **mptcpd** 软件包

- 安装了 **iperf3** 软件包
- 服务器网络接口设置：
 - enp4s0: **192.0.2.1/24**
 - enp1s0: **198.51.100.1/24**
- 客户端网络接口设置：
 - enp4s0f0: **192.0.2.2/24**
 - enp4s0f1: **198.51.100.2/24**

流程

1. 将客户端配置为接受 1 个额外的远程地址，如服务器提供的：

```
# ip mptcp limits set add_addr_accepted 1
```

2. 在服务器上将 IP 地址 **198.51.100.1** 添加为新的 MPTCP 端点：

```
# ip mptcp endpoint add 198.51.100.1 dev enp1s0 signal
```

signal 选项可确保在三向手后发送 **ADD_ADDR** 数据包。

3. 启动 **iperf3** 服务器，并强制它创建 MPTCP 套接字而不是 TCP 套接字：

```
# mptcpize run iperf3 -s  
Server listening on 5201
```

4. 将客户端连接到服务器，并强制它创建 MPTCP 套接字而不是 TCP 套接字：

```
# mptcpize iperf3 -c 192.0.2.1 -t 3
```

验证

1. 验证连接是否已建立：

```
# ss -nti '( sport :5201 )'
```

2. 验证连接和 IP 地址限制：

```
# ip mptcp limit show
```

3. 验证新添加的端点：

```
# ip mptcp endpoint show
```

4. 在服务器上使用 **nstat MPTcp*** 命令验证 MPTCP 计数器：

```
# nstat MPTcp*
```

```
#kernel
MPTcpExtMPCapableSYNRX      2          0.0
MPTcpExtMPCapableACKRX      2          0.0
MPTcpExtMPJoinSynRx         2          0.0
MPTcpExtMPJoinAckRx         2          0.0
MPTcpExtEchoAdd              2          0.0
```

其他资源

- [ip-mptcp\(8\) 手册页](#)
- [mptcpize\(8\) 手册页](#)

35.4. 为 MPTCP 应用程序永久配置多路径

您可以使用 `nmcli` 命令配置 MultiPath TCP (MPTCP)，以在源和目标系统之间永久建立多个子流。子流可以使用不同的资源，不同的路由到目的地，甚至不同网络。例如以太网、单元格、wifi 等。因此，您可以实现组合连接，这提高了网络弹性和吞吐量。

服务器在我们的示例中使用以下网络接口：

- `enp4s0`: **192.0.2.1/24**
- `enp1s0`: **198.51.100.1/24**
- `enp7s0`: **192.0.2.3/24**

客户端在我们的示例中使用以下网络接口：

- `enp4s0f0`: **192.0.2.2/24**
- `enp4s0f1`: **198.51.100.2/24**
- `enp6s0`: **192.0.2.5/24**

前提条件

- 您在相关接口上配置了默认网关。

流程

1. 在内核中启用 MPTCP 套接字：

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. 可选：用于子流限制的 RHEL 内核默认是 2。如果您需要更多：

- a. 使用以下内容创建 `/etc/systemd/system/set_mptcp_limit.service` 文件：

```
[Unit]
Description=Set MPTCP subflow limit to 3
After=network.target
```

```
[Service]
ExecStart=ip mptcp limits set subflows 3
Type=oneshot
```

```
[Install]
WantedBy=multi-user.target
```

在每个引导过程中，当网络 (**network.target**) 可以正常工作后，**oneshot** 单元会执行 **ip mptcp limits set subflows 3** 命令。

ip mptcp limits set subflows 3 命令为每个连接设置附加子流的最大数量，即 4 个。可以最多添加 3 个额外的子流。

- b. 启用 **set_mptcp_limit** 服务：

```
# systemctl enable --now set_mptcp_limit
```

3. 在要用于连接聚合的所有连接配置集上启用 MPTCP：

```
# nmcli connection modify <profile_name> connection.mptcp-flags
signal,subflow,also-without-default-route
```

connection.mptcp-flags 参数配置 MPTCP 端点和 IP 地址标志。如果在 NetworkManager 连接配置集中启用 MPTCP，则设置会将相关网络接口的 IP 地址配置为 MPTCP 端点。

默认情况下，如果没有默认网关，NetworkManager 不会将 MPTCP 标记添加到 IP 地址。如果要绕过该检查，则需要使用 **also-without-default-route** 标志。

验证

1. 验证您启用了 MPTCP 内核参数：

```
# sysctl net.mptcp.enabled
net.mptcp.enabled = 1
```

2. 验证您正确设置了子流限制，如果默认值是不够的：

```
# ip mptcp limit show
add_addr_accepted 2 subflows 3
```

3. 验证您正确配置了每个地址 MPTCP 设置：

```
# ip mptcp endpoint show
192.0.2.1 id 1 subflow dev enp4s0
198.51.100.1 id 2 subflow dev enp1s0
192.0.2.3 id 3 subflow dev enp7s0
192.0.2.4 id 4 subflow dev enp3s0
...
```

其他资源

- [nm-settings-nmcli\(5\)](#)
- [ip-mptcp\(8\)](#)

- 第 35.1 节 “了解 MPTCP”
- 了解多路径 TCP：端点和高可用性以及未来的网络
- RFC8684：通过多个地址进行多路径操作的 TCP 扩展
- 使用多路径 TCP 更好地停机并增加带宽

35.5. 监控 MPTCP 子流

多路径 TCP(MPTCP)套接字的生命周期可能比较复杂：创建主 MPTCP 套接字，验证 MPTCP 路径，创建一个或多个子流并最终删除。最后，终止 MPTCP 套接字。

MPTCP 协议允许使用 **iproute** 软件包提供的 **ip** 工具监控与套接字和子流创建和删除相关的特定于 MPTCP 的事件。这个工具使用 **netlink** 接口来监控 MPTCP 事件。

此流程演示了如何监控 MPTCP 事件。因此，它会模拟 MPTCP 服务器应用程序，以及客户端连接到此服务。本例中涉及的客户端使用以下接口和 IP 地址：

- 服务器：**192.0.2.1**
- 客户端（以太网连接）：**192.0.2.2**
- 客户端（WiFi 连接）：**192.0.2.3**

为了简化这一示例，所有接口都在同一子网内。这不是必须的。但是，重要的是路由已正确配置，并且客户端能够通过两个接口访问服务器。

前提条件

- 有两个网络接口的 RHEL 客户端，如带有以太网和 WiFi 的笔记本电脑
- 客户端可以通过两个接口连接到服务器
- RHEL 服务器
- 客户端和服务器运行 RHEL 9.0 或更高版本
- 在客户端和服务器上安装了 **mptcpd** 软件包

流程

1. 将客户端和服务器的每个连接额外子流的限制设为 **1**：

```
# ip mptcp limits set add_addr_accepted 0 subflows 1
```

2. 在服务器上，要模拟 MPTCP 服务器应用程序，请使用强制的 MPTCP 套接字而不是 TCP 套接字，以侦听模式启动 **netcat (nc)**：

```
# mptcpize run nc -l -k -p 12345
```

-k 选项可使 **nc** 在第一次接受连接后不关闭监听程序。这要求演示子流的监控。

3. 在客户端中：
 - a. 识别具有最低指标的接口：

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
```

enp1s0 接口具有比 **wlp1s0** 更低的指标。因此，RHEL 默认使用 **enp1s0**。

- b. 在第一个终端上，启动监控：

ip mptcp monitor

- c. 在第二个终端上，启动到服务器的 MPTCP 连接：

mptcpize run nc 192.0.2.1 12345

RHEL 使用 **enp1s0** 接口及其关联的 IP 地址作为此连接的源。

在监控终端上，**ip mptcp monitor** 命令现在记录日志：

```
[ CREATED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2 daddr4=192.0.2.1
sport=36444 dport=12345
```

这个令牌将 MPTCP 套接字标识为唯一的 ID，之后它会在同一套接字上关联 MPTCP 事件。

- d. 在运行 **nc** 连接到服务器的终端上，按 **Enter** 键。第一个数据包完全建立连接。请注意，如果没有发送数据，连接就不会建立。

在监控终端上，**ip mptcp monitor** 现在记录日志：

```
[ ESTABLISHED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2
daddr4=192.0.2.1 sport=36444 dport=12345
```

- e. 可选：显示到服务器上端口 **12345** 的连接：

```
# ss -taunp | grep ":12345"
tcp ESTAB 0 0      192.0.2.2:36444 192.0.2.1:12345
```

此时，只建立一个到服务器的连接。

- f. 在第三个终端上，创建另一个端点：

ip mptcp endpoint add dev wlp1s0 192.0.2.3 subflow

此命令设置客户端 WiFi 接口的名称和 IP 地址。

在监控终端上，**ip mptcp monitor** 现在记录日志：

```
[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
```

locid 字段显示新子流的本地地址 ID，即使连接使用了网络地址转换(NAT)，也标识此子流。**saddr4** 字段与 **ip mptcp endpoint add** 命令的端点的 IP 地址匹配。

- g. 可选：显示到服务器上端口 **12345** 的连接：


```
# 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
```

该命令现在显示两个连接：

- 源地址为 **192.0.2.2** 的、与之前建立的第一个 MPTCP 子流相对应的连接。
- 来自 **wlp1s0** 接口及源地址 **192.0.2.3** 的子流的连接。

h. 在第三个终端上，删除端点：

```
# ip mptcp endpoint delete id 2
```

使用 **ip mptcp monitor** 输出中的 **locid** 字段的 ID，或者使用 **ip mptcp endpoint show** 命令来检索端点 ID。

在监控终端上，**ip mptcp monitor** 现在记录日志：

```
[ 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. 在第一个带有 **nc** 客户端的终端上，按 **Ctrl+C** 来终止会话。
在监控终端上，**ip mptcp monitor** 现在记录日志：

```
[ CLOSED] token=63c070d2
```

其他资源

- [ip-mptcp\(1\) 手册页](#)
- [NetworkManager 如何管理多个默认网关](#)

35.6. 在内核中禁用多路径 TCP

您可以在内核中明确禁用 MPTCP 选项。

步骤

- 禁用 **mptcp.enabled** 选项。

```
# echo "net.mptcp.enabled=0" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

验证

- 验证 **mptcp.enabled** 是否在内核中被禁用。

```
# sysctl -a | grep mptcp.enabled
net.mptcp.enabled = 0
```

第 36 章 管理 MPTCPD 服务

本节介绍 **mptcpd** 服务的基本管理。**mptcpd** 软件包提供 **mptcpize** 工具，该工具在 **TCP** 环境中的 **mptcp** 协议上切换。

36.1. 配置 MPTCPD

mptcpd 服务是 **mptcp** 协议的一个组件，它提供对配置 **mptcp** 端点的检测。**mptcpd** 服务默认为每个地址创建一个子流端点。端点列表根据正在运行的主机上的 IP 地址修改动态更新。**mptcpd** 服务自动创建端点列表。它使多个路径作为使用 **ip** 工具的替代选择。

先决条件

- 已安装 **mptcpd** 软件包

步骤

1. 使用以下命令在内核中启用 **mptcp.enabled** 选项：

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. 启动 **mptcpd** 服务：

```
# systemctl start mptcp.service
```

3. 验证端点创建：

```
# ip mptcp endpoint
```

4. 要停止 **mptcpd** 服务，请使用以下命令：

```
# systemctl stop mptcp.service
```

5. 要手动配置 **mptcpd** 服务，修改 **/etc/mptcpd/mptcpd.conf** 配置文件。

请注意，端点 **mptcpd** 服务创建，最后一个主机关闭。

其他资源

- **mptcpd(8)** man page。

36.2. 使用 MPTCPIZE 工具管理应用程序

使用 **mptcpize** 工具管理应用程序和服务。

以下指令演示了如何使用 **mptcpize** 工具在 **TCP** 环境中管理应用程序。

假设需要使用启用的 **MPTCP** 套接字运行 **iperf3** 工具。您可以按照以下过程来实现此目标。

先决条件

- 已安装 **mptcpd** 软件包
- **iperf3** 软件包已安装

步骤

- 启动启用 **MPTCP** 套接字的 **iperf3** 工具：

```
# mptcpize run iperf3 -s &
```

36.3. 使用 MPTCPIZE 程序为服务启用 MPTCP 套接字

以下一组命令演示了如何使用 **mptcpize** 工具管理服务。您可以为服务启用或禁用 **mptcp** 套接字。

假设需要管理 **nginx** 服务的 **mptcp** 套接字。您可以按照以下过程来实现此目标。

先决条件

- 已安装 **mptcpd** 软件包
- 安装了 **nginx** 软件包

步骤

1. 为服务启用 **MPTCP** 套接字：

```
# mptcpize enable nginx
```

2. 为服务禁用 **MPTCP** 套接字：

```
# mptcpize disable nginx
```

3. 重启该服务以使更改生效：

```
# systemctl restart nginx
```

第 37 章 密钥文件格式的 NETWORKMANAGER 连接配置文件

默认情况下，Red Hat Enterprise Linux 9 及之后的版本中的 NetworkManager 以 keyfile 格式存储连接配置文件。与已弃用的 `ifcfg` 格式不同，keyfile 格式支持 NetworkManager 提供的所有连接设置。

37.1. NETWORKMANAGER 配置文件的密钥文件格式

keyfile 格式与 INI 格式类似。例如，以下是 keyfile 格式的以太网连接配置文件：

```
[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
```



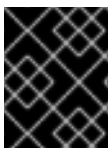
警告

参数的拼写错误或不正确的参数放置可能会导致意外行为。因此，请不要手动编辑或创建 NetworkManager 配置集。

使用 `nmcli` 工具、`network` RHEL 系统角色或 `nmstate` API 来管理 NetworkManager 连接。例如，您可以在 [离线模式下使用 nmcli 工具](#) 创建连接配置集。

每个部分都对应一个 NetworkManager 设置名称，如 `nm-settings(5)` 和 `nm-settings-keyfile(5)` 手册页中所述。该部分中的每一个键值对是手册页设置规范中列出的一个属性。

NetworkManager 密钥文件中的大部分变量都具有一对一映射。这意味着 NetworkManager 属性以相同的名称和格式的变量存储在密钥文件中。不过，也有例外情况，主要是为了使密钥文件语法更易于阅读。有关这些例外的列表，请查看 `nm-settings-keyfile(5)` 手册页。



重要

为安全起见，因为连接配置文件可以包含敏感信息，如私钥和密码短语，所以 NetworkManager 仅使用 `root` 用户所拥有的配置文件，并且仅可由 `root` 读写。

根据连接配置文件的目的是，将其保存在以下目录中：

- `/etc/NetworkManager/system-connections/`：持久配置文件的位置。如果您使用 NetworkManager API 修改了持久配置文件，NetworkManager 会写并覆盖此目录中的文件。

- `/run/NetworkManager/system-connections/` : 用于在重启系统时自动删除的临时配置文件。
- `/usr/lib/NetworkManager/system-connections/` : 用于预先部署的不可变的配置文件。当您使用 NetworkManager API 编辑此类配置文件时, NetworkManager 会将此配置文件复制到持久性存储或临时存储中。

NetworkManager 不会自动从磁盘重新加载配置文件。当您以密钥文件格式创建或更新连接配置文件时, 请使用 `nmcli connection reload` 命令告知 NetworkManager 这些变化。

37.2. 使用 NMCLI 在离线模式下创建密钥文件连接配置集

使用 NetworkManager 工具 (如 `nmcli`、`network` RHEL 系统角色或 `nmstate` API) 来管理 NetworkManager 连接, 以创建和更新配置文件。但是, 您也可以使用 `nmcli --offline connection add` 命令以离线模式创建不同的连接配置集。

脱机模式可确保 `nmcli` 在没有 **NetworkManager** 服务的情况下运行, 以通过标准输出生成 keyfile 连接配置集。此功能在以下情况中很有用:

- 您需要创建需要预先部署的连接配置集。例如在容器镜像中, 或者作为 RPM 软件包。
- 您需要在 **NetworkManager** 服务不可用的环境中创建连接配置集。例如, 您需要使用 `chroot` 实用程序。或者, 当您想通过 Kickstart `%post` 脚本创建或修改 RHEL 系统的网络配置时。

您可以创建以下连接配置集类型:

- 静态以太网连接
- 动态以太网连接
- 网络绑定
- 网桥
- VLAN 或任何支持的连接类型

流程

1. 以 keyfile 格式创建新连接配置集。例如, 对于不使用 DHCP 的以太网设备的连接配置文件, 请运行类似的 `nmcli` 命令:

```
# 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
```



注意

使用 `con-name` 键指定的连接名称被保存到生成的配置集的 `id` 变量中。当您使用 `nmcli` 命令稍后管理这个连接时, 请按如下所示指定连接:

- 如果没有省略 `id` 变量, 请使用连接名称, 如 `Example-Connection`。
- 当没有使用 `id` 变量时, 请使用没有 `.nmconnection` 后缀的文件名, 如 `output`。

2. 对配置文件设置权限, 以便只有 `root` 用户可以读和更新它:

```
# chmod 600 /etc/NetworkManager/system-connections/output.nmconnection
# chown root:root /etc/NetworkManager/system-connections/output.nmconnection
```

3. 启动 **NetworkManager** 服务：

```
# systemctl start NetworkManager.service
```

4. 如果将配置文件中的 **autoconnect** 变量设置为 **false**，激活连接：

```
# nmcli connection up Example-Connection
```

验证

1. 验证 **NetworkManager** 服务是否正在运行：

```
# 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
  ...
```

2. 验证 **NetworkManager** 是否可以从配置文件中读取配置集：

```
# 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
...
```

如果输出没有显示新创建的连接，请验证密钥文件权限和您所用的语法是否正确。

3. 显示连接配置文件：

```
# 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
...
```

其他资源

- [nmcli\(1\)](#)
- [nm-settings-keyfile\(5\)](#)
- [NetworkManager 配置文件的密钥文件格式](#)
- [使用 nmcli 配置以太网连接](#)

- [使用 nmcli 配置 VLAN 标记](#)
- [使用 nmcli 配置网桥](#)
- [使用 nmcli 配置网络绑定](#)

37.3. 手动创建 KEYFILE 格式的 NETWORKMANAGER 配置文件

您可以手动创建密钥文件格式的 NetworkManager 连接配置集。



注意

手动创建或更新配置文件可能会导致意外或无法正常工作的网络配置。作为替代方案，您可以在离线模式下使用 **nmcli**。请参阅 [使用 nmcli 在离线模式下创建 keyfile 连接配置文件](#)

步骤

1. 如果您为硬件接口（如以太网）创建了一个配置文件，请显示此接口的 MAC 地址：

```
# 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. 创建连接配置文件。例如，对于使用 DHCP 的以太网设备的连接配置文件，请使用以下内容创建 **/etc/NetworkManager/system-connections/example.nmconnection** 文件：

```
[connection]
id=example_connection
type=ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[ethernet]
mac-address=00:53:00:8f:fa:66
```



注意

您可以使用任何以 **.nmconnection** 为后缀的文件名。但是，当您稍后使用 **nmcli** 命令来管理连接时，您必须在引用此连接时使用 **id** 变量中设置的连接名称。当省略 **id** 变量时，请使用不带 **.nmconnection** 的文件名来引用此连接。

3. 对配置文件设置权限，以便只有 **root** 用户可以读和更新它：

```
# chown root:root /etc/NetworkManager/system-connections/example.nmconnection
# chmod 600 /etc/NetworkManager/system-connections/example.nmconnection
```

4. 重新加载连接配置文件：

nmcli connection reload

5. 验证 NetworkManager 是否从配置文件读取配置文件：

```
# nmcli -f NAME,UUID,FILENAME connection
NAME          UUID          FILENAME
example-connection 86da2486-068d-4d05-9ac7-957ec118afba
/etc/NetworkManager/system-connections/example.nmconnection
...
```

如果命令未显示新添加的连接，请验证文件权限和您在文件中使用的语法是否正确。

6. 如果将配置文件中的 **autoconnect** 变量设置为 **false**，激活连接：

```
# nmcli connection up example_connection
```

验证

1. 显示连接配置文件：

```
# nmcli connection show example_connection
```

其他资源

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

37.4. IFCFG 和 KEYFILE 格式的使用配置文件重命名接口的差异

您可以定义自定义网络接口名称，如 **provider** 或 **lan**，使接口名称更具有描述性。在这种情况下，**udev** 服务会重命名接口。重命名过程的工作方式根据您是否使用 **ifcfg** 或 **keyfile** 格式的连接配置文件而有所不同。

当以 ifcfg 格式使用配置文件时，重命名进程的接口

1. `/usr/lib/udev/rules.d/60-net.rules` **udev** 规则调用 `/lib/udev/rename_device` 助手工具。
2. 助手工具在 `/etc/sysconfig/network-scripts/ifcfg-*` 文件中搜索 **HWADDR** 参数。
3. 如果变量中设置的值与接口的 MAC 地址匹配，则帮助工具会将接口重命名为文件的 **DEVICE** 参数中设置的名称。

当使用 keyfile 格式的配置文件时，重命名进程的接口

1. 创建一个 [systemd 链接文件](#) 或 [udev 规则](#) 来重命名接口。
2. 在 NetworkManager 连接配置文件的 **interface-name** 属性中使用自定义接口名称。

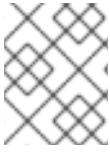
其他资源

- [udev 设备管理器如何重命名网络接口](#)
- [使用 udev 规则配置用户定义的网络接口名称](#)

- 使用 [systemd 链接文件](#) 配置用户定义的网络接口名称

37.5. 将 NETWORKMANAGER 配置集从 IFCFG 迁移到 KEYFILE 格式

如果您仍然使用已弃用的 `ifcfg` 格式的连接配置文件，您可以将它们转换为 `keyfile` 格式。



注意

如果 `ifcfg` 文件包含 `NM_CONTROLLED=no` 设置，则 NetworkManager 不会控制这个配置文件，因此迁移过程会忽略它。

先决条件

- 您有 `ifcfg` 格式的连接配置集，采用 `/etc/sysconfig/network-scripts/` 目录中。
- 如果连接配置文件包含一个设置为自定义设备名称的 `DEVICE` 变量，如 `provider` 或 `lan`，则您将每个自定义设备名称创建一个 [systemd 链接文件](#) 或 [udev 规则](#)。

步骤

- 迁移连接配置集：

```
# nmcli connection migrate
Connection 'enp1s0' (43ed18ab-f0c4-4934-af3d-2b3333948e45) successfully migrated.
Connection 'enp2s0' (883333e8-1b87-4947-8ceb-1f8812a80a9b) successfully migrated.
...
```

验证

- 另外，您可以验证您是否成功迁移了所有连接配置集：

```
# 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
...
```

其他资源

- [nm-settings-keyfile\(5\)](#)
- [nm-settings-ifcfg-rh\(5\)](#)
- [udev 设备管理器如何重命名网络接口](#)

第 38 章 SYSTEMD 网络目标和服务

NetworkManager 在系统引导过程中配置网络。但是，当使用远程 `root(/)` 引导时，例如，如果 `root` 目录存储在 iSCSI 设备上，网络设置会在 RHEL 启动之前在初始 RAM 磁盘(`initrd`)中应用。例如，如果网络配置是使用 `rd.neednet=1` 在内核命令行上指定了，或配置被指定为挂载远程文件系统，那么网络设置将在 `initrd` 上应用。

在应用网络设置时，RHEL 使用 `network` 和 `network-online` 目标以及 `NetworkManager-wait-online` 服务。另外，如果这些服务无法动态重新加载，您可以将 `systemd` 服务配置为在网络完全可用后启动。

38.1. NETWORK 和 NETWORK-ONLINE SYSTEMD TARGET 的不同

`systemd` 维护 `network` 和 `network-online` 目标单元。特殊单元，如 `NetworkManager-wait-online.service`，具有 `WantedBy=network-online.target` 和 `Before=network-online.target` 参数。如果启用了，这些单元将启动 `network-online.target`，并延迟要达到的目标，直到建立了某种形式的网络连接。它们会延迟 `network-online` 目标，直到网络连接了。

`network-online` 目标启动一个服务，这会对进一步执行增加更长的延迟。`systemd` 会自动使用这个目标单元的 `Wants` 和 `After` 参数来向所有 System V(SysV) `init` 脚本服务单元添加依赖项，这些服务单元具有一个指向 `$network` 工具的 Linux Standard Base(LSB)头。LSB 头是 `init` 脚本的元数据。您可以使用它指定依赖项。这与 `systemd` 目标类似。

`network` 目标不会显著延迟引导进程的执行。到达 `network` 目标意味着，负责设置网络的服务已启动。但并不意味着已经配置了一个网络设备。这个目标在关闭系统的过程中非常重要。例如，如果您在引导过程中有一个排在 `network` 目标之后的服务，则这个依赖关系在关闭过程中会反过来。在服务停止后，网络才会断开连接。远程网络文件系统的所有挂载单元都会自动启动 `network-online` 目标单元，并在其之后排序。



注意

`network-online` 目标单元只在系统启动过程中有用。系统完成引导后，这个目标不会跟踪网络的在线状态。因此，您无法使用 `network-online` 来监控网络连接。这个目标提供了一个一次性系统启动概念。

38.2. NETWORKMANAGER-WAIT-ONLINE 概述

`NetworkManager-wait-online` 服务会等待网络配置网络超时。这个网络配置涉及插入以太网设备、扫描 Wi-Fi 设备等。`NetworkManager` 会自动激活配置为自动启动的适当配置集。因 DHCP 超时或类似事件导致自动激活失败，网络管理器 (`NetworkManager`) 可能会在一定时间内处于忙碌状态。根据配置，`NetworkManager` 会重新尝试激活同一配置集或不同的配置集。

当启动完成后，所有配置集都处于断开连接的状态，或被成功激活。您可以配置配置集来自动连接。以下是一些参数示例，这些参数设定超时或者在连接被视为活跃时定义：

- `connection.wait-device-timeout` - 设置用来检测设备的驱动程序的超时时间
- `ipv4.may-fail` 和 `ipv6.may-fail` - 使用一个 IP 地址系列设置激活，或者一个特定的地址系列是否必须已完成配置。
- `ipv4.gateway-ping-timeout` - 延迟激活。

其他资源

- `nm-settings(5)` 手册页

38.3. 将 SYSTEMD 服务配置为在网络已启动后再启动

Red Hat Enterprise Linux 在 `/usr/lib/systemd/system/` 目录中安装 **systemd** 服务文件。此流程为 `/etc/systemd/system/service_name.service.d/` 中的服务文件创建一个置入段，该文件与 `/usr/lib/systemd/system/` 中的服务文件一起使用，以便在网络在线后启动特定的 *服务*。如果置入段中的设置与 `/usr/lib/systemd/system/` 中服务文件中的设置重叠，则它具有更高的优先级。

步骤

1. 要在编辑器中打开服务文件，请输入：

```
# systemctl edit service_name
```

2. 输入以下内容并保存更改：

```
[Unit]
After=network-online.target
```

3. 重新加载 **systemd** 服务。

```
# systemctl daemon-reload
```

第 39 章 NMSTATE 简介

nmstate 是一个声明性网络管理器 API。**nmstate** 软件包提供了 **libnmstate** Python 库和一个命令行工具 **nmstatectl**，来管理 RHEL 上的 NetworkManager。使用 Nmstate 时，您可以使用 YAML 或 JSON 格式的指令描述预期的网络状态。

nmstate 有很多优点。例如，它：

- 提供稳定且可扩展的接口来管理 RHEL 网络功能
- 支持主机和集群级别的原子和事务操作
- 支持对大多数属性进行部分编辑，并保留在说明中没有指定的现有设置
- 提供插件支持，使管理员能够使用自己的插件

39.1. 在 PYTHON 应用程序中使用 LIBNMSTATE 库

libnmstate Python 库可让开发人员在他们自己的应用程序中使用 Nmstate

要使用库，请在源代码中导入它：

```
import libnmstate
```

请注意，您必须安装 **nmstate** 软件包才能使用这个库。

例 39.1. 使用 libnmstate 库查询网络状态

以下 Python 代码导入了 **libnmstate** 库，并显示可用的网络接口及其状态：

```
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])
```

39.2. 使用 NMSTATECTL 更新当前的网络配置

您可以使用 **nmstatectl** 工具将一个或多个接口的当前网络配置存储在一个文件中。然后您可以使用此文件：

- 修改配置并将其应用到同一系统。
- 将文件复制到其他主机上，并使用相同的或经过修改的设置配置主机。

例如，您可以将 **enp1s0** 接口的设置导出到一个文件中，修改配置，将设置应用到主机。

先决条件

- **nmstate** 软件包已安装。

步骤

1. 将 **enp1s0** 接口的设置导出到 `~/network-config.yml` 文件：

```
# nmstatectl show enp1s0 > ~/network-config.yml
```

此命令会以 YAML 格式存储 **enp1s0** 的配置。要以 JSON 格式存储输出，请将 `--json` 选项传给命令。

如果没有指定接口名称，**nmstatectl** 将导出所有接口的配置。

2. 使用文本编辑器修改 `~/network-config.yml` 文件，以更新配置。
3. 应用 `~/network-config.yml` 文件中的设置：

```
# nmstatectl apply ~/network-config.yml
```

如果您以 JSON 格式导出设置，请将 `--json` 选项传给命令。

39.3. NMSTATE SYSTEMD 服务

您可以通过配置 **nmstate systemd** 服务，在 Red Hat Enterprise Linux 系统引导时自动应用新的网络设置。

安装 **nmstate** 软件包后，您可以在 `/etc/nmstate/` 目录中存储带有 Nmstate 指令的 `*.yml` 文件。然后，**nmstate** 服务在下次重启时或手动重启服务时自动应用文件。在 Nmstate 成功应用文件后，它会将文件的 `.yml` 后缀重命名为 `.applied`，以防止服务再次处理同一文件。

nmstate 服务是一个 **oneshot systemd** 服务。因此，只有在系统引导以及手动重启服务时，**systemd** 才会执行它。



注意

默认情况下禁用 **nmstate** 服务。使用 `systemctl enable nmstate` 命令启用它。之后，**systemd** 在每次系统启动时都执行此服务。

39.4. 网络 RHEL 系统角色的网络状态

network RHEL 系统角色支持 playbook 中的状态配置来配置设备。为此，请使用 `network_state` 变量，后面跟上状态配置。

在 playbook 中使用 `network_state` 变量的好处：

- 通过与状态配置结合使用声明方法，您可以配置接口，NetworkManager 会在后台为这些接口创建一个配置集。
- 使用 `network_state` 变量，您可以指定您需要更改的选项，所有其他选项将保持不变。但是，使用 `network_connections` 变量，您必须指定所有设置来更改网络连接配置集。

例如，要使用动态 IP 地址设置创建以太网连接，请在 playbook 中使用以下 `vars` 块：

带有状态配置的 playbook	常规 playbook
<pre>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</pre>	<pre>vars: network_connections: - name: enp7s0 interface_name: enp7s0 type: ethernet autoconnect: yes ip: dhcp4: yes auto6: yes state: up</pre>

例如，要仅更改您之前创建的动态 IP 地址设置的连接状态，请在 playbook 中使用以下 **vars** 块：

带有状态配置的 playbook	常规 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>

其他资源

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

39.5. 其他资源

- `/usr/share/doc/nmstate/README.md`
- `/usr/share/doc/nmstate/examples/`

第 40 章 捕获网络数据包

要调试网络问题和通讯，您可以捕获网络数据包。以下部分提供有关捕获网络数据包的步骤和附加信息。

40.1. 使用 XDPDUMP 捕获包括 XDP 程序丢弃的数据包在内的网络数据包

`xpdump` 工具捕获网络数据包。与 `tcpdump` 工具不同，`xpdump` 为此使用扩展的 Berkeley 数据包过滤(eBPF)程序。这也可使 `xpdump` 能够捕获快速数据路径(XDP)程序丢弃的数据包。用户空间工具（如 `tcpdump`）无法捕获这些被丢弃的数据包，以及 XDP 程序修改的原始数据包。

您可以使用 `xpdump` 来调试已附加到接口上的 XDP 程序。因此，实用程序可以在 XDP 程序启动和完成后捕获数据包。在后一种情况下，`xpdump` 也捕获 XDP 操作。默认情况下，`xpdump` 会在 XDP 程序的入口处捕获传入的数据包。



重要

在 AMD 和 Intel 64 位以外的其他构架上，`xpdump` 工具仅作为技术预览提供。红帽产品服务级别协议 (SLA) 不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些预览可让用户早期访问将来的产品功能，让用户在开发过程中测试并提供反馈意见。

如需有关 [技术预览功能支持范围](#) 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

请注意，`xpdump` 没有数据包过滤或解码功能。但是，您可以将它与 `tcpdump` 结合使用来解码数据包。

先决条件

- 支持 XDP 程序的网络驱动程序。
- XDP 程序被加载到 `enp1s0` 接口。如果没有程序载入，`xpdump` 会以与 `tcpdump` 类似的方式捕获数据包，以便向后兼容。

步骤

1. 要捕获 `enp1s0` 接口上的数据包，并将它们写入到 `/root/capture.pcap` 文件，请输入：

```
# xpdump -i enp1s0 -w /root/capture.pcap
```

2. 要停止捕获数据包，请按 `Ctrl+C`。

其他资源

- [xpdump\(8\) 手册页](#)
- 如果您是开发人员，并且您对 `xpdump` 的源代码感兴趣，请从红帽客户门户网站下载并安装相应的源 RPM(SRPM)。

40.2. 其他资源

- [如何使用 tcpdump 捕获网络数据包？](#)

第 41 章 了解 RHEL 9 中的 EBPF 网络功能

扩展的 Berkeley Packet 过滤器 (eBPF) 是一个内核中的虚拟机，允许在内核空间中执行代码。此代码运行在一个受限的沙箱环境中，仅可访问有限功能集。

在网络中，您可以使用 eBPF 来补充或替换内核数据包处理。根据您使用的 hook，eBPF 程序有：

- 对元数据的读和写的访问权限
- 可以查找套接字和路由
- 可以设置套接字选项
- 可以重定向数据包

41.1. RHEL 9 中网络 EBPF 功能概述

您可以将扩展的 Berkeley 数据包过滤器(eBPF)网络程序附加到 RHEL 中的以下钩子：

- Express Data Path(XDP)：在内核网络堆栈处理它们之前，对接收的数据包提供早期的访问权限。
- 带有直接动作标志的 **tc** eBPF 分类器：对入口和出口提供强大的数据包处理。
- 控制组版本 2(cgroup v2)：在控制组中，对程序所执行的基于套接字的操作启用过滤和覆盖。
- 套接字过滤：启用对从套接字接收的数据包进行过滤。这个功能也可用于经典 Berkeley Packet Filter (cBPF)，但已扩展为支持 eBPF 程序。
- 流解析器：启用将流分成单独的消息、过滤并将其重定向到套接字。
- **SO_REUSEPORT** 套接字选择：对来自 **reuseport** 套接字组的接收套接字提供可编程选择。
- 流程分析器：在某些情况下，启用覆盖内核解析数据包头的方式。
- TCP 拥塞控制回调：启用实现一个自定义 TCP 拥塞控制算法。
- 带有封装的路由：允许创建自定义隧道封装。

XDP

您可以将 **BPF_PROG_TYPE_XDP** 类型的程序附加到网络接口。然后，在内核网络堆栈开始处理之前，内核会在接收的数据包上执行该程序。在某些情况下，这允许快速数据包转发，如快速数据包丢弃以防止分布式拒绝服务(DDoS)攻击，以及负载均衡场景的快速数据包重定向。

您还可以使用 XDP 进行不同类型的数据包监控和抽样。内核允许 XDP 程序修改数据包，并将其传送到内核网络堆栈进行进一步处理。

以下的 XDP 模式可用：

- 原生（驱动程序）XDP：内核在数据包接收过程从最早可能的点执行程序。目前，内核无法解析数据包，因此无法使用内核提供的元数据。这个模式要求网络接口驱动程序支持 XDP，但并非所有驱动程序都支持这种原生模式。
- 通用 XDP：内核网络栈在进程早期执行 XDP 程序。此时内核数据结构已被分配，数据包已被预先处理。如果数据包被丢弃或重定向，与原生模式相比，这需要大量开销。但是，通用模式不需要支持网络接口驱动，它可适用于所有网络接口。

- **Offloaded XDP**：内核在网络接口而不是主机 CPU 上执行 XDP 程序。请注意，这需要特定的硬件，这个模式中只有某些 eBPF 功能可用。

在 RHEL 上，使用 **libxdp** 库加载所有 XDP 程序。这个程序库启用系统控制的 XDP 使用。



注意

目前，XDP 程序有一些系统配置限制。例如：您必须禁用接收接口中某些硬件卸载功能。另外，并非所有功能都可用于支持原生模式的所有驱动程序。

在 RHEL 9 中，红帽仅在使用 **libxdp** 库将程序加载到内核中时，才支持 XDP 功能。

AF_XDP

使用过滤并将数据包重定向到给定的 **AF_XDP** 套接字的 XDP 程序，您可以使用 **AF_XDP** 协议系列中的一个或多个套接字来快速将数据包从内核复制到用户空间。

流量控制

流量控制(tc)子系统提供以下 eBPF 程序类型：

- **BPF_PROG_TYPE_SCHED_CLS**
- **BPF_PROG_TYPE_SCHED_ACT**

这些类型允许您在 eBPF 中编写自定义的 **tc** 分类器和 **tc** 操作。与 **tc** 生态系统的各个部分一起，这为强大的数据包处理提供了能力，是一些容器编排解决方案的核心部分。

在大多数情况下，只有类符被使用，与 **direct-action** 标记一样，eBPF 分类器可以直接从同一 eBPF 程序执行操作。**clsact** 排队规程(**qdisc**)被设计为在入口端启用此功能。

请注意，使用流解析器 eBPF 程序可能会影响其他 **qdiscs** 和 **tc** 分类器的操作，如 **flower**。

套接字过滤器

一些实用程序会使用或在过去使用了 classic Berkeley Packet Filter (cBPF) 过滤套接字上接收到的数据包。例如，**tcpdump** 工具允许用户指定表达式，**tcpdump** 然后将它们转换为 cBPF 码。

作为 cBPF 的替代方案，内核允许 **BPF_PROG_TYPE_SOCKET_FILTER** 类型的 eBPF 程序实现相同的目的。

控制组群

在 RHEL 中，您可以使用多种 eBPF 程序，供您附加到 cgroup。当给定 cgroup 中的某个程序执行某个操作时，内核会执行这些程序。请注意，您只能使用 cgroups 版本 2。

RHEL 中提供以下与网络相关的 cgroup eBPF 程序：

- **BPF_PROG_TYPE SOCK_OPS**：内核对各种 TCP 事件调用该程序。程序可以调整内核 TCP 堆栈的行为，包括自定义 TCP 头选项等。
- **BPF_PROG_TYPE CGROUP SOCK_ADDR**：在 **connect**、**bind**、**sendto**、**recvmsg**、**getpeername** 和 **getsockname** 操作过程中，内核调用该程序。该程序允许更改 IP 地址和端口。当您在 eBPF 中实现基于套接字的网络地址转换(NAT)时，这很有用。
- **BPF_PROG_TYPE CGROUP SOCKOPT**：在 **setsockopt** 和 **getsockopt** 过程中，内核调用该程序，并允许更改选项。

- **BPF_PROG_TYPE_CGROUP_SOCK**：在套接字创建、套接字释放和绑定到地址的过程中，内核调用该程序。您可以使用这些程序来允许或拒绝操作，或者只检查套接字创建统计信息。
- **BPF_PROG_TYPE_CGROUP_SKB**：该程序在入口和出口处过滤单个数据包，并可以接受或拒绝数据包。
- **BPF_PROG_TYPE_CGROUP_SYSCTL**：该程序允许访问系统控制的过滤(**sysctl**)。

流解析器 (Stream Parser)

流解析器对添加到特殊 eBPF 映射中的一组套接字进行操作。然后 eBPF 程序处理内核在那些套接字上接收或发送的数据包。

RHEL 中提供了以下流解析程序 eBPF 程序：

- **BPF_PROG_TYPE_SK_SKB**：eBPF 程序将来自套接字的数据包解析为单独的消息，并指示内核丢弃这些消息或将其发送给组中的另一个套接字。
- **BPF_PROG_TYPE_SK_MSG**：此程序过滤出口消息。eBPF 程序将数据包解析到单个信息中，并批准或拒绝它们。

SO_REUSEPORT 套接字选择

使用这个套接字选项，您可以绑定多个套接字到相同的 IP 地址和端口。如果没有 eBPF，内核会根据连接散列选择接收套接字。有了 **BPF_PROG_TYPE_SK_REUSEPORT** 程序，接收套接字的选择是完全可编程的。

dissector 流程

当内核需要处理数据包头，而不需要查看全部协议解码时，会对它们进行剖析。例如，这会在 **tc** 子系统、多路径路由、绑定或者计算数据包哈希时发生。在这种情况下，内核解析数据包的标头，并使用数据包标头中的信息填充内部结构。您可以使用 **BPF_PROG_TYPE_FLOW_DISSECTOR** 程序替换此内部解析。请注意，您只能在 RHEL 的 eBPF 的 IPv4 和 IPv6 上分离 TCP 和 UDP。

TCP 阻塞控制

您可以使用一组实现 **struct tcp_congestion_oops** 回调的 **BPF_PROG_TYPE_STRUCT_OPS** 程序来编写一个自定义的 TCP 阻塞控制算法。通过这种方法的算法可以和内置内核算法一起提供给系统。

带有封装的路由

您可以将以下 eBPF 程序类型之一附加到路由表中作为隧道封装属性的路由：

- **BPF_PROG_TYPE_LWT_IN**
- **BPF_PROG_TYPE_LWT_OUT**
- **BPF_PROG_TYPE_LWT_XMIT**

这样的 eBPF 程序的功能仅限于特定的隧道配置，它不允许创建通用封装或封装解决方案。

套接字查找

要绕过 **bind** 系统调用的限制，请使用 **BPF_PROG_TYPE_SK_LOOKUP** 类型的 eBPF 程序。此类程序可以为新传入的 TCP 连接选择侦听套接字，或为 UDP 数据包选择一个未连接的套接字。

41.2. 使用网卡 RHEL 9 中的 XDP 功能概述

以下是启用了 XDP 的网卡和您可以使用的 XDP 特性的概述：

网卡	驱动	基本的	重定向	目标	HW 卸载	零复制	大 MTU
Amazon 弹性网络适配卡	ena	是	是	是 [a]	否	否	否
Quantia AQtion 以太网卡	atlantic	是	是	否	否	否	否
Broadcom NetXtreme-C/E 10/25/40/50 千兆以太网	bnxt_en	是	是	是 [a]	否	否	是
Cavium Thunder 虚拟功能	nicvf	是	否	否	否	否	否
Google Virtual NIC (gVNIC)支持	gve	是	是	是	否	是	否
Intel® 10GbE PCI Express 虚拟功能 以太网	ixgbev	是	否	否	否	否	否
Intel® 10GbE PCI Express 适配卡	ixgbe	是	是	是 [a]	否	是	是 [b]
Intel® 以太网连接 E800 系列	lce	是	是	是 [a]	否	是	是
Intel® Ethernet Controller I225- LM/I225-V 系列	igc	是	是	是	否	是	是 [b]
Intel® PCI Express 千兆适配卡	igb	是	是	是 [a]	否	否	是 [b]
Intel® 以太网控制器 XL710 系列	i40e	是	是	是 [a] [c]	否	是	否
Marvell OcteonTX2	rvu_nicpf	是	是	是 [a] [c]	否	否	否
Mellanox 第 5 代网络适配卡 (ConnectX 系列)	mlx5_core	是	是	是 [c]	否	是	是
Mellanox Technologies 1/10/40Gbit 以太网	mlx4_en	是	是	否	否	否	否
Microsoft Azure Network Adapter	mana	是	是	是	否	否	否
Microsoft Hyper-V 虚拟网络	hv_netvsc	是	是	是	否	否	否
Netronome® NFP4000/NFP6000 NIC [d]	nfp	是	否	否	是	是	否
QEMU Virtio 网络	virtio_net	是	是	是 [a]	否	否	是

网卡	驱动	基本的	重定向	目标	HW 卸载	零复制	大 MTU
QLogic QED 25/40/100Gb 以太网 NIC	qede	是	是	是	否	否	否
STMicroelectronics Multi-Gigabit Ethernet	stmmac	是	是	是	否	是	否
Solarflare SFC9000/SFC9100/EF100-系列	sfc	是	是	是 [c]	否	否	否
通用 TUN/TAP 设备	tun	是	是	是	否	否	否
虚拟以太网对设备	veth	是	是	是	否	否	是
VMware VMXNET3 以太驱动程序	vmxnet3	是	是	是 [a] [c]	否	否	否
Xen 半虚拟网络设备	xen-netfront	是	是	是	否	否	否
<p>[a] 只有在接口上加载 XDP 程序时。</p> <p>[b] 仅传输侧。无法通过 XDP 接收大型数据包。</p> <p>[c] 需要分配几个大于或等于最大 CPU 索引的 XDP TX 队列。</p> <p>[d] 一些列出的功能不适用于 Netronome® NFP3800 NIC。</p>							

图例：

- 基本的：支持基本的返回代码：**DROP**、**PASS**、**ABORTED** 和 **TX**。
- 重定向：支持 **XDP_REDIRECT** 返回码。
- 目标：可以是 **XDP_REDIRECT** 返回码的目标。
- HW 卸载：支持 XDP 硬件卸载。
- 零-复制：支持 **AF_XDP** 协议系列的零复制模式。
- 大 MTU：支持大于页大小的数据包。

第 42 章 使用 BPF 编译器集合进行网络追踪

BPF Compiler Collection (BCC) 是一个库，可帮助创建扩展的 Berkeley Packet Filter (eBPF) 程序。eBPF 程序的主要工具是分析操作系统性能和网络性能,而不会遇到开销或安全问题。

BCC 不再需要用户了解 eBPF 的技术详情，并提供了许多开箱即用的起点，如带有预先创建的 eBPF 程序的 **bcc-tools** 软件包。



注意

eBPF 程序在事件中触发，如磁盘 I/O、TCP 连接以及进程创建。程序不太可能导致内核崩溃、循环或者变得无响应，因为它们在内核的安全性虚拟机中运行。

42.1. 安装 BCC-TOOLS 软件包

安装 **bcc-tools** 软件包，该软件包还会将 BPF Compiler Collection (BCC)库作为依赖项安装。

流程

1. 安装 **bcc-tools**。

```
# dnf install bcc-tools
```

BCC 工具安装在 **/usr/share/bcc/tools/** 目录中。

2. (可选) 检查工具：

```
# 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
...
```

上表中的 **doc** 目录包含每个工具的文档。

42.2. 显示添加到内核的接受队列中的 TCP 连接

内核在 TCP 3 向握手中接收 **ACK** 数据包后，内核会将来自 **SYN** 队列的连接移到 **accept** 队列，直到连接的状态变为 **ESTABLISHED**。因此，只有成功的 TCP 连接才能在此队列中看到。

tcpaccept 工具使用 eBPF 特性显示内核添加到 **accept** 队列的所有连接。该工具是轻量级的，因为它跟踪内核的 **accept ()** 函数，而不是捕获和过滤数据包。例如，使用 **tcpaccept** 进行常规故障排除，来显示服务器已接受的新连接。

步骤

1. 输入以下命令来启动对内核 **accept** 队列的追踪：

■

```
# /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
...
```

每次内核接受一个连接时，**tcpaccept** 都会显示连接的详情。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpaccept(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpaccept_example.txt` file

42.3. 追踪出去的 TCP 连接尝试

tcpconnect 工具使用 eBPF 特性来跟踪出去的 TCP 连接尝试。该工具的输出还包括失败的连接。

tcpconnect 工具是轻量级的，例如，因为它跟踪内核的 **connect ()** 函数，而不是捕获和过滤数据包。

步骤

1. 输入以下命令启动显示所有传出连接的追踪过程：

```
# /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
...
```

每次内核处理一个出去的连接时，**tcpconnect** 都会显示连接的详情。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpconnect(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpconnect_example.txt` file

42.4. 测量出站 TCP 连接的延迟

TCP 连接延迟是建立连接所需的时间。这通常涉及内核 TCP/IP 处理和网络往返时间，而不是应用程序运行时。

tcpconlat 工具使用 eBPF 特性来测量发送 **SYN** 数据包和接收响应数据包之间的时间。

步骤

1. 开始测量出站连接的延迟：

```
# /usr/share/bcc/tools/tcpconlat
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
...
```

每次内核处理一个出去的连接时，**tcpconlat** 都会在内核接收响应数据包后显示连接的详细信息。

- 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpconlat(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpconlat_example.txt` 文件

42.5. 显示被内核丢弃的 TCP 数据包和片段详情

tcpdrop 工具使管理员能够显示内核所丢弃的 TCP 数据包和段的详情。使用这个实用程序调试丢弃数据包的高速率，以便远程系统发送基于计时器的重新传输。释放数据包和片段的高速率可能会影响服务器的性能。

tcpdrop 工具使用 eBPF 特性，而不是捕获和过滤资源密集型的数据包，来直接从内核检索信息。

步骤

- 输入以下命令来显示丢弃 TCP 数据包和片段详情：

```
# /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'
...
```

每次内核丢弃 TCP 数据包和段时，**tcpdrop** 都会显示连接的详情，包括导致软件包丢弃的内核堆栈追踪。

- 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpdrop(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpdrop_example.txt` file

42.6. 追踪 TCP 会话

tcplife 工具使用 eBPF 跟踪打开和关闭的 TCP 会话，并打印一行输出来总结每一个会话。管理员可以使用 **tcplife** 来识别连接和传输的流量数。

例如，您可以显示到端口 **22** (SSH) 的连接来检索以下信息：

- 本地进程 ID (PID)
- 本地进程名称
- 本地 IP 地址和端口号
- 远程 IP 地址和端口号
- 接收和传输的流量的数量（以 KB 为单位）。
- 连接处于活跃状态的时间（毫秒）

步骤

1. 输入以下命令来开始追踪到本地端口 **22** 的连接：

```
/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
...
```

每次关闭连接时，**tcplife** 都会显示连接的详情。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcplife(8)** 手册页
- `/usr/share/bcc/tools/doc/tcplife_example.txt` 文件

42.7. 追踪 TCP 重新传输

tcpretrans 工具显示有关 TCP 重新传输的详细信息，如本地和远程的 IP 地址和端口号，以及重新传输时 TCP 的状态。

该工具使用 eBPF 功能，因此开销非常低。

流程

1. 使用以下命令来显示 TCP 重新传输详情：

```
# /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
...
```


每次内核调用 TCP 重新传输函数时，**tcpretrans** 都会显示连接的详情。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpretrans(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpretrans_example.txt` file

42.8. 显示 TCP 状态更改信息

在 TCP 会话中，TCP 状态会改变。**tcpstates** 工具使用 eBPF 功能跟踪这些状态变化，并打印包括每个状态持续时间的详细信息。例如，使用 **tcpstates** 来确定连接是否在初始化状态中花费了太多时间。

步骤

1. 使用以下命令开始追踪 TCP 状态变化：

```
# /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
...
```

每次连接改变其状态时，**tcpstates** 都会显示一个新行，其中包含更新的连接详情。

如果多个连接同时改变了其状态，请使用第一列中的套接字地址(**SKADDR**)来确定哪些条目属于同一个连接。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- **tcpstates(8)** 手册页
- `/usr/share/bcc/tools/doc/tcpstates_example.txt` file

42.9. 聚合发送到特定子网的 TCP 流量

tcpsubnet 工具汇总并合计了本地主机发往子网的 IPv4 TCP 流量，并按固定间隔显示输出。该工具使用 eBPF 功能来收集并总结数据，以减少开销。

默认情况下，**tcpsubnet** 为以下子网汇总流量：

- **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

请注意，最后一个子网(0.0.0.0/0)是一个全包括选项。`tcpsubnet` 工具计算与这个全包括条目中前四个不同的子网的所有流量。

按照以下流程计算 `192.0.2.0/24` 和 `198.51.100.0/24` 子网的流量。到其他子网的流量将在 `0.0.0.0/0` 全包括子网条目中跟踪。

步骤

1. 开始监控发送到 `192.0.2.0/24`、`198.51.100.0/24` 以及其他子网的流量数：

```
# /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
...
```

这个命令以字节为单位显示指定子网每秒一次的流量。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- [tcpsubnet\(8\) 手册页](#)
- `/usr/share/bcc/tools/doc/tcpsubnet.txt` 文件

42.10. 通过 IP 地址和端口显示网络吞吐量

`tcptop` 工具以 KB 为单位显示主机发送并接收的 TCP 流量。这个报告会自动刷新并只包含活跃的 TCP 连接。该工具使用 eBPF 功能，因此开销非常低。

流程

1. 要监控发送和接收的流量，请输入：

```
# /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
...
```

命令的输出只包括活跃的 TCP 连接。如果本地或者远程系统关闭了连接，则该连接在输出中不再可见。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- [tcptop\(8\) 手册页](#)
- `/usr/share/bcc/tools/doc/tcptop.txt` 文件

42.11. 追踪已建立的 TCP 连接

`tcptracer` 工具跟踪连接、接受和关闭 TCP 连接的内核函数。该工具使用 eBPF 功能，因此开销非常低。

流程

1. 使用以下命令启动追踪过程：

```
# /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
...
```

每当内核连接、接受或关闭连接时，`tcptracer` 都会显示连接的详情。

2. 按 **Ctrl+C** 停止追踪过程。

其他资源

- [tcptracer\(8\) 手册页](#)
- `/usr/share/bcc/tools/doc/tcptracer_example.txt` file

42.12. 追踪 IPV4 和 IPV6 侦听尝试

`solisten` 工具追踪所有 IPv4 和 IPv6 侦听尝试。它跟踪监听尝试，包括最终失败或者不接受连接的监听程序。当程序要侦听 TCP 连接时，程序会追踪内核调用的功能。

流程

1. 输入以下命令启动显示所有监听 TCP 尝试的追踪过程：

```
# /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
....
```

- 按 **Ctrl+C** 停止追踪过程。

其他资源

- [solisten\(9\) 手册页](#)
- `/usr/share/bcc/tools/doc/solisten_example.txt` file

42.13. 软中断的服务时间概述

softirqs 工具总结了服务软中断 (soft IRQ) 所花费的时间，并将这个时间显示为总计或直方图分布。这个工具使用 `irq:softirq_enter` 和 `irq:softirq_exit` 内核追踪点，是一个稳定的追踪机制。

步骤

- 输入以下命令启动追踪 **soft irq** 事件时间：

```
# /usr/share/bcc/tools/softirqs
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ      TOTAL_usec
tasklet      166
block        9152
net_rx       12829
rcu          53140
sched        182360
timer        306256
```

- 按 **Ctrl+C** 停止追踪过程。

其他资源

- [softirqs\(8\) 手册页](#)
- `/usr/share/bcc/tools/doc/softirqs_example.txt` file
- [mpstat\(1\) 手册页](#)

42.14. 总结在网络接口上数据包大小和计数

netqtop 工具显示有关特定网络接口的每个网络队列上收到的(RX)和传输的(TX)数据包属性的统计信息。统计包括：

- 每秒字节数(BPS)
- 每秒数据包数(PPS)
- 平均数据包大小
- 数据包总数

要生成这些统计数据，**netqtop** 跟踪执行传输数据包 `net_dev_start_xmit` 和接收数据包 `netif_receive_skb` 的事件的内核功能。

步骤

1. 显示 2 秒时间间隔的字节大小范围内的数据包数：

```
# /usr/share/bcc/tools/netqtop -n enp1s0 -i 2

Fri Jan 31 18:08:55 2023
TX
QueueID avg_size [0, 64) [64, 512) [512, 2K) [2K, 16K) [16K, 64K)
0 0 0 0 0 0 0
Total 0 0 0 0 0 0

RX
QueueID avg_size [0, 64) [64, 512) [512, 2K) [2K, 16K) [16K, 64K)
0 38.0 1 0 0 0 0
Total 38.0 1 0 0 0 0
-----
Fri Jan 31 18:08:57 2023
TX
QueueID avg_size [0, 64) [64, 512) [512, 2K) [2K, 16K) [16K, 64K)
0 0 0 0 0 0 0
Total 0 0 0 0 0 0

RX
QueueID avg_size [0, 64) [64, 512) [512, 2K) [2K, 16K) [16K, 64K)
0 38.0 1 0 0 0 0
Total 38.0 1 0 0 0 0
-----
```

2. 按 **Ctrl+C** 停止 `netqtop`。

其他资源

- `netqtop(8)` 手册页
- `/usr/share/bcc/tools/doc/netqtop_example.txt`

42.15. 其他资源

- `/usr/share/doc/bcc/README.md`

第 43 章 配置网络设备以接受来自所有 MAC 地址的流量

网络设备通常会拦截和读取编程的控制器接收的数据包。您可以在虚拟交换机或端口组层面上，将网络设备配置为接受来自所有 MAC 地址的流量。

您可以使用这个网络模式来：

- 诊断网络连接问题
- 出于安全原因监控网络活动
- 拦截传输中的私有数据或网络中的入侵

您可以为任何类型的网络设备启用此模式，除了 **InfiniBand**。

43.1. 临时配置设备以接受所有流量

您可以使用 **ip** 工具临时配置网络设备以接受所有流量，而不考虑 MAC 地址。

步骤

1. 可选：显示网络接口以标识您要接收所有流量的接口：

```
# ip address show
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. 修改设备以启用或禁用此属性：

- 为 **enp1s0** 启用 **accept-all-mac-addresses** 模式：

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

- 为 **enp1s0** 禁用 **accept-all-mac-addresses** 模式：

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

验证

- 验证 **accept-all-mac-addresses** 模式是否已启用：

```
# 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
```

设备描述中的 **PROMISC** 标志表示启用了该模式。

43.2. 使用 NMCLI 永久配置网络设备以接受所有流量

您可以使用 **nmcli** 工具永久配置网络设备以接受所有流量，而不考虑 MAC 地址。

步骤

1. 可选：显示网络接口以标识您要接收所有流量的接口：

```
# ip address show
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
    ...
```

如果没有任何连接，您可以创建一个新的连接。

2. 修改网络设备以启用或禁用此属性。

- 为 **enp1s0** 启用 **ethernet.accept-all-mac-addresses** 模式：

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

- 为 **enp1s0** 禁用 **accept-all-mac-addresses** 模式：

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

3. 应用更改，重新激活连接：

```
# nmcli connection up enp1s0
```

验证

- 验证是否启用了 **ethernet.accept-all-mac-addresses** 模式：

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

802-3-ethernet.accept-all-mac-addresses: true 表示该模式已启用。

43.3. 使用 NMSTATECTL 永久配置网络设备以接受所有流量

使用 **nmstatectl** 工具将设备配置为接受所有流量，而不考虑通过 Nmstate API 的 MAC 地址。Nmstate API 确保设置配置后，结果与配置文件匹配。如果有任何失败，**nmstatectl** 会自动回滚更改以避免系统处于不正确的状态。

先决条件

- **nmstate** 软件包已安装。
- 用于配置设备的 **enp1s0.yml** 文件可用。

步骤

1. 编辑 **enp1s0** 连接的现有 **enp1s0.yml** 文件，并将以下内容添加到其中：

```
---
```

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

这些设置将 **enp1s0** 设备配置为接受所有流量。

2. 应用网络设置：

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

验证

- 验证是否启用了 **802-3-ethernet.accept-all-mac-addresses** 模式：

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

802-3-ethernet.accept-all-mac-addresses: true 表示该模式已启用。

其他资源

- [nmstatectl\(8\) 手册页](#)
- [/usr/share/doc/nmstate/examples/ 目录](#)

第 44 章 使用 NMCLI 对网络接口进行镜像

网络管理员可以使用端口镜像复制从一个网络设备传输到另一个网络设备的入站和出站网络流量。在以下情况下，接口的镜像流量很有帮助：

- 调试网络问题并调优网络流
- 检查和分析网络流量
- 检测入侵

先决条件

- 一个镜像网络流量的网络接口。

步骤

1. 添加您要镜像网络流量的网络连接配置集：

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

2. 将 **prio qdisc** 附加到带有 **10: handle** 的出口（传出）流量的 **enp1s0**：

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

在没有子项的情况下附加 **prio qdisc** 可允许附加过滤器。

3. 为入口流量添加 **qdisc**，使用 **ffff: handle**：

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

4. 添加以下过滤器，以匹配入口和出口 **qdiscs** 上的数据包，并将其镜像到 **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"
```

matchall 过滤器与所有数据包匹配，**mirrored** 操作会将数据包重定向到目的地。

5. 激活连接：

```
# nmcli connection up enp1s0
```

验证

1. 安装 **tcpdump** 工具：

```
# dnf install tcpdump
```

2. 显示目标设备上镜像的流量 (**enp7s0**)：

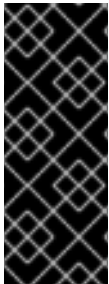
```
# tcpdump -i enp7s0
```

其他资源

- [如何使用 tcpdump 捕获网络数据包](#)

第 45 章 使用 NMSTATE-AUTOCONF 自动配置使用 LLDP 的网络状态

网络设备可以使用链路层发现协议(LLDP)，来在 LAN 中公告其身份、功能和邻居。**nmstate-autoconf** 工具可使用此信息来自动配置本地网络接口。



重要

nmstate-autoconf 工具仅作为技术预览提供。红帽产品服务级别协议 (SLA) 不支持技术预览功能，且其功能可能并不完善，因此红帽不建议在生产环境中使用它们。这些预览可让用户早期访问将来的产品功能，让用户在开发过程中测试并提供反馈意见。

如需有关 [技术预览功能支持范围](#) 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

45.1. 使用 NMSTATE-AUTOCONF 来自动配置网络接口

nmstate-autoconf 工具使用 LLDP 来识别连接到交换机的接口的 VLAN 设置来配置本地设备。

此流程假设以下场景，以及交换机使用 LLDP 广播 VLAN 设置：

- RHEL 服务器的 **enp1s0** 和 **enp2s0** 接口连接到使用 VLAN ID **100** 和 VLAN 名称 **prod-net** 配置的交换机端口。
- RHEL 服务器的 **enp3s0** 接口连接到使用 VLAN ID **200** 和 VLAN 名称 **mgmt-net** 配置的交换机端口。

然后，**nmstate-autoconf** 工具使用此信息来在服务器上创建以下接口：

- **bond100** - **enp1s0** 和 **enp2s0** 作为端口的绑定接口。
- **prod-net** - 在 VLAN ID 为 **100** 的 **bond100** 上面的 VLAN 接口。
- **mgmt-net** - 在 VLAN ID 为 **200** 的 **enp3s0** 上面的 VLAN 接口

如果您将多个网络接口连接到 LLDP 用来广播同一 VLAN ID 的不同交换机的端口，则 **nmstate-autoconf** 会用这些接口来创建一个绑定，并在其上配置通用 VLAN ID。

先决条件

- **nmstate** 软件包已安装。
- 网络交换机上启用了 LLDP。
- 以太网接口已启用。

步骤

1. 在以太网接口上启用 LLDP：
 - a. 创建一个包含以下内容的 YAML 文件，如 **~/enable-lldp.yml**：

```
interfaces:
  - name: enp1s0
    type: ethernet
```

```
lldp:
  enabled: true
- name: enp2s0
  type: ethernet
  lldp:
    enabled: true
- name: enp3s0
  type: ethernet
  lldp:
    enabled: true
```

- b. 将设置应用到系统：

```
# nmstatectl apply ~/enable-lldp.yml
```

2. 使用 LLDP 配置网络接口：

- a. 可选，启动一个空运行来显示并验证 **nmstate-autoconf** 生成的 YAML 配置：

```
# 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. 使用 **nmstate-autoconf** 根据从 LLDP 接收的信息来生成配置，并将设置应用到系统：

```
# nmstate-autoconf enp1s0,enp2s0,enp3s0
```

后续步骤

- 如果您的网络中没有为接口提供 IP 设置的 DHCP 服务器，请手动配置它们。详情请查看：
 - [配置以太网连接](#)
 - [配置网络绑定](#)

验证

1. 显示单个接口的设置：

```
# nmstatectl show <interface_name>
```

其他资源

- [nmstate-autoconf\(8\) 手册页](#)

第 46 章 配置 802.3 链路设置

自动协商是 IEEE 802.3u 快速以太网协议的一个特性。它以设备端口为目标，为链路上的信息交换提供速度、双工模式和流控制的性能。使用自动协商协议时，您具有在以太网上进行数据传输的最佳性能。



注意

要最大限度地利用自动协商的性能，请在链接两端使用同样的配置。

46.1. 使用 NMCLI 工具配置 802.3 链路设置

要配置以太网连接的 802.3 链路设置，请修改以下配置参数：

- **802-3-ethernet.auto-negotiate**
- **802-3-ethernet.speed**
- **802-3-ethernet.duplex**

步骤

1. 显示连接的当前设置：

```
# nmcli connection show Example-connection
...
802-3-ethernet.speed: 0
802-3-ethernet.duplex: --
802-3-ethernet.auto-negotiate: no
...
```

如果需要在有问题的时候重置参数，您可以使用这些值。

2. 设置速度和双工链路设置：

```
# nmcli connection modify Example-connection 802-3-ethernet.auto-negotiate yes 802-3-ethernet.speed 10000 802-3-ethernet.duplex full
```

此命令启用自动协商，并将连接速度设置为 **10000** Mbit 全双工。

3. 重新激活连接：

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

验证

- 使用 **ethtool** 工具验证以太网接口 **enp1s0** 的值：

```
# ethtool enp1s0

Settings for enp1s0:
...
Speed: 10000 Mb/s
Duplex: Full
```

Auto-negotiation: on

...

Link detected: yes

其他资源

- [nm-settings\(5\) 手册页](#)

第 47 章 DPDK 入门

数据平面开发套件(DPDK)提供库和网络驱动程序来加快用户空间中的数据包处理。

管理员使用 DPDK，例如，在虚拟机中使用单一根 I/O 虚拟化 (SR-IOV) 来减少延迟并增加 I/O 吞吐量。



注意

红帽不支持实验性的 DPDK API。

47.1. 安装 DPDK 软件包

要使用 DPDK，请安装 **dpdk** 软件包。

步骤

- 使用 **dnf** 工具安装 **dpdk** 软件包：

```
# dnf install dpdk
```

47.2. 其他资源

- [网络适配器快速数据路径特性支持矩阵](#)

第 48 章 TIPC 入门

透明进程间通信(TIPC) (也称为 **集群域套接字**) 是用于 集群范围操作的进程间通信(IPC)服务。

在高可用性和动态集群环境中运行的应用程序有特殊需要。集群中的节点数量可能会有所不同，路由器可能会失败，且出于负载均衡的考虑，功能也可以移到集群中的不同节点。TIPC 可最大程度降低应用程序开发人员处理此类问题的的工作，并尽可能以正确和最佳的方式处理它们。另外，TIPC 比一般协议（如 TCP）提供效率更高且容错的通讯。

48.1. TIPC 的构架

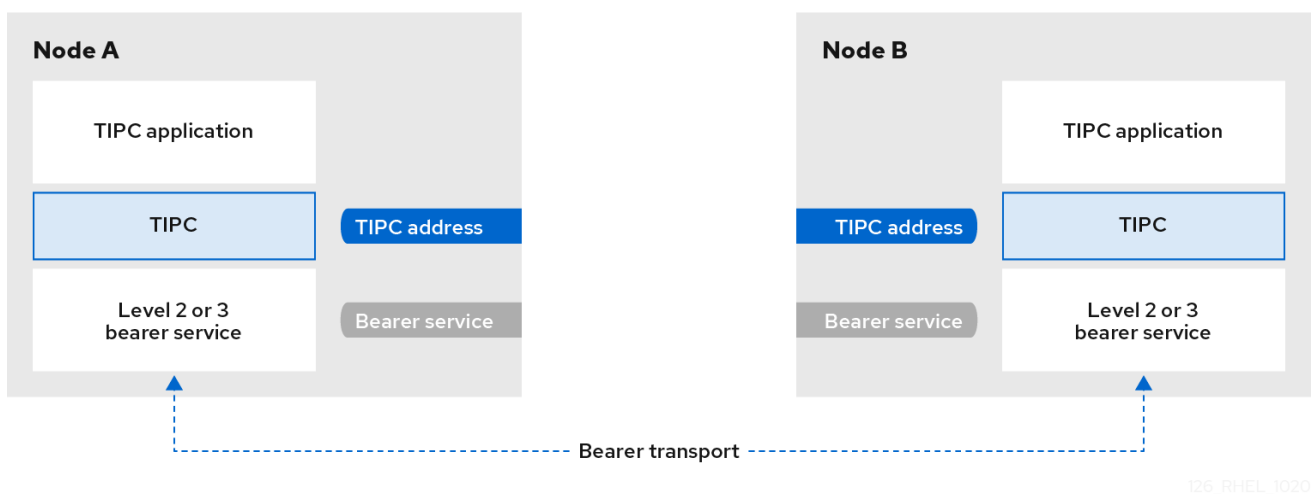
TIPC 是使用 TIPC 和数据包传输服务(**bearer**)的应用程序之间的一个层，横跨传输层、网络层和信令链路层。然而，TIPC 可以使用不同的传输协议作为 bearer，这样 TCP 连接就可以充当 TIPC 信号连接的 bearer。

TIPC 支持以下 bearer:

- Ethernet
- InfiniBand
- UDP 协议

TIPC 提供了在 TIPC 端口间可靠传送信息，这是所有 TIPC 通讯的端点。

以下是 TIPC 构架图：



126_RHEL_1020

48.2. 系统引导时载入 TIPC 模块

在使用 TIPC 协议前，您必须载入 **tipc** 内核模块。您可以将 Red Hat Enterprise Linux 配置为在系统引导时自动载入这个内核模块。

步骤

1. 使用以下内容创建 `/etc/modules-load.d/tipc.conf` 文件：

```
tipc
```

2. 重启 **systemd-modules-load** 服务，以在不重启系统的情况下加载模块：

```
# systemctl start systemd-modules-load
```

验证

1. 使用以下命令验证 RHEL 是否已载入 **tipc** 模块：

```
# lsmod | grep tipc
tipc 311296 0
```

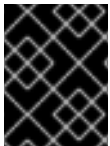
如果命令没有显示 **tipc** 模块的条目，则 RHEL 没有加载它。

其他资源

- **modules-load.d(5)** 手册页

48.3. 创建 TIPC 网络

要创建 TIPC 网络，请在应该加入 TIPC 网络的每个主机上执行这个流程。



重要

这些命令只临时配置 TIPC 网络。要在节点上永久配置 TIPC，在脚本中使用此流程的命令，并将 RHEL 配置为在系统引导时执行该脚本。

先决条件

- **tipc** 模块已加载。详情请查看 [当系统启动时加载 tipc 模块](#)

步骤

1. 可选：设置一个唯一的节点身份，如 UUID 或节点的主机名：

```
# tipc node set identity host_name
```

身份可以是任何由最多 16 个字母和数字组成的唯一字符串。

您不能在此步骤后设置或改变身份。

2. 添加一个 bearer。例如，要将 Ethernet 用作介质，并将 **enp0s1** 设备用作物理 bearer 设备，请输入：

```
# tipc bearer enable media eth device enp1s0
```

3. 可选：要获得冗余和更好的性能，请使用上一步中的命令附加更多 bearer。您可以配置最多三个 bearer，但在同一介质上不能超过两个。
4. 在应该加入 TIPC 网络的每个节点中重复前面的所有步骤。

验证

1. 显示集群成员的链接状态：

```
# tipc link list
broadcast-link: up
5254006b74be:enp1s0-525400df55d1:enp1s0: up
```

此输出表示，节点 **5254006b74be** 上的 bearer **enp1s0** 和节点 **525400df55d1** 上的 bearer **enp1s0** 之间的链接为 **up**。

2. 显示 TIPC 发布表：

```
# 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
```

- 服务类型为 **0** 的两个条目表示两个节点是这个集群的成员。
- 服务类型为 **1** 的条目代表内置的拓扑服务跟踪服务。
- 服务类型为 **2** 的条目显示从发布节点看到的链接。范围限制 **3741353223** 表示十进制格式的对等端点的地址（基于节点身份的唯一 32 位哈希值）。

其他资源

- [tipc-bearer\(8\)](#) 手册页
- [tipc-namespace\(8\)](#) 手册页

48.4. 其他资源

- 红帽建议使用其他 bearer 级别协议来根据传输介质加密节点之间的通信。例如：
 - MACsec：请参阅 [使用 MACsec 来加密第 2 层流量](#)
 - IPsec：请参阅 [使用 IPsec 配置 VPN](#)
- 有关如何使用 TIPC 的示例，请使用 `git clone git://git.code.sf.net/p/tipc/tipc/tipcutils` 命令克隆上游的 GIT 存储库。该仓库包含使用 TIPC 功能的演示和测试程序的源代码。请注意，这个软件仓库不是由红帽提供的。
- `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/output/networking/tipc.html` 由 `kernel-doc` 软件包提供。

第 49 章 使用 NM-CLOUD-SETUP 在公有云中自动配置网络接口

通常，虚拟机(VM)只有一个可由 DHCP 配置的接口。但是，DHCP 无法使用多个网络实体配置虚拟机，如接口、IP 子网和 IP 地址。另外，您无法在虚拟机实例运行时应用设置。要解决这个运行时配置问题，**nm-cloud-setup** 工具会自动从云服务供应商的元数据服务器检索配置信息，并更新主机的网络配置。工具自动在一个接口上获取多个网络接口、多个 IP 地址或 IP 子网，并帮助重新配置正在运行的虚拟机实例的网络。

49.1. 配置和预部署 NM-CLOUD-SETUP

要在公有云中启用和配置网络接口，请运行 **nm-cloud-setup** 作为计时器和服务。



注意

在 Red Hat Enterprise Linux On Demand 和 AWS 黄金镜像上，**nm-cloud-setup** 已启用，无需任何操作。

前提条件

- 存在网络连接。
- 连接使用 DHCP。
默认情况下，NetworkManager 会创建一个使用 DHCP 的连接配置文件。如果因为您在 `/etc/NetworkManager/NetworkManager.conf` 中设置了 **no-auto-default** 参数而没有创建配置文件，请手动创建此初始连接。

步骤

1. 安装 **nm-cloud-setup** 软件包：

```
# dnf install NetworkManager-cloud-setup
```

2. 为 **nm-cloud-setup** 服务创建并运行 管理单元文件：

- a. 使用以下命令开始编辑管理单元文件：

```
# systemctl edit nm-cloud-setup.service
```

显式启动服务或重启系统，以使配置设置生效是非常重要的。

- b. 使用 **systemd** 管理单元文件来在 **nm-cloud-setup** 中配置云提供商。例如，要使用 Amazon EC2，请输入：

```
[Service]
Environment=NM_CLOUD_SETUP_EC2=yes
```

您可以设置以下环境变量来启用您所使用的云提供商：

- 用于 Microsoft Azure 的 **NM_CLOUD_SETUP_AZURE**
- 用于 Amazon EC2(AWS)的 **NM_CLOUD_SETUP_EC2**
- 用于 Google Cloud Platform(GCP)的 **NM_CLOUD_SETUP_GCP**

- 用于 Alibaba Cloud (Aliyun) 的 **NM_CLOUD_SETUP_ALIYUN**

c. 保存文件并退出编辑器。

3. 重新载入 **systemd** 配置：

```
# systemctl daemon-reload
```

4. 启用并启动 **nm-cloud-setup** 服务：

```
# systemctl enable --now nm-cloud-setup.service
```

5. 启用并启动 **nm-cloud-setup** 计时器：

```
# systemctl enable --now nm-cloud-setup.timer
```

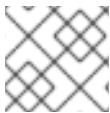
其他资源

- **nm-cloud-setup(8)** 手册页
- [配置以太网连接](#)

49.2. 了解 RHEL EC2 实例中 IMDSV2 和 NM-CLOUD-SETUP 的角色

Amazon EC2 中的实例元数据服务(IMDS)允许您管理访问正在运行的 Red Hat Enterprise Linux (RHEL) EC2 实例的实例元数据的权限。RHEL EC2 实例使用 IMDS 版本 2 (IMDSv2)，一个面向会话的方法。通过使用 **nm-cloud-setup** 工具，管理员可以重新配置网络，并自动更新正在运行的 RHEL EC2 实例的配置。**nm-cloud-setup** 工具通过使用 IMDSv2 令牌处理 IMDSv2 API 调用，而无需用户干预。

- IMDS 运行在本地链路地址 **169.254.169.254** 上，来提供对 RHEL EC2 实例上原生应用程序的访问。
- 为应用程序和用户的每个 RHEL EC2 实例指定并配置了 IMDSv2 后，您无法再访问 IMDSv1。
- 通过使用 IMDSv2，RHEL EC2 实例可在不使用 IAM 角色的情况下维护元数据，同时通过 IAM 角色保留可访问。
- 当 RHEL EC2 实例引导时，**nmn-cloud-setup** 工具会自动运行，以获取使用 RHEL EC2 实例 API 的 EC2 实例 API 访问令牌。



注意

使用 IMDSv2 令牌作为 HTTP 标头，来检查 EC2 环境的详情。

其他资源

- **nm-cloud-setup(8)** 手册页