Red Hat Enterprise Linux 9

ネットワークの設定および管理

ネットワークインターフェイスおよび高度なネットワーク機能の管理

Last Updated: 2024-04-22

Red Hat Enterprise Linux 9 ネットワークの設定および管理

ネットワークインターフェイスおよび高度なネットワーク機能の管理

法律上の通知

概要

Red Hat Enterprise Linux (RHEL) のネットワーク機能を使用すると、組織のネットワーク要件とセキュリティー要件に合わせてホストを設定できます。以下に例を示します。 ボンディング、VLAN、ブリッジ、トンネル、およびその他のネットワークタイプを設定して、ホストをネットワークに接続できます。 IPSec と WireGuard は、ホストとネットワーク間にセキュアな VPN を提供します。 RHEL は、ポリシーベースのルーティングやマルチパス TCP (MPTCP) などの高度なネットワーク機能もサポートします。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 一貫したネットワークインターフェイス命名の実装

udev デバイスマネージャーは、Red Hat Enterprise Linux で一貫したデバイス命名を実装します。デバイスマネージャーは、さまざまな命名スキームをサポートしています。デフォルトでは、ファームウェア、トポロジー、および場所の情報に基づいて固定名を割り当てます。

一貫したデバイス命名を使用しない場合、Linux カーネルは固定の接頭辞とインデックスを組み合わせて名前をネットワークインターフェイスに割り当てます。カーネルがネットワークデバイスを初期化すると、インデックスが増加します。たとえば、**eth0** は、起動時にプローブされる最初のイーサネットデバイスを表します。別のネットワークインターフェイスコントローラーをシステムに追加すると、再起動後にデバイスが異なる順序で初期化される可能性があるため、カーネルデバイス名の割り当てが一定でなくなります。その場合、カーネルはデバイスに別の名前を付けることがあります。

この問題を解決するために、**udev** は一貫したデバイス名を割り当てます。これには、次の利点があります。

- 再起動してもデバイス名が変わりません。
- ハードウェアを追加または削除しても、デバイス名が固定されたままになります。
- 不具合のあるハードウェアをシームレスに交換できます。
- ネットワークの命名はステートレスであり、明示的な設定ファイルは必要ありません。

警告

通常、Red Hat は、一貫したデバイス命名が無効になっているシステムはサポートしていません。例外については、Is it safe to set net.ifnames=0 ソリューションを参照してください。

1.1. UDEV デバイスマネージャーによるネットワークインターフェイスの名前変更の仕組み

ネットワークインターフェイスの一貫した命名スキームを実装するために、**udev** デバイスマネージャーは次のルールファイルを記載されている順番どおりに処理します。

1. オプション: **/usr/lib/udev/rules.d/60-net.rules**
   このファイルは、**initscripts-rename-device** パッケージをインストールした場合にのみ存在します。**/usr/lib/udev/rules.d/60-net.rules** ファイルは、非推奨の **/usr/lib/udev/rename_device** ヘルパーユーティリティーが **/etc/sysconfig/network-scripts/ifcfg-*** ファイルの **HWADDR** パラメーターを検索することを定義します。変数に設定した値がインターフェイスの MAC アドレスに一致すると、ヘルパーユーティリティーは、インターフェイスの名前を、**ifcfg** ファイルの **DEVICE** パラメーターに設定した名前に変更します。

   システムがキーファイル形式の NetworkManager 接続プロファイルのみを使用する場合、**udev** はこの手順をスキップします。

2. Dell システムのみ: **/usr/lib/udev/rules.d/71-biosdevname.rules**
   このファイルは、**biosdevname** パッケージがインストールされている場合にのみ存在します。このルールファイルは、前の手順でインターフェイスの名前が変更されていない場合

に、**biosdevname** ユーティリティーが命名ポリシーに従ってインターフェイスの名前を変更することを定義します。

注記

biosdevname は Dell システムにのみインストールして使用してください。

3. **/usr/lib/udev/rules.d/75-net-description.rules**
   このファイルは、**udev** がネットワークインターフェイスを検査し、**udev** の内部変数にプロパティーを設定する方法を定義します。これらの変数は、次のステップで **/usr/lib/udev/rules.d/80-net-setup-link.rules** ファイルによって処理されます。一部のプロパティーは未定義である場合があります。

4. **/usr/lib/udev/rules.d/80-net-setup-link.rules**
   このファイルは **udev** サービスの **net_setup_link** ビルトインを呼び出します。**udev** は **/usr/lib/systemd/network/99-default.link** ファイルの **NamePolicy** パラメーターのポリシーの順序に基づいてインターフェイスの名前を変更します。詳細は、ネットワークインターフェイスの命名ポリシー を参照してください。

   どのポリシーも適用されない場合、**udev** はインターフェイスの名前を変更しません。

関連情報

- Why are systemd network interface names different between major RHEL versions ソリューション

1.2. ネットワークインターフェイスの命名ポリシー

デフォルトでは、**udev** デバイスマネージャーは **/usr/lib/systemd/network/99-default.link** ファイルを使用して、インターフェイスの名前を変更するときに適用するデバイス命名ポリシーを決定します。このファイルの **NamePolicy** パラメーターは、**udev が** どのポリシーをどの順序で使用するかを定義します。

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

次の表では、**NamePolicy** パラメーターで指定された最初に一致するポリシーに基づく、**udev** のさまざまなアクションを説明します。

ポリシー	説明	名前の例
keep	デバイスにユーザー空間で割り当てられた名前がすでにある場合、**udev** はこのデバイスの名前を変更しません。たとえば、名前がデバイスの作成中または名前変更操作によって割り当てられた場合がこれに該当します。	
kernel	デバイス名が予測可能であるとカーネルが通知した場合、**udev** はこのデバイスの名前を変更しません。	**lo**
database	このポリシーは、**udev** ハードウェアデータベース内のマッピングに基づいて名前を割り当てます。詳細は、man ページの **hwdb(7)** を参照してください。	**idrac**

ポリシー	説明	名前の例
onboard	デバイス名には、ファームウェアまたは BIOS が提供するオンボードデバイスのインデックス番号が含まれます。	**eno1**
slot	デバイス名には、ファームウェアまたは BIOS が提供する PCI Express (PCIe) ホットプラグのスロットインデックス番号が含まれます。	**ens1**
path	デバイス名には、ハードウェアのコネクターの物理的な場所が含まれます。	**enp1s0**
mac	デバイス名には MAC アドレスが含まれます。デフォルトでは、Red Hat Enterprise Linux はこのポリシーを使用しませんが、管理者はこのポリシーを有効にすることができます。	**enx525400d5e0fb**

関連情報

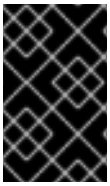
- udev デバイスマネージャーによるネットワークインターフェイスの名前変更の仕組み
- **systemd.link(5)** man ページ

1.3. ネットワークインターフェイスの命名スキーム

udev デバイスマネージャーは、一定のインターフェイス属性を使用して、一貫したデバイス名を生成します。さまざまなデバイスタイプおよびプラットフォームの命名スキームの詳細は、man ページの **systemd.net-naming-scheme(7)** を参照してください。

1.4. インストール時のイーサネットインターフェイスの接頭辞のカスタマイズ

イーサネットインターフェイスにデフォルトのデバイス命名ポリシーを使用しない場合は、Red Hat Enterprise Linux (RHEL) のインストール時にカスタムデバイス接頭辞を設定できます。



重要

Red Hat は、RHEL のインストール時に接頭辞を設定した場合にのみ、カスタマイズされたイーサネット接頭辞を持つシステムをサポートします。すでにデプロイされているシステムでの **prefixdevname** ユーティリティーの使用はサポートされていません。

インストール時にデバイス接頭辞を設定した場合、**udev** サービスはインストール後にイーサネットインターフェイスに **<prefix><index>** という形式を使用します。たとえば、接頭辞 **net** を設定すると、サービスはイーサネットインターフェイスに **net0**、**net1** などの名前を割り当てます。

udev サービスはカスタム接頭辞にインデックスを追加し、既知のイーサネットインターフェイスのインデックス値を保存します。インターフェイスを追加すると、**udev** は、以前に割り当てたインデックス値より 1 大きいインデックス値を新しいインターフェイスに割り当てます。

前提条件

- 接頭辞が ASCII 文字で構成されている。

- 接頭辞が英数字の文字列である。
- 接頭辞が 16 文字未満である。
- 接頭辞が、**eth**、**eno**、**ens**、**em** などの他の既知のネットワークインターフェイス接頭辞と競合しない。

手順

1. Red Hat Enterprise Linux インストールメディアを起動します。
2. ブートマネージャーで、次の手順を実行します。
   a. **Install Red Hat Enterprise Linux <version>** エントリーを選択します。
   b. **Tab** を押してエントリーを編集します。
   c. **net.ifnames.prefix=<prefix>** をカーネルオプションに追加します。
   d. **Enter** を押してインストールプログラムを起動します。
3. Red Hat Enterprise Linux をインストールします。

検証

- インターフェイス名を確認するには、ネットワークインターフェイスを表示します。

```
# 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
...
```

関連情報

- 標準的な RHEL 9 インストールの実行

1.5. UDEV ルールを使用したユーザー定義のネットワークインターフェイス名の設定

udev ルールを使用して、組織の要件を反映したカスタムネットワークインターフェイス名を実装できます。

手順

1. 名前を変更するネットワークインターフェイスを特定します。

```
# 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
...
```

インターフェイスの MAC アドレスを記録します。

2. インターフェイスのデバイスタイプ ID を表示します。

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

3. **/etc/udev/rules.d/70-persistent-net.rules** ファイルを作成し、名前を変更する各インターフェイスのルールを追加します。

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

重要

ブートプロセス中に一貫したデバイス名が必要な場合は、ファイル名として **70-persistent-net.rules** のみを使用してください。RAM ディスクイメージを再生成すると、**dracut** ユーティリティーはこの名前のファイルを **initrd** イメージに追加します。

たとえば、次のルールを使用して、MAC アドレス **00:00:5e:00:53:1a** のインターフェイスの名前を **provider0** に変更します。

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

4. オプション: **initrd** RAM ディスクイメージを再生成します。

```
# dracut -f
```

この手順は、RAM ディスクにネットワーク機能が必要な場合にのみ必要です。たとえば、ルートファイルシステムが iSCSI などのネットワークデバイスに保存されている場合がこれに当てはまります。

5. 名前を変更するインターフェイスを使用する NetworkManager 接続プロファイルを特定します。

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

6. 接続プロファイルの **connection.interface-name** プロパティーの設定を解除します。

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

7. 一時的に、新しいインターフェイス名と以前のインターフェイス名の両方に一致するように接続プロファイルを設定します。

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

8. システムを再起動します。

-

```
# reboot
```

9. リンクファイルで指定した MAC アドレスを持つデバイスの名前が **Provider0** に変更されていることを確認します。

```
# 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. 新しいインターフェイス名のみと一致するように接続プロファイルを設定します。

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

これで、接続プロファイルから古いインターフェイス名が削除されました。

11. 接続プロファイルを再度アクティベートします。

```
# nmcli connection up example_profile
```

関連情報

- **udev(7)** man ページ

1.6. SYSTEMD リンクファイルを使用したユーザー定義のネットワークインターフェイス名の設定

systemd リンクファイルを使用して、組織の要件を反映したカスタムネットワークインターフェイス名を実装できます。

前提条件

- 次の条件のいずれかを満たしている必要があります。NetworkManager がこのインターフェイスを管理していない。または、対応する接続プロファイルが キーファイル形式 を使用している。

手順

1. 名前を変更するネットワークインターフェイスを特定します。

```
# 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
...
```

インターフェイスの MAC アドレスを記録します。

2. **/etc/systemd/network/** ディレクトリーがない場合は作成します。

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

3. 名前を変更するインターフェイスごとに、次の内容を含む **70-*.link** ファイルを **/etc/systemd/network/** ディレクトリーに作成します。

```
[Match]
MACAddress=<MAC_address>

[Link]
Name=<new_interface_name>
```

重要

udev のルールベースのソリューションとファイル名の一貫性を保つために、接頭辞 **70-** を付けたファイル名を使用してください。

たとえば、MAC アドレス **00:00:5e:00:53:1a** のインターフェイスの名前を **provider0** に変更するには、次の内容を含む **/etc/systemd/network/70-provider0.link** ファイルを作成します。

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

[Link]
Name=provider0
```

4. オプション: **initrd** RAM ディスクイメージを再生成します。

```
# dracut -f
```

この手順は、RAM ディスクにネットワーク機能が必要な場合にのみ必要です。たとえば、ルートファイルシステムが iSCSI などのネットワークデバイスに保存されている場合がこれに当てはまります。

5. 名前を変更するインターフェイスを使用する NetworkManager 接続プロファイルを特定します。

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

6. 接続プロファイルの **connection.interface-name** プロパティーの設定を解除します。

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

7. 一時的に、新しいインターフェイス名と以前のインターフェイス名の両方に一致するように接続プロファイルを設定します。

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

8. システムを再起動します。

```
# reboot
```

9. リンクファイルで指定した MAC アドレスを持つデバイスの名前が **Provider0** に変更されていることを確認します。

```
# 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. 新しいインターフェイス名のみと一致するように接続プロファイルを設定します。

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

これで、接続プロファイルから古いインターフェイス名が削除されました。

11. 接続プロファイルを再度アクティベートします。

```
# nmcli connection up example_profile
```

関連情報

- **systemd.link(5)** man ページ

1.7. SYSTEMD リンクファイルを使用したネットワークインターフェイスへの代替名の割り当て

代替インターフェイス名の命名を使用すると、カーネルはネットワークインターフェイスに追加の名前を割り当てることができます。この代替名は、ネットワークインターフェイス名を必要とするコマンドで通常のインターフェイス名と同じように使用できます。

前提条件

- 代替名に ASCII 文字が使用されている。
- 代替名が 128 文字未満である。

手順

1. ネットワークインターフェイス名とその MAC アドレスを表示します。

```
# 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
...
```

代替名を割り当てるインターフェイスの MAC アドレスを記録します。

2. **/etc/systemd/network/** ディレクトリーがない場合は作成します。

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

3. 代替名を指定する必要があるインターフェイスごとに、次の内容を含む ***.link** ファイルを **/etc/systemd/network/** ディレクトリーに作成します。

```
[Match]
MACAddress=<MAC_address>

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

たとえば、次の内容を含む **/etc/systemd/network/70-altname.link** ファイルを作成して、MAC アドレス **00:00:5e:00:53:1a** のインターフェイスに代替名として **provider** を割り当てます。

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

[Link]
AlternativeName=provider
```

4. **initrd** RAM ディスクイメージを再生成します。

```
# dracut -f
```

5. システムを再起動します。

```
# reboot
```

検証

- 代替インターフェイス名を使用します。たとえば、代替名 **provider** を使用してデバイスの IP アドレス設定を表示します。

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

関連情報

- インターフェイス命名スキームの AlternativeNamesPolicy とは何ですか?

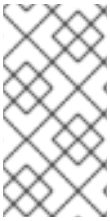
第2章 イーサネット接続の設定

NetworkManager は、ホストにインストールされている各イーサネットアダプターの接続プロファイルを作成します。デフォルトでは、このプロファイルは IPv4 接続と IPv6 接続の両方に DHCP を使用します。次の場合は、この自動作成されたプロファイルを変更するか、新しいプロファイルを追加してください。

- ネットワークに、静的 IP アドレス設定などのカスタム設定が必要な場合
- ホストが異なるネットワーク間をローミングするため、複数のプロファイルが必要な場合

Red Hat Enterprise Linux は、イーサネット接続を設定するためのさまざまなオプションを管理者に提供します。以下に例を示します。

- **nmcli** を使用して、コマンドラインで接続を設定します。
- **nmtui** を使用して、テキストベースのユーザーインターフェイスで接続を設定します。
- GNOME Settings メニューまたは **nm-connection-editor** アプリケーションを使用して、グラフィカルインターフェイスで接続を設定します。
- **nmstatectl** を使用して、Nmstate API を介して接続を設定します。
- RHEL システムロールを使用して、1つまたは複数のホストで接続の設定を自動化します。



注記

Microsoft Azure クラウドで実行しているホストでイーサネット接続を手動で設定する場合は、**cloud-init** サービスを無効にするか、クラウド環境から取得したネットワーク設定を無視するように設定します。それ以外の場合は、**cloud-init** は、手動で設定したネットワーク設定を次回の再起動時に上書きされます。

2.1. NMCLI を使用したイーサネット接続の設定

イーサネット経由でホストをネットワークに接続する場合は、**nmcli** ユーティリティーを使用してコマンドラインで接続の設定を管理できます。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。

手順

1. NetworkManager 接続プロファイルをリストします。

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

デフォルトでは、NetworkManager はホスト内の各 NIC のプロファイルを作成します。この NIC を特定のネットワークにのみ接続する予定がある場合は、自動作成されたプロファイルを調整してください。この NIC をさまざまな設定のネットワークに接続する予定がある場合は、ネットワークごとに個別のプロファイルを作成してください。

2. 追加の接続プロファイルを作成する場合は、次のように入力します。

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

既存のプロファイルを変更するには、この手順をスキップしてください。

3. オプション: 接続プロファイルの名前を変更します。

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

ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。

4. 接続プロファイルの現在の設定を表示します。

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

5. IPv4 を設定します。

- DHCP を使用するには、次のように入力します。

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

ipv4.method がすでに **auto** (デフォルト) に設定されている場合は、この手順をスキップしてください。

- 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、次のように入力します。

```
# 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. IPv6 設定を行います。

- ステートレスアドレス自動設定 (SLAAC) を使用するには、次のように入力します。

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

ipv6.method がすでに **auto** (デフォルト) に設定されている場合は、この手順をスキップしてください。

- 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、次のように入力します。

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

-

7. プロファイルの他の設定をカスタマイズするには、次のコマンドを使用します。

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

値はスペースまたはセミコロンで引用符で囲みます。

8. プロファイルをアクティブ化します。

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

検証

1. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

3. IPv6 デフォルトゲートウェイを表示します。

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

4. DNS 設定を表示します。

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

複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

5. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

トラブルシューティング

- ネットワークケーブルがホストとスイッチに差し込まれていることを確認します。

- リンク障害がこのホストだけに存在するか、同じスイッチに接続された他のホストにも存在するかを確認します。
- ネットワークケーブルとネットワークインターフェイスが予想どおりに機能していることを確認します。ハードウェア診断手順を実施して、不具合ケーブルとネットワークインターフェイスカードを置き換えます。
- ディスクの設定がデバイスの設定と一致しない場合は、NetworkManager を起動するか再起動して、インメモリー接続を作成することで、デバイスの設定を反映します。この問題を回避する方法および詳細は、NetworkManager サービスの再起動後に、NetworkManager が接続を複製する ソリューションを参照してください。

関連情報

- **nm-settings(5)** man ページ
- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- DNS サーバーの順序の設定

2.2. NMCLI インタラクティブエディターを使用したイーサネット接続の設定

イーサネット経由でホストをネットワークに接続する場合は、**nmcli** ユーティリティーを使用してコマンドラインで接続の設定を管理できます。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。

手順

1. NetworkManager 接続プロファイルをリストします。

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

   デフォルトでは、NetworkManager はホスト内の各 NIC のプロファイルを作成します。この NIC を特定のネットワークにのみ接続する予定がある場合は、自動作成されたプロファイルを調整してください。この NIC をさまざまな設定のネットワークに接続する予定がある場合は、ネットワークごとに個別のプロファイルを作成してください。

2. **nmcli** インタラクティブモードで起動します。
   - 追加の接続プロファイルを作成するには、次のように入力します。

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

   - 既存の接続プロファイルを変更するには、次のように入力します。

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

3. オプション: 接続プロファイルの名前を変更します。

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

ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。

nmcli が引用符を名前の一部としてしまうことを避けるため、スペースを含む ID を設定する場合は引用符を使用しないでください。たとえば、**Example Connection** を ID として設定するには、**set connection.id Example Connection** と入力します。

4. 接続プロファイルの現在の設定を表示します。

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

5. 新しい接続プロファイルを作成する場合は、ネットワークインターフェイスを設定します。

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

6. IPv4 を設定します。

- DHCP を使用するには、次のように入力します。

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

ipv4.method がすでに **auto** (デフォルト) に設定されている場合は、この手順をスキップしてください。

- 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、次のように入力します。

```
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. IPv6 設定を行います。

- ステートレスアドレス自動設定 (SLAAC) を使用するには、次のように入力します。

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

ipv6.method がすでに **auto** (デフォルト) に設定されている場合は、この手順をスキップしてください。

- 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、次のように入力します。

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

8. 接続をアクティベートして保存します。

```
nmcli> save persistent
```

9. インタラクティブモードを終了します。

```
nmcli> quit
```

検証

1. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

3. IPv6 デフォルトゲートウェイを表示します。

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

4. DNS 設定を表示します。

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

複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

5. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

トラブルシューティング

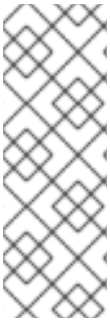
- ネットワークケーブルがホストとスイッチに差し込まれていることを確認します。
- リンク障害がこのホストだけに存在するか、同じスイッチに接続された他のホストにも存在するかを確認します。
- ネットワークケーブルとネットワークインターフェイスが予想どおりに機能していることを確認します。ハードウェア診断手順を実施して、不具合ケーブルとネットワークインターフェイスカードを置き換えます。
- ディスクの設定がデバイスの設定と一致しない場合は、NetworkManager を起動するか再起動して、インメモリー接続を作成することで、デバイスの設定を反映します。この問題を回避する方法および詳細は、NetworkManager サービスの再起動後に、NetworkManager が接続を複製する ソリューションを参照してください。

関連情報

- **nm-settings(5)** man ページ
- **nmcli(1)** man ページ
- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- DNS サーバーの順序の設定

2.3. NMTUI を使用したイーサネット接続の設定

イーサネット経由でホストをネットワークに接続する場合は、**nmtui** アプリケーションを使用して、テキストベースのユーザーインターフェイスで接続の設定を管理できます。**nmtui** では、グラフィカルインターフェイスを使用せずに、新しいプロファイルの作成や、ホスト上の既存のプロファイルの更新を行います。



注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。

手順

1. 接続に使用するネットワークデバイス名がわからない場合は、使用可能なデバイスを表示します。

```
# nmcli device status
DEVICE     TYPE      STATE                CONNECTION
```

```
enp1s0    ethernet  unavailable          --
...
```

2. **nmtui** を開始します。

```
# nmtui
```

3. **Edit a connection**選択し、**Enter** を押します。

4. 新しい接続プロファイルを追加するか、既存の接続プロファイルを変更するかを選択します。

   - 新しいプロファイルを作成するには、以下を実行します。

     i. **Add** ボタンを押します。

     ii. ネットワークタイプのリストから **Ethernet** を選択し、**Enter** を押します。

   - 既存のプロファイルを変更するには、リストからプロファイルを選択し、**Enter** を押します。

5. オプション: 接続プロファイルの名前を更新します。
   ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。

6. 新しい接続プロファイルを作成する場合は、ネットワークデバイス名を **connection** フィールドに入力します。

7. 環境に応じて、**IPv4 configuration** および **IPv6 configuration** 領域に IP アドレス設定を設定します。これを行うには、これらの領域の横にあるボタンを押して、次を選択します。

   - この接続に IP アドレスが必要ない場合は、**Disabled** にします。

   - DHCP サーバーが IP アドレスをこの NIC に動的に割り当てる場合は、**Automatic** にします。

   - ネットワークで静的 IP アドレス設定が必要な場合は、**Manual** にします。この場合、さらにフィールドに入力する必要があります。

     i. 設定するプロトコルの横にある **Show** ボタンを押して、追加のフィールドを表示します。

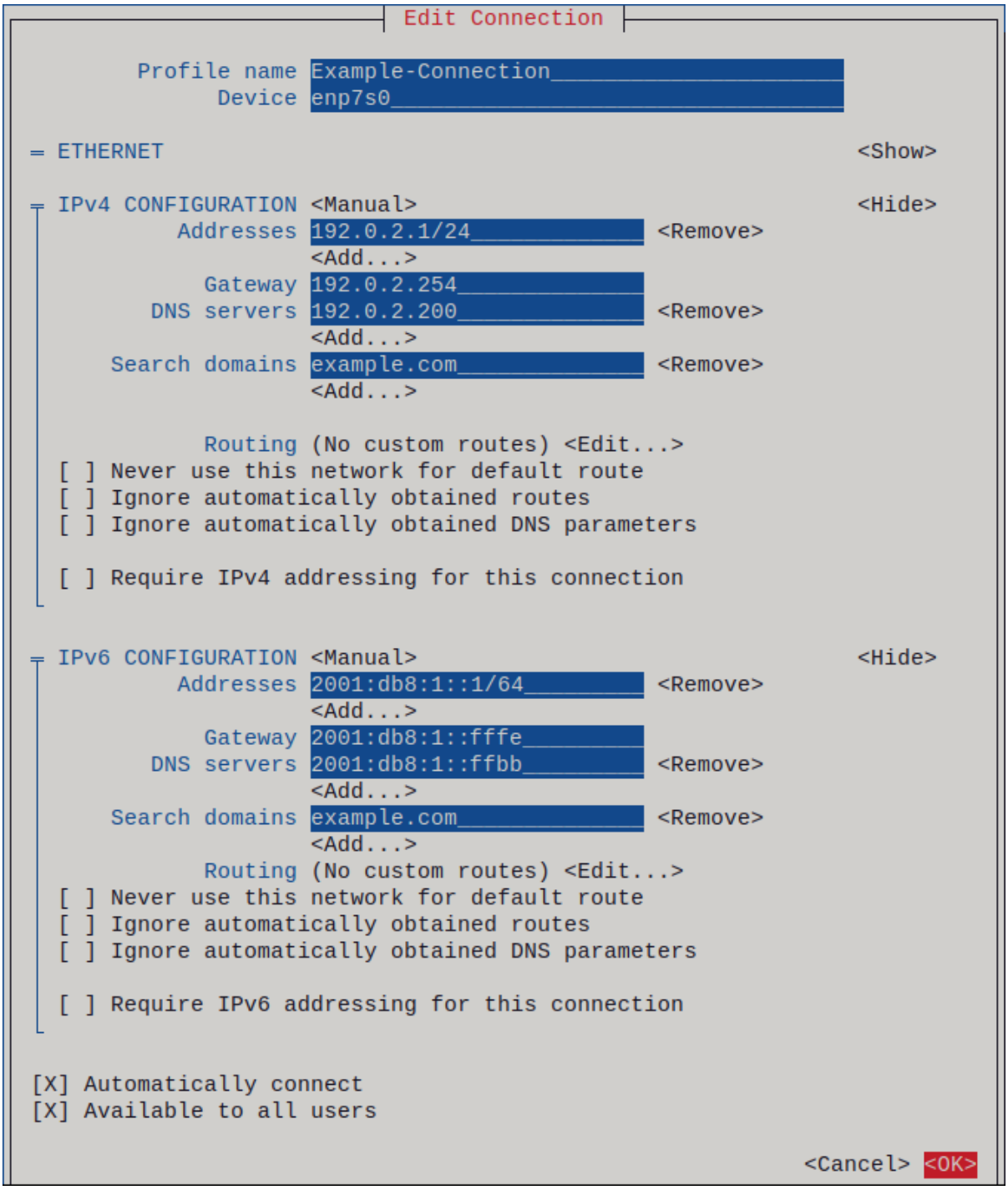
     ii. **Addresses** の横にある **Add** ボタンを押して、IP アドレスとサブネットマスクを Classless Inter-Domain Routing (CIDR) 形式で入力します。
     サブネットマスクを指定しない場合、NetworkManager は IPv4 アドレスに **/32** サブネットマスクを設定し、IPv6 アドレスに **/64** サブネットマスクを設定します。

     iii. デフォルトゲートウェイのアドレスを入力します。

     iv. **DNS servers** の横にある **Add** ボタンを押して、DNS サーバーのアドレスを入力します。

     v. **Search domains** の横にある **Add** ボタンを押して、DNS 検索ドメインを入力します。

図2.1 静的 IP アドレス設定によるイーサネット接続の例



8. **OK** ボタンを押して、新しい接続を作成し、自動的にアクティブにします。
9. **Back** ボタンを押してメインメニューに戻ります。
10. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

1. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

3. IPv6 デフォルトゲートウェイを表示します。

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

4. DNS 設定を表示します。

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

   複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

5. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

トラブルシューティング

- ネットワークケーブルがホストとスイッチに差し込まれていることを確認します。
- リンク障害がこのホストだけに存在するか、同じスイッチに接続された他のホストにも存在するかを確認します。
- ネットワークケーブルとネットワークインターフェイスが予想どおりに機能していることを確認します。ハードウェア診断手順を実施して、不具合ケーブルとネットワークインターフェイスカードを置き換えます。
- ディスクの設定がデバイスの設定と一致しない場合は、NetworkManager を起動するか再起動して、インメモリー接続を作成することで、デバイスの設定を反映します。この問題を回避する方法および詳細は、NetworkManager サービスの再起動後に、NetworkManager が接続を複製する ソリューションを参照してください。

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- DNS サーバーの順序の設定

2.4. CONTROL-CENTER によるイーサネット接続の設定

イーサネット経由でホストをネットワークに接続する場合は、GNOME 設定メニューを使用して、グラフィカルインターフェイスで接続の設定を管理できます。

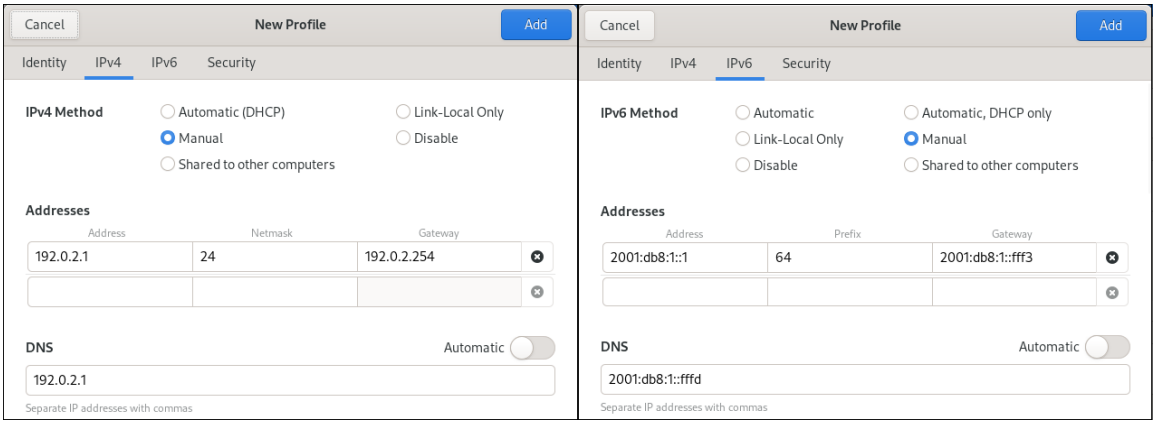
control-center は、**nm-connection-editor** アプリケーションまたは **nmcli** ユーティリティーほど多くの設定オプションに対応していないことに注意してください。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- GNOME がインストールされている。

手順

1. **Super** キーを押して **Settings** を入力し、**Enter** を押します。
2. 左側のナビゲーションにある **Network** を選択します。
3. 新しい接続プロファイルを追加するか、既存の接続プロファイルを変更するかを選択します。
   - 新しいプロファイルを作成するには、**Ethernet** エントリーの横にある **+** ボタンをクリックします。
   - 既存のプロファイルを変更するには、プロファイルエントリーの横にある歯車アイコンをクリックします。
4. オプション: **ID** タブで、接続プロファイルの名前を更新します。
   ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。
5. 環境に応じて、**IPv4** タブと **IPv6** タブで IP アドレス設定を設定します。
   - DHCP または IPv6 ステートレスアドレス自動設定 (SLAAC) を使用するには、方法として **Automatic (DHCP)** を選択します (デフォルト)。
   - 静的 IP アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、方法として **Manual** を選択し、タブのフィールドに入力します。



6. 接続プロファイルを追加するか変更するかに応じて、**Add** または **Apply** ボタンをクリックして接続を保存します。
   GNOME の **control-center** は、接続を自動的にアクティブにします。

検証

1. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

3. IPv6 デフォルトゲートウェイを表示します。

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

4. DNS 設定を表示します。

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

   複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

5. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

トラブルシューティングの手順

- ネットワークケーブルがホストとスイッチに差し込まれていることを確認します。
- リンク障害がこのホストだけに存在するか、同じスイッチに接続された他のホストにも存在するかを確認します。
- ネットワークケーブルとネットワークインターフェイスが予想どおりに機能していることを確認します。ハードウェア診断手順を実施して、不具合ケーブルとネットワークインターフェイスカードを置き換えます。
- ディスクの設定がデバイスの設定と一致しない場合は、NetworkManager を起動するか再起動して、インメモリー接続を作成することで、デバイスの設定を反映します。この問題を回避する方法および詳細は、NetworkManager サービスの再起動後に、NetworkManager が接続を複製する ソリューションを参照してください。

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- DNS サーバーの順序の設定

2.5. NM-CONNECTION-EDITOR を使用したイーサネット接続の設定

イーサネット経由でホストをネットワークに接続する場合は、**nm-connection-editor** アプリケーションを使用して、グラフィカルインターフェイスで接続の設定を管理できます。

前提条件

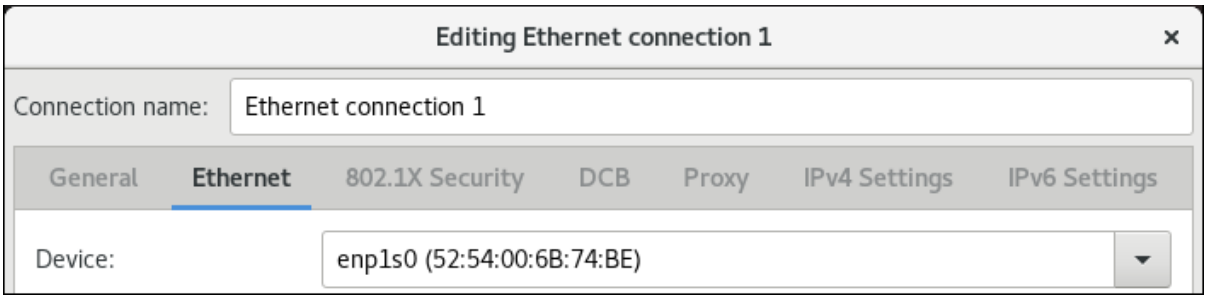
- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- GNOME がインストールされている。

手順

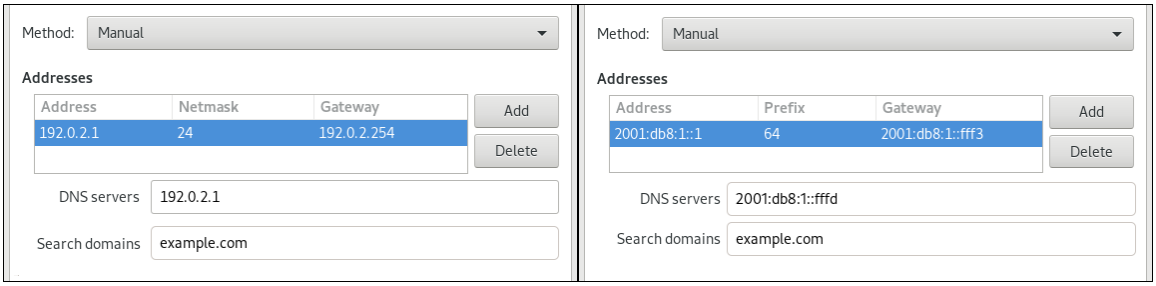
1. ターミナルを開き、次のコマンドを入力します。

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

2. 新しい接続プロファイルを追加するか、既存の接続プロファイルを変更するかを選択します。
   - 新しいプロファイルを作成するには、以下を実行します。
     i. **+** ボタンをクリックします。
     ii. 接続タイプとして **Ethernet** を選択し、**Create** をクリックします。
   - 既存のプロファイルを変更するには、プロファイルエントリーをダブルクリックします。
3. オプション: **Connection** フィールドでプロファイルの名前を更新します。
   ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。
4. 新しいプロファイルを作成する場合は、**Ethernet** タブでデバイスを選択します。



5. 環境に応じて、**IPv4 Settings** タブと **IPv6 Settings** タブで IP アドレス設定を設定します。
   - DHCP または IPv6 ステートレスアドレス自動設定 (SLAAC) を使用するには、方法として **Automatic (DHCP)** を選択します (デフォルト)。
   - 静的 IP アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および検索ドメインを設定するには、方法として **Manual** を選択し、タブのフィールドに入力します。



6. **Save** をクリックします。
7. **nm-connection-editor** を閉じます。

検証

1. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

3. IPv6 デフォルトゲートウェイを表示します。

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

4. DNS 設定を表示します。

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

   複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

5. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

トラブルシューティングの手順

- ネットワークケーブルがホストとスイッチに差し込まれていることを確認します。

- リンク障害がこのホストだけに存在するか、同じスイッチに接続された他のホストにも存在するかを確認します。
- ネットワークケーブルとネットワークインターフェイスが予想どおりに機能していることを確認します。ハードウェア診断手順を実施して、不具合ケーブルとネットワークインターフェイスカードを置き換えます。
- ディスクの設定がデバイスの設定と一致しない場合は、NetworkManager を起動するか再起動して、インメモリー接続を作成することで、デバイスの設定を反映します。この問題を回避する方法および詳細は、NetworkManager サービスの再起動後に、NetworkManager が接続を複製する ソリューションを参照してください。

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- DNS サーバーの順序の設定

2.6. NMSTATECTL を使用した静的 IP アドレスによるイーサネット接続の設定

nmstatectl ユーティリティーを使用して、Nmstate API を介してイーサネット接続を設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- **nmstate** パッケージがインストールされている。

手順

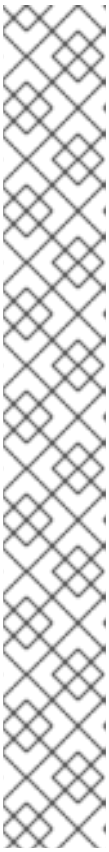
1. 以下の内容を含む YAML ファイル (例: ~/**create-ethernet-profile.yml**) を作成します。

```
---
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
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: **192.0.2.1** (サブネットマスクが **/24**)
- 静的 IPv6 アドレス: **2001:db8:1::1** (サブネットマスクが **/64**)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**



注記

identifier: mac-address および **mac-address: <ACTUAL-NIC-MACADDRESS>** プロパティーを **interfaces** プロパティーに直接定義すると、ネットワークインターフェイスカードを名前 (**enp1s0** など) ではなく MAC アドレスで識別できます。

以下に例を示します。

```
---
interfaces:
- name: <profile-name>
  type: ethernet
  identifier: mac-address
  mac-address: <ACTUAL-NIC-MACADDRESS>
  state: up
  ipv4:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    dhcp: true
    ...
```

2. 設定をシステムに適用します。

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

検証

1. 現在の状態を YAML 形式で表示します。

```
# nmstatectl show enp1s0
```

2. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

4. IPv6 デフォルトゲートウェイを表示します。

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

5. DNS 設定を表示します。

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

複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

6. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

2.7. NETWORK RHEL システムロールとインターフェイス名を使用した静的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが /**24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (/**64** サブネットマスクあり)

- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

2.8. NETWORK RHEL システムロールとデバイスパスを使用した静的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。

デバイスパスは、次のコマンドで識別できます。

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

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
```

```
- 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
```

これらの設定では、次の設定を使用してイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
  この例の **match** パラメーターは、PCI ID **0000:00:0[1-3].0** に一致するデバイスには Ansible によってプレイを適用し、**0000:00:02.0** には適用しないことを定義します。使用できる特殊な修飾子およびワイルドカードの詳細は、**/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイルーの **match** パラメーターの説明を参照してください。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

2.9. NMSTATECTL を使用した動的 IP アドレスによるイーサネット接続の設定

nmstatectl ユーティリティーを使用して、Nmstate API を介してイーサネット接続を設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- DHCP サーバーをネットワークで使用できる。
- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイル (例: **~/create-ethernet-profile.yml**) を作成します。

```
---
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
```

これらの設定では、**enp1s0** デバイスのイーサネット接続プロファイルを定義します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

注記

identifier: mac-address および **mac-address: <ACTUAL-NIC-MACADDRESS>** プロパティーを **interfaces** プロパティーに直接定義すると、ネットワークインターフェイスカードを名前 (**enp1s0** など) ではなく MAC アドレスで識別できます。

以下に例を示します。

```
---
interfaces:
- name: <profile-name>
  type: ethernet
  identifier: mac-address
  mac-address: <ACTUAL-NIC-MACADDRESS>
  state: up
  ipv4:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    dhcp: true
    ...
```

2. 設定をシステムに適用します。

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

検証

1. 現在の状態を YAML 形式で表示します。

```
# nmstatectl show enp1s0
```

2. NIC の IP 設定を表示します。

```
# 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. IPv4 デフォルトゲートウェイを表示します。

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

4. IPv6 デフォルトゲートウェイを表示します。

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

5. DNS 設定を表示します。

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

複数の接続プロファイルが同時にアクティブな場合、**nameserver** エントリーの順序は、これらのプロファイルの DNS 優先度の値と接続タイプによって異なります。

6. **ping** ユーティリティーを使用して、このホストがパケットを他のホストに送信できることを確認します。

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

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

2.10. NETWORK RHEL システムロールとインターフェイス名を使用した動的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。動的 IP アドレス設定との接続の場合、NetworkManager は、DHCP サーバーから接続の IP 設定を要求します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- DHCP サーバーをネットワークで使用できる。
- 管理対象ノードが NetworkManager を使用してネットワークを設定している。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、**enp1s0** デバイスのイーサネット接続プロファイルを定義します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

2.11. NETWORK RHEL システムロールとデバイスパスを使用した動的 IP アドレスでのイーサネット接続設定

network RHEL システムロールを使用して、イーサネット接続をリモートで設定できます。動的 IP アドレス設定との接続の場合、NetworkManager は、DHCP サーバーから接続の IP 設定を要求します。

デバイスパスは、次のコマンドで識別できます。

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

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

- サーバーに、物理または仮想のイーサネットデバイスが設定されている。
- DHCP サーバーをネットワークで使用できる。
- 管理対象ホストは、NetworkManager を使用してネットワークを設定します。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、イーサネット接続プロファイルを定義します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

match パラメーターは、PCI ID **0000:00:0[1-3].0** に一致するデバイスには Ansible によってプレイを適用し、**0000:00:02.0** には適用しないことを定義します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

2.12. インターフェイス名による単一の接続プロファイルを使用した複数のイーサネットインターフェイスの設定

ほとんどの場合、1つの接続プロファイルには1つのネットワークデバイスの設定が含まれています。ただし、接続プロファイルでインターフェイス名を設定する場合、NetworkManager はワイルドカードもサポートします。ホストが動的 IP アドレス割り当てを使用してイーサネットネットワーク間をローミングする場合、この機能を使用して、複数のイーサネットインターフェイスに使用できる単一の接続プロファイルを作成できます。

前提条件

- サーバーの設定には、物理または仮想のイーサネットデバイスが複数存在します。
- DHCP サーバーをネットワークで使用できる。
- ホストに接続プロファイルが存在しません。

手順

1. **enp** で始まるすべてのインターフェイス名に適用される接続プロファイルを追加します。

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

検証

1. 単一接続プロファイルのすべての設定を表示します。

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

   3 は、接続プロファイルで同時にアクティブなインターフェイスの数を示し、接続プロファイルのネットワークインターフェイスの数ではありません。接続プロファイルは、**match.interface-name** パラメーターのパターンに一致するすべてのデバイスを使用するため、接続プロファイルには同じ Universally Unique Identifier (UUID) があります。

2. 接続のステータスを表示します。

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

関連情報

- **nmcli(1)** man ページ
- **nm-settings(5)** man ページ

2.13. PCI ID を使用した複数のイーサネットインターフェイスの単一接続プロファイルの設定

PCI ID は、システムに接続されているデバイスの一意の識別子です。接続プロファイルは、PCI ID のリストに基づいてインターフェイスを照合することにより、複数のデバイスを追加します。この手順を使用して、複数のデバイス PCI ID を単一の接続プロファイルに接続できます。

前提条件

- サーバーの設定には、物理または仮想のイーサネットデバイスが複数存在します。
- DHCP サーバーをネットワークで使用できる。
- ホストに接続プロファイルが存在しません。

手順

1. デバイスパスを特定します。たとえば、**enp** で始まるすべてのインターフェイスのデバイスパスを表示するには、次のように入力します。

```
# 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. **0000:00:0[7-8].0** 式に一致するすべての PC IID に適用される接続プロファイルを追加します。

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

検証

1. 接続のステータスを表示します。

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

2. 接続プロファイルのすべての設定を表示するには、次のコマンドを実行します。

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

この接続プロファイルは、**match.path** パラメーターのパターンに一致する PCI ID を持つすべてのデバイスを使用するため、接続プロファイルには同じ Universally Unique Identifier (UUID) があります。

関連情報

関連情報

- **nmcli(1)** man ページ
- **nm-settings(5)** man ページ

第3章 ネットワークボンディングの設定

ネットワークボンディングは、物理ネットワークインターフェイスと仮想ネットワークインターフェイスを組み合わせるか集約して、より高いスループットまたは冗長性を備えた論理インターフェイスを提供する方法です。ボンディングでは、カーネルがすべての操作を排他的に処理します。イーサネットデバイスや VLAN など、さまざまなタイプのデバイスでネットワークボンディングを作成できます。

Red Hat Enterprise Linux は、チームデバイスを設定するためのさまざまなオプションを管理者に提供します。以下に例を示します。

- **nmcli** を使用し、コマンドラインを使用してボンディング接続を設定します。
- RHEL Web コンソールを使用し、Web ブラウザーを使用してボンディング接続を設定します。
- **nmtui** を使用して、テキストベースのユーザーインターフェイスでボンディング接続を設定します。
- **nm-connection-editor** アプリケーションを使用して、グラフィカルインターフェイスでボンディング接続を設定します。
- **nmstatectl** を使用して、Nmstate API を介してボンディング接続を設定します。
- RHEL システムロールを使用して、1つまたは複数のホストで ボンディング設定を自動化します。

3.1. コントローラーおよびポートインターフェイスのデフォルト動作の理解

NetworkManager サービスを使用してチームまたはボンディングのポートインターフェイスを管理またはトラブルシューティングする場合は、以下のデフォルトの動作を考慮してください。

- コントローラーインターフェイスを起動しても、ポートインターフェイスは自動的に起動しない。
- ポートインターフェイスを起動すると、コントローラーインターフェイスは毎回、起動する。
- コントローラーインターフェイスを停止すると、ポートインターフェイスも停止する。
- ポートのないコントローラーは、静的 IP 接続を開始できる。
- コントローラーにポートがない場合は、DHCP 接続の開始時にポートを待つ。
- DHCP 接続でポートを待機中のコントローラーは、キャリアを伴うポートの追加時に完了する。
- DHCP 接続でポートを待機中のコントローラーは、キャリアを伴わないポートを追加する時に待機を継続する。

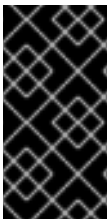
3.2. ボンディングモードに応じたアップストリームのスイッチ設定

使用するボンディングモードに応じて、スイッチでポートを設定する必要があります。

ボンディングモード	スイッチの設定

ボンディングモード	スイッチの設定
0 - balance-rr	Link Aggregation Control Protocol (LACP) がネゴシエートされたものではなく、静的 EtherChannel を有効にする必要があります。
1 - active-backup	このスイッチで必要な設定は必要ありません。
2 - balance-xor	(LACP がネゴシエートされたものではなく) 静的な Etherchannel を有効にする必要があります。
3 - broadcast	(LACP がネゴシエートされたものではなく) 静的な Etherchannel を有効にする必要があります。
4 - 802.3ad	LACP がネゴシエートされた Etherchannel が有効になっている必要があります。
5 - balance-tlb	このスイッチで必要な設定は必要ありません。
6 - balance-alb	このスイッチで必要な設定は必要ありません。

スイッチの設定方法の詳細は、スイッチのドキュメントを参照してください。



重要

特定のネットワークボンディング機能 (例: fail-over メカニズム) は、ネットワークスイッチなしでのダイレクトケーブル接続に対応していません。詳細は、ボンディングは、クロスオーバーケーブルを使用したダイレクトコレクションをサポートしますか? を参照してください。を参照してください。

3.3. NMCLI を使用したネットワークボンディングの設定

コマンドラインでネットワークボンディングを設定するには、**nmcli** ユーティリティーを使用します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ボンディングのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされている。
- ボンディングのポートにチーム、ブリッジ、または VLAN デバイスを使用するには、ボンディングの作成時にこれらのデバイスを作成するか、次の説明に従って事前にデバイスを作成することができます。
  - nmcli を使用したネットワークチームの設定
  - nmcli を使用したネットワークブリッジの設定

- nmcli を使用した VLAN タグ付けの設定

手順

1. ボンドインターフェイスを作成します。

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

このコマンドは、**active-backup** モードを使用する **bond0** という名前のボンディングを作成します。

Media Independent Interface (MII) 監視間隔も設定する場合は、**miimon=interval** オプションを **bond.options** プロパティーに追加します。以下に例を示します。

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

2. ネットワークインターフェイスを表示して、ボンドに追加する予定のインターフェイス名を書き留めます。

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

この例では、以下のように設定されています。

- **enp7s0** および **enp8s0** は設定されません。これらのデバイスをポートとして使用するには、次のステップに接続プロファイルを追加します。
- **bridge0** および **bridge1** には既存の接続プロファイルがあります。これらのデバイスをポートとして使用するには、次の手順でプロファイルを変更します。

3. インターフェイスをボンディングに割り当てます。

a. ボンディングに割り当てるインターフェイスが設定されていない場合は、インターフェイス用に新しい接続プロファイルを作成します。

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

これらのコマンドは、**enp7s0** および **enp8s0** のプロファイルを作成し、**bond0** 接続に追加します。

b. 既存の接続プロファイルをボンディングに割り当てるには、以下を実行します。

i. これらの接続の **master** パラメーターを **bond0** に設定します。

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

これらのコマンドは、**bridge0** および **bridge1** という名前の既存の接続プロファイルを **bond0** 接続に割り当てます。

ii. 接続を再度アクティブにします。

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

4. IPv4 を設定します。

- このボンドデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

- DHCP を使用するために必要な操作はありません。
- 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **bond0** 接続に設定するには、次のように入力します。

```
# 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. IPv6 設定を行います。

- このボンドデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

- ステートレスアドレス自動設定 (SLAAC) を使用する場合、アクションは必要ありません。
- 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **bond0** 接続に設定するには、次のように入力します。

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

6. オプション: ボンディングポートにパラメーターを設定する場合は、次のコマンドを使用します。

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

7. 接続をアクティベートします。

```
# nmcli connection up bond0
```

8. ポートが接続されており、**CONNECTION** コラムがポートの接続名を表示していることを確認します。

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

接続のいずれかのポートをアクティブにすると、NetworkManager はボンディングもアクティブにしますが、他のポートはアクティブにしません。ボンディングが有効な場合に、Red Hat Enterprise Linux がすべてのポートを自動的に有効にするように設定できます。

a. ボンディングの接続で **connection.autoconnect-slaves** パラメーターを有効にします。

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

b. ブリッジを再度アクティブにします。

```
# nmcli connection up bond0
```

検証

1. ホストからネットワークケーブルを一時的に削除します。
ソフトウェアユーティリティーを使用して、リンク障害イベントを適切にテストする方法がないことに注意してください。**nmcli** などの接続を非アクティブにするツールでは、ポート設定の変更を処理するボンディングドライバーの機能のみが表示され、実際のリンク障害イベントは表示されません。

2. ボンドのステータスを表示します。

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

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- ネットワークボンディングのドキュメント

3.4. RHEL WEB コンソールを使用したネットワークボンディングの設定

Web ブラウザーベースのインターフェイスを使用してネットワーク設定を管理する場合は、RHEL Web コンソールを使用してネットワークボンディングを設定します。

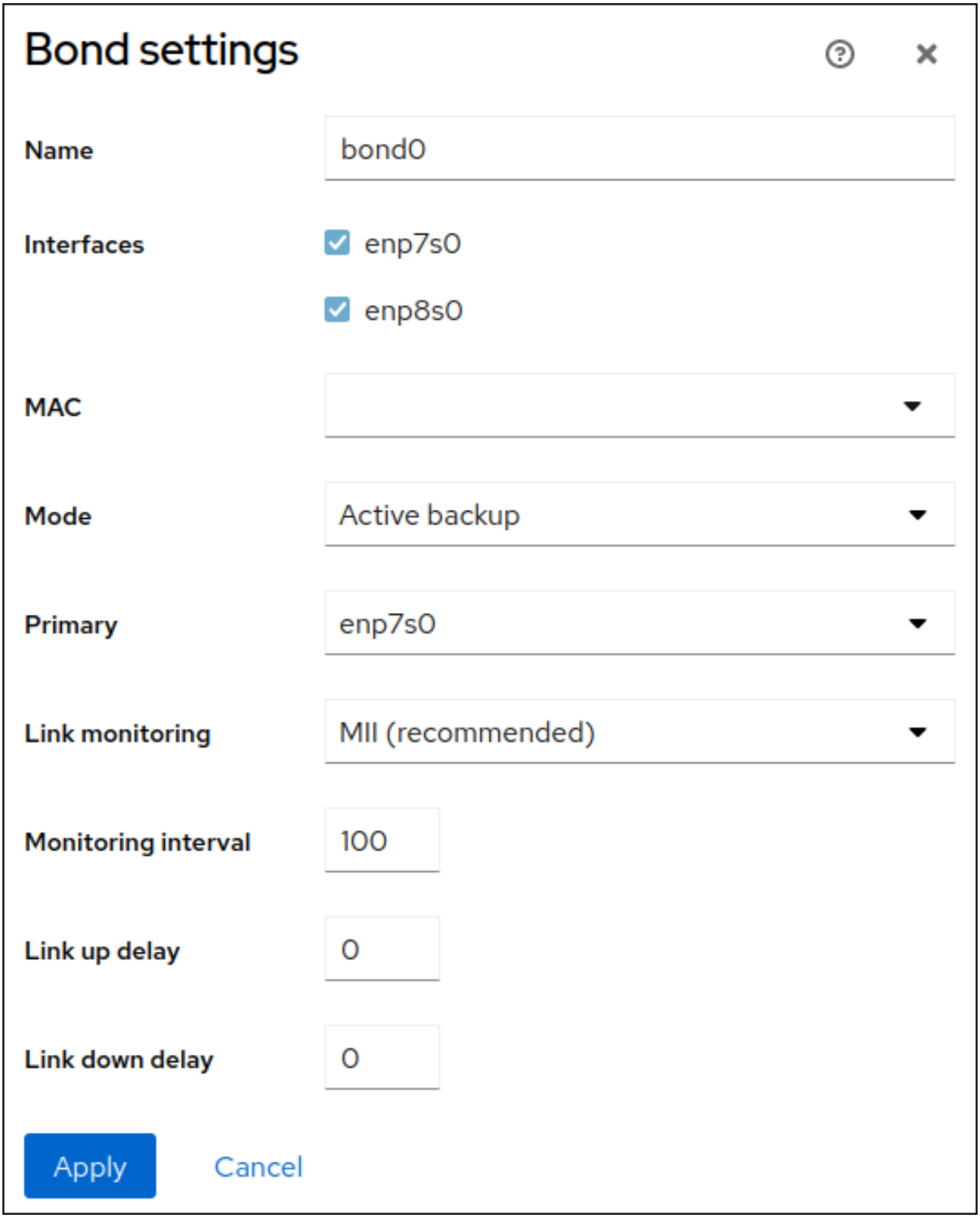
前提条件

- RHEL Web コンソールにログインしています。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ボンディングのメンバーとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされている。

- チーム、ブリッジ、または VLAN デバイスを結合のメンバーとして使用するには、次の説明に従って事前に作成します。
  - RHEL Web コンソールを使用したネットワークチームの設定
  - RHEL Web コンソールを使用したネットワークブリッジの設定
  - RHEL Web コンソールを使用した VLAN タグ付けの設定

手順

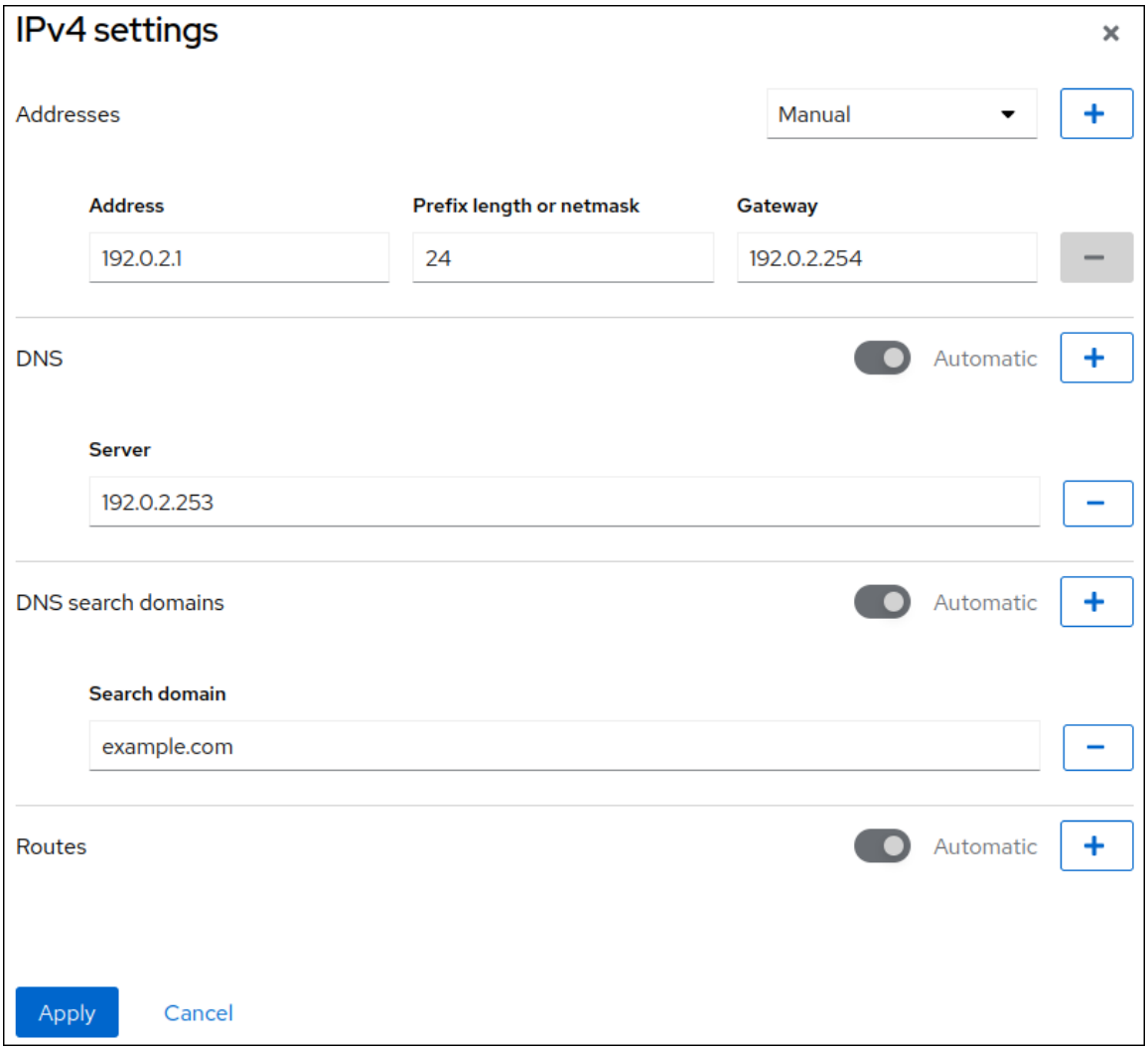
1. 画面左側のナビゲーションで **Networking** タブを選択します。
2. **Interfaces** セクションで **Add bond** をクリックします。
3. 作成するボンドデバイスの名前を入力します。
4. 結合のメンバーにするインターフェイスを選択します。
5. 結合のモードを選択します。
   Active backup を選択すると、Web コンソールに追加フィールド **Primary** が表示され、優先するアクティブデバイスを選択できます。
6. リンクモニタリング監視モードを設定します。たとえば、**Adaptive load balancing** モードを使用する場合は、**ARP** に設定します。
7. オプション: モニター間隔、リンクアップ遅延、およびリンクダウン遅延の設定を調整します。
   通常、トラブルシューティングの目的でのみデフォルトを変更します。



8. **Apply** をクリックします。
9. デフォルトでは、ボンドは動的 IP アドレスを使用します。静的 IP アドレスを設定する場合:
   a. **Interfaces** セクションでボンドの名前をクリックします。
   b. 設定するプロトコルの横にある **Edit** をクリックします。
   c. **Addresses** の横にある **Manual** を選択し、IP アドレス、接頭辞、およびデフォルトゲートウェイを入力します。
   d. **DNS** セクションで **+** ボタンをクリックし、DNS サーバーの IP アドレスを入力します。複数の DNS サーバーを設定するには、この手順を繰り返します。

e. **DNS search domains** セクションで、**+** ボタンをクリックし、検索ドメインを入力します。

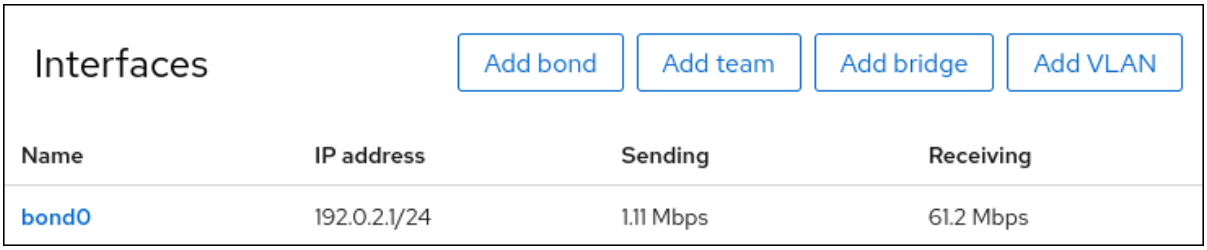
f. インターフェイスにスタティックルートが必要な場合は、**Routes** セクションで設定します。



g. **Apply** をクリックします。

検証

1. 画面左側のナビゲーションで **Networking** タブを選択し、インターフェイスに着信および発信トラフィックがあるかどうかを確認します。



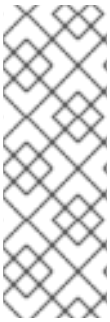
2. ホストからネットワークケーブルを一時的に削除します。
ソフトウェアユーティリティーを使用して、リンク障害イベントを適切にテストする方法がないことに注意してください。Web コンソールなどの接続を非アクティブ化するツールは、実際のリンク障害イベントではなく、メンバー設定の変更を処理するボンディングドライバーの機能のみを示します。

3. ボンドのステータスを表示します。

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

3.5. NMTUI を使用したネットワークボンディングの設定

nmtui アプリケーションは、NetworkManager 用のテキストベースのユーザーインターフェイスを提供します。**nmtui** を使用して、グラフィカルインターフェイスを使用せずにホスト上でネットワークボンドを設定できます。



注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ボンディングのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされている。

手順

1. ネットワークボンドを設定するネットワークデバイス名がわからない場合は、使用可能なデバイスを表示します。

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

2. **nmtui** を開始します。

```
# nmtui
```

3. **Edit a connection** 選択し、**Enter** を押します。
4. **Add** ボタンを押します。
5. ネットワークタイプのリストから **Bond** を選択し、**Enter** を押します。
6. オプション: 作成する NetworkManager プロファイルの名前を入力します。
   ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。
7. 作成するボンドデバイス名を **Device** フィールドに入力します。
8. 作成するボンドにポートを追加します。

a. **Slaves** リストの横にある **Add** ボタンを押します。

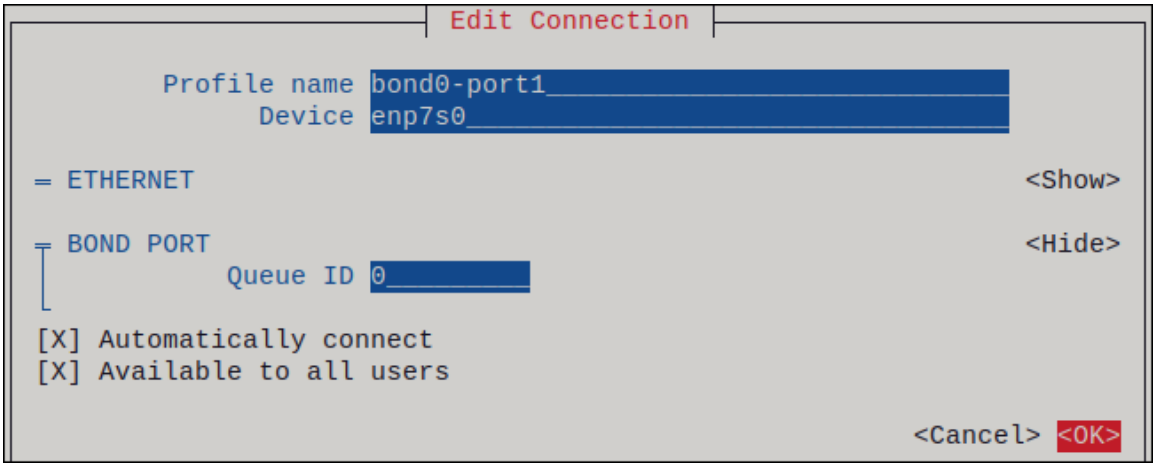
b. ボンドにポートとして追加するインターフェイスのタイプ (例: **Ethernet**) を選択します。

c. オプション: このボンドポート用に作成する NetworkManager プロファイルの名前を入力します。

d. ポートのデバイス名を **Device** フィールドに入力します。

e. **OK** ボタンを押して、ボンド設定のウィンドウに戻ります。

図3.1イーサネットデバイスをポートとしてボンドに追加する



f. ボンドにさらにポートを追加するには、これらの手順を繰り返します。

9. ボンディングモードを設定します。設定した値に応じて、**nmtui** は、選択したモードに関連する設定の追加フィールドを表示します。

10. 環境に応じて、**IPv4 configuration** および **IPv6 configuration** 領域に IP アドレス設定を設定します。これを行うには、これらの領域の横にあるボタンを押して、次を選択します。

- ボンドが IP アドレスを必要としない場合は **Disabled** にします。
- DHCP サーバーまたはステートレスアドレス自動設定 (SLAAC) が IP アドレスをボンディングに動的に割り当てる場合は、**Automatic** にします。
- ネットワークで静的 IP アドレス設定が必要な場合は、**Manual** にします。この場合、さらにフィールドに入力する必要があります。

  i. 設定するプロトコルの横にある **Show** ボタンを押して、追加のフィールドを表示します。

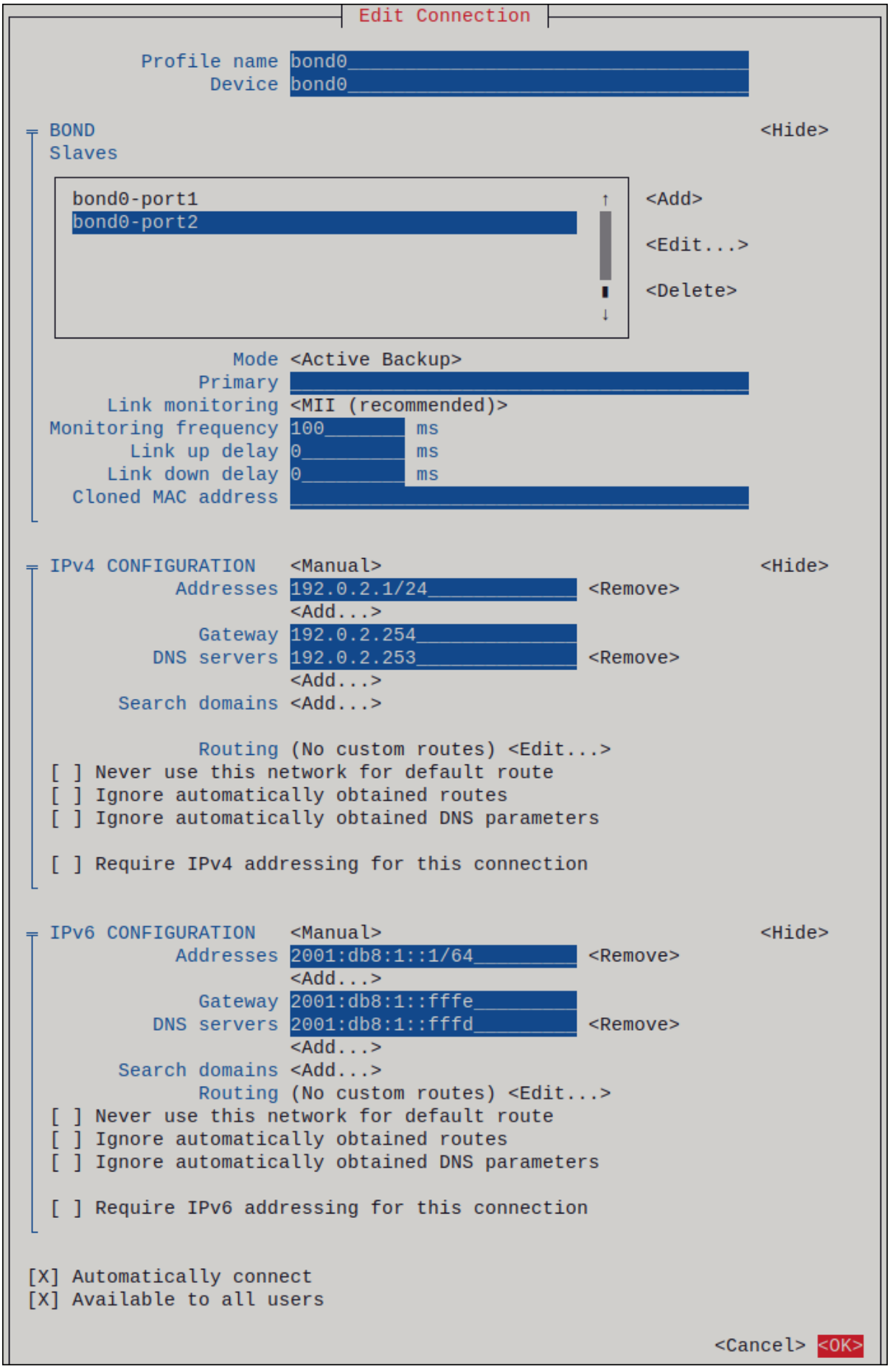
  ii. **Addresses** の横にある **Add** ボタンを押して、IP アドレスとサブネットマスクを Classless Inter-Domain Routing (CIDR) 形式で入力します。
  サブネットマスクを指定しない場合、NetworkManager は IPv4 アドレスに **/32** サブネットマスクを設定し、IPv6 アドレスに **/64** サブネットマスクを設定します。

  iii. デフォルトゲートウェイのアドレスを入力します。

  iv. **DNS servers** の横にある **Add** ボタンを押して、DNS サーバーのアドレスを入力します。

  v. **Search domains** の横にある **Add** ボタンを押して、DNS 検索ドメインを入力します。

図3.2 静的 IP アドレス設定によるボンド接続例



11. **OK** ボタンを押して、新しい接続を作成し、自動的にアクティブにします。

12. **Back** ボタンを押してメインメニューに戻ります。

13. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

1. ホストからネットワークケーブルを一時的に削除します。
ソフトウェアユーティリティーを使用して、リンク障害イベントを適切にテストする方法がないことに注意してください。**nmcli** などの接続を非アクティブにするツールでは、ポート設定の変更を処理するボンディングドライバーの機能のみが表示され、実際のリンク障害イベントは表示されません。

2. ボンドのステータスを表示します。

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

3.6. NM-CONNECTION-EDITOR を使用したネットワークボンディングの設定

グラフィカルインターフェイスで Red Hat Enterprise Linux を使用する場合は、**nm-connection-editor** アプリケーションを使用してネットワークボンディングを設定できます。

nm-connection-editor は、新しいポートだけをボンドに追加できることに注意してください。既存の接続プロファイルをポートとして使用するには、nmcli を使用したネットワークボンディングの設定 の説明に従って **nmcli** ユーティリティーを使用してボンディングを作成します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ボンディングのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされている。
- ボンディングのポートとしてチーム、ボンディング、または VLAN デバイスを使用するには、これらのデバイスがまだ設定されていないことを確認してください。

手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

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

2. **+** ボタンをクリックして、新しい接続を追加します。

3. 接続タイプ **Bond** を選択し、**作成** をクリックします。

4. **Bond** タブで、以下を行います。

   a. オプション: **Interface name** フィールドにボンドインターフェイスの名前を設定します。

   b. **追加** ボタンをクリックして、ネットワークインターフェイスをポートとしてボンドに追加します。

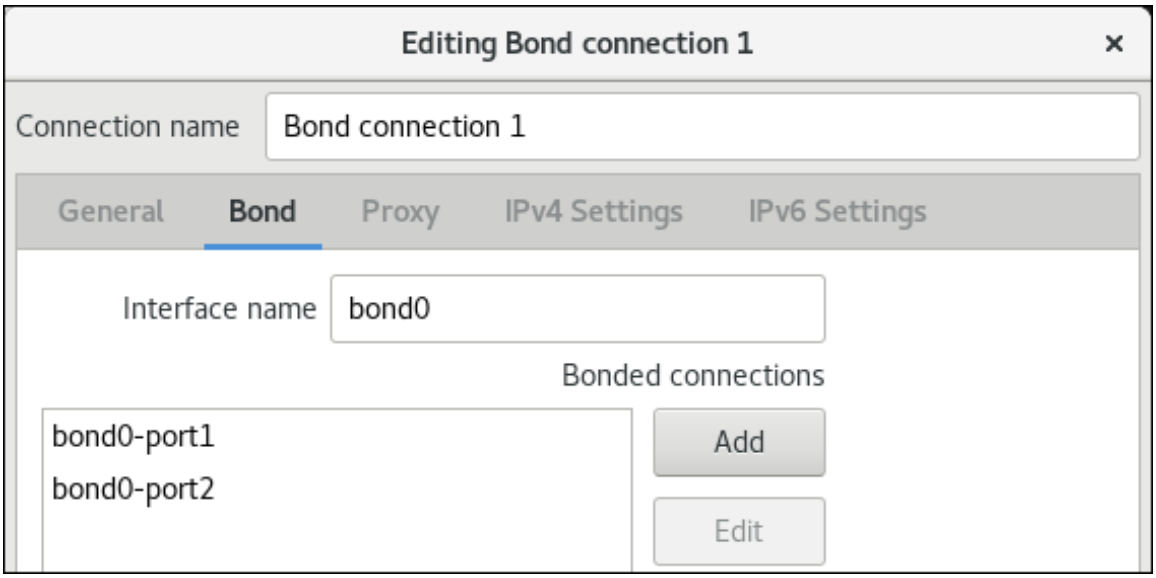
i. インターフェイスの接続タイプを選択します。たとえば、有線接続に **Ethernet** を選択します。

ii. オプション: ポートの接続名を設定します。

iii. イーサネットデバイスの接続プロファイルを作成する場合は、**Ethernet** タブを開き、**Device** フィールドでポートとしてボンディングに追加するネットワークインターフェイスを選択します。別のデバイスタイプを選択した場合は、それに応じて設定します。イーサネットインターフェイスは、設定されていないボンディングでのみ使用できることに注意してください。

iv. **Save** をクリックします。

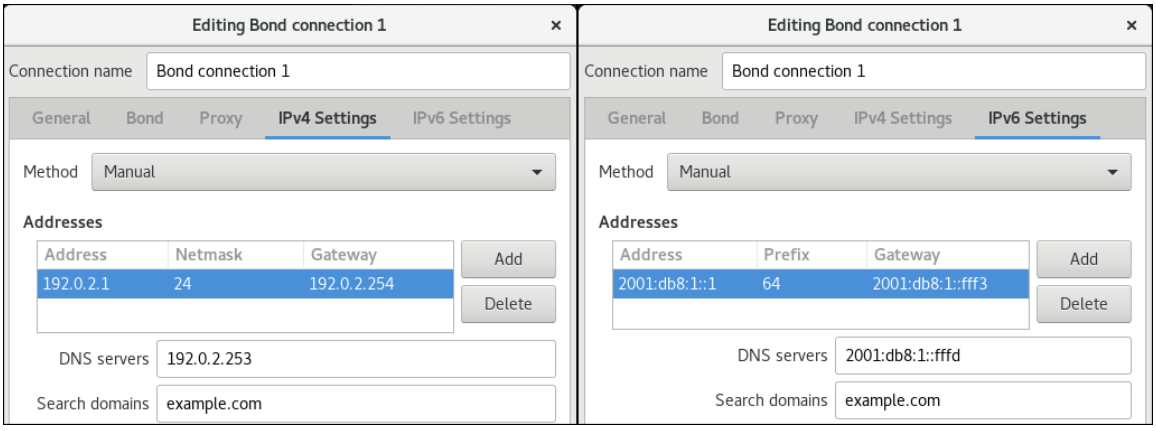
c. ボンディングに追加する各インターフェイスで直前の手順を繰り返します。



d. オプション: Media Independent Interface (MII) の監視間隔などの他のオプションを設定します。

5. **IPv4 Settings** タブと **IPv6 Settings** タブの両方で IP アドレス設定を設定します。

- このブリッジデバイスを他のデバイスのポートとして使用するには、**Method** フィールドを **Disabled** に設定します。
- DHCP を使用するには、**Method** フィールドをデフォルトの **Automatic (DHCP)** のままにします。
- 静的 IP 設定を使用するには、**Method** フィールドを **Manual** に設定し、それに応じてフィールドに値を入力します。



6. **Save** をクリックします。

7. **nm-connection-editor** を閉じます。

検証

1. ホストからネットワークケーブルを一時的に削除します。
ソフトウェアユーティリティーを使用して、リンク障害イベントを適切にテストする方法がないことに注意してください。**nmcli** などの接続を非アクティブにするツールでは、ポート設定の変更を処理するボンディングドライバーの機能のみが表示され、実際のリンク障害イベントは表示されません。

2. ボンドのステータスを表示します。

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

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- nm-connection-editor を使用したネットワークチームの設定
- nm-connection-editor を使用したネットワークブリッジの設定
- nm-connection-editor を使用した VLAN タグ付けの設定

3.7. NMSTATECTL を使用したネットワークボンディングの設定

nmstatectl ユーティリティーを使用して、Nmstate API を介してネットワークボンディングを設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

環境に応じて、YAML ファイルを適宜調整します。たとえば、ボンディングでイーサネットアダプターとは異なるデバイスを使用するには、ボンディングで使用するポートの **Base-iface** 属性と **type** 属性を調整します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- 物理または仮想のイーサネットデバイスをサーバーにインストールしてボンディングでポートとしてイーサネットデバイスを使用する。
- **ポート** リストでインターフェイス名を設定し、対応するインターフェイスを定義して、ボンディングのポートとしてチーム、ブリッジ、または VLAN デバイスを使用する。
- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイルを作成します (例: **~/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
```

これらの設定では、次の設定を使用してネットワークボンディングを定義します。

- ボンドのネットワークインターフェイス: **enp1s0** および **enp7s0**
- モード: **active-backup**
- 静的 IPv4 アドレス:**192.0.2.1** および **/24** サブネットマスク
- 静的 IPv6 アドレス:**2001: db8:1::1** および **/64** サブネットマスク

- IPv4 デフォルトゲートウェイ:**192.0.2.254**
- IPv6 デフォルトゲートウェイ:**2001:db8:1::fffe**
- IPv4 DNS サーバー:**192.0.2.200**
- IPv6 DNS サーバー:**2001:db8:1::ffbb**
- DNS 検索ドメイン: **example.com**

2. 設定をシステムに適用します。

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

検証

1. デバイスおよび接続の状態を表示します。

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

2. 接続プロファイルのすべての設定を表示します。

```
# 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. 接続設定を YAML 形式で表示します。

```
# nmstatectl show bond0
```

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

3.8. NETWORK RHEL システムロールを使用したネットワークボンディングの設定

network RHEL システムロールを使用して、ネットワークボンディングをリモートで設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、次の設定を使用してネットワークボンディングを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)

- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- ボンディングのポート - **enp7s0** および **enp8s0**
- ボンディングモード - **active-backup**

注記

Linux ボンディングのポートではなく、ボンディングに IP 設定を設定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

3.9. VPN を中断せずにイーサネットとワイヤレス接続間の切り替えを可能にするネットワークボンディングの作成

ワークステーションを会社のネットワークに接続する RHEL ユーザーは、通常、リモートリソースにアクセスするのに VPN を使用します。ただし、イーサネット接続と Wi-Fi 接続間のワークステーションスイッチ (たとえば、イーサネット接続のあるドッキングステーションからノート PC を解放した場合など) は、VPN 接続が中断されます。この問題を回避するには、**active-backup** モードでイーサネット接続および Wi-Fi 接続を使用するネットワークボンディングを作成します。

前提条件

- ホストに、イーサネットデバイスと Wi-Fi デバイスが含まれている。
- イーサネットおよび Wi-Fi NetworkManager 接続プロファイルが作成され、両方の接続が独立して機能します。
  この手順では、以下の接続プロファイルを使用して **bond0** という名前のネットワークボンディングを作成します。

- **enp11s0u1** イーサネットデバイスに関連付けられた **Docking_station**
- **wlp1s0** Wi-Fi デバイスに関連付けられた **Wi-Fi**

手順

1. **active-backup** モードでボンドインターフェイスを作成します。

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

このコマンドは、インターフェイスおよび接続プロファイル **bond0** の両方に名前を付けます。

2. ボンディングの IPv4 設定を設定します。
   - ネットワークの DHCP サーバーが IPv4 アドレスをホストに割り当てる場合は、何もする必要はありません。
   - ローカルネットワークに静的 IPv4 アドレスが必要な場合は、アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および DNS 検索ドメインを **bond0** 接続に設定します。

```
# 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. ボンディングの IPv6 設定を設定します。
   - ネットワークのルーターまたは DHCP サーバーが IPv6 アドレスをホストに割り当てる場合、アクションは必要ありません。
   - ローカルネットワークに静的 IPv6 アドレスが必要な場合は、アドレス、ネットワークマスク、デフォルトゲートウェイ、DNS サーバー、および DNS 検索ドメインを **bond0** 接続に設定します。

```
# 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. 接続プロファイルを表示します。

```
# 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
...
```

次のステップでは、接続プロファイルとイーサネットデバイス名が必要です。

5. イーサネット接続の接続プロファイルをボンドに割り当てます。

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

6. Wi-Fi 接続の接続プロファイルをボンディングに割り当てます。

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

7. Wi-Fi ネットワークが MAC フィルタリングを使用して、許可リストの MAC アドレスのみがネットワークにアクセスできるようにするには、NetworkManager がアクティブなポートの MAC アドレスをボンドに動的に割り当てるように設定します。

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

この設定では、イーサネットデバイスと Wi-Fi デバイスの両方の MAC アドレスの代わりに、Wi-Fi デバイスの MAC アドレスのみを許可リストに設定する必要があります。

8. イーサネット接続に関連付けられたデバイスを、ボンドのプライマリーデバイスとして設定します。

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

この設定では、ボンディングが利用可能な場合は、イーサネット接続を常に使用します。

9. **bond0** デバイスがアクティブになると、NetworkManager がポートを自動的にアクティブになるように設定します。

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

10. **bond0** 接続をアクティベートします。

```
# nmcli connection up bond0
```

検証

- 現在アクティブなデバイス、ボンドおよびそのポートのステータスを表示します。

```
# 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
```

関連情報

- イーサネット接続の設定
- Wi-Fi 接続の管理
- ネットワークボンディングの設定

3.10. 異なるネットワークボンディングモード

Linux ボンディングドライバーは、リンクアグリゲーションを提供します。ボンディングは、複数のネットワークインターフェイスを並行して集約して、単一の論理結合インターフェイスを提供するプロセスです。ボンディングされたインターフェイスのアクションは、モードとも呼ばれるボンディングポリシーによって異なります。さまざまなモードが、ロードバランシングサービスまたはホットスタンバイサービスのいずれかを提供します。

次のモードがあります。

Balance-rr (モード 0)

Balance-rr は、使用可能な最初のポートから最後のポートへとパケットを順次送信するラウンドロビンアルゴリズムを使用します。このモードは、ロードバランシングとフォールトトレランスを提供します。
このモードでは、EtherChannel または同様ポートのグループ化とも呼ばれるポートアグリゲーショングループのスイッチ設定が必要です。EtherChannel は、複数の物理イーサネットリンクを 1 つの論理イーサネットリンクにグループ化するポートリンクアグリゲーションテクノロジーです。

このモードの欠点は、負荷の高いワークロードや、TCP スループットと順序付けられたパケット配信が不可欠な場合には適していないことです。

Active-backup (Mode 1)

Active-backup は、結合内でアクティブなポートが 1 つだけであることを決定するポリシーを使用します。このモードはフォールトトレランスを提供し、スイッチ設定は必要ありません。
アクティブポートに障害が発生すると、代替ポートがアクティブになります。ボンディングは、Gratuitous Address Resolution Protocol (ARP) 応答をネットワークに送信します。Gratuitous ARP は、ARP フレームの受信者に転送テーブルの更新を強制します。**Active-backup** モードは、Gratuitous ARP を送信して、ホストの接続を維持するための新しいパスを通知します。

primary オプションは、ボンディングインターフェイスの優先ポートを定義します。

Balance-xor (Mode 2)

Balance-xor は、選択された送信ハッシュポリシーを使用してパケットを送信します。このモードは、ロードバランシングとフォールトトレランスを提供し、Etherchannel または同様のポートグループをセットアップするためのスイッチ設定を必要とします。
パケット送信を変更して送信のバランスを取るために、このモードでは **xmit_hash_policy** オプ

ションを使用します。インターフェイス上のトラフィックの送信元または宛先に応じて、インターフェイスには追加の負荷分散設定が必要です。xmit_hash_policy bonding parameter の説明を参照してください。

Broadcast (Mode 3)

Broadcast は、すべてのインターフェイスですべてのパケットを送信するポリシーを使用します。このモードは、フォールトトレランスを提供し、EthercChannel または同様のポートグループをセットアップするためのスイッチ設定を必要とします。
このモードの欠点は、負荷の高いワークロードや、TCP スループットと順序付けられたパケット配信が不可欠な場合には適していないことです。

802.3ad (Mode 4)

802.3ad は、同じ名前の IEEE 標準の動的リンクアグリゲーションポリシーを使用します。このモードはフォールトトレランスを提供します。このモードでは、Link Aggregation Control Protocol (LACP) ポートグループを設定するためのスイッチ設定が必要です。
このモードは、同じ速度とデュプレックス設定を共有するアグリゲーショングループを作成し、アクティブなアグリゲーターのすべてのポートを利用します。インターフェイス上のトラフィックの送信元または宛先に応じて、モードには追加の負荷分散設定が必要です。

デフォルトでは、発信トラフィックのポート選択は送信ハッシュポリシーに依存します。送信ハッシュポリシーの **xmit_hash_policy** オプションを使用して、ポートの選択を変更し、送信を分散します。

802.3ad と **Balance-xor** の違いはコンプライアンスです。**802.3ad** ポリシーは、ポートアグリゲーショングループ間で LACP をネゴシエートします。xmit_hash_policy bonding parameter の説明を参照してください。

Balance-tlb (Mode 5)

Balance-tlb は、送信負荷分散ポリシーを使用します。このモードは、フォールトトレランスと負荷分散を提供し、スイッチサポートを必要としないチャネルボンディングを確立します。
アクティブポートは着信トラフィックを受信します。アクティブポートに障害が発生した場合、別のポートが障害ポートの MAC アドレスを引き継ぎます。発信トラフィックを処理するインターフェイスを決定するには、次のいずれかのモードを使用します。

- 値 **0**:ハッシュ分散ポリシーを使用して、負荷分散なしでトラフィックを配分します
- 値 **1**:負荷分散を使用してトラフィックを各ポートに配分します
  ボンディングオプション **tlb_dynamic_lb=0** を使用すると、このボンディングモードは **xmit_hash_policy** ボンディングオプションを使用して送信を分散します。**primary** オプションは、ボンディングインターフェイスの優先ポートを定義します。

xmit_hash_policy bonding parameter の説明を参照してください。

Balance-alb (Mode 6)

Balance-alb は、適応負荷分散ポリシーを使用します。このモードは、フォールトトレランスとロードバランシングを提供し、特別なスイッチサポートを必要としません。
このモードには、IPv4 および IPv6 トラフィックのバランス - 送信ロードバランシング (**balance-tlb**) と受信ロードバランシングが含まれます。ボンディングは、ローカルシステムから送信された ARP 応答を傍受し、ボンディング内のポートの 1 つの送信元ハードウェアアドレスを上書きします。ARP ネゴシエーションは、受信負荷分散を管理します。したがって、異なるポートは、サーバーに対して異なるハードウェアアドレスを使用します。

primary オプションは、ボンディングインターフェイスの優先ポートを定義します。ボンディングオプション **tlb_dynamic_lb=0** を使用すると、このボンディングモードは **xmit_hash_policy** ボン

ディングオプションを使用して送信を分散します。xmit_hash_policy bonding parameter の説明を参照してください。

関連情報

- **/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.rst** (**kernel-doc** パッケージで提供)
- **/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.txt** (**kernel-doc** パッケージで提供)
- Which bonding modes work when used with a bridge that virtual machine guests or containers connect to?
- How are the values for different policies in "xmit_hash_policy" bonding parameter calculated?

3.11. XMIT_HASH_POLICY ボンディングパラメーター

xmit_hash_policy 負荷分散パラメーターは、**balance-xor**、**802.3ad**、**balance-alb**、および **balance-tlb** モードでのノード選択の送信ハッシュポリシーを選択します。**tlb_dynamic_lb parameter is 0** の場合、モード 5 および 6 にのみ適用されます。このパラメーターで使用できる値は、**layer2**、**layer2+3**、**layer3+4**、**encap2+3**、**encap3+4**、および **vlan+srcmac** です。

詳細については、次の表を参照してください。

ポリシー層またはネットワーク層	Layer2	Layer2+3	Layer3+4	encap2+3	encap3+4	VLAN+src mac
用途	送信元および宛先の MAC アドレスとイーサネットプロトコルタイプの XOR	送信元および宛先の MAC アドレスと IP アドレスの XOR	送信元および宛先のポートと IP アドレスの XOR	サポートされているトンネル内の送信元と宛先の MAC アドレスと IP アドレスの XOR (仮想拡張 LAN (VXLAN) など)。このモードは、**skb_flow_dissect()** 関数に依存してヘッダーフィールドを取得します。	サポートされているトンネル内の送信元ポートと宛先ポートおよび IP アドレスの XOR (VXLAN など)。このモードは、**skb_flow_dissect()** 関数に依存してヘッダーフィールドを取得します。	VLAN ID、送信元 MAC ベンダー、送信元 MAC デバイスの XOR

トラフィックの配置	基盤となる同一ネットワークインターフェイス上にある特定のネットワークピアに向かうすべてのトラフィック	基盤となる同一ネットワークインターフェイス上の特定の IP アドレスに向かうすべてのトラフィック	基盤となる同一ネットワークインターフェイス上の特定の IP アドレスとポートに向かうすべてのトラフィック			
プライマリーの選択	このシステムと、同じブロードキャストドメイン内の他の複数システムとの間でネットワークトラフィックが発生している場合	このシステムと他の複数システム間のネットワークトラフィックがデフォルトゲートウェイを通過する場合	このシステムと別のシステムの間のネットワークトラフィックが同じ IP アドレスを使用しているが、複数のポートを通過する場合	カプセル化されたトラフィックが、ソースシステムと、複数の IP アドレスを使用する他の複数システムとの間に発生している場合	カプセル化されたトラフィックが、ソースシステムと、複数のポート番号を使用する他のシステムとの間で発生している場合	ボンディングが複数のコンテナーまたは仮想マシン (VM) からのネットワークトラフィックを伝送し、それらの MAC アドレスをブリッジネットワークなどの外部ネットワークに直接公開し、モード 2 またはモード 4 のスイッチを設定できない場合
セカンダリーの選択	ネットワークトラフィックの大部分が、このシステムとデフォルトゲートウェイの背後にある複数の他のシステムとの間で発生する場合	ネットワークトラフィックの大部分がこのシステムと別のシステムとの間で発生する場合				
Compliant	802.3ad	802.3ad	802.3ad 以外			

デフォルトポリシー	設定されていない場合、これがデフォルトポリシー	非 IP トラフィックの場合、式は **layer2** 送信ポリシーと同じ	非 IP トラフィックの場合、式は **layer2** 送信ポリシーと同じ			

第4章 ネットワークチーミングの設定

ネットワークチームは、物理ネットワークインターフェイスと仮想ネットワークインターフェイスを組み合わせるか集約して、より高いスループットまたは冗長性を備えた論理インターフェイスを提供する方法です。ネットワークチーミングでは、小さなカーネルモジュールを使用してパケットフローの高速処理や、他のタスクのためのユーザー空間サービスを実装します。これにより、ネットワークチーミングは、負荷分散および冗長性の要件に対して、簡単に拡張可能でスケーラブルなソリューションとなります。

Red Hat Enterprise Linux は、チームデバイスを設定するためのさまざまなオプションを管理者に提供します。以下に例を示します。

- **nmcli** を使用し、コマンドラインを使用してチーム接続を設定します。
- RHEL Web コンソールを使用し、Web ブラウザーを使用してチーム接続を設定します。
- **nm-connection-editor** アプリケーションを使用して、グラフィカルインターフェイスでチーム接続を設定します。

重要

Red Hat Enterprise Linux 9 では、ネットワークチーミングが非推奨になりました。代わりに、ネットワークボンディングドライバーの使用を検討してください。詳細は、Configuring network bonding を参照してください。

4.1. ネットワークボンディングへのネットワークチーム設定の移行

Red Hat Enterprise Linux 9 では、ネットワークチーミングが非推奨になりました。以前のバージョンの RHEL からアップグレードした場合など、稼働中のネットワークチームを設定している場合は、設定を、NetworkManager が管理するネットワークボンディングに移行できます。

重要

team2bond ユーティリティーは、ネットワークチーム設定のみをボンディングに変換します。その後、IP アドレスや DNS 設定など、ボンディングの詳細設定を手動で行う必要があります。

前提条件

- **team-team0** NetworkManager の接続プロファイルが設定され、**team0** デバイスを管理している。
- **teamd** パッケージがインストールされている。

手順

1. オプション: **team-team0** NetworkManager 接続の IP 設定を表示します。

```
# 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
...
```

2. **team0** デバイスの設定を JSON ファイルにエクスポートします。

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

3. ネットワークチームを削除します。たとえば、NetworkManager でチームを設定した場合は、**team-team0** 接続プロファイルと、関連するポートのプロファイルを削除します。

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

4. **team2bond** ユーティリティーをドライランモードで実行して、チームデバイスと同様の設定でネットワークボンディングを設定する **nmcli** コマンドを表示します。

```
# 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,resend_igmp=1,miimon=100,miimon=100"
nmcli con add type ethernet ifname enp7s0 master bond0
nmcli con add type ethernet ifname enp8s0 master bond0
```

最初のコマンドには 2 つの **miimon** オプションが含まれます。これは、チーム設定ファイルに 2 つの **link_watch** エントリーが含まれているためです。これはボンディングの作成には影響しないことに注意してください。

サービスをチームのデバイス名にバインドし、これらのサービスの更新や破損を回避する場合は、**--rename=bond0** を省略します。この場合、**team2bond** は、チームと同じインターフェイス名をボンディングに使用します。

5. **team2bond** ユーティリティーが推奨するボンディングのオプションが正しいことを確認します。

6. ボンディングを作成します。推奨される **nmcli** コマンドを実行するか、**--exec-cmd** オプションを指定して **team2bond** コマンドを再実行できます。

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

次の手順では、ボンディング接続プロファイル (**bond-bond0**) の名前が必要です。

7. **team-team0** で以前設定した IPv4 設定を、 **bond-bond0** 接続に設定します。

```
# 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
```

8. **team-team0** で以前設定した IPv6 設定を、**bond-bond0** 接続に設定します。

```
# 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. 接続をアクティベートします。

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

検証

1. **bond-bond0** NetworkManager 接続の IP 設定を表示します。

```
# 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. ボンドのステータスを表示します。

```
# 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
```

この例では、両方のポートが起動しています。

3. ボンディングフェイルオーバーが機能することを確認するには、以下を行います。

   a. ホストからネットワークケーブルを一時的に削除します。コマンドラインでリンク障害イベントを適切にテストする方法がないことに注意してください。

   b. ボンドのステータスを表示します。

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

4.2. コントローラーおよびポートインターフェイスのデフォルト動作の理解

NetworkManager サービスを使用してチームまたはボンディングのポートインターフェイスを管理またはトラブルシューティングする場合は、以下のデフォルトの動作を考慮してください。

- コントローラーインターフェイスを起動しても、ポートインターフェイスは自動的に起動しない。
- ポートインターフェイスを起動すると、コントローラーインターフェイスは毎回、起動する。
- コントローラーインターフェイスを停止すると、ポートインターフェイスも停止する。
- ポートのないコントローラーは、静的 IP 接続を開始できる。
- コントローラーにポートがない場合は、DHCP 接続の開始時にポートを待つ。
- DHCP 接続でポートを待機中のコントローラーは、キャリアを伴うポートの追加時に完了する。
- DHCP 接続でポートを待機中のコントローラーは、キャリアを伴わないポートを追加する時に待機を継続する。

4.3. TEAMD サービス、ランナー、およびリンク監視の理解

チームサービス **teamd** は、チームドライバーのインスタンスを制御します。このドライバーのインスタンスは、ハードウェアデバイスドライバーのインスタンスを追加して、ネットワークインターフェイスのチームを形成します。チームドライバーは、ネットワークインターフェイス (**team0** など) をカーネルに提示します。

teamd サービスは、チーミングのすべてのメソッドに共通のロジックを実装します。この関数は、ラウンドロビンなどの異なる負荷分散とバックアップメソッドに一意で、**ランナー** と呼ばれる別のコードのユニットにより実装されます。管理者は、JSON (JavaScript Object Notation) 形式でランナーを指定し

ます。インスタンスの作成時に、JSON コードが **teamd** のインスタンスにコンパイルされます。または、**NetworkManager** を使用する場合は、**team.runner** パラメーターにランナーを設定でき、対応する JSON コードを **NetworkManager** が自動的に作成します。

以下のランナーが利用できます。

- **broadcast**:すべてのポートでデータを送信します。
- **roundrobin**:次に、すべてのポートでデータを送信します。
- **activebackup**:1 つのポートにデータを送信します。もう 1 つのポートはバックアップとして維持されます。
- **loadbalance**:アクティブな Tx 負荷分散と Berkeley Packet Filter (BPF) ベースの Tx ポートセレクターを持つすべてのポートでデータを送信します。
- **random**:無作為に選択されたポートでデータを送信します。
- **lacp**:802.3ad リンクアグリゲーション制御プロトコル (LACP) を実装します。

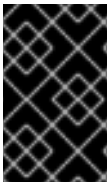
teamd サービスはリンク監視を使用して、下位デバイスの状態を監視します。さらに、以下のリンク監視が利用可能です。

- **ethtool**:**libteam** ライブラリーは、**ethtool** ユーティリティーを使用してリンク状態の変更を監視します。これはデフォルトのリンク監視です。
- **arp_ping**:**libteam** ライブラリーは、**arp_ping** ユーティリティーでアドレス解決プロトコル (ARP) を使用して、遠端のハードウェアアドレスの存在を監視します。
- **nsna_ping**:IPv6 接続では、**libteam** ライブラリーが IPv6 neighbor Discovery プロトコルの Neighbor Advertisement 機能と Neighbor Solicitation 機能を使用して、近くのインターフェイスの存在を監視します。

各ランナーは、**lacp** を除くリンク監視を使用できます。このランナーは、**ethtool** リンク監視のみを使用できます。

4.4. NMCLI を使用したネットワークチームの設定

コマンドラインでネットワークチームを設定するには、**nmcli** ユーティリティーを使用します。



重要

Red Hat Enterprise Linux 9 では、ネットワークチーミングが非推奨になりました。代わりに、ネットワークボンディングドライバーの使用を検討してください。詳細は、Configuring network bonding を参照してください。

前提条件

- **teamd** および **NetworkManager-team** パッケージがインストールされている。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- チームのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされ、スイッチに接続されている必要があります。

- チームのポートにボンディング、ブリッジ、または VLAN デバイスを使用するには、チームの作成時にこれらのデバイスを作成するか、次の説明に従って事前にデバイスを作成することができます。
  - nmcli を使用したネットワークボンディングの設定
  - nmcli を使用したネットワークブリッジの設定
  - nmcli を使用した VLAN タグ付けの設定

手順

1. チームインターフェイスを作成します。

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

このコマンドは、**activebackup** ランナーを使用する **team0** という名前のネットワークチームを作成します。

2. 必要に応じて、リンク監視を設定します。たとえば、**team0** 接続プロファイルで **ethtool** リンク監視を設定するには、次のコマンドを実行します。

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

リンク監視は、さまざまなパラメーターに対応します。リンク監視にパラメーターを設定するには、**name** プロパティーでスペースで区切って指定します。name プロパティーは引用符で囲む必要があることに注意してください。たとえば、**ethtool** リンク監視を使用し、**delay-up** パラメーターを **2500** ミリ秒 (2.5 秒) で設定するには、次のコマンドを実行します。

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

複数のリンク監視および各リンク監視を、特定のパラメーターで設定するには、リンク監視をコンマで区切る必要があります。以下の例では、**delay-up** パラメーターで **ethtool** リンク監視を設定します。**arp_ping** リンク監視は、**source-host** パラメーターおよび **target-host** パラメーターで設定します。

```
# 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. ネットワークインターフェイスを表示し、次のステップでチームに追加するインターフェイスの名前を書き留めておきます。

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

この例では、以下のように設定されています。

- **enp7s0** および **enp8s0** は設定されません。これらのデバイスをボートとして使用するには、次のステップに接続プロファイルを追加します。いずれの接続にも割り当てられていないチームのイーサネットインターフェイスのみを使用できる点に注意してください。

- **bond0** および **bond1** には既存の接続プロファイルがあります。これらのデバイスをポートとして使用するには、次の手順でプロファイルを変更します。

4. ポートインターフェイスをチームに割り当てます。

   a. チームに割り当てるインターフェイスが設定されていない場合は、それらの接続プロファイルを新たに作成します。

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

   これらのコマンドは、**enp7s0** および **enp8s0** にプロファイルを作成し、**team0** 接続に追加します。

   b. 既存の接続プロファイルをチームに割り当てるには、以下を実行します。

      i. これらの接続の **master** パラメーターを **team0** に設定します。

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

      これらのコマンドは、**bond0** および **bond1** という名前の既存の接続プロファイルを **team0** 接続に割り当てます。

      ii. 接続を再度アクティブにします。

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

5. IPv4 を設定します。

   - このチームデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

   - DHCP を使用するために必要な操作はありません。

   - 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **team0** 接続に設定するには、次のように入力します。

   ```
   # 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. IPv6 設定を行います。

   - このチームデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

- ステートレスアドレス自動設定 (SLAAC) を使用する場合、アクションは必要ありません。
- 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **team0** 接続に設定するには、次のように入力します。

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

7. 接続をアクティベートします。

```
# nmcli connection up team0
```

検証

- チームのステータスを表示します。

```
# 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
```

この例では、両方のポートが起動しています。

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- teamd サービス、ランナー、およびリンク監視の理解
- **nm-settings(5)** man ページ
- **teamd.conf(5)** man ページ

4.5. RHEL WEB コンソールを使用したネットワークチームの設定

Web ブラウザーベースのインターフェイスを使用してネットワーク設定を管理する場合は、RHEL Web コンソールを使用してネットワークチームを設定します。

重要

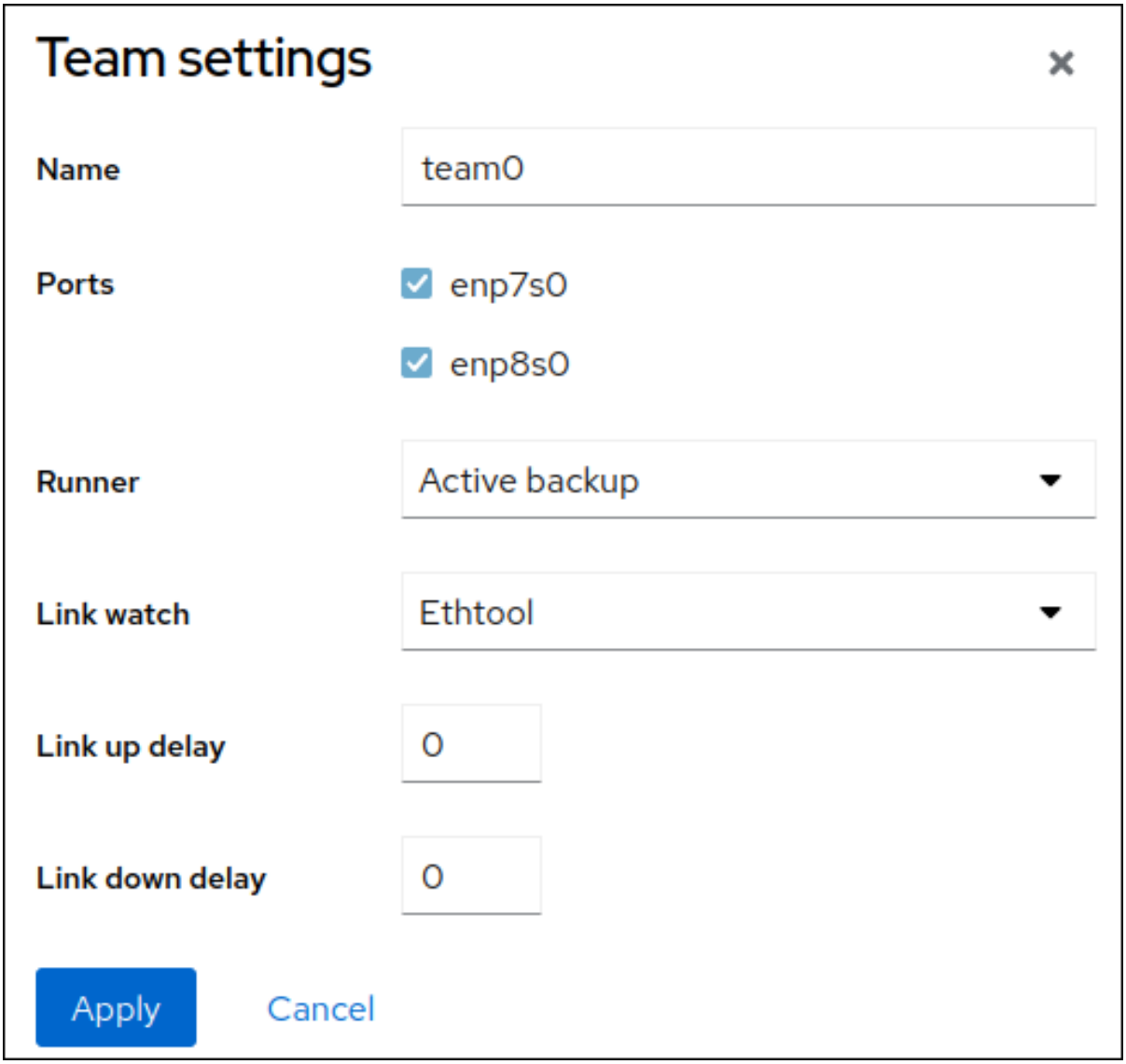
Red Hat Enterprise Linux 9 では、ネットワークチーミングが非推奨になりました。代わりに、ネットワークボンディングドライバーの使用を検討してください。詳細は、Configuring network bonding を参照してください。

前提条件

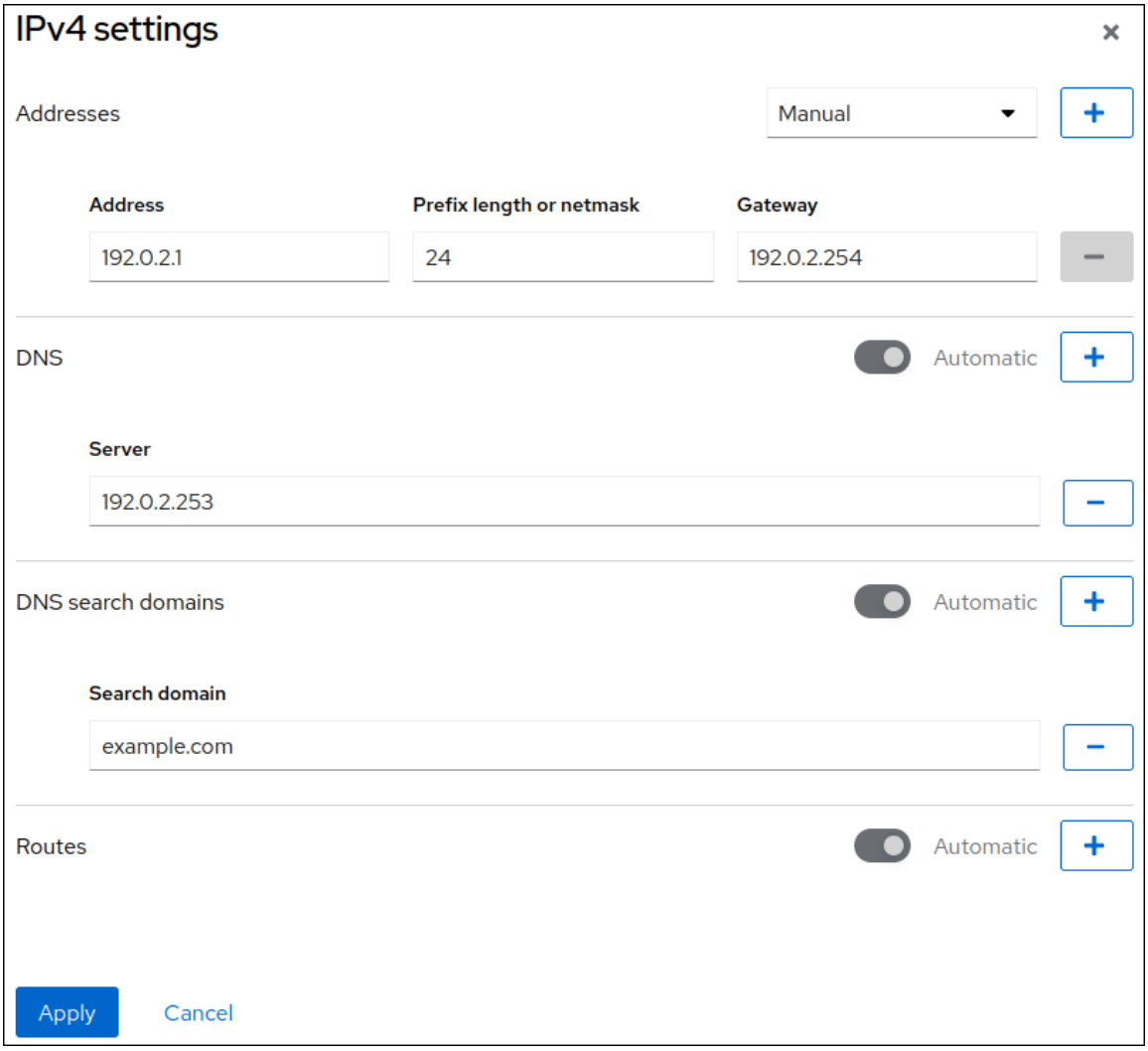
- **teamd** および **NetworkManager-team** パッケージがインストールされている。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- チームのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスがサーバーにインストールされ、スイッチに接続されている必要があります。
- ボンド、ブリッジ、または VLAN デバイスをチームのポートとして使用するには、次の説明に従って事前に作成します。
  - RHEL Web コンソールを使用したネットワークボンディングの設定
  - RHEL Web コンソールを使用したネットワークブリッジの設定
  - RHEL Web コンソールを使用した VLAN タグ付けの設定

手順

1. 画面左側のナビゲーションで **Networking** タブを選択します。
2. **Interfaces** セクションで **Add team** をクリックします。
3. 作成するチームデバイスの名前を入力します。
4. チームのポートにするインターフェイスを選択します。
5. チームのランナーを選択します。
   Load balancing または **802.3ad LACP** を選択すると、Web コンソールに追加のフィールド **Balancer** が表示されます。
6. リンクウォッチャーを設定します。
   - **Ethtool** を選択した場合は、さらに、リンクアップおよびリンクダウンの遅延を設定します。
   - **ARP ping** または **NSNA ping** を選択し、さらに ping の間隔と ping ターゲットを設定します。



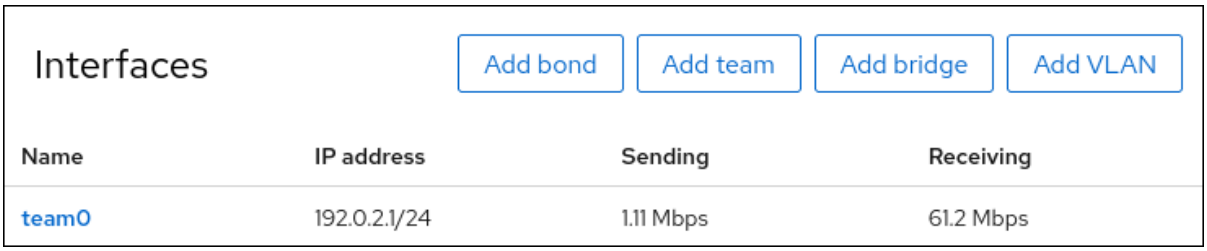
7. **Apply** をクリックします。
8. デフォルトでは、チームは動的 IP アドレスを使用します。静的 IP アドレスを設定する場合:
   a. **Interfaces** セクションでチームの名前をクリックします。
   b. 設定するプロトコルの横にある **Edit** をクリックします。
   c. **Addresses** の横にある **Manual** を選択し、IP アドレス、接頭辞、およびデフォルトゲートウェイを入力します。
   d. **DNS** セクションで **+** ボタンをクリックし、DNS サーバーの IP アドレスを入力します。複数の DNS サーバーを設定するには、この手順を繰り返します。
   e. **DNS search domains** セクションで、**+** ボタンをクリックし、検索ドメインを入力します。
   f. インターフェイスにスタティックルートが必要な場合は、**Routes** セクションで設定します。



g. **Apply** をクリックします。

検証

1. 画面左側のナビゲーションで **Networking** タブを選択し、インターフェイスに着信および発信トラフィックがあるかどうかを確認します。



2. チームのステータスを表示します。

```
# 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
```

この例では、両方のポートが起動しています。

関連情報

- ネットワークチームランナー

4.6. NM-CONNECTION-EDITOR を使用したネットワークチームの設定

グラフィカルインターフェイスで Red Hat Enterprise Linux を使用する場合は、**nm-connection-editor** アプリケーションを使用してネットワークチームを設定できます。

nm-connection-editor は、新しいポートだけをチームに追加できることに注意してください。既存の接続プロファイルをポートとして使用するには、nmcli を使用したネットワークチームの設定 の説明に従って、**nmcli** ユーティリティーを使用してチームを作成します。

重要

Red Hat Enterprise Linux 9 では、ネットワークチーミングが非推奨になりました。代わりに、ネットワークボンディングドライバーの使用を検討してください。詳細は、Configuring network bonding を参照してください。

前提条件

- **teamd** および **NetworkManager-team** パッケージがインストールされている。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- 物理または仮想のイーサネットデバイスをサーバーにインストールし、チームのポートとしてイーサネットデバイスを使用する。
- チーム、ボンディング、または VLAN デバイスをチームのポートとして使用するには、これらのデバイスがまだ設定されていないことを確認してください。

手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

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

2. **+** ボタンをクリックして、新しい接続を追加します。
3. 接続タイプ **Team** を選択し、**作成** をクリックします。

4. **Team** タブで、以下を行います。

   a. オプション: **Interface name** フィールドにチームインターフェイスの名前を設定します。

   b. **Add** ボタンをクリックして、ネットワークインターフェイスの新しい接続プロファイルを追加し、プロファイルをポートとしてチームに追加します。

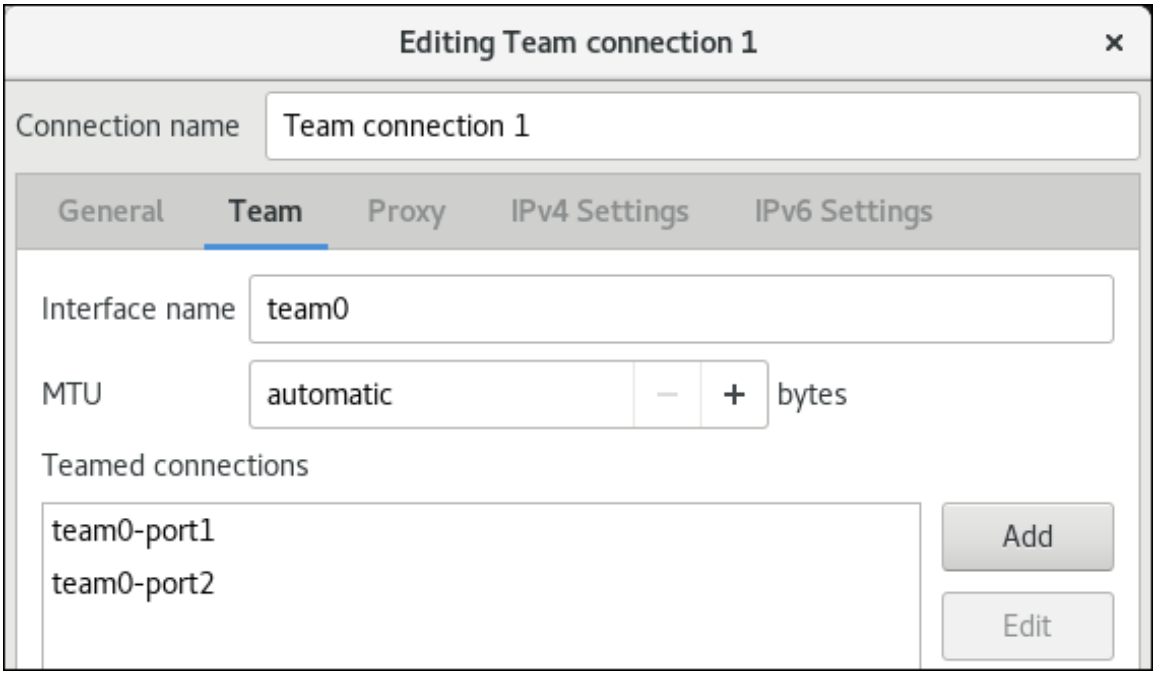
      i. インターフェイスの接続タイプを選択します。たとえば、有線接続に **Ethernet** を選択します。

      ii. オプション: ポートの接続名を設定します。

      iii. イーサネットデバイスの接続プロファイルを作成する場合は、**Ethernet** タブを開き、**Device** フィールドでポートとしてチームに追加するネットワークインターフェイスを選択します。別のデバイスタイプを選択した場合は、それに応じて設定します。いずれの接続にも割り当てられていないチームのイーサネットインターフェイスのみを使用できる点に注意してください。

      iv. **Save** をクリックします。

   c. チームに追加する各インターフェイスに直前の手順を繰り返します。



   d. **Advanced** ボタンをクリックして、チーム接続に高度なオプションを設定します。

      i. **Runner** タブで、ランナーを選択します。

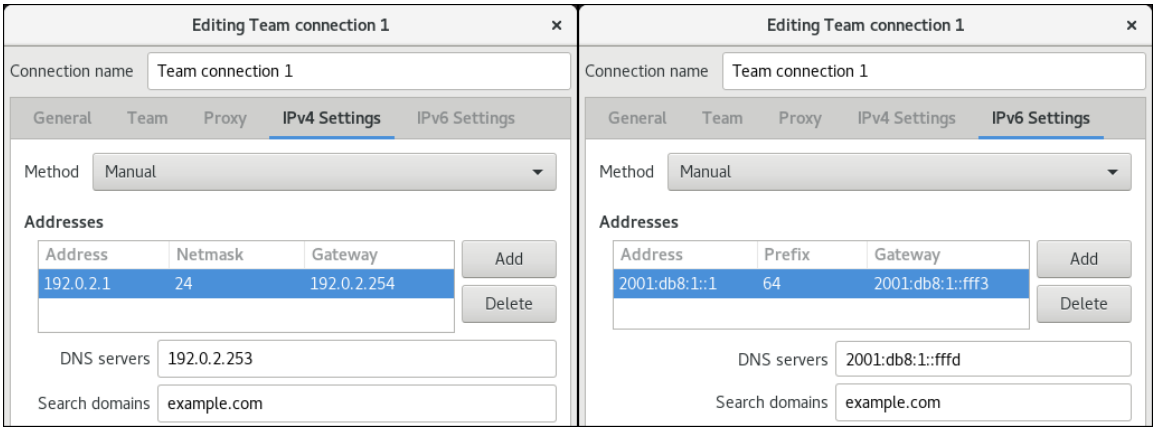
      ii. **Link Watcher** タブで、リンク監視とそのオプションを設定します。

      iii. **OK** をクリックします。

5. **IPv4 Settings** タブと **IPv6 Settings** タブの両方で IP アドレス設定を設定します。

   - このブリッジデバイスを他のデバイスのポートとして使用するには、**Method** フィールドを **Disabled** に設定します。
   - DHCP を使用するには、**Method** フィールドをデフォルトの **Automatic (DHCP)** のままにします。

- 静的 IP 設定を使用するには、**Method** フィールドを **Manual** に設定し、それに応じてフィールドに値を入力します。



6. **Save** をクリックします。
7. **nm-connection-editor** を閉じます。

検証

- チームのステータスを表示します。

```
# 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
```

関連情報

- nm-connection-editor を使用したネットワークボンディングの設定
- nm-connection-editor を使用したネットワークチームの設定
- nm-connection-editor を使用した VLAN タグ付けの設定
- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- teamd サービス、ランナー、およびリンク監視の理解

- NetworkManager duplicates a connection after restart of NetworkManager service

第5章 VLAN タグの設定

仮想ローカルエリアネットワーク (VLAN) は、物理ネットワーク内の論理ネットワークです。VLAN インターフェイスは、インターフェイスを通過する際に VLAN ID でパケットをタグ付けし、返信パケットのタグを削除します。VLAN インターフェイスを、イーサネット、ボンド、チーム、ブリッジデバイスなどの別のインターフェイスに作成します。これらのインターフェイスは **parent interface** と呼ばれます。

Red Hat Enterprise Linux は、VLAN デバイスを設定するためのさまざまなオプションを管理者に提供します。以下に例を示します。

- **nmcli** を使用し、コマンドラインを使用して VLAN のタグ付けを設定します。
- RHEL Web コンソールを使用し、Web ブラウザーを使用して VLAN のタグ付けを設定します。
- **nmtui** を使用し、テキストベースのユーザーインターフェイスで VLAN のタグ付けを設定します。
- **nm-connection-editor** アプリケーションを使用して、グラフィカルインターフェイスで接続を設定します。
- **nmstatectl** を使用して、Nmstate API を介して接続を設定します。
- RHEL システムロールを使用して、1 つまたは複数のホストで VLAN 設定を自動化します。

5.1. NMCLI を使用した VLAN タグ付けの設定

nmcli ユーティリティーを使用して、コマンドラインで仮想ローカルエリアネットワーク (VLAN) のタグ付けを設定できます。

前提条件

- 仮想 VLAN インターフェイスに対する親として使用するインターフェイスが VLAN タグに対応している。
- ボンドインターフェイスに VLAN を設定する場合は、以下のようになります。
  - ボンディングのポートが起動している。
  - ボンドが、**fail_over_mac=follow** オプションで設定されていない。VLAN 仮想デバイスは、親の新規 MAC アドレスに一致する MAC アドレスを変更できません。このような場合、トラフィックは間違ったソースの MAC アドレスで送信されます。
  - ボンドは通常、DHCP サーバーまたは IPv6 自動設定から IP アドレスを取得することは想定されていません。ボンディングの作成時に **ipv4.method=disable** オプションおよび **ipv6.method=ignore** オプションを設定してこれを確認します。そうしないと、DHCP または IPv6 の自動設定がしばらくして失敗した場合に、インターフェイスがダウンする可能性があります。
- ホストが接続するスイッチは、VLAN タグに対応するように設定されています。詳細は、スイッチのドキュメントを参照してください。

手順

1. ネットワークインターフェイスを表示します。

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

2. VLAN インターフェイスを作成します。たとえば、VLAN インターフェイス **vlan10** を作成し、**enp1s0** を親インターフェイスとして使用し、VLAN ID **10** のタグパケットを作成するには、次のコマンドを実行します。

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

   VLAN は、**0** から **4094** の範囲内に存在する必要があります。

3. デフォルトでは、VLAN 接続は、親インターフェイスから最大伝送単位 (MTU) を継承します。必要に応じて、別の MTU 値を設定します。

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

4. IPv4 を設定します。

   - この VLAN デバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

   - DHCP を使用するために必要な操作はありません。
   - 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **vlan10** 接続に設定するには、次のように入力します。

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

5. IPv6 設定を行います。

   - この VLAN デバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

   - ステートレスアドレス自動設定 (SLAAC) を使用する場合、アクションは必要ありません。
   - 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **vlan10** 接続に設定するには、次のように入力します。

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

6. 接続をアクティベートします。

```
# nmcli connection up vlan10
```

検証

- 設定を確認します。

```
# 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
```

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- **nm-settings(5)** man ページ

5.2. NMCLI を使用したネストされた VLAN の設定

802.1ad は、仮想ローカルエリアネットワーク (VLAN) のタグ付けに使用されるプロトコルです。これは Q-in-Q タグ付けとしても知られています。この技術を使用して、1 つのイーサネットフレーム内に複数の VLAN タグを作成すると、以下の利点が得られます。

- VLAN 内に複数の分離ネットワークセグメントを作成することで、ネットワークのスケーラビリティーが向上します。これにより、大規模なネットワークを、より小さく管理可能なユニットに分割して整理できます。
- さまざまな種類のネットワークトラフィックを分離および制御することで、トラフィック管理が改善されました。これにより、ネットワークパフォーマンスが向上し、ネットワークの輻輳を減らすことができます。
- より小規模で、よりターゲットを絞ったネットワークセグメントの作成を可能にすることで、リソースを効率的に利用します。
- ネットワークトラフィックを分離し、機密データへの不正アクセスのリスクを軽減することで、セキュリティーを強化します。

前提条件

- 仮想 VLAN インターフェイスに対する親として使用するインターフェイスが VLAN タグに対応している。
- ボンドインターフェイスに VLAN を設定する場合は、以下のようになります。
  - ボンディングのポートが起動している。

- ボンドが、**fail_over_mac=follow** オプションで設定されていない。VLAN 仮想デバイスは、親の新規 MAC アドレスに一致する MAC アドレスを変更できません。このような場合、トラフィックは間違ったソースの MAC アドレスで送信されます。

- ボンドは通常、DHCP サーバーまたは IPv6 自動設定から IP アドレスを取得することは想定されていません。ボンディングの作成時に **ipv4.method=disable** オプションおよび **ipv6.method=ignore** オプションを設定してこれを確認します。そうしないと、DHCP または IPv6 の自動設定がしばらくして失敗した場合に、インターフェイスがダウンする可能性があります。

- ホストが接続するスイッチは、VLAN タグに対応するように設定されています。詳細は、スイッチのドキュメントを参照してください。

手順

1. 物理ネットワークデバイスを表示します。

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

2. ベース VLAN インターフェイスを作成します。たとえば、**enp1s0** を親インターフェイスとして使用し、パケットに VLAN ID **10** のタグを付ける **vlan10** という名前のベース VLAN インターフェイスを作成するには、次のように入力します。

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

   VLAN は、**0** から **4094** の範囲内に存在する必要があります。

3. デフォルトでは、VLAN 接続は、親インターフェイスから最大伝送単位 (MTU) を継承します。必要に応じて、別の MTU 値を設定します。

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

4. ベース VLAN インターフェイスの上にネストされた VLAN インターフェイスを作成します。

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

   このコマンドは、親 VLAN 接続 **vlan10** で、名前が **vlan10.20** で VLAN ID が **20** の新しい VLAN 接続を作成します。**dev** オプションは、親ネットワークデバイスを指定します。この場合、**enp1s0.10** です。**vlan.protocol** オプションは、VLAN カプセル化プロトコルを指定します。この場合、**802.1ad** (Q-in-Q)です。

5. ネストされた VLAN インターフェイスの IPv4 設定を設定します。

   - DHCP を使用するために必要な操作はありません。

   - 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **vlan10.20** 接続に設定するには、次のように入力します。

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

6. ネストされた VLAN インターフェイスの IPv6 設定を設定します。

   - ステートレスアドレス自動設定 (SLAAC) を使用する場合、アクションは必要ありません。
   - 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを vlan10 接続に設定するには、次のように入力します。

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

7. プロファイルをアクティブ化します。

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

検証

1. ネストされた VLAN インターフェイスの設定を確認します。

   ```
   # 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
   ```

関連情報

- **nm-settings(5)** man ページ

5.3. RHEL WEB コンソールを使用した VLAN タグ付けの設定

Web ブラウザーベースのインターフェイスを使用してネットワーク設定を管理する場合は、RHEL Web コンソールを使用して VLAN タグ付けを設定します。

前提条件

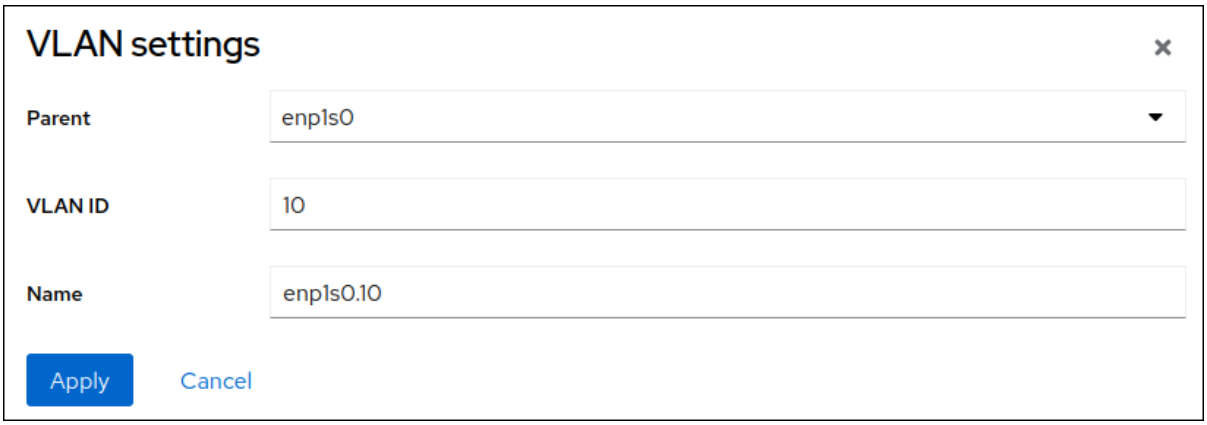
- 仮想 VLAN インターフェイスに対する親として使用するインターフェイスが VLAN タグに対応している。
- ボンドインターフェイスに VLAN を設定する場合は、以下のようになります。
  - ボンディングのポートが起動している。
  - ボンドが、**fail_over_mac=follow** オプションで設定されていない。VLAN 仮想デバイスは、親の新規 MAC アドレスに一致する MAC アドレスを変更できません。このような場合、トラフィックは間違ったソースの MAC アドレスで送信されます。

- ボンドは通常、DHCP サーバーまたは IPv6 自動設定から IP アドレスを取得することは想定されていません。結合を作成する IPv4 および IPv6 プロトコルを無効にして、これを確認します。そうしないと、DHCP または IPv6 の自動設定がしばらくして失敗した場合に、インターフェイスがダウンする可能性があります。

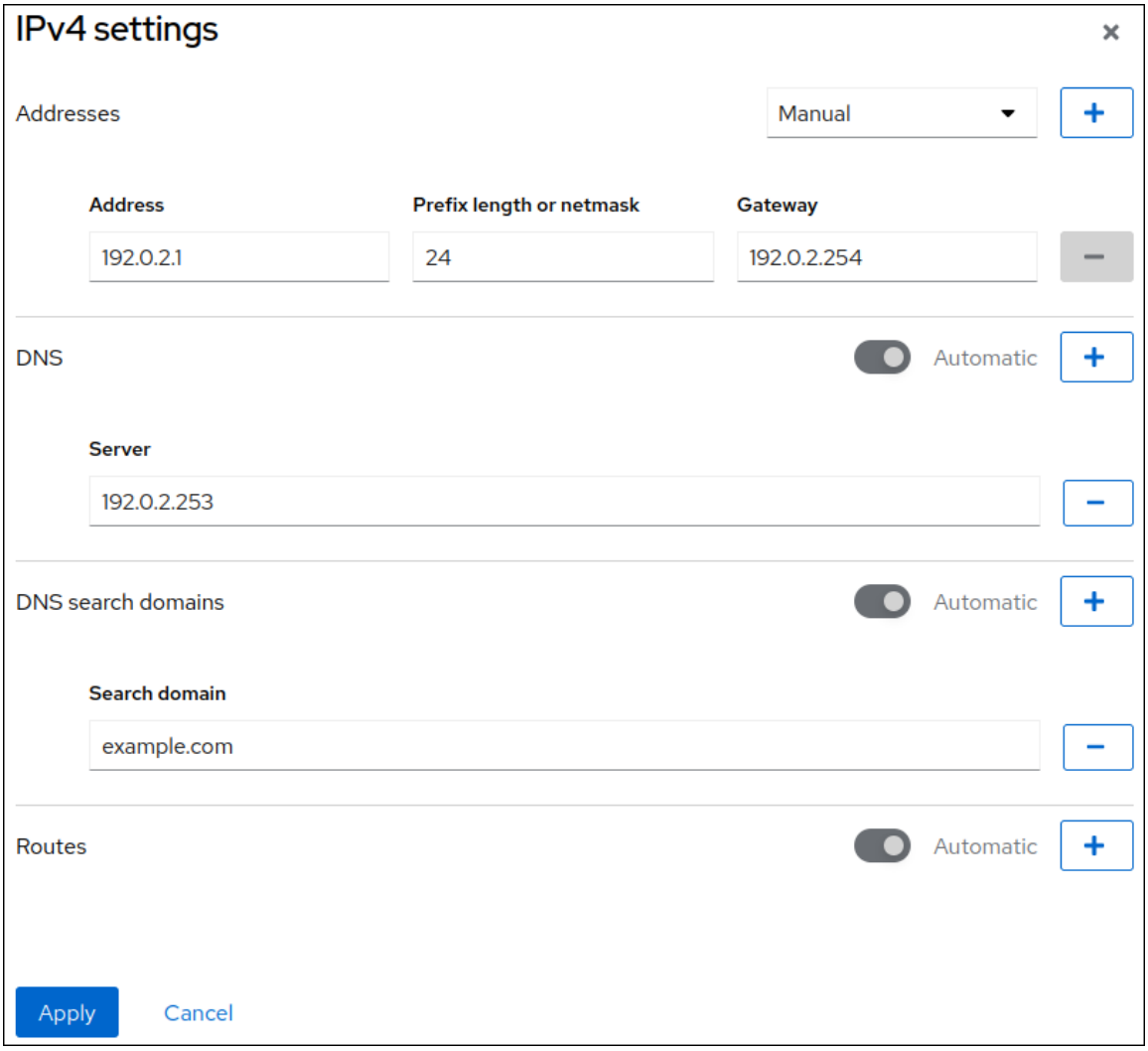
- ホストが接続するスイッチは、VLAN タグに対応するように設定されています。詳細は、スイッチのドキュメントを参照してください。

手順

1. 画面左側のナビゲーションで **Networking** タブを選択します。
2. **Interfaces** セクションで **Add VLAN** をクリックします。
3. 親デバイスを選択します。
4. VLAN ID を入力します。
5. VLAN デバイスの名前を入力するか、自動生成された名前のままにします。



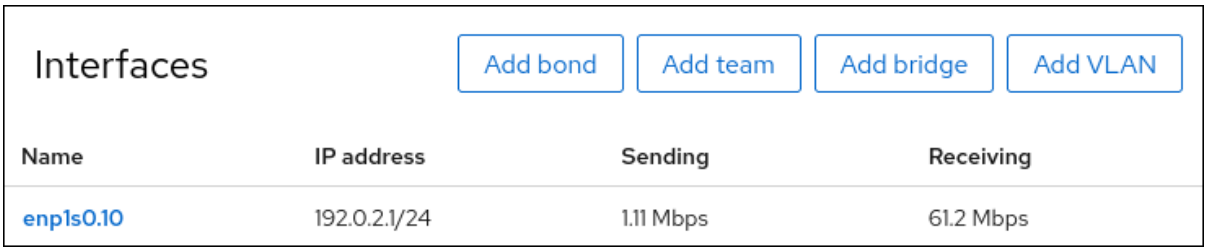
6. **Apply** をクリックします。
7. デフォルトでは、VLAN デバイスは動的 IP アドレスを使用します。静的 IP アドレスを設定する場合:
   a. **Interfaces** セクションで VLAN デバイスの名前をクリックします。
   b. 設定するプロトコルの横にある **Edit** をクリックします。
   c. **Addresses** の横にある **Manual** を選択し、IP アドレス、接頭辞、およびデフォルトゲートウェイを入力します。
   d. **DNS** セクションで **+** ボタンをクリックし、DNS サーバーの IP アドレスを入力します。複数の DNS サーバーを設定するには、この手順を繰り返します。
   e. **DNS search domains** セクションで、**+** ボタンをクリックし、検索ドメインを入力します。
   f. インターフェイスにスタティックルートが必要な場合は、**Routes** セクションで設定します。



g. **Apply** をクリックします。

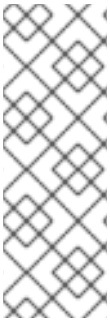
検証

- 画面左側のナビゲーションで **Networking** タブを選択し、インターフェイスに着信および発信トラフィックがあるかどうかを確認します。



5.4. NMTUI を使用した VLAN タグ付けの設定

nmtui アプリケーションは、NetworkManager 用のテキストベースのユーザーインターフェイスを提供します。**nmtui** を使用して、グラフィカルインターフェイスを使用せずにホスト上で VLAN タグ付けを設定できます。



注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

前提条件

- 仮想 VLAN インターフェイスに対する親として使用するインターフェイスが VLAN タグに対応している。
- ボンドインターフェイスに VLAN を設定する場合は、以下のようになります。
  - ボンディングのポートが起動している。
  - ボンドが、**fail_over_mac=follow** オプションで設定されていない。VLAN 仮想デバイスは、親の新規 MAC アドレスに一致する MAC アドレスを変更できません。このような場合、トラフィックは間違ったソースの MAC アドレスで送信されます。
  - ボンドは通常、DHCP サーバーまたは IPv6 自動設定から IP アドレスを取得することは想定されていません。ボンディングの作成時に **ipv4.method=disable** オプションおよび **ipv6.method=ignore** オプションを設定してこれを確認します。そうしないと、DHCP または IPv6 の自動設定がしばらくして失敗した場合に、インターフェイスがダウンする可能性があります。
- ホストが接続するスイッチは、VLAN タグに対応するように設定されています。詳細は、スイッチのドキュメントを参照してください。

手順

1. VLAN タグ付けを設定するネットワークデバイス名がわからない場合は、使用可能なデバイスを表示します。

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

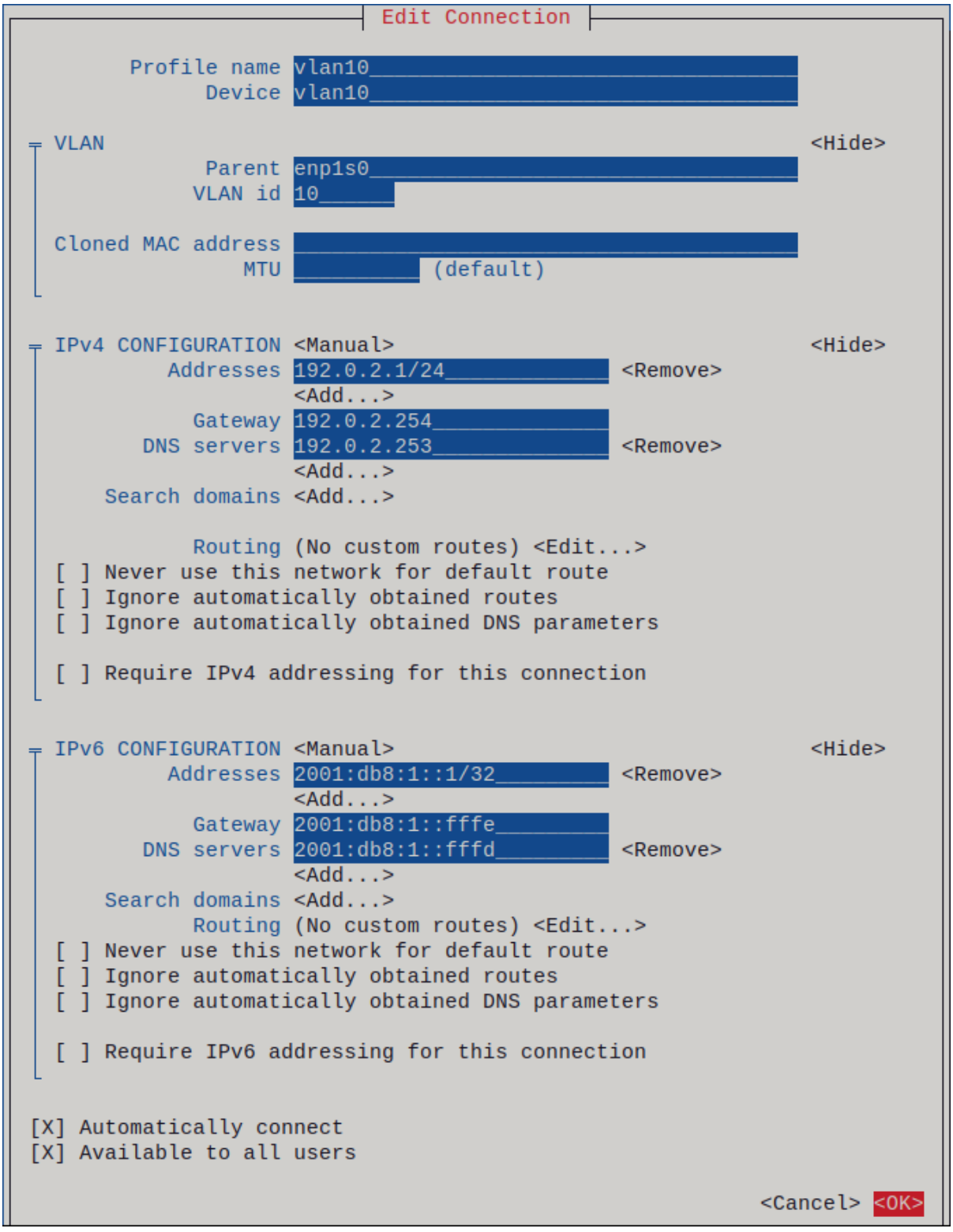
2. **nmtui** を開始します。

```
# nmtui
```

3. **Edit a connection** 選択し、**Enter** を押します。
4. **Add** ボタンを押します。
5. ネットワークタイプのリストから **VLAN** を選択し、**Enter** を押します。
6. オプション: 作成する NetworkManager プロファイルの名前を入力します。
   ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。

7. 作成する VLAN デバイス名を **Device** フィールドに入力します。
8. VLAN タグ付けを設定するデバイスの名前を **Parent** フィールドに入力します。
9. VLAN ID を入力します。ID は **0** から **4094** の範囲内である必要があります。
10. 環境に応じて、**IPv4 configuration** および **IPv6 configuration** 領域に IP アドレス設定を設定します。これを行うには、これらの領域の横にあるボタンを押して、次を選択します。
    - この VLAN デバイスが IP アドレスを必要としない場合、または他のデバイスのポートとして使用する場合は、**Disabled** にします。
    - DHCP サーバーまたはステートレスアドレス自動設定 (SLAAC) が IP アドレスを VLAN デバイスに動的に割り当てる場合は、**Automatic** にします。
    - ネットワークで静的 IP アドレス設定が必要な場合は、**Manual** にします。この場合、さらにフィールドに入力する必要があります。
        i. 設定するプロトコルの横にある **Show** ボタンを押して、追加のフィールドを表示します。
        ii. **Addresses** の横にある **Add** ボタンを押して、IP アドレスとサブネットマスクを Classless Inter-Domain Routing (CIDR) 形式で入力します。
        サブネットマスクを指定しない場合、NetworkManager は IPv4 アドレスに **/32** サブネットマスクを設定し、IPv6 アドレスに **/64** サブネットマスクを設定します。
        iii. デフォルトゲートウェイのアドレスを入力します。
        iv. **DNS servers** の横にある **Add** ボタンを押して、DNS サーバーのアドレスを入力します。
        v. **Search domains** の横にある **Add** ボタンを押して、DNS 検索ドメインを入力します。

図5.1 静的 IP アドレス設定による VLAN 接続例



11. **OK** ボタンを押して、新しい接続を作成し、自動的にアクティブにします。
12. **Back** ボタンを押してメインメニューに戻ります。
13. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

- 設定を確認します。

```
# 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. NM-CONNECTION-EDITOR を使用した VLAN タグ付けの設定

nm-connection-editor アプリケーションを使用して、グラフィカルインターフェイスで仮想ローカルエリアネットワーク (VLAN) のタグ付けを設定できます。

前提条件

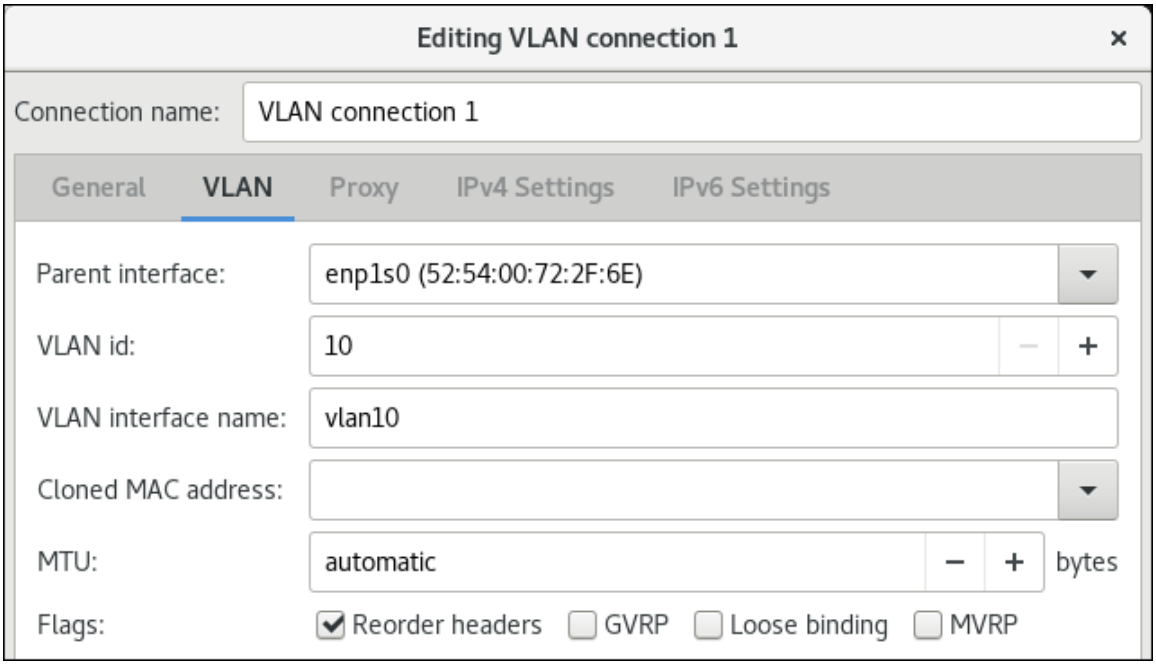
- 仮想 VLAN インターフェイスに対する親として使用するインターフェイスが VLAN タグに対応している。
- ボンドインターフェイスに VLAN を設定する場合は、以下のようになります。
  - ボンディングのポートが起動している。
  - ボンドが、**fail_over_mac=follow** オプションで設定されていない。VLAN 仮想デバイスは、親の新規 MAC アドレスに一致する MAC アドレスを変更できません。このような場合、トラフィックは間違ったソースの MAC アドレスで送信されます。
- ホストが接続するスイッチは、VLAN タグに対応するように設定されています。詳細は、スイッチのドキュメントを参照してください。

手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

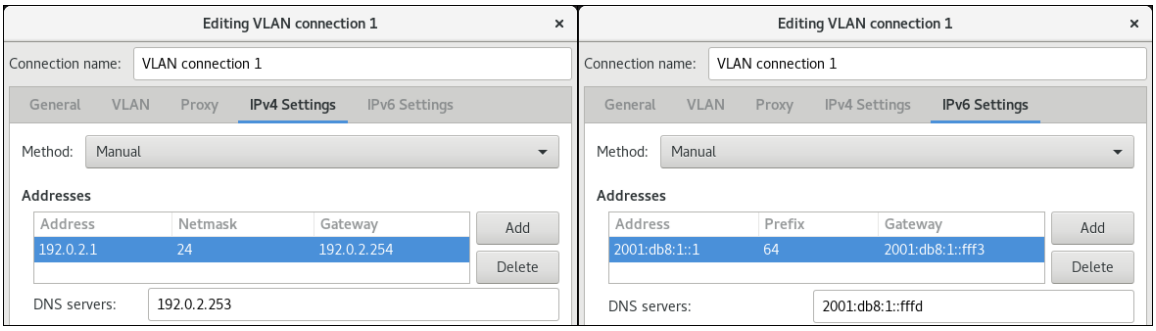
```
$ nm-connection-editor
```

2. **+** ボタンをクリックして、新しい接続を追加します。
3. **VLAN** 接続タイプを選択し、**作成** をクリックします。
4. **VLAN** タブで、以下を行います。
   a. 親インターフェイスを選択します。
   b. VLAN id を選択します。VLAN は、**0** から **4094** の範囲内に存在する必要があります。
   c. デフォルトでは、VLAN 接続は、親インターフェイスから最大伝送単位 (MTU) を継承します。必要に応じて、別の MTU 値を設定します。
   d. 必要に応じて、VLAN インターフェイスの名前および VLAN 固有のオプションを設定します。



5. **IPv4 Settings** タブと **IPv6 Settings** タブの両方で IP アドレス設定を設定します。

   - このブリッジデバイスを他のデバイスのポートとして使用するには、**Method** フィールドを **Disabled** に設定します。
   - DHCP を使用するには、**Method** フィールドをデフォルトの **Automatic (DHCP)** のままにします。
   - 静的 IP 設定を使用するには、**Method** フィールドを **Manual** に設定し、それに応じてフィールドに値を入力します。



6. **Save** をクリックします。
7. **nm-connection-editor** を閉じます。

検証

1. 設定を確認します。

```
# 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
```

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定

5.6. NMSTATECTL を使用した VLAN タグ付けの設定

nmstatectl ユーティリティーを使用して、Nmstate API を介して仮想ローカルエリアネットワーク VLAN を設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

環境に応じて、YAML ファイルを適宜調整します。たとえば、VLAN でイーサネットアダプターとは異なるデバイスを使用するには、VLAN で使用するポートの **Base-iface** 属性と **type** 属性を調整します。

前提条件

- 物理または仮想のイーサネットデバイスをサーバーにインストールし、VLAN でイーサネットデバイスをポートとして使用する。
- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイル (例: **~/create-vlan.yml**) を作成します。

```
---
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
```

これらの設定では、**enp1s0** デバイスを使用する ID 10 の VLAN を定義します。子デバイスの VLAN 接続の設定は以下のようになります。

- 静的 IPv4 アドレス: **192.0.2.1** (サブネットマスクが **/24**)
- 静的 IPv6 アドレス: **2001:db8:1::1** (サブネットマスクが **/64**)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

2. 設定をシステムに適用します。

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

検証

1. デバイスおよび接続の状態を表示します。

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

2. 接続プロファイルのすべての設定を表示します。

```
# 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. 接続設定を YAML 形式で表示します。

```
# nmstatectl show vlan0
```

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

5.7. NETWORK RHEL システムロールを使用した VLAN タグ付けの設定

network RHEL システムロールを使用して、VLAN タグ付けを設定できます。この例では、イーサネット接続と、このイーサネット接続の上に ID **10** の VLAN を追加します。子デバイスの VLAN 接続には、IP、デフォルトゲートウェイ、および DNS の設定が含まれます。

環境に応じて、プレイを適宜調整します。以下に例を示します。

- ボンディングなどの他の接続でポートとして VLAN を使用する場合は、**ip** 属性を省略し、子設定で IP 設定を行います。
- VLAN でチーム、ブリッジ、またはボンディングデバイスを使用するには、**interface_name** と VLAN で使用するポートの **type** 属性を調整します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、**enp1s0** デバイス上で動作する VLAN を定義します。VLAN インターフェイスの設定は以下のようになります。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- VLAN ID - **10**
  VLAN プロファイルの **parent** 属性は、**enp1s0** デバイス上で動作する VLAN を設定します。子デバイスの VLAN 接続には、IP、デフォルトゲートウェイ、および DNS の設定が含まれます。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル

- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

5.8. 関連情報

- VLANs for sysadmins:The basics

第6章 ネットワークブリッジの設定

ネットワークブリッジは、MAC アドレスのテーブルに基づいてネットワーク間のトラフィックを転送するリンク層デバイスです。ブリッジは、ネットワークトラフィックをリッスンし、どのホストが各ネットワークに接続しているかを把握して、MAC アドレステーブルを構築します。たとえば、Red Hat Enterprise Linux ホストのソフトウェアブリッジを使用して、ハードウェアブリッジまたは仮想環境をエミュレートし、仮想マシンをホストと同じネットワークに統合できます。

ブリッジには、ブリッジが接続する必要がある各ネットワークにネットワークデバイスが必要です。ブリッジを設定する場合には、ブリッジは **コントローラー** と呼ばれ、**ポート** を使用するデバイスです。

以下のように、さまざまなタイプのデバイスにブリッジを作成できます。

- 物理および仮想イーサネットデバイス
- ネットワークボンド
- ネットワークチーム
- VLAN デバイス

Wi-Fi で効率的に使用するために、Wi-Fi で 3-address フレームの使用を指定する IEEE 802.11 規格により、Ad-Hoc モードまたは Infrastructure モードで稼働している Wi-Fi ネットワークにはブリッジを設定できません。

6.1. NMCLI を使用したネットワークブリッジの設定

コマンドラインでネットワークブリッジを設定するには、**nmcli** ユーティリティーを使用します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ブリッジのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスをサーバーにインストールする必要があります。
- ブリッジのポートにチーム、ボンディング、または VLAN デバイスを使用するには、ブリッジの作成時にこれらのデバイスを作成するか、次の説明に従って事前にデバイスを作成することができます。
  - nmcli を使用したネットワークチームの設定
  - nmcli を使用したネットワークボンディングの設定
  - nmcli を使用した VLAN タグ付けの設定

手順

1. ブリッジインターフェイスを作成します。

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

このコマンドにより **bridge0** という名前のブリッジが作成されます。以下を入力します。

2. ネットワークインターフェイスを表示し、ブリッジに追加するインターフェイスの名前を書き留めます。

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

この例では、以下のように設定されています。

- **enp7s0** および **enp8s0** は設定されません。これらのデバイスをポートとして使用するには、次のステップに接続プロファイルを追加します。
- **bond0** および **bond1** には既存の接続プロファイルがあります。これらのデバイスをポートとして使用するには、次の手順でプロファイルを変更します。

3. インターフェイスをブリッジに割り当てます。

a. ブリッジに割り当てるインターフェイスが設定されていない場合は、それらのブリッジに新しい接続プロファイルを作成します。

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

これらのコマンドにより、**enp7s0** および **enp8s0** のプロファイルが作成され、それらを **bridge0** 接続に追加します。

b. 既存の接続プロファイルをブリッジに割り当てるには、以下を実行します。

i. これらの接続の **master** パラメーターを **bridge0** に設定します。

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

これらのコマンドは、**bond0** および **bond1** という名前の既存の接続プロファイルを **bridge0** 接続に割り当てます。

ii. 接続を再度アクティブにします。

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

4. IPv4 を設定します。

- このブリッジデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

- DHCP を使用するために必要な操作はありません。

- 静的 IPv4 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **bridge0** 接続に設定するには、次のように入力します。

```
# 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. IPv6 設定を行います。

- このブリッジデバイスを他のデバイスのポートとして使用するには、次のように入力します。

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

- ステートレスアドレス自動設定 (SLAAC) を使用する場合、アクションは必要ありません。
- 静的 IPv6 アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **bridge0** 接続に設定するには、次のように入力します。

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

6. オプション: ブリッジのその他のプロパティーを設定します。たとえば、**bridge0** の STP (Spanning Tree Protocol) の優先度を **16384** に設定するには、次のコマンドを実行します。

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

デフォルトでは STP が有効になっています。

7. 接続をアクティベートします。

```
# nmcli connection up bridge0
```

8. ポートが接続されており、**CONNECTION** コラムがポートの接続名を表示していることを確認します。

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

接続のいずれかのポートをアクティブにすると、NetworkManager はブリッジもアクティブにしますが、他のポートはアクティブにしません。ブリッジが有効な場合には、Red Hat Enterprise Linux がすべてのポートを自動的に有効にするように設定できます。

a. ブリッジ接続の **connection.autoconnect-slaves** パラメーターを有効にします。

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

b. ブリッジを再度アクティブにします。

```
# nmcli connection up bridge0
```

-

検証

- **ip** ユーティリティーを使用して、特定のブリッジのポートであるイーサネットデバイスのリンクステータスを表示します。

```
# 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
```

- **bridge** ユーティリティーを使用して、任意のブリッジデバイスのポートであるイーサネットデバイスの状態を表示します。

```
# 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
...
```

  特定のイーサネットデバイスのステータスを表示するには、**bridge link show dev ethernet_device_name** コマンドを使用します。

関連情報

- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- **nm-settings(5)** man ページ
- **bridge(8)** man ページ
- NetworkManager duplicates a connection after restart of NetworkManager service
- VLAN 情報を使用して、ブリッジを設定する方法

6.2. RHEL WEB コンソールを使用したネットワークブリッジの設定

Web ブラウザーベースのインターフェイスを使用してネットワーク設定を管理する場合は、RHEL Web コンソールを使用してネットワークブリッジを設定します。

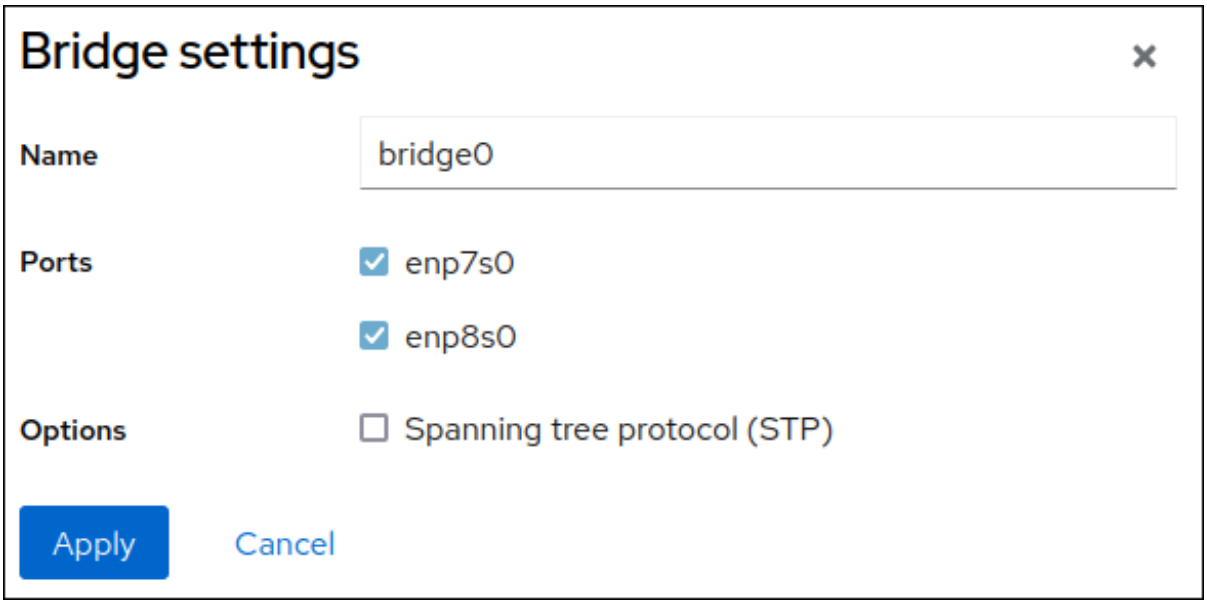
前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。

- ブリッジのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスをサーバーにインストールする必要があります。
- ブリッジのポートにチーム、ボンディング、または VLAN デバイスを使用するには、ブリッジの作成時にこれらのデバイスを作成するか、次の説明に従って事前にデバイスを作成することができます。
  - RHEL Web コンソールを使用したネットワークチームの設定
  - RHEL Web コンソールを使用したネットワークボンディングの設定
  - RHEL Web コンソールを使用した VLAN タグ付けの設定

手順

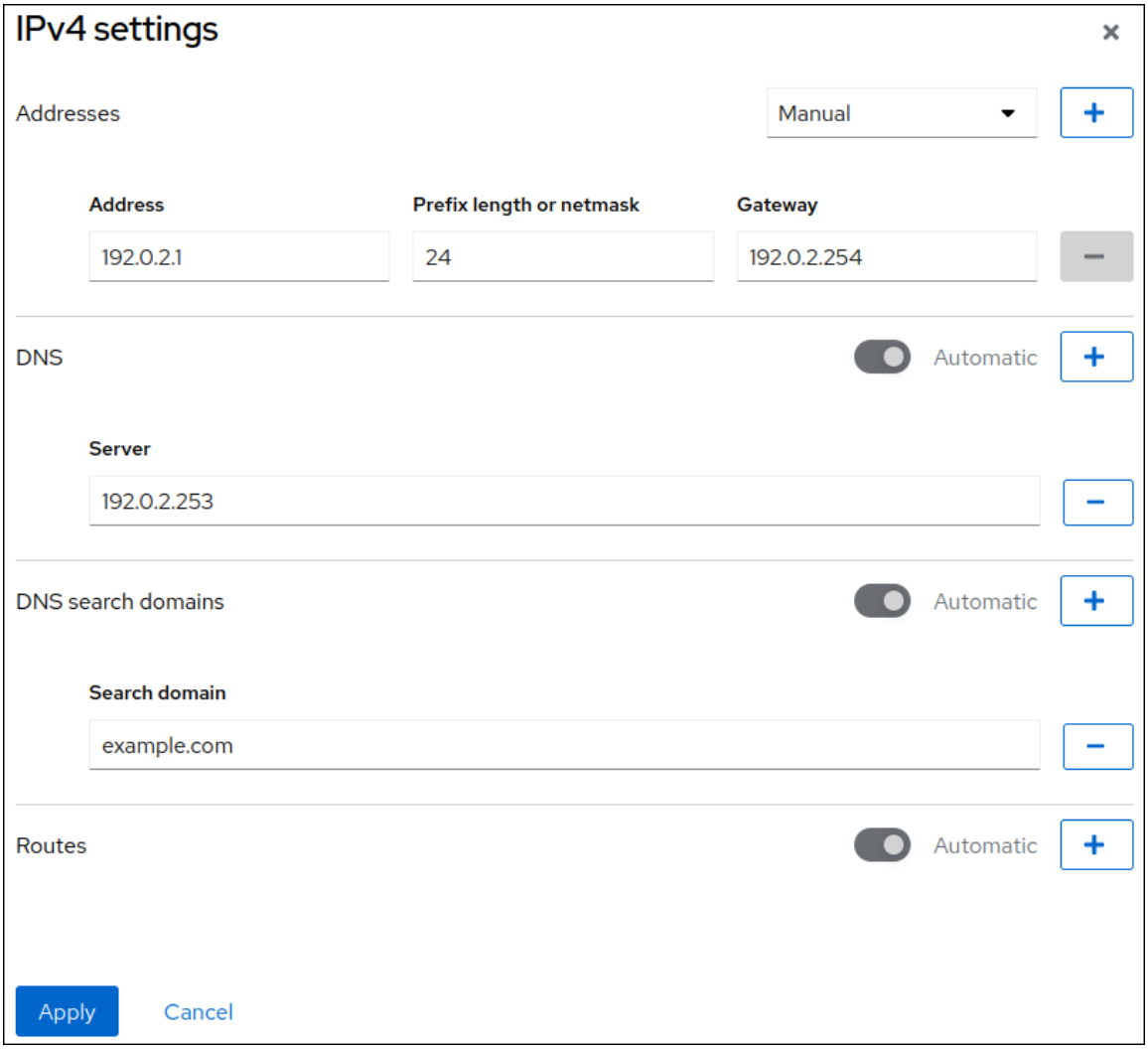
1. 画面左側のナビゲーションで **Networking** タブを選択します。
2. **Interfaces** セクションで **Add bridge** をクリックします。
3. 作成するブリッジデバイスの名前を入力します。
4. ブリッジのポートにするインターフェイスを選択します。
5. オプション: **Spanning tree protocol (STP)** 機能を有効にして、ブリッジループとブロードキャスト放射を回避します。



6. **Apply** をクリックします。
7. デフォルトでは、ブリッジは動的 IP アドレスを使用します。静的 IP アドレスを設定する場合:
   a. **Interfaces** セクションでブリッジの名前をクリックします。
   b. 設定するプロトコルの横にある **Edit** をクリックします。
   c. **Addresses** の横にある **Manual** を選択し、IP アドレス、接頭辞、およびデフォルトゲートウェイを入力します。
   d. **DNS** セクションで **+** ボタンをクリックし、DNS サーバーの IP アドレスを入力します。複数の DNS サーバーを設定するには、この手順を繰り返します。

e. **DNS search domains** セクションで、**+** ボタンをクリックし、検索ドメインを入力します。

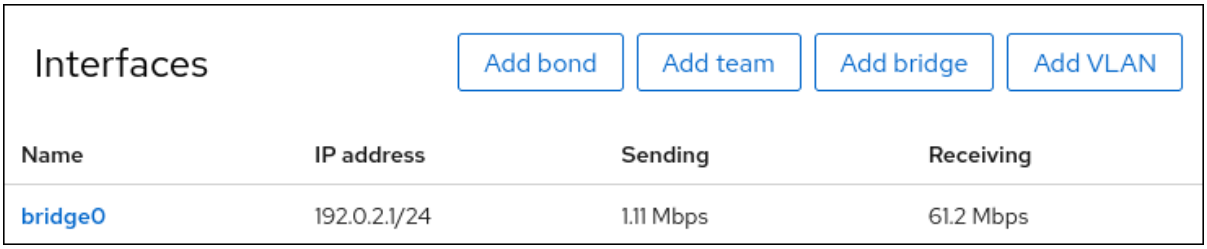
f. インターフェイスにスタティックルートが必要な場合は、**Routes** セクションで設定します。



g. **Apply** をクリックします。

検証

1. 画面左側のナビゲーションで **Networking** タブを選択し、インターフェイスに着信および発信トラフィックがあるかどうかを確認します。



6.3. NMTUI を使用したネットワークブリッジの設定

nmtui アプリケーションは、NetworkManager 用のテキストベースのユーザーインターフェイスを提供します。**nmtui** を使用して、グラフィカルインターフェイスを使用せずにホスト上でネットワークブリッジを設定できます。

注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ブリッジのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスをサーバーにインストールする必要があります。

手順

1. ネットワークブリッジを設定するネットワークデバイス名がわからない場合は、使用可能なデバイスを表示します。

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

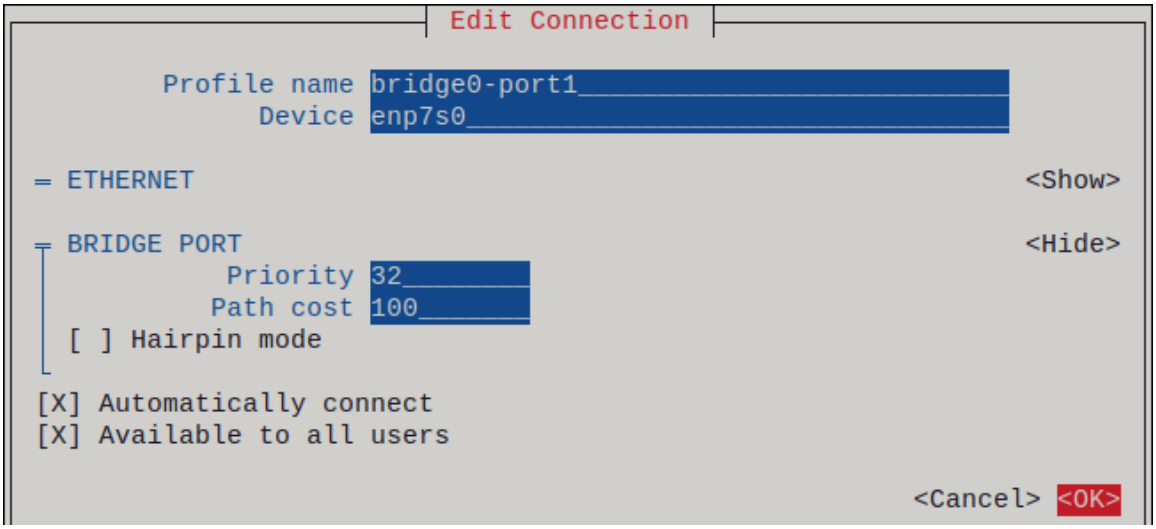
2. **nmtui** を開始します。

```
# nmtui
```

3. **Edit a connection** 選択し、**Enter** を押します。
4. **Add** ボタンを押します。
5. ネットワークタイプのリストから **Bridge** を選択し、**Enter** を押します。
6. オプション: 作成する NetworkManager プロファイルの名前を入力します。
ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。
7. 作成するブリッジデバイス名を **Device** フィールドに入力します。
8. 作成するブリッジにポートを追加します。
   a. **Slaves** リストの横にある **Add** ボタンを押します。
   b. ブリッジにポートとして追加するインターフェイスのタイプ (例: **Ethernet**) を選択します。
   c. オプション: このブリッジポート用に作成する NetworkManager プロファイルの名前を入力します。
   d. ポートのデバイス名を **Device** フィールドに入力します。

e. **OK** ボタンを押して、ブリッジ設定のウィンドウに戻ります。

図6.1イーサネットデバイスをポートとしてブリッジに追加する



f. ブリッジにさらにポートを追加するには、これらの手順を繰り返します。

9. 環境に応じて、**IPv4 configuration** および **IPv6 configuration** 領域に IP アドレス設定を設定します。これを行うには、これらの領域の横にあるボタンを押して、次を選択します。

- ブリッジが IP アドレスを必要としない場合は **Disabled** にします。
- DHCP サーバーまたはステートレスアドレス自動設定 (SLAAC) が IP アドレスをブリッジに動的に割り当てる場合は、**Automatic** にします。
- ネットワークで静的 IP アドレス設定が必要な場合は、**Manual** にします。この場合、さらにフィールドに入力する必要があります。

  i. 設定するプロトコルの横にある **Show** ボタンを押して、追加のフィールドを表示します。

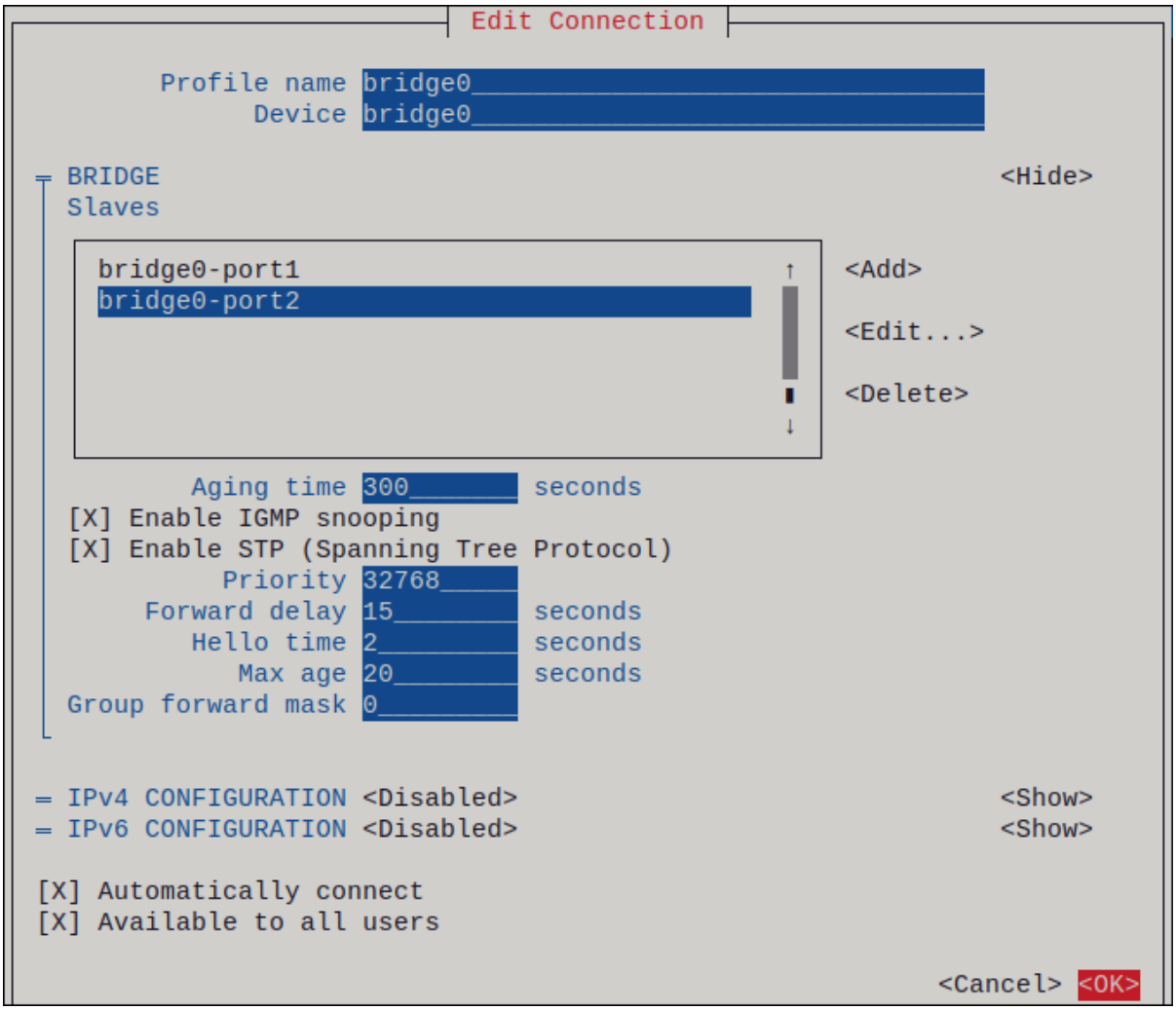
  ii. **Addresses** の横にある **Add** ボタンを押して、IP アドレスとサブネットマスクを Classless Inter-Domain Routing (CIDR) 形式で入力します。
  サブネットマスクを指定しない場合、NetworkManager は IPv4 アドレスに **/32** サブネットマスクを設定し、IPv6 アドレスに **/64** サブネットマスクを設定します。

  iii. デフォルトゲートウェイのアドレスを入力します。

  iv. **DNS servers** の横にある **Add** ボタンを押して、DNS サーバーのアドレスを入力します。

  v. **Search domains** の横にある **Add** ボタンを押して、DNS 検索ドメインを入力します。

図6.2 IP アドレス設定なしのブリッジ接続例



10. **OK** ボタンを押して、新しい接続を作成し、自動的にアクティブにします。
11. **Back** ボタンを押してメインメニューに戻ります。
12. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

1. **ip** ユーティリティーを使用して、特定のブリッジのポートであるイーサネットデバイスのリンクステータスを表示します。

```
# 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. **bridge** ユーティリティーを使用して、任意のブリッジデバイスのポートであるイーサネットデバイスの状態を表示します。

```
# 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
...
```

特定のイーサネットデバイスのステータスを表示するには、**bridge link show dev ethernet_device_name** コマンドを使用します。

6.4. NM-CONNECTION-EDITOR を使用したネットワークブリッジの設定

グラフィカルインターフェイスで Red Hat Enterprise Linux を使用する場合は、**nm-connection-editor** アプリケーションを使用してネットワークブリッジを設定できます。

nm-connection-editor は、新しいポートだけをブリッジに追加できることに注意してください。既存の接続プロファイルをポートとして使用するには、nmcli を使用したネットワークブリッジの設定 の説明に従って、**nmcli** ユーティリティーを使用してブリッジを作成します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- ブリッジのポートとしてイーサネットデバイスを使用するには、物理または仮想のイーサネットデバイスをサーバーにインストールする必要があります。
- ブリッジのポートとしてチーム、ボンディング、または VLAN デバイスを使用するには、これらのデバイスがまだ設定されていないことを確認してください。

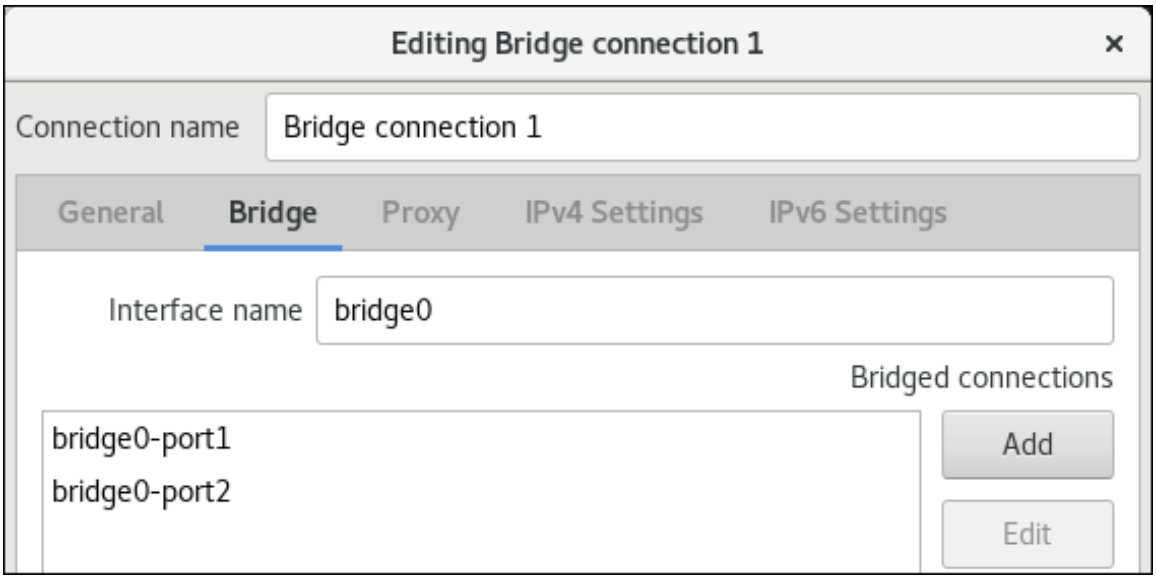
手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

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

2. **+** ボタンをクリックして、新しい接続を追加します。
3. 接続タイプ **Bridge** を選択し、**作成** をクリックします。
4. **Bridge** タブで以下を行います。
   a. オプション: **Interface name** フィールドにブリッジインターフェイスの名前を設定します。
   b. **追加** ボタンをクリックして、ネットワークインターフェイスの新しい接続プロファイルを作成し、プロファイルをポートとしてブリッジに追加します。
      i. インターフェイスの接続タイプを選択します。たとえば、有線接続に **Ethernet** を選択します。
      ii. 必要に応じて、ポートデバイスの接続名を設定します。
      iii. イーサネットデバイスの接続プロファイルを作成する場合は、**Ethernet** タブを開き、**Device** フィールドで選択し、ポートとしてブリッジに追加するネットワークインターフェイスを選択します。別のデバイスタイプを選択した場合は、それに応じて設定します。
      iv. **Save** をクリックします。

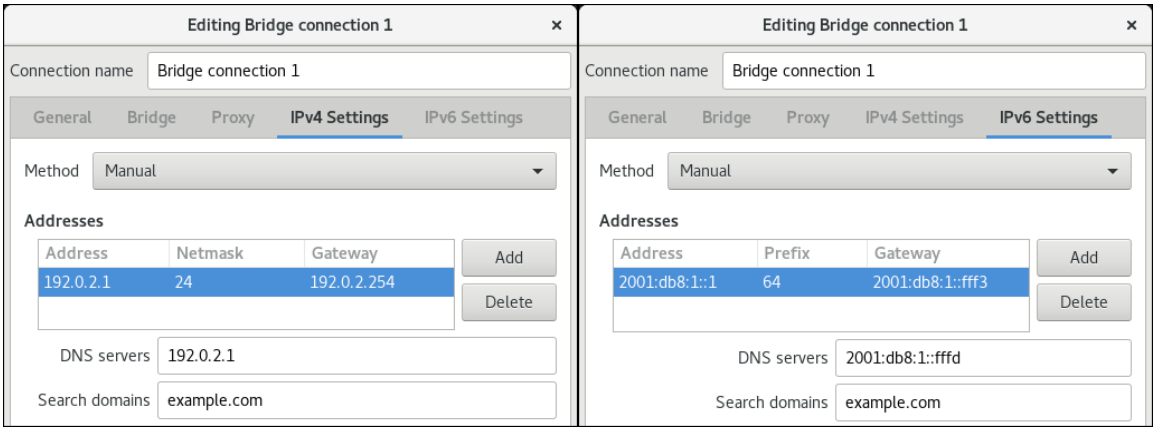
c. ブリッジに追加する各インターフェイスに、直前の手順を繰り返します。



5. オプション: スパニングツリープロトコル (STP) オプションなどの追加のブリッジ設定を行います。

6. **IPv4 Settings** タブと **IPv6 Settings** タブの両方で IP アドレス設定を設定します。

   - このブリッジデバイスを他のデバイスのポートとして使用するには、**Method** フィールドを **Disabled** に設定します。
   - DHCP を使用するには、**Method** フィールドをデフォルトの **Automatic (DHCP)** のままにします。
   - 静的 IP 設定を使用するには、**Method** フィールドを **Manual** に設定し、それに応じてフィールドに値を入力します。



7. **Save** をクリックします。

8. **nm-connection-editor** を閉じます。

検証

- **ip** ユーティリティーを使用して、特定のブリッジのポートであるイーサネットデバイスのリンクステータスを表示します。

```
# 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
```

- **bridge** ユーティリティーを使用して、任意のブリッジデバイスのポートであるイーサネットデバイスの状態を表示します。

```
# 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
...
```

特定のイーサネットデバイスのステータスを表示するには、**bridge link show dev ethernet_device_name** コマンドを使用します。

関連情報

- nm-connection-editor を使用したネットワークボンディングの設定
- nm-connection-editor を使用したネットワークチームの設定
- nm-connection-editor を使用した VLAN タグ付けの設定
- 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NetworkManager の設定
- VLAN 情報を使用して、ブリッジを設定する方法

6.5. NMSTATECTL を使用したネットワークブリッジの設定

nmstatectl ユーティリティーを使用して、Nmstate API を介してネットワークブリッジを設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

環境に応じて、YAML ファイルを適宜調整します。たとえば、ブリッジでイーサネットアダプターとは異なるデバイスを使用するには、ブリッジで使用するポートの **Base-iface** 属性と **type** 属性を調整します。

前提条件

- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。
- 物理または仮想のイーサネットデバイスをサーバーにインストールし、ブリッジでイーサネットデバイスをポートとして使用する。
- **ポート** リストでインターフェイス名を設定し、対応するインターフェイスを定義して、ブリッジのポートとしてチーム、ボンディング、または VLAN デバイスを使用する。

- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイル (例: ~/**create-bridge.yml**) を作成します。

```
---
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
```

これらの設定では、次の設定でネットワークブリッジを定義します。

- ブリッジのネットワークインターフェイス: **enp1s0** および **enp7s0**
- Spanning Tree Protocol (STP):有効
- 静的 IPv4 アドレス:**192.0.2.1** および **/24** サブネットマスク
- 静的 IPv6 アドレス:**2001: db8:1::1** および **/64** サブネットマスク
- IPv4 デフォルトゲートウェイ:**192.0.2.254**
- IPv6 デフォルトゲートウェイ:**2001:db8:1::fffe**
- IPv4 DNS サーバー:**192.0.2.200**
- IPv6 DNS サーバー:**2001:db8:1::ffbb**
- DNS 検索ドメイン: **example.com**

2. 設定をシステムに適用します。

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

検証

1. デバイスおよび接続の状態を表示します。

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

2. 接続プロファイルのすべての設定を表示します。

```
# 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. 接続設定を YAML 形式で表示します。

```
# nmstatectl show bridge0
```

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory
- VLAN 情報を使用して、ブリッジを設定する方法

6.6. NETWORK RHEL システムロールを使用したネットワークブリッジの設定

network RHEL システムロールを使用して、ネットワークブリッジをリモートで設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- サーバーに、2 つ以上の物理ネットワークデバイスまたは仮想ネットワークデバイスがインストールされている。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

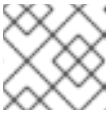
```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: Configure a network bridge that uses two Ethernet ports
      ansible.builtin.include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          # Define the bridge profile
          - 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
```

これらの設定では、次の設定でネットワークブリッジを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- ブリッジのポート - **enp7s0** および **enp8s0**



注記

Linux ブリッジのポートではなく、ブリッジに IP 設定を指定します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第7章 IPSEC VPN のセットアップ

仮想プライベートネットワーク (VPN) は、インターネット経由でローカルネットワークに接続する方法です。**Libreswan** により提供される **IPsec** は、VPN を作成するための望ましい方法です。**Libreswan** は、VPN のユーザー空間 **IPsec** 実装です。VPN は、インターネットなどの中間ネットワークにトンネルを設定して、使用中の LAN と別のリモート LAN との間の通信を可能にします。セキュリティー上の理由から、VPN トンネルは常に認証と暗号化を使用します。暗号化操作では、**Libreswan** は **NSS** ライブラリーを使用します。

7.1. CONTROL-CENTER による VPN 接続の確立

グラフィカルインターフェイスで Red Hat Enterprise Linux を使用する場合は、この VPN 接続を GNOME **control-center** で設定できます。

前提条件

- **NetworkManager-libreswan-gnome** パッケージがインストールされている。

手順

1. **Super** キーを押して **Settings** と入力し、**Enter** を押して **control-center** アプリケーションを開きます。
2. 左側の **Network** エントリーを選択します。
3. **+** アイコンをクリックします。
4. **VPN** を選択します。
5. **Identity** メニューエントリーを選択して、基本的な設定オプションを表示します。
   一般

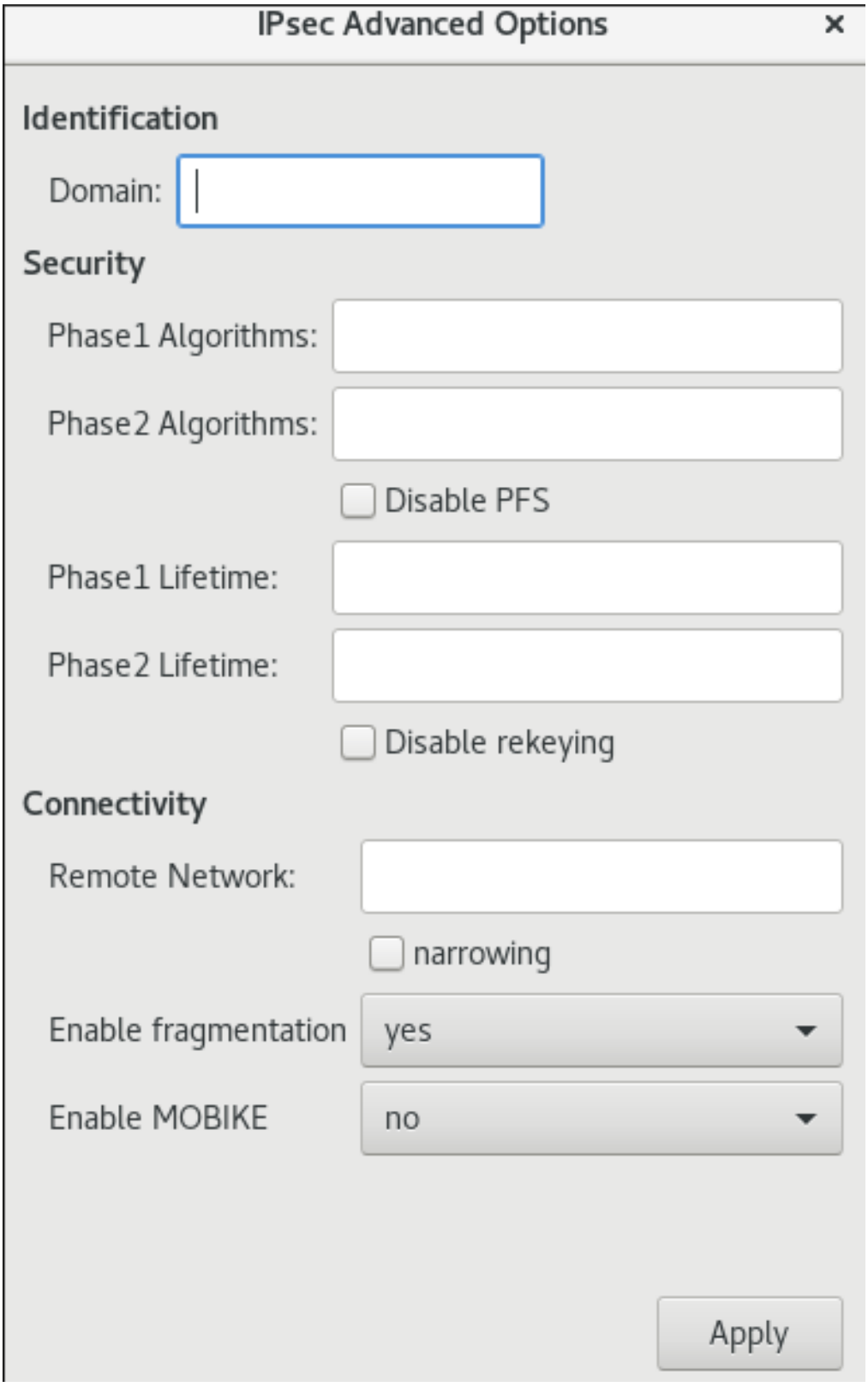
   Gateway - リモート VPN ゲートウェイの名前または **IP** アドレスです。

   認証

   Type

   - **IKEv2 (証明書)**- クライアントは、証明書により認証されます。これはより安全です (デフォルト)。
   - **IKEv1(XAUTH)**: クライアントは、ユーザー名とパスワード、または事前共有キー (PSK) で認証されます。
     Advanced セクションでは、以下の設定が可能です。

図7.1 VPN 接続の詳細なオプション



警告

gnome-control-center アプリケーションを使用して IPsec ベースの VPN 接続を設定すると、**Advanced** ダイアログには設定が表示されますが、変更することはできません。したがって、詳細な IPsec オプションを変更できません。**nm-connection-editor** ツールまたは **nmcli** ツールを使用して、詳細なプロパティーの設定を実行します。

識別

- **Domain** - 必要な場合は、ドメイン名を入力します。
  セキュリティー

- **Phase1 Algorithms** - Libreswan パラメーター **ike** に対応します。暗号化チャンネルの認証および設定に使用するアルゴリズムを入力します。

- **Phase2 Algorithms** - Libreswan パラメーター **esp** に対応します。**IPsec** ネゴシエーションに使用するアルゴリズムを入力します。
  Disable PFS フィールドで PFS (Perfect Forward Secrecy) を無効にし、PFS に対応していない古いサーバーとの互換性があることを確認します。

- **Phase1 Lifetime** - Libreswan パラメーター **ikelifetime** に対応します。このパラメーターは、トラフィックの暗号化に使用される鍵がどのぐらい有効であるかどうかを示します。

- **Phase2 Lifetime** - Libreswan パラメーター **salifetime** に対応します。このパラメーターは、接続の特定インスタンスが最後に終了するまでの時間を指定します。
  セキュリティー上の理由から、暗号化キーは定期的に変更する必要があります。

- **Remote network** - Libreswan パラメーター **rightsubnet** に対応します。このパラメーターは、VPN から到達できる宛先のプライベートリモートネットワークです。
  絞り込むことのできる **narrowing** フィールドを確認します。これは IKEv2 ネゴシエーションの場合にのみ有効であることに注意してください。

- **Enable fragmentation** - Libreswan パラメーターの **断片化** に対応します。IKE 断片化を許可するかどうかを指定します。有効な値は、**yes** (デフォルト) または **no** です。

- **Enable Mobike** - Libreswan パラメーター **mobike** に対応します。最初から接続を再起動しなくても、接続がエンドポイントを移行することを Mobility and Multihoming Protocol (MOBIKE, RFC 4555) が許可するかどうかを設定します。これは、有線、無線、またはモバイルデータの接続の切り替えを行うモバイルデバイスで使用されます。値は、**no** (デフォルト) または **yes** です。

6. **IPv4** メニューエントリーを選択します。
   IPv4 Method

   - **Automatic (DHCP)** - 接続しているネットワークが動的 **IP** アドレスの割り当てに **DHCP** サーバーを使用する場合は、このオプションを選択します。

   - **Link-Local Only** - 接続しているネットワークに **DHCP** サーバーがなく、**IP** アドレスを手動で割り当てない場合は、このオプションを選択します。接頭辞 **169.254/16** 付きのランダムなアドレスが、RFC 3927 に従って割り当てられます。

- **Manual** - **IP** アドレスを手動で割り当てる場合は、このオプションを選択します。
- **Disable** - この接続では **IPv4** は無効です。
  DNS

  DNS セクションでは、**Automatic** が **ON** になっているときに、これを **OFF** に切り替えて、使用する DNS サーバーの IP アドレスを入力します。IP アドレスはコンマで区切ります。

  Routes

  Routes セクションでは、**Automatic** が **ON** になっている場合は、DHCP からのルートが使用されますが、他の静的ルートを追加することもできることに注意してください。**OFF** の場合は、静的ルートだけが使用されます。

- **Address** - リモートネットワークまたはホストの **IP** アドレスを入力します。
- **Netmask** - 上に入力した **IP** アドレスのネットマスクまたは接頭辞長。
- **Gateway** - 上に入力したリモートネットワーク、またはホストにつながるゲートウェイの **IP** アドレス。
- **Metric** - このルートに付与する優先値であるネットワークコスト。数値が低い方が優先されます。
  Use this connection only for resources on its network (**この接続はネットワーク上のリソースのためだけに使用**)

  このチェックボックスを選択すると、この接続はデフォルトルートになりません。このオプションを選択すると、この接続で自動的に学習したルートを使用することが明確なトラフィックか、手動で入力したトラフィックのみがこの接続を経由します。

7. **VPN** 接続の **IPv6** 設定を設定するには、**IPv6** メニューエントリーを選択します。
   IPv6 Method

   - **Automatic** - **IPv6** ステートレスアドレス自動設定 (SLAAC) を使用して、ハードウェアのアドレスとルーター通知 (RA) に基づくステートレスの自動設定を作成するには、このオプションを選択します。
   - **Automatic, DHCP only** - RA を使用せず、直接 **DHCPv6** に情報を要求してステートフルな設定を作成する場合は、このオプションを選択します。
   - **Link-Local Only** - 接続しているネットワークに **DHCP** サーバーがなく、**IP** アドレスを手動で割り当てない場合は、このオプションを選択します。接頭辞 **FE80::0** 付きのランダムなアドレスが、RFC 4862 に従って割り当てられます。
   - **Manual** - **IP** アドレスを手動で割り当てる場合は、このオプションを選択します。
   - **Disable** - この接続では **IPv6** は無効です。
     DNS、**Routes**、**Use this connection only for resources on its network** が、一般的な **IPv4** 設定となることに注意してください。

8. **VPN** 接続の編集が終了したら、**追加** ボタンをクリックして設定をカスタマイズするか、**適用** ボタンをクリックして、既存の接続に保存します。
9. プロファイルを **ON** に切り替え、**VPN** 接続をアクティブにします。

関連情報

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

7.2. NM-CONNECTION-EDITOR による VPN 接続の設定

Red Hat Enterprise Linux をグラフィカルインターフェイスで使用する場合は、**nm-connection-editor** アプリケーションを使用して VPN 接続を設定できます。

前提条件

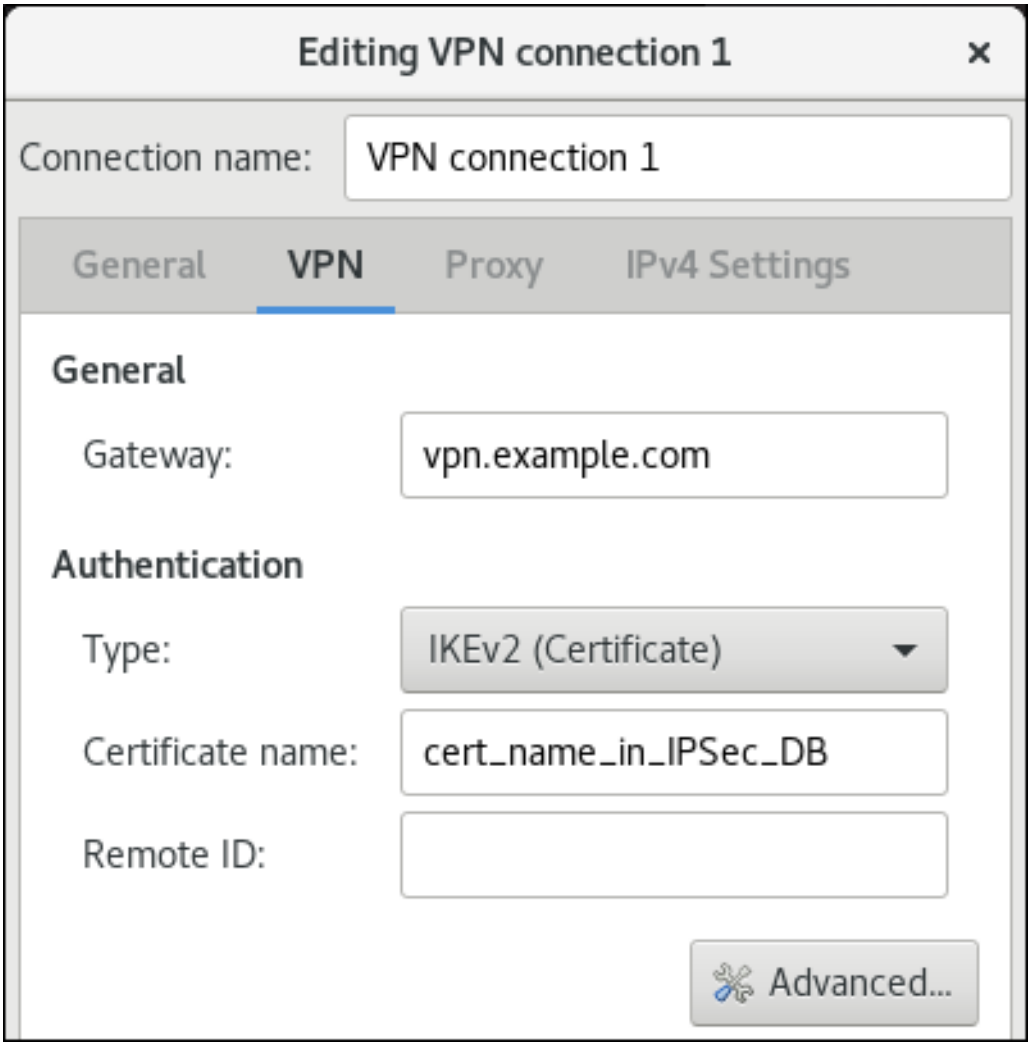
- **NetworkManager-libreswan-gnome** パッケージがインストールされている。
- インターネット鍵交換バージョン 2 (IKEv2) 接続を設定する場合は、以下のようになります。
  - 証明書が、IPsec ネットワークセキュリティーサービス (NSS) データベースにインポートされている。
  - NSS データベースの証明書のニックネームが知られている。

手順

1. ターミナルを開き、次のコマンドを入力します。

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

2. **+** ボタンをクリックして、新しい接続を追加します。
3. **IPsec ベースの VPN** 接続タイプを選択し、**作成** をクリックします。
4. **VPN** タブで、以下を行います。
   a. **Gateway** フィールドに VPN ゲートウェイのホスト名または IP アドレスを入力し、認証タイプを選択します。認証タイプに応じて、異なる追加情報を入力する必要があります。
      - **IKEv2 (Certifiate)** は、証明書を使用してクライアントを認証します。これは、より安全です。この設定には、IPsec NSS データベースの証明書のニックネームが必要です。
      - **IKEv1 (XAUTH)** は、ユーザー名とパスワード (事前共有鍵) を使用してユーザーを認証します。この設定は、以下の値を入力する必要があります。
        - ユーザー名
        - Password
        - グループ名
        - シークレット
   b. リモートサーバーが IKE 交換のローカル識別子を指定する場合は、**Remote ID** フィールドに正確な文字列を入力します。リモートサーバーで Libreswan を実行すると、この値はサーバーの **leftid** パラメーターに設定されます。



c. 必要に応じて、**詳細** ボタンをクリックして、追加設定を設定します。以下の設定を設定できます。

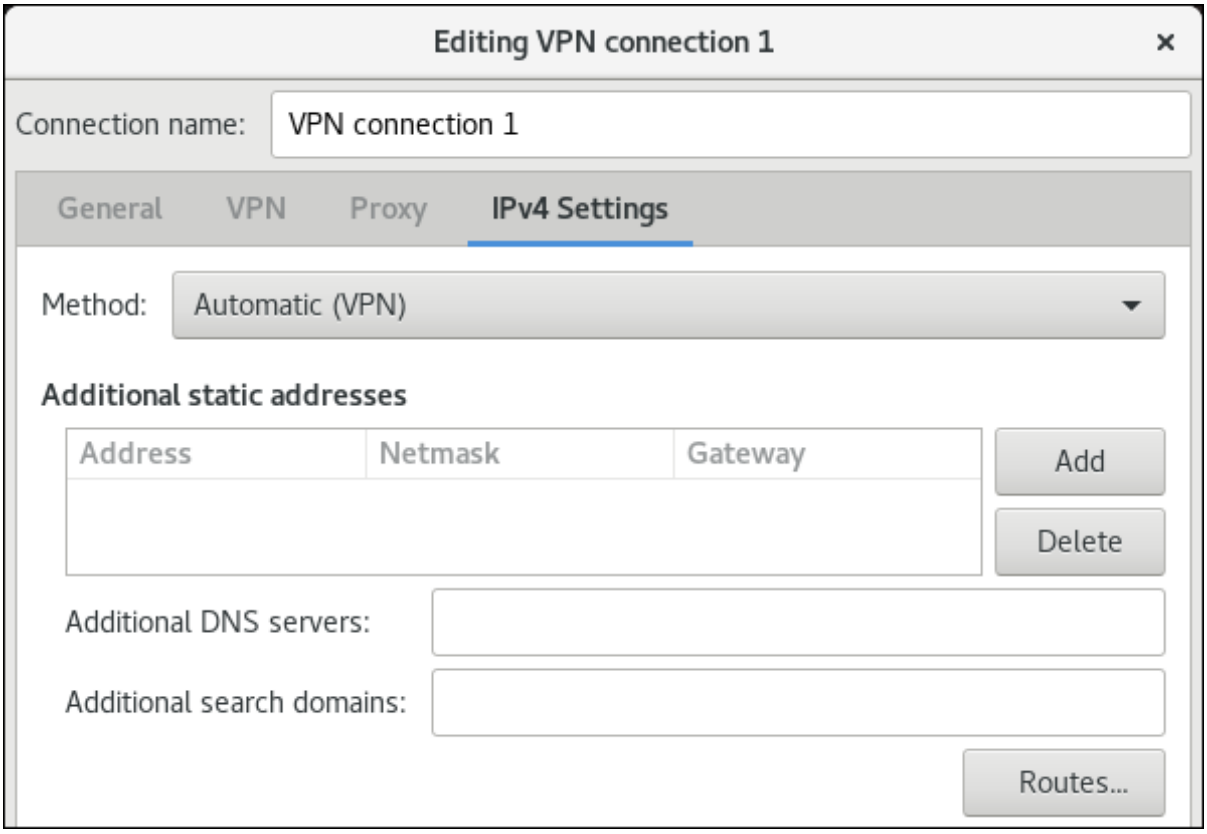
- 識別
  - **ドメイン** - 必要な場合は、ドメイン名を入力します。
- セキュリティー
  - **Phase1 アルゴリズム** は、Libreswan パラメーター **ike** に対応します。暗号化チャンネルの認証および設定に使用するアルゴリズムを入力します。
  - **Phase2 アルゴリズム** は、Libreswan パラメーター **esp** に対応します。**IPsec** ネゴシエーションに使用するアルゴリズムを入力します。
    Disable PFS フィールドで PFS (Perfect Forward Secrecy) を無効にし、PFS に対応していない古いサーバーとの互換性があることを確認します。
  - **Phase1 ライフタイム** は、Libreswan パラメーター **ikelifetime** に対応します。このパラメーターは、トラフィックの暗号化に使用される鍵が有効である期間を定義します。
  - **Phase2 ライフタイム** は、Libreswan パラメーター **salifetime** に対応します。このパラメーターは、セキュリティー関連が有効である期間を定義します。
- 接続性

- **リモートネットワーク** は、Libreswan パラメーター **rightsubnet** に対応し、VPN から到達できる宛先のプライベートリモートネットワークです。
  絞り込むことのできる **narrowing** フィールドを確認します。これは IKEv2 ネゴシエーションの場合にのみ有効であることに注意してください。

- **フラグメンテーションの有効化** は、Libreswan パラメーターの **断片化** に対応します。IKE 断片化を許可するかどうかを指定します。有効な値は、**yes** (デフォルト) または **no** です。

- **Mobike の有効化** は、Libreswan パラメーター **mobike** に対応します。パラメーターは、最初から接続を再起動しなくても、接続がエンドポイントを移行するようにするため、MOBIKE (Mobility and Multihoming Protocol) (RFC 4555) を許可するかどうかを定義します。これは、有線、無線、またはモバイルデータの接続の切り替えを行うモバイルデバイスで使用されます。値は、**no** (デフォルト) または **yes** です。

5. **IPv4 設定** タブで、IP 割り当て方法を選択し、必要に応じて、追加の静的アドレス、DNS サーバー、検索ドメイン、ルートを設定します。



6. 接続を読み込みます。

7. **nm-connection-editor** を閉じます。

注記

+ ボタンをクリックして新しい接続を追加する場合は、**NetworkManager** により、その接続用の新しい設定が作成され、既存の接続の編集に使用するのと同じダイアログが表示されます。このダイアログの違いは、既存の接続プロファイルに **Details** メニューエントリーがあることです。

関連情報

- **nm-settings-libreswan(5)** の man ページ

7.3. IPSEC 接続を高速化するために、ESP ハードウェアオフロードの自動検出と使用を設定

Encapsulating Security Payload (ESP) をハードウェアにオフロードすると、Ethernet で IPsec 接続が加速します。デフォルトでは、Libreswan は、ハードウェアがこの機能に対応しているかどうかを検出するため、ESP ハードウェアのオフロードを有効にします。機能が無効になっているか、明示的に有効になっている場合は、自動検出に戻すことができます。

前提条件

- ネットワークカードは、ESP ハードウェアオフロードに対応します。
- ネットワークドライバーは、ESP ハードウェアのオフロードに対応します。
- IPsec 接続が設定され、動作する。

手順

1. ESP ハードウェアオフロードサポートの自動検出を使用する接続の **/etc/ipsec.d/** ディレクトリーにある Libreswan 設定ファイルを編集します。
2. 接続の設定で **nic-offload** パラメーターが設定されていないことを確認します。
3. **nic-offload** を削除した場合は、**ipsec** を再起動します。

```
# systemctl restart ipsec
```

検証

ネットワークカードが ESP ハードウェアオフロードサポートに対応している場合は、以下の手順に従って結果を検証します。

1. IPsec 接続が使用するイーサネットデバイスの **tx_ipsec** および **rx_ipsec** カウンターを表示します。

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

2. IPsec トンネルを介してトラフィックを送信します。たとえば、リモート IP アドレスに ping します。

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

3. イーサネットデバイスの **tx_ipsec** および **rx_ipsec** カウンターを再度表示します。

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

カウンターの値が増えると、ESP ハードウェアオフロードが動作します。

関連情報

- IPsec を使用した VPN の設定

7.4. IPSEC 接続を加速化するためにボンディングでの ESP ハードウェアオフロードの設定

Encapsulating Security Payload (ESP) をハードウェアにオフロードすると、IPsec 接続が加速します。フェイルオーバーの理由でネットワークボンディングを使用する場合、ESP ハードウェアオフロードを設定する要件と手順は、通常のイーサーネットデバイスを使用する要件と手順とは異なります。たとえば、このシナリオでは、ボンディングでオフロードサポートを有効にし、カーネルはボンディングのポートに設定を適用します。

前提条件

- ボンディングのすべてのネットワークカードが、ESP ハードウェアオフロードをサポートしている。
- ネットワークドライバーが、ボンドデバイスで ESP ハードウェアオフロードに対応している。RHEL では、**ixgbe** ドライバーのみがこの機能をサポートします。
- ボンディングが設定されており動作する。
- ボンディングで **active-backup** モードを使用している。ボンディングドライバーは、この機能の他のモードはサポートしていません。
- IPsec 接続が設定され、動作する。

手順

1. ネットワークボンディングで ESP ハードウェアオフロードのサポートを有効にします。

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

   このコマンドにより、**bond0** 接続での ESP ハードウェアオフロードのサポートが有効になります。

2. **bond0** 接続を再度アクティブにします。

```
# nmcli connection up bond0
```

3. ESP ハードウェアオフロードに使用すべき接続の **/etc/ipsec.d/** ディレクトリーにある Libreswan 設定ファイルを編集し、**nic-offload=yes** ステートメントを接続エントリーに追加します。

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

4. **ipsec** サービスを再起動します。

```
# systemctl restart ipsec
```

検証

1. ボンディングのアクティブなポートを表示します。

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

2. アクティブなポートの **tx_ipsec** カウンターおよび **rx_ipsec** カウンターを表示します。

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

3. IPsec トンネルを介してトラフィックを送信します。たとえば、リモート IP アドレスに ping します。

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

4. アクティブなポートの **tx_ipsec** カウンターおよび **rx_ipsec** カウンターを再度表示します。

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

カウンターの値が増えると、ESP ハードウェアオフロードが動作します。

関連情報

- ネットワークボンディングの設定
- ネットワークのセキュリティー保護ドキュメントの Configuring a VPN with IPsec セクション

第8章 WIREGUARD VPN の設定

WireGuard は、Linux カーネルで実行する高パフォーマンスの VPN ソリューションです。最新の暗号を使用し、他の多くの VPN ソリューションよりも簡単に設定できます。さらに、WireGuard のコードベースが小さくなり、攻撃の影響が減るため、セキュリティーが向上します。認証および暗号化には、WireGuard が SSH と同様の鍵を使用します。

重要

WireGuard はテクノロジープレビューとしてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

WireGuard VPN に参加するすべてのホストがピアであることに注意してください。このドキュメントでは、接続を確立するホストを説明する **client** という用語と、クライアントが接続する固定ホスト名または IP アドレスを使用してホストを説明する **server** という用語を使用し、必要に応じてすべてのトラフィックをこのサーバーにルーティングします。

WireGuard VPN を設定するには、次の手順を完了する必要があります。さまざまなオプションを使用して、ほとんどの手順を実行できます。

1. VPN 内のすべてのホストに公開鍵と秘密鍵を作成します。
2. nmcli、nmtui、nm-connection-editor、または、wg-quick サービスを使用して、WireGuard サーバーを設定します。
3. コマンドライン または グラフィカルインターフェイス を使用して、WireGuard サーバーで firewalld を設定します。
4. nmcli、nm-connection-editor、または wg-quick を使用して、WireGuard クライアントを設定します。

WireGuard は、ネットワーク層 (レイヤー 3) で動作します。そのため、DHCP を使用できず、静的 IP アドレスまたは IPv6 リンクローカルアドレスを、サーバーとクライアントの両方のトンネルデバイスに割り当てる必要があります。

重要

WireGuard は、RHEL の FIPS (Federal Information Processing Standard) モードが無効になっている場合にのみ使用できます。

8.1. WIREGUARD が使用するプロトコルおよびプリミティブ

WireGuard は、次のプロトコルおよびプリミティブを使用します。

- RFC7539 で説明されているように Authenticated Encryption with Associated Data (AEAD) 構造を使用して、Poly1305 で認証された対称暗号化用の ChaCha20
- Elliptic-curve Diffie–Hellman (ECDH) 鍵交換用の Curve25519

- RFC7693 で説明されているように、ハッシュ用および鍵付きのハッシュ用の BLAKE2
- ハッシュテーブルキーの SipHash24
- RFC5869 で説明されているように、鍵の派生に使用される HKDF

8.2. WIREGUARD がトンネル IP アドレス、公開鍵、およびリモートエンドポイントを使用する方法

WireGuard がピアにネットワークパケットを送信する場合は、次のコマンドを実行します。

1. WireGuard は、パケットから宛先 IP を読み込み、ローカル設定で許可されている IP アドレスのリストと比較します。ピアが見つからない場合、WireGuard はパケットを破棄します。
2. ピアが有効な場合、WireGuard は、ピアの公開鍵を使用してパケットを暗号化します。
3. 送信側ホストは、ホストの最新のインターネット IP アドレスを検索し、暗号化したパケットを送信します。

WireGuard がパケットを受信すると、以下が行われます。

1. WireGuard は、リモートホストの秘密鍵を使用してパケットを復号します。
2. WireGuard は、パケットから内部ソースアドレスを読み込み、ローカルホストのピア設定で許可されている IP アドレスのリストに IP が設定されているかどうかを調べます。ソース IP が許可リストにある場合、WireGuard はパケットを受け入れます。IP アドレスがリストにない場合は、WireGuard がパケットを破棄します。

公開鍵と許可された IP アドレスの関連付けは、**Cryptokey Routing Table** と呼ばれます。つまり、IP アドレスのリストは、パケットの送信時にはルーティングテーブルと同様に動作し、パケットの受信時にはアクセス制御リストのように動作します。

8.3. NAT およびファイアウォールの背後で WIREGUARD クライアントを使用する

WireGuard は UDP プロトコルを使用し、ピアがパケットを送信する場合にのみデータを送信します。ルーターのステートフルファイアウォールとネットワークアドレス変換 (NAT) は、接続を追跡して、NAT の背後のピアまたはファイアウォールがパケットを受信できるようにします。

コネクションをアクティブな状態に保つために、WireGuard は **persistent keepalives** に対応しています。つまり、WireGuard がキープアライブパケットを送信する間隔を設定できます。デフォルトでは、ネットワークトラフィックを削減するために、永続的なキープアライブ機能は無効になっています。NAT を使用したネットワークでクライアントを使用する場合、またはしばらく非アクティブにした後にファイアウォールが接続を閉じる場合は、クライアントでこの機能を有効にします。

8.4. WIREGUARD 接続で使用される秘密鍵および公開鍵の作成

WireGuard は、base64 でエンコードされた秘密鍵と公開鍵を使用して、ホストを相互に認証します。そのため、WireGuard VPN に参加する各ホストで鍵を作成する必要があります。

重要

セキュアな接続には、ホストごとに異なる鍵を作成し、公開鍵のみをリモートの WireGuard ホストと共有するようにしてください。このドキュメントで使用しているサンプルキーは使用しないでください。

手順

1. **wireguard-tools** パッケージをインストールします。

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

2. ホストの秘密鍵と、対応する公開鍵を作成します。

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

キーファイルの内容は必要ですが、ファイル自体は必要ありません。ただし、Red Hat では、将来的に鍵を覚えておく必要がある場合に備え、ファイルを保持することを推奨しています。

3. キーファイルにセキュアなパーミッションを設定します。

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

4. 秘密鍵を表示します。

```
# cat /etc/wireguard/$HOSTNAME.private.key
YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=
```

ローカルホストで WireGuard 接続を設定するには、秘密鍵が必要です。秘密鍵を共有しないでください。

5. 公開鍵を表示します。

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

リモートホストで WireGuard 接続を設定するには、公開鍵が必要です。

関連情報

- man ページの **wg(8)**

8.5. NMCLI を使用した WIREGUARD サーバーの設定

NetworkManager で接続プロファイルを作成することで、WireGuard サーバーを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

この手順では、次の設定を前提としています。

- サーバー
  - プライベートキー:**YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**

- トンネル IPv4 アドレス:**192.0.2.1/24**
- トンネル IPv6 アドレス:**2001:db8:1::1/32**

- クライアント:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - サーバーの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - クライアントの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. NetworkManager WireGuard 接続プロファイルを追加します。

```
# nmcli connection add type wireguard con-name server-wg0 ifname wg0 autoconnect no
```

server-wg0 という名前のプロファイルを作成し、そのプロファイルに仮想インターフェイス **wg0** を割り当てます。設定を確定せずに接続を追加した後、接続が自動的に開始しないようにするには、**autoconnect** パラメーターを無効にします。

2. サーバーのトンネル IPv4 アドレスおよびサブネットマスクを設定します。

```
# nmcli connection modify server-wg0 ipv4.method manual ipv4.addresses 192.0.2.1/24
```

3. サーバーのトンネル IPv6 アドレスおよびサブネットマスクを設定します。

```
# nmcli connection modify server-wg0 ipv6.method manual ipv6.addresses 2001:db8:1::1/32
```

4. サーバーの秘密鍵を接続プロファイルに追加します。

```
# nmcli connection modify server-wg0 wireguard.private-key "YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg="
```

5. 着信 WireGuard 接続のポートを設定します。

```
# nmcli connection modify server-wg0 wireguard.listen-port 51820
```

着信 WireGuard 接続を受信するホストでは、常に固定ポート番号を設定してください。ポートを設定しないと、**wg0** インターフェイスをアクティブにするたびにランダムな空きポートが使用されます。

6. このサーバーとの通信を許可する各クライアントに、ピア設定を追加します。この設定は手動で追加する必要があります。**nmcli** ユーティリティーでは、対応する接続プロパティーの設定に対応していないためです。

   a. **/etc/NetworkManager/system-connections/server-wg0.nmconnection** ファイルーを編集し、以下を追加します。

   ```
   [wireguard-peer.bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=]
   allowed-ips=192.0.2.2;2001:db8:1::2;
   ```

   - **[wireguard-peer.<public_key_of_the_client>]** エントリーは、クライアントのピアセクションを定義し、セクション名にはクライアントの公開鍵が含まれます。
   - **allowed-ips** パラメーターは、このサーバーへのデータ送信を許可するクライアントのトンネル IP アドレスを設定します。
     各クライアントにセクションを追加します。

   b. **server-wg0** 接続プロファイルを再読み込みします。

   ```
   # nmcli connection load /etc/NetworkManager/system-connections/server-wg0.nmconnection
   ```

7. オプション: 自動的に起動するように接続を設定し、次のコマンドを実行します。

   ```
   # nmcli connection modify server-wg0 autoconnect yes
   ```

8. **server-wg0** 接続を再アクティブ化します。

   ```
   # nmcli connection up server-wg0
   ```

次のステップ

- WireGuard サーバーで firewalld サービスを設定します。

検証

1. **wg0** デバイスのインターフェイス設定を表示します。

   ```
   # 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
   ```

   出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

2. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**
- **nm-settings(5)** の man ページの **WireGuard setting** セクション

8.6. NMTUI を使用した WIREGUARD サーバーの設定

NetworkManager で接続プロファイルを作成することで、WireGuard サーバーを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

この手順では、次の設定を前提としています。

- サーバー
  - プライベートキー:**YFAnE0psgldiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
  - トンネル IPv4 アドレス:**192.0.2.1/24**
  - トンネル IPv6 アドレス:**2001:db8:1::1/32**
- クライアント:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - サーバーの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - クライアントの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

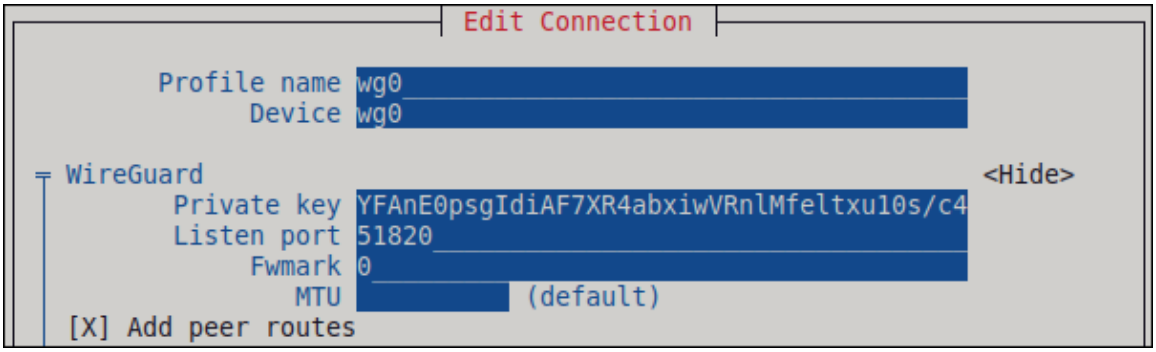
- **NetworkManager-tui** パッケージをインストールしました。

手順

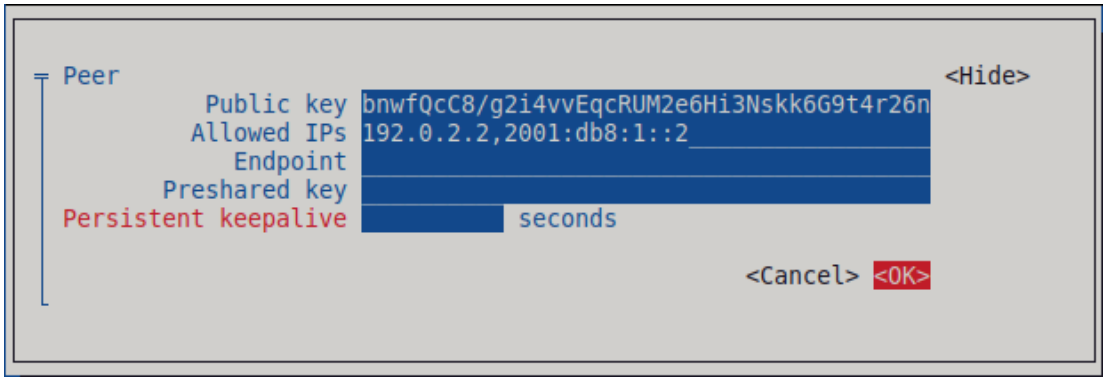
1. **nmtui** アプリケーションを開始します。

```
# nmtui
```

2. **Edit a connection** 選択し、**Enter** を押します。
3. **Add** を選択し、**Enter** を押します。
4. リストから **WireGuard** 接続タイプを選択し、**Enter** を押します。
5. **Edit connection** ウィンドウで:
   a. NetworkManager が接続に割り当てる仮想インターフェイス (**wg0** など) の接続名を入力します。
   b. サーバーの秘密鍵を入力します。
   c. 着信 WireGuard 接続のリッスンポート番号 (**51820** など) を設定します。
   着信 WireGuard 接続を受信するホストでは、常に固定ポート番号を設定してください。
   ポートを設定しないと、WireGuard はインターフェイスをアクティブにするたびにランダムな空きポートを使用します。



   d. **Peers** ペインの横にある **Add** をクリックします。
      i. クライアントの公開鍵を入力します。
      ii. **Allowed IPs** フィールドには、このサーバーへのデータ送信を許可するクライアントのトンネル IP アドレスを設定します。
      iii. **OK** を選択し、**Enter** を押します。



e. **IPv4 Configuration** の横にある **Show** を選択し、**Enter** を押します。

i. IPv4 設定方法 **Manual** を選択します。

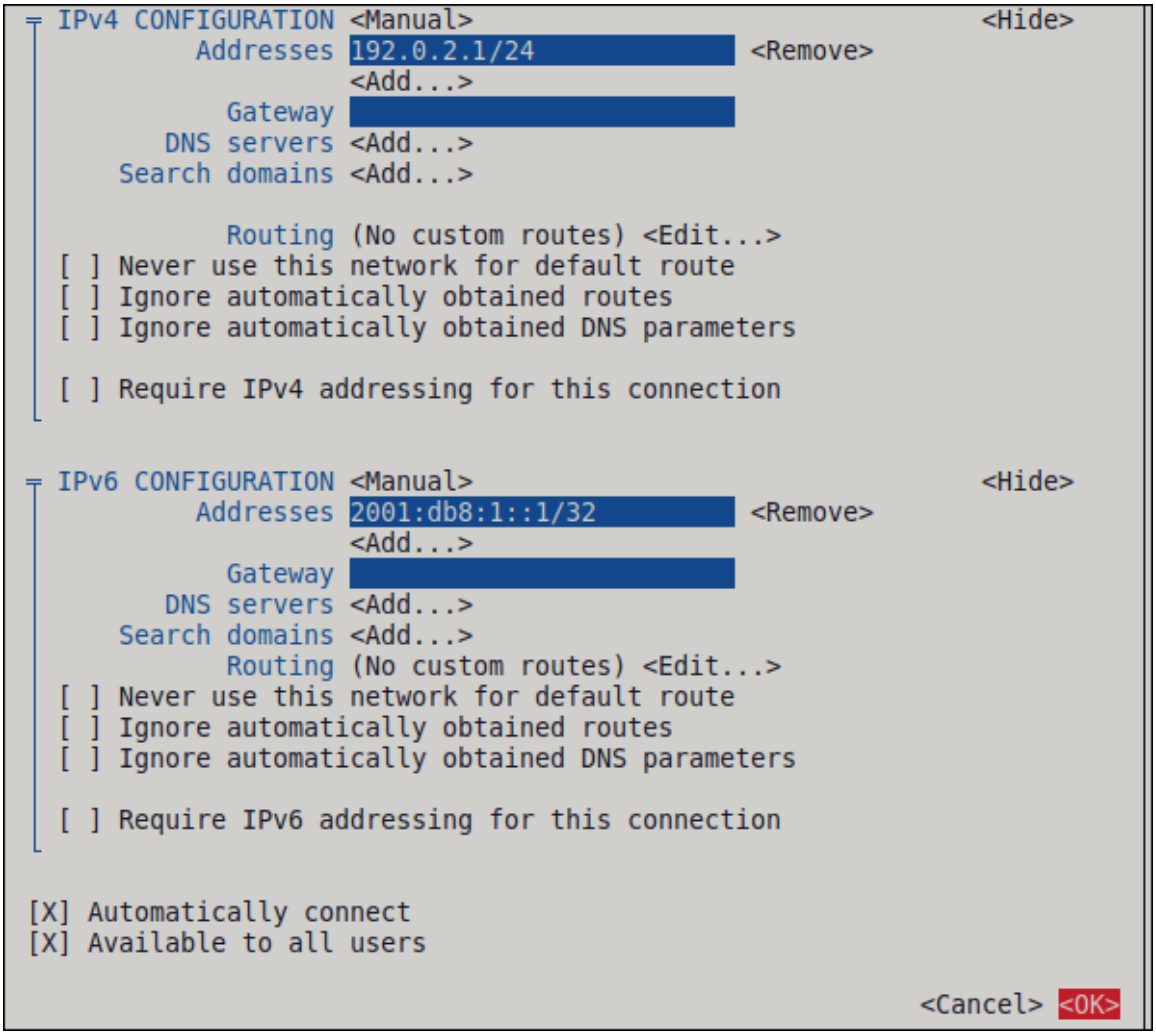
ii. トンネルの IPv4 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。

f. **IPv6 Configuration** の横にある **Show** を選択し、**Enter** を押します。

i. IPv6 設定方法 **Manual** を選択します。

ii. トンネルの IPv6 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。

g. **OK** を選択し、**Enter** を押します



6. 接続のリストが表示されたウィンドウで、**Back** を選択し、**Enter** を押します。

7. **NetworkManager TUI** のメインウィンドウで、**Quit** を選択し、**Enter** を押します。

次のステップ

- WireGuard サーバーで firewalld サービスを設定します。

検証

1. **wg0** デバイスのインターフェイス設定を表示します。

■

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

2. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**

8.7. NM-CONNECTION-EDITOR を使用した WIREGUARD サーバーの設定

NetworkManager で接続プロファイルを作成することで、WireGuard サーバーを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - サーバーの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - クライアントの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. ターミナルを開き、次のコマンドを入力します。

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

2. **+** ボタンをクリックして、新しいコネクションを追加します。
3. **WireGuard** の接続の種類を選択し、**作成** をクリックします。
4. オプション: 接続名を更新します。
5. **General** タブで、**Connect automatically with priority** を選択します。必要に応じて、優先度の値を設定します。
6. **WireGuard** タブで、以下を行います。
   a. NetworkManager が接続に割り当てる仮想インターフェイス (**wg0** など) の名前を入力します。
   b. サーバーの秘密鍵を入力します。
   c. 着信 WireGuard 接続のリッスンポート番号 (**51820** など) を設定します。
   着信 WireGuard 接続を受信するホストでは、常に固定ポート番号を設定してください。ポートを設定しないと、WireGuard はインターフェイスをアクティブにするたびにランダムな空きポートを使用します。
   d. **追加** を選択して、ピアを追加します。
      i. クライアントの公開鍵を入力します。
      ii. **Allowed IPs** フィールドには、このサーバーへのデータ送信を許可するクライアントのトンネル IP アドレスを設定します。
      iii. **Apply** をクリックします。
7. **IPv4 Settings** タブで、以下を行います。
   a. **Method** リストで **Manual** を選択します。
   b. **追加** を選択して、トンネルの IPv4 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。
8. **IPv6 Settings** タブで、以下を行います。
   a. **Method** リストで **Manual** を選択します。
   b. **追加** を選択して、トンネル IPv6 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。
9. **保存** を選択して、接続プロファイルを保存します。

次のステップ

- WireGuard サーバーで firewalld サービスを設定します。

検証

1. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

2. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**

8.8. WG-QUICK サービスを使用した WIREGUARD サーバーの設定

/etc/wireguard/ ディレクトリーに設定ファイルを作成することで、WireGuard サーバーを設定できます。この方法を使用して、NetworkManager からサービスを独立して設定します。

この手順では、次の設定を前提としています。

- サーバー
  - プライベートキー:**YFAnE0psgIdiAF7XR4abxiwVRnIMfeltxu10s/c4JXg=**
  - トンネル IPv4 アドレス:**192.0.2.1/24**
  - トンネル IPv6 アドレス:**2001:db8:1::1/32**
- クライアント:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - サーバーの秘密鍵

- クライアントの静的トンネルの IP アドレスとサブネットマスク
- クライアントの公開鍵
- サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. **wireguard-tools** パッケージをインストールします。

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

2. 以下の内容で **/etc/wireguard/wg0.conf** ファイルを作成します。

```
[Interface]
Address = 192.0.2.1/24, 2001:db8:1::1/32
ListenPort = 51820
PrivateKey = YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=

[Peer]
PublicKey = bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
AllowedIPs = 192.0.2.2, 2001:db8:1::2
```

- **[Interface]** セクションでは、サーバー上のインターフェイスの WireGuard 設定を説明します。
  - **Address**:サーバーのトンネル IP アドレスをコンマで区切ったリストです。
  - **PrivateKey**:サーバーの秘密鍵。
  - **ListenPort**:WireGuard が着信 UDP 接続をリッスンするポートです。
    着信 WireGuard 接続を受信するホストでは、常に固定ポート番号を設定してください。ポートを設定しないと、**wg0** インターフェイスをアクティブにするたびにランダムな空きポートが使用されます。
- 各 **[Peer]** セクションでは、1 台のクライアントの設定を説明します。
  - **PublicKey**:クライアントの公開鍵。
  - **AllowedIPs**:このサーバーにデータを送信できるクライアントのトンネル IP アドレスです。

3. WireGuard 接続を有効にして起動します。

```
# systemctl enable --now wg-quick@wg0
```

systemd インスタンス名は、**/etc/wireguard/** ディレクトリーの設定ファイル名に、**.conf** の接尾辞を付けずに、同じ名前にする必要があります。このサービスは、仮想ネットワークインターフェイスにもこの名前を使用します。

次のステップ

- WireGuard サーバーで firewalld サービスを設定します。

検証

1. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

2. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**
- **wg-quick(8)** の man ページ

8.9. コマンドラインを使用した WIREGUARD サーバーでの FIREWALLD の設定

クライアントからの着信接続を許可するには、WireGuard サーバーで **firewalld** サービスを設定する必要があります。また、クライアントが WireGuard サーバーをデフォルトゲートウェイとして使用し、すべてのトラフィックをトンネル経由でルーティングできるようにするには、マスカレードを有効にする必要があります。

手順

1. **firewalld** サービスで着信接続用の WireGuard ポートを開きます。

```
# firewall-cmd --permanent --add-port=51820/udp --zone=public
```

2. クライアントがすべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用する場合は、**public** ゾーンのマスカレードを有効にします。

```
# firewall-cmd --permanent --zone=public --add-masquerade
```

3. **firewalld** ルールを再読み込みします。

```
# firewall-cmd --reload
```

検証

- **public** ゾーンの設定を表示します。

```
# firewall-cmd --list-all
public (active)
  ...
  ports: 51820/udp
  masquerade: yes
  ...
```

関連情報

- **firewall-cmd(1)** の man ページ

8.10. グラフィカルインターフェイスを使用した WIREGUARD サーバーでの FIREWALLD の設定

クライアントからの着信接続を許可するには、WireGuard サーバーで **firewalld** サービスを設定する必要があります。また、クライアントが WireGuard サーバーをデフォルトゲートウェイとして使用し、すべてのトラフィックをトンネル経由でルーティングできるようにするには、マスカレードを有効にする必要があります。

手順

1. **Super** キーを押して、**firewall** を入力し、結果から **Firewall** アプリケーションを選択します。
2. **Configuration** リストで **Permanent** を選択します。
3. **public** ゾーンを選択します。
4. WireGuard ポートへの着信接続を許可します。
   a. **Ports** タブで、**追加** をクリックします。
   b. 着信 WireGuard 接続に設定したポート番号を入力します。
   c. **Protocol** リストから **udp** を選択します。
   d. **OK** をクリックします。
5. クライアントがすべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用する場合は、次のコマンドを実行します。
   a. **public** ゾーンの **Masquerading** タブに移動します。
   b. **Masquerade zone** を選択します。
6. **オプション → Firewalld の再読み込み** を選択します。

検証

- **public** ゾーンの設定を表示します。

```
# firewall-cmd --list-all
public (active)
  ...
  ports: 51820/udp
  masquerade: yes
  ...
```

8.11. NMCLI を使用した WIREGUARD クライアントの設定

NetworkManager で接続プロファイルを作成することで、WireGuard クライアントを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

この手順では、次の設定を前提としています。

- クライアント:
  - 秘密鍵: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**
- サーバー
  - 公開鍵:**UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
  - トンネル IPv4 アドレス:**192.0.2.1/24**
  - トンネル IPv6 アドレス:**2001:db8:1::1/32**

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - クライアントの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - サーバーの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. NetworkManager WireGuard 接続プロファイルを追加します。

```
# nmcli connection add type wireguard con-name client-wg0 ifname wg0 autoconnect no
```

client-wg0 という名前のプロファイルを作成し、そのプロファイルに仮想インターフェイス **wg0** を割り当てます。設定を確定せずに接続を追加した後、接続が自動的に開始しないようにするには、**autoconnect** パラメーターを無効にします。

2. オプション: NetworkManager が **client-wg** 接続を自動的に起動しないように設定します。

```
# nmcli connection modify client-wg0 autoconnect no
```

3. クライアントのトンネル IPv4 アドレスとサブネットマスクを設定します。

```
# nmcli connection modify client-wg0 ipv4.method manual ipv4.addresses 192.0.2.2/24
```

4. クライアントのトンネル IPv6 アドレスとサブネットマスクを設定します。

```
# nmcli connection modify client-wg0 ipv6.method manual ipv6.addresses
2001:db8:1::2/32
```

5. すべてのトラフィックをトンネル経由でルーティングする場合は、サーバーのトンネル IP アドレスをデフォルトゲートウェイとして設定します。

```
# nmcli connection modify client-wg0 ipv4.gateway 192.0.2.1 ipv6.gateway
2001:db8:1::1
```

すべてのトラフィックをトンネル経由でルーティングするには、後の手順で、このクライアントの **allowed-ips** を **0.0.0.0/0;::/0** に設定する必要があります。

すべてのトラフィックをトンネル経由でルーティングすると、サーバーのルーティングとファイアウォールの設定によっては他のホストへの接続に影響が及ぶ可能性があることに注意してください。

6. クライアントの秘密鍵を接続プロファイルに追加します。

```
# nmcli connection modify client-wg0 wireguard.private-key
"aPUcp5vHz8yMLrzk8SsDyYnV33IhE/k20e52iKJFV0A="
```

7. このクライアントとの通信を許可するサーバーごとにピア設定を追加します。この設定は手動で追加する必要があります。**nmcli** ユーティリティーでは、対応する接続プロパティーの設定に対応していないためです。

   a. **/etc/NetworkManager/system-connections/client-wg0.nmconnection** ファイルーを編集し、以下を追加します。

```
[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>]** エントリーは、サーバーのピアセクションを定義します。セクション名には、サーバーの公開鍵が含まれます。
   - **endpoint** パラメーターは、サーバーのホスト名または IP アドレスとポートを設定します。クライアントはこの情報を使用して接続を確立します。
   - **allowed-ips** パラメーターは、このクライアントにデータを送信できる IP アドレスのリストを設定します。たとえば、このパラメーターを次のように設定します。
     - サーバーのみがこのクライアントと通信できるようにするサーバーのトンネル IP アドレス。上記の例の値は、このシナリオを設定します。

- リモートの IPv4 アドレスおよび IPv6 アドレスが、このクライアントと通信できるように許可する **0.0.0.0/0;::/0;**この設定を使用して、すべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用します。

- オプションの **persistent-keepalive** パラメーターは、WireGuard がサーバーにキープアライブパケットを送信する間隔を秒単位で定義します。ネットワークアドレス変換 (NAT) を使用するネットワークでクライアントを使用する場合、またはしばらく非アクティブにした後にファイアウォールが UDP 接続を閉じる場合は、このパラメーターを設定します。

b. **client-wg0** 接続プロファイルを再読み込みします。

```
# nmcli connection load /etc/NetworkManager/system-connections/client-wg0.nmconnection
```

8. **client-wg0** 接続を再アクティブ化します。

```
# nmcli connection up client-wg0
```

検証

1. サーバーの IP アドレスの ping を実行します。

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

VPN トンネルを介してトラフィックを送信している場合は、**latest handshake** エントリーと **transfer** エントリーのみが含まれることに注意してください。

3. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**
- **nm-settings(5)** の man ページの **WireGuard setting** セクション

8.12. NMTUI を使用した WIREGUARD クライアントの設定

NetworkManager で接続プロファイルを作成することで、WireGuard クライアントを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

この手順では、次の設定を前提としています。

- クライアント:
  - 秘密鍵: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**
- サーバー
  - 公開鍵:**UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
  - トンネル IPv4 アドレス:**192.0.2.1/24**
  - トンネル IPv6 アドレス:**2001:db8:1::1/32**

前提条件

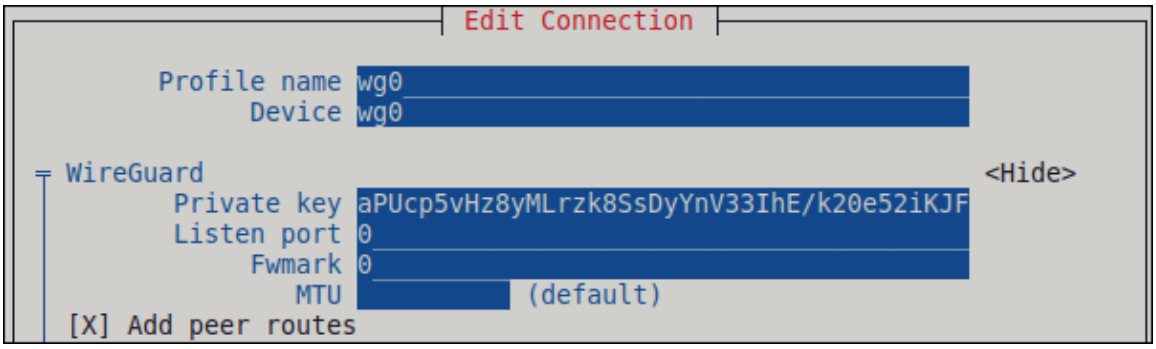
- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - クライアントの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - サーバーの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク
- **NetworkManager-tui** パッケージをインストールしました

手順

1. **nmtui** アプリケーションを開始します。

```
# nmtui
```

2. **Edit a connection** 選択し、**Enter** を押します。
3. **Add** を選択し、**Enter** を押します。
4. リストから **WireGuard** 接続タイプを選択し、**Enter** を押します。
5. **Edit connection** ウィンドウで:
   a. NetworkManager が接続に割り当てる仮想インターフェイス (**wg0** など) の接続名を入力します。
   b. クライアントの秘密鍵を入力します。



   c. **Peers** ペインの横にある **Add** をクリックします。
      i. サーバーの公開鍵を入力します。
      ii. **Allowed IPs** フィールドを設定します。たとえば、次のように設定します。
         - サーバーのみがこのクライアントと通信できるようにするサーバーのトンネル IP アドレス。
         - リモートの IPv4 アドレスおよび IPv6 アドレスが、このクライアントと通信できるように許可する **0.0.0.0/0,::/0**この設定を使用して、すべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用します。
      iii. **Endpoint** フィールドに、WireGuard サーバーのホスト名または IP アドレスとポートを入力します。**hostname_or_IP:port_number** の形式を使用します。
      iv. オプション: ネットワークアドレス変換 (NAT) を使用するネットワークでクライアントを使用するか、しばらく非アクティブにした後にファイアウォールが UDP 接続を閉じる場合は、永続的なキープアライブの間隔を秒単位で設定します。この間隔で、クライアントは、キープアライブパケットをサーバーに送信します。
      v. **OK** を選択し、**Enter** を押します。

d. **IPv4 Configuration** の横にある **Show** を選択し、**Enter** を押します。

   i. IPv4 設定方法 **Manual** を選択します。

   ii. トンネルの IPv4 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。

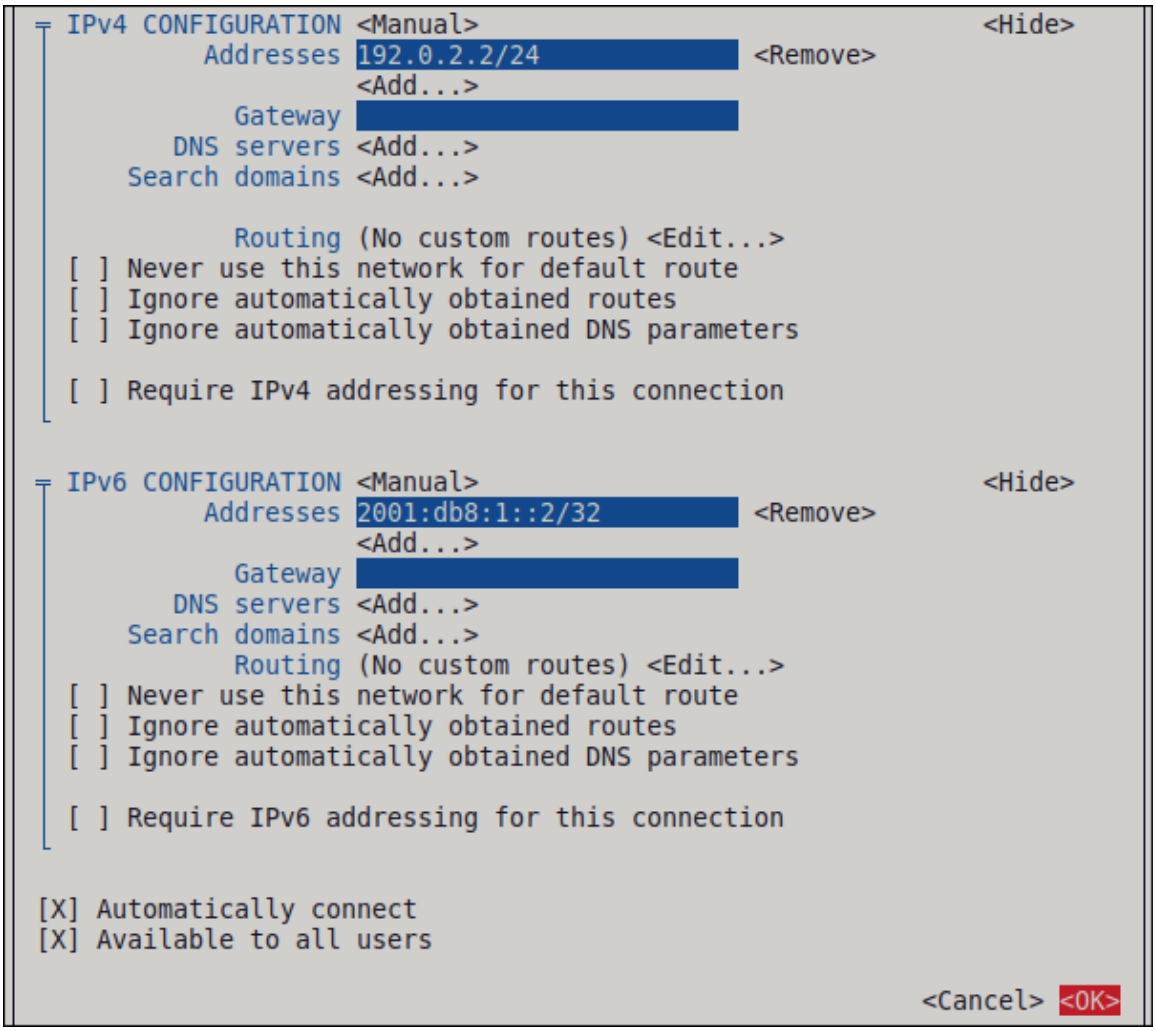
e. **IPv6 Configuration** の横にある **Show** を選択し、**Enter** を押します。

   i. IPv6 設定方法 **Manual** を選択します。

   ii. トンネルの IPv6 アドレスとサブネットマスクを入力します。**Gateway** フィールドは空のままにします。

f. オプション: **Automatically connect** を選択します。

g. **OK** を選択し、**Enter** を押します



6. 接続のリストが表示されたウィンドウで、**Back** を選択し、**Enter** を押します。
7. **NetworkManager TUI** のメインウィンドウで、**Quit** を選択し、**Enter** を押します。

検証

1. サーバーの IP アドレスの ping を実行します。

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

VPN トンネルを介してトラフィックを送信している場合は、**latest handshake** エントリーと **transfer** エントリーのみが含まれることに注意してください。

3. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**

8.13. NM-CONNECTION-EDITOR を使用した WIREGUARD クライアントの設定

NetworkManager で接続プロファイルを作成することで、WireGuard クライアントを設定できます。この方法を使用して、NetworkManager に WireGuard 接続を管理させます。

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - クライアントの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - サーバーの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. ターミナルを開き、次のコマンドを入力します。

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

2. **+** ボタンをクリックして、新しいコネクションを追加します。
3. **WireGuard** の接続の種類を選択し、**作成** をクリックします。
4. オプション: 接続名を更新します。
5. オプション: **General** タブで、**Connect automatically with priority** を選択します。
6. **WireGuard** タブで、以下を行います。
   a. NetworkManager が接続に割り当てる仮想インターフェイス (**wg0** など) の名前を入力します。
   b. クライアントの秘密鍵を入力します。
   c. **追加** を選択して、ピアを追加します。
      i. サーバーの公開鍵を入力します。
      ii. **Allowed IPs** フィールドを設定します。たとえば、次のように設定します。
         - サーバーのみがこのクライアントと通信できるようにするサーバーのトンネル IP アドレス。
         - リモートの IPv4 アドレスおよび IPv6 アドレスが、このクライアントと通信できるように許可する **0.0.0.0/0;::/0;**この設定を使用して、すべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用します。
           すべてのトラフィックをトンネル経由でルーティングすると、サーバーのルーティングとファイアウォールの設定によっては他のホストへの接続に影響が及ぶ可能性があることに注意してください。
      iii. **Endpoint** フィールドに、WireGuard サーバーのホスト名または IP アドレスとポートを入力します。**hostname_or_IP:port_number** の形式を使用します。
      iv. オプション: ネットワークアドレス変換 (NAT) を使用するネットワークでクライアントを使用するか、しばらく非アクティブにした後にファイアウォールが UDP 接続を閉じ

る場合は、永続的なキープアライブの間隔を秒単位で設定します。この間隔で、クライアントは、キープアライブパケットをサーバーに送信します。

v. **Apply** をクリックします。

7. **IPv4 Settings** タブで、以下を行います。

a. **Method** リストで **Manual** を選択します。

b. **追加** を選択して、トンネルの IPv4 アドレスとサブネットマスクを入力します。

c. すべてのトラフィックをトンネル経由でルーティングする場合は、サーバーのトンネル IPv4 アドレスを **Gateway** フィールドに設定します。それ以外の場合は、フィールドを空のままにします。
すべての IPv4 トラフィックをトンネル経由でルーティングするには、このクライアントの **Allowed IPs** フィールドに **0.0.0.0/0** を含める必要があります。

8. **IPv6 Settings** タブで、以下を行います。

a. **Method** リストで **Manual** を選択します。

b. **追加** を選択して、トンネル IPv6 アドレスとサブネットマスクを入力します。

c. すべてのトラフィックをトンネル経由でルーティングする場合は、**Gateway** フィールドに、サーバーのトンネル IPv6 アドレスを設定します。それ以外の場合は、フィールドを空のままにします。
すべての IPv4 トラフィックをトンネル経由でルーティングするには、このクライアントの **Allowed IPs** フィールドに **::/0** を含める必要があります。

9. **保存** を選択して、接続プロファイルを保存します。

検証

1. サーバーの IP アドレスの ping を実行します。

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

VPN トンネルを介してトラフィックを送信している場合は、**latest handshake** エントリーと **transfer** エントリーのみが含まれることに注意してください。

3. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**

8.14. WG-QUICK サービスを使用した WIREGUARD クライアントの設定

/etc/wireguard/ ディレクトリーに設定ファイルを作成することで、WireGuard クライアントを設定できます。この方法を使用して、NetworkManager からサービスを独立して設定します。

この手順では、次の設定を前提としています。

- クライアント:
  - 秘密鍵: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
  - トンネル IPv4 アドレス:**192.0.2.2/24**
  - トンネル IPv6 アドレス:**2001:db8:1::2/32**
- サーバー
  - 公開鍵:**UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
  - トンネル IPv4 アドレス:**192.0.2.1/24**
  - トンネル IPv6 アドレス:**2001:db8:1::1/32**

前提条件

- サーバーとクライアントの両方に公開鍵と秘密鍵を生成している。
- 以下の情報を把握している。
  - クライアントの秘密鍵
  - クライアントの静的トンネルの IP アドレスとサブネットマスク
  - サーバーの公開鍵
  - サーバーの静的トンネル IP アドレスおよびサブネットマスク

手順

1. **wireguard-tools** パッケージをインストールします。

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

2. 以下の内容で **/etc/wireguard/wg0.conf** ファイルを作成します。

```
[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]** セクションでは、クライアントのインターフェイスの WireGuard 設定を説明します。
  - **Address**:クライアントのトンネル IP アドレスをコンマで区切ったリストです。
  - **PrivateKey**:クライアントの秘密鍵。
- **[Peer]** セクションでは、サーバーの設定を説明します。
  - **PublicKey**:サーバーの公開鍵。
  - **AllowedIPs**:このクライアントにデータを送信できる IP アドレス。たとえば、このパラメーターを次のように設定します。
    - サーバーのみがこのクライアントと通信できるようにするサーバーのトンネル IP アドレス。上記の例の値は、このシナリオを設定します。
    - リモートの IPv4 アドレスおよび IPv6 アドレスが、このクライアントと通信できるように許可する **0.0.0.0/0, ::/0**この設定を使用して、すべてのトラフィックをトンネル経由でルーティングし、WireGuard サーバーをデフォルトゲートウェイとして使用します。
  - **Endpoint**:サーバーのホスト名または IP アドレスとポートを設定します。クライアントはこの情報を使用して接続を確立します。
  - オプションの **persistent-keepalive** パラメーターは、WireGuard がサーバーにキープアライブパケットを送信する間隔を秒単位で定義します。ネットワークアドレス変換 (NAT) を使用するネットワークでクライアントを使用する場合、またはしばらく非アクティブにした後にファイアウォールが UDP 接続を閉じる場合は、このパラメーターを設定します。

3. WireGuard 接続を有効にして起動します。

```
# systemctl enable --now wg-quick@wg0
```

systemd インスタンス名は、**/etc/wireguard/** ディレクトリーの設定ファイル名に、**.conf** の接尾辞を付けずに、同じ名前にする必要があります。このサービスは、仮想ネットワークインターフェイスにもこの名前を使用します。

検証

1. サーバーの IP アドレスの ping を実行します。

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. **wg0** デバイスのインターフェイス設定を表示します。

```
# 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
```

出力で秘密鍵を表示するには、**WG_HIDE_KEYS=never wg show wg0** コマンドを使用します。

VPN トンネルを介してトラフィックを送信している場合は、**latest handshake** エントリーと **transfer** エントリーのみが含まれることに注意してください。

3. **wg0** デバイスの IP 設定を表示します。

```
# 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
```

関連情報

- man ページの **wg(8)**
- **wg-quick(8)** の man ページ

第9章 IP トンネルの設定

VPN と同様に、IP トンネルは、インターネットなどの 3 番目のネットワークを介して 2 つのネットワークを直接接続します。ただし、すべてのトンネルプロトコルが暗号化に対応しているわけではありません。

トンネルを確立する両方のネットワークのルーターには、最低でも 2 つのインターフェイスが必要です。

- ローカルネットワークに接続されているインターフェイス 1 つ
- トンネルが確立されたネットワークに接続されたインターフェイス 1 つ。

トンネルを確立するには、リモートサブネットから IP アドレスを使用して、両方のルーターに仮想インターフェイスを作成します。

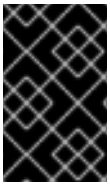
NetworkManager は、以下の IP トンネルに対応します。

- GRE (Generic Routing Encapsulation)
- IP6GRE (Generic Routing Encapsulation over IPv6)
- GRETAP (Generic Routing Encapsulation Terminal Access Point)
- IP6GRETAP (Generic Routing Encapsulation Terminal Access Point over IPv6)
- IPIP (IPv4 over IPv4)
- IPIP6 (IPv4 over IPv6)
- IP6IP6 (IPv6 over IPv6)
- SIT (Simple Internet Transition)

このトンネルは、タイプに応じて、OSI (Open Systems Interconnection) モデルのレイヤー 2 または 3 で動作します。

9.1. NMCLI を使用して IPIP トンネルを設定して、IPV4 パケットの IPV4 トラフィックをカプセル化します。

IPIP (IP over IP) トンネルは OSI レイヤー 3 で動作し、RFC 2003 で説明されているように IPv4 パケットの IPv4 トラフィックをカプセル化します。

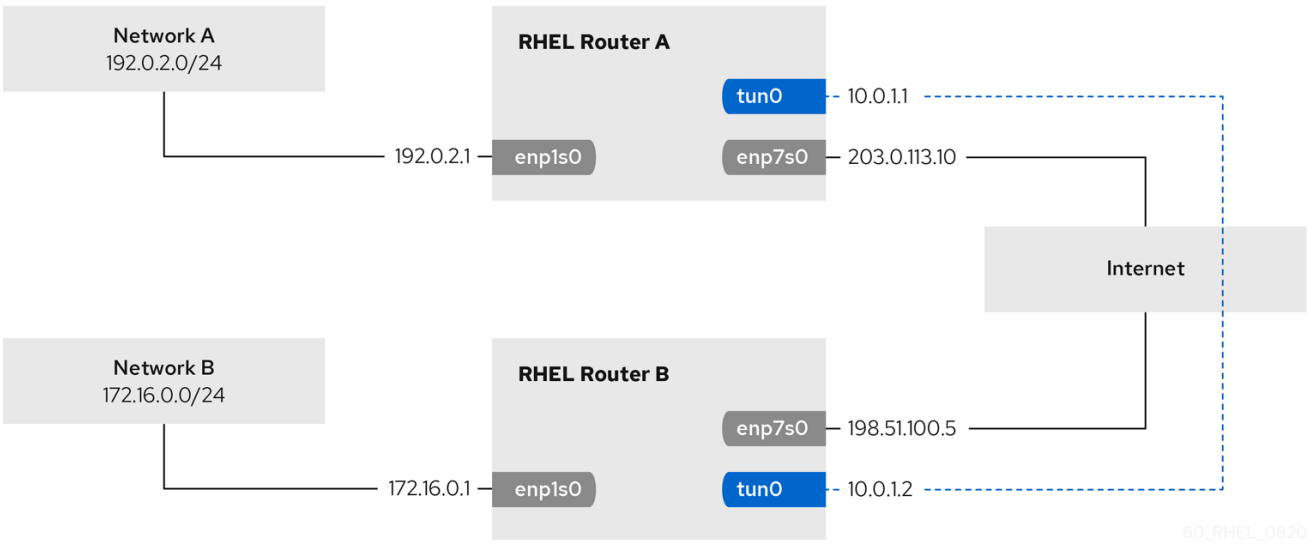


重要

IPIP トンネルを介して送信されるデータは暗号化されません。セキュリティー上の理由から、すでに暗号化されたデータにはトンネルを使用してください (HTTPS などの他のプロトコル)。

IPIP トンネルはユニキャストパケットのみをサポートすることに注意してください。マルチキャストをサポートする IPv4 トンネルが必要な場合は、nmcli を使用した GRE トンネルを設定して IPv4 パケット内のレイヤー 3 トラフィックをカプセル化 を参照します。

たとえば、以下の図に示すように、2 つの RHEL ルーター間で IPIP トンネルを作成し、インターネット経由で 2 つの内部サブネットに接続できます。



前提条件

- 各 RHEL ルーターには、ローカルサブネットに接続されているネットワークインターフェイスがあります。
- 各 RHEL ルーターには、インターネットに接続しているネットワークインターフェイスがあります。
- トンネル経由で送信するトラフィックは IPv4 ユニキャストです。

手順

1. ネットワーク A の RHEL ルーターで、次のコマンドを実行します。
   a. **tun0** という名前の IPIP トンネルインターフェイスを作成します。

   ```
   # 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 パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

   b. IPv4 アドレスを **tun0** デバイスに設定します。

   ```
   # nmcli connection modify tun0 ipv4.addresses '10.0.1.1/30'
   ```

   トンネルには、2 つの使用可能な IP アドレスを持つ **/30** サブネットで十分であることに注意してください。

   c. IPv 4 設定を使用するように手動で **tun0** 接続を設定します。

   ```
   # nmcli connection modify tun0 ipv4.method manual
   ```

   d. トラフィックを **172.16.0.0/24** ネットワークにルーティングする静的ルートをルーター B のトンネル IP に追加します。

   ```
   # nmcli connection modify tun0 +ipv4.routes "172.16.0.0/24 10.0.1.2"
   ```

e. **tun0** 接続を有効にします。

```
# nmcli connection up tun0
```

f. パケット転送を有効にします。

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. ネットワーク B の RHEL ルーターで、次のコマンドを実行します。

a. **tun0** という名前の IPIP トンネルインターフェイスを作成します。

```
# 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 パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

b. IPv4 アドレスを **tun0** デバイスに設定します。

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.2/30'
```

c. IPv 4 設定を使用するように手動で **tun0** 接続を設定します。

```
# nmcli connection modify tun0 ipv4.method manual
```

d. トラフィックを **192.0.2.0/24** ネットワークにルーティングする静的ルートをルーター A のトンネル IP に追加します。

```
# nmcli connection modify tun0 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

e. **tun0** 接続を有効にします。

```
# nmcli connection up tun0
```

f. パケット転送を有効にします。

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

検証

- 各 RHEL ルーターから、他のルーターの内部インターフェイスの IP アドレスに ping します。

a. ルーター A で **172.16.0.1** に ping します。

```
# ping 172.16.0.1
```

b. ルーター B で **192.0.2.1** に ping します。

```
# ping 192.0.2.1
```

関連情報

- **nmcli(1)** man ページ
- **nm-settings(5)** man ページ

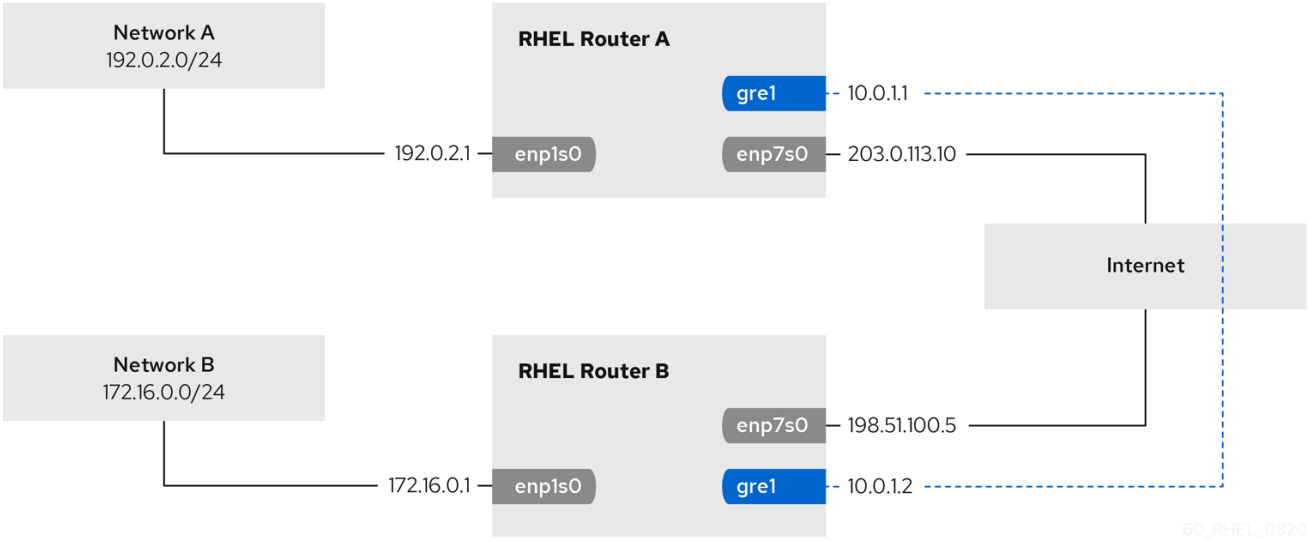
9.2. NMCLI を使用して GRE トンネルを設定して、IPV4 パケット内のレイヤー 3 トラフィックをカプセル化

Generic Routing Encapsulation (GRE) トンネルは、RFC 2784 で説明されているように、IPv4 パケットでレイヤー 3 トラフィックをカプセル化します。GRE トンネルは、有効なイーサネットタイプで任意のレイヤー 3 プロトコルをカプセル化できます。

重要

GRE トンネルを介して送信されるデータは暗号化されません。セキュリティー上の理由から、すでに暗号化されたデータにはトンネルを使用してください (HTTPS などの他のプロトコル)。

たとえば、以下の図に示すように、2 つの RHEL ルーター間で GRE トンネルを作成し、インターネット経由で 2 つの内部サブネットに接続できます。



注記

gre0 デバイス名は予約されています。デバイスに **gre1** または別の名前を使用します。

前提条件

- 各 RHEL ルーターには、ローカルサブネットに接続されているネットワークインターフェイスがあります。
- 各 RHEL ルーターには、インターネットに接続しているネットワークインターフェイスがあります。

手順

1. ネットワーク A の RHEL ルーターで、次のコマンドを実行します。

   a. **gre1** という名前の GRE トンネルインターフェイスを作成します。

   ```
   # 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 パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

   b. IPv4 アドレスを **gre1** デバイスに設定します。

   ```
   # nmcli connection modify gre1 ipv4.addresses '10.0.1.1/30'
   ```

   トンネルには、2 つの使用可能な IP アドレスを持つ **/30** サブネットで十分であることに注意してください。

   c. 手動の IPv 4 設定を使用するように **gre1** 接続を設定します。

   ```
   # nmcli connection modify gre1 ipv4.method manual
   ```

   d. トラフィックを **172.16.0.0/24** ネットワークにルーティングする静的ルートをルーター B のトンネル IP に追加します。

   ```
   # nmcli connection modify gre1 +ipv4.routes "172.16.0.0/24 10.0.1.2"
   ```

   e. **gre1** コネクションを有効にします。

   ```
   # nmcli connection up gre1
   ```

   f. パケット転送を有効にします。

   ```
   # echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
   # sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
   ```

2. ネットワーク B の RHEL ルーターで、次のコマンドを実行します。

   a. **gre1** という名前の GRE トンネルインターフェイスを作成します。

   ```
   # 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 パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

   b. IPv4 アドレスを **gre1** デバイスに設定します。

   ```
   # nmcli connection modify gre1 ipv4.addresses '10.0.1.2/30'
   ```

   c. 手動の IPv 4 設定を使用するように **gre1** 接続を設定します。

   ```
   # nmcli connection modify gre1 ipv4.method manual
   ```

d. トラフィックを **192.0.2.0/24** ネットワークにルーティングする静的ルートをルーター A のトンネル IP に追加します。

```
# nmcli connection modify gre1 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

e. **gre1** コネクションを有効にします。

```
# nmcli connection up gre1
```

f. パケット転送を有効にします。

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

検証

1. 各 RHEL ルーターから、他のルーターの内部インターフェイスの IP アドレスに ping します。

a. ルーター A で **172.16.0.1** に ping します。

```
# ping 172.16.0.1
```

b. ルーター B で **192.0.2.1** に ping します。

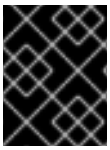
```
# ping 192.0.2.1
```

関連情報

- **nmcli(1)** man ページ
- **nm-settings(5)** man ページ

9.3. IPV4 でイーサネットフレームを転送するための GRETAP トンネルの設定

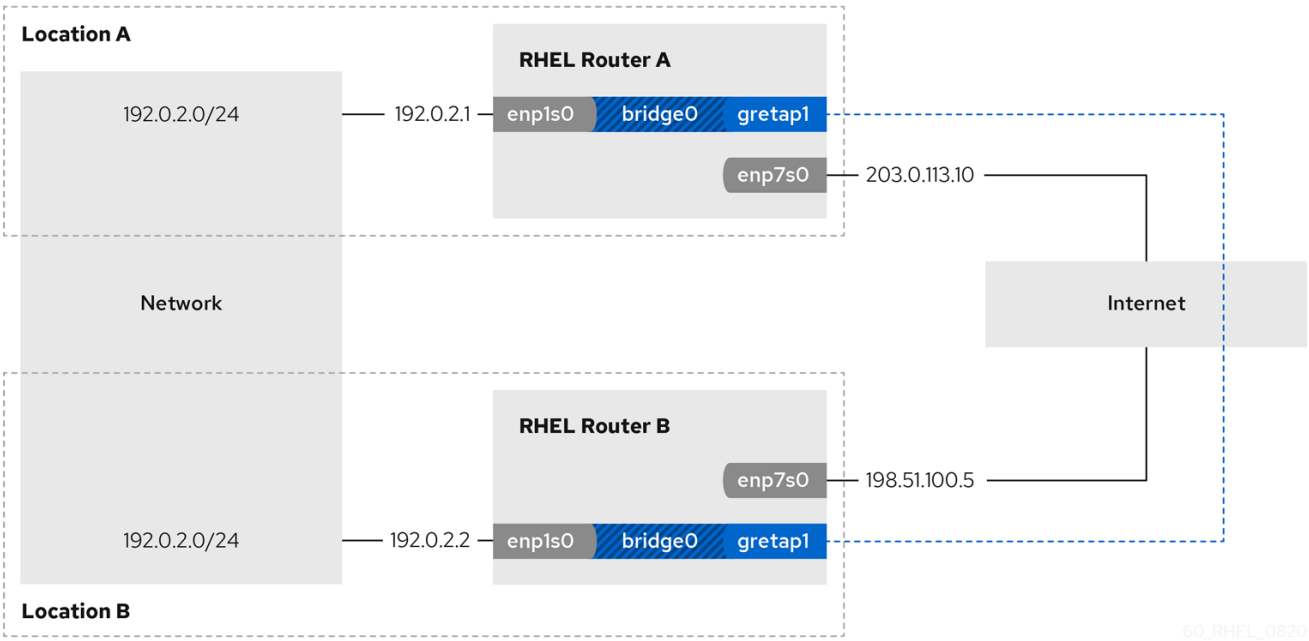
GRETAP (Generic Routing Encapsulation Terminal Access Point) トンネルは OSI レベル 2 で動作し、RFC 2784 で説明されているように IPv4 パケットのイーサネットトラフィックをカプセル化します。

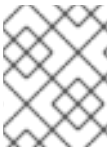


重要

GRETAP トンネルを介して送信されるデータは暗号化されません。セキュリティー上の理由から、VPN または別の暗号化された接続にトンネルを確立します。

たとえば、以下の図に示すように、2 つの RHEL ルーター間で GRETAP トンネルを作成し、ブリッジを使用して 2 つのネットワークに接続します。





注記

gretap0 デバイス名が予約されています。デバイスに **gretap1** または別の名前を使用します。

前提条件

- 各 RHEL ルーターには、ローカルネットワークに接続されたネットワークインターフェイスがあり、IP 設定は割り当てられません。
- 各 RHEL ルーターには、インターネットに接続しているネットワークインターフェイスがあります。

手順

1. ネットワーク A の RHEL ルーターで、次のコマンドを実行します。
   a. **bridge0** という名前のブリッジインターフェイスを作成します。

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

   b. ブリッジの IP 設定を設定します。

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify bridge0 ipv4.method manual
```

   c. ローカルネットワークに接続されたインターフェイス用の新しい接続プロファイルをブリッジに追加します。

```
# nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
ifname enp1s0 master bridge0
```

   d. GRETAP トンネルインターフェイスの新しい接続プロファイルをブリッジに追加します。

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

remote パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

e. オプション: STP (Spanning Tree Protocol) を無効にする必要がない場合は、これを無効にします。

```
# nmcli connection modify bridge0 bridge.stp no
```

デフォルトでは、STP は有効になり、接続を使用する前に遅延が生じます。

f. **bridge0** 接続がアクティベートするように、ブリッジのポートが自動的にアクティブになるようにします。

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

g. **bridge0** 接続をアクティブにします。

```
# nmcli connection up bridge0
```

2. ネットワーク B の RHEL ルーターで、次のコマンドを実行します。

a. **bridge0** という名前のブリッジインターフェイスを作成します。

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

b. ブリッジの IP 設定を設定します。

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.2/24'
# nmcli connection modify bridge0 ipv4.method manual
```

c. ローカルネットワークに接続されたインターフェイス用の新しい接続プロファイルをブリッジに追加します。

```
# nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
ifname enp1s0 master bridge0
```

d. GRETAP トンネルインターフェイスの新しい接続プロファイルをブリッジに追加します。

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

remote パラメーターおよび **local** パラメーターは、リモートルーターおよびローカルルーターのパブリック IP アドレスを設定します。

e. オプション: STP (Spanning Tree Protocol) を無効にする必要がない場合は、これを無効にします。

```
# nmcli connection modify bridge0 bridge.stp no
```

f. **bridge0** 接続がアクティベートするように、ブリッジのポートが自動的にアクティブになるようにします。

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

g. **bridge0** 接続をアクティブにします。

```
# nmcli connection up bridge0
```

検証

1. 両方のルーターで、**enp1s0** 接続および **gretap1** 接続が接続され、**CONNECTION** 列にポートの接続名が表示されていることを確認します。

```
# nmcli device
nmcli device
DEVICE   TYPE      STATE      CONNECTION
...
bridge0  bridge    connected  bridge0
enp1s0   ethernet  connected  bridge0-port1
gretap1  iptunnel  connected  bridge0-port2
```

2. 各 RHEL ルーターから、他のルーターの内部インターフェイスの IP アドレスに ping します。

a. ルーター A で **192.0.2.2** に ping します。

```
# ping 192.0.2.2
```

b. ルーター B で **192.0.2.1** に ping します。

```
# ping 192.0.2.1
```

関連情報

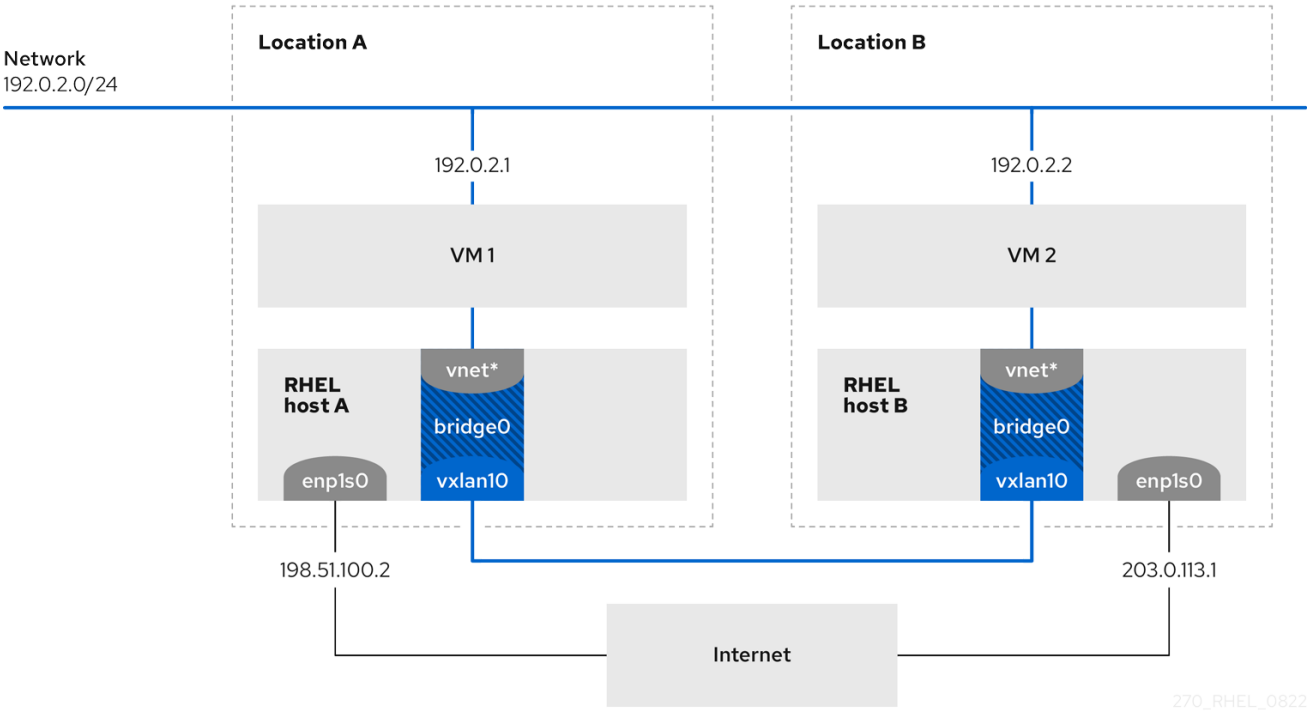
- **nmcli(1)** man ページ
- **nm-settings(5)** man ページ

9.4. 関連情報

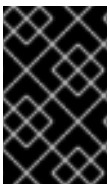
- **ip-link(8)** man ページ

第10章 VXLAN を使用した仮想マシンの仮想レイヤー 2 ドメインの作成

仮想拡張可能な LAN (VXLAN) は、UDP プロトコルを使用して IP ネットワーク経由でレイヤー 2 トラフィックをトンネルするネットワークプロトコルです。たとえば、別のホストで実行している特定の仮想マシンは、VXLAN トンネルを介して通信できます。ホストは、世界中の異なるサブネットやデータセンターに存在できます。仮想マシンの視点からは、同じ VXLAN 内のその他の仮想マシンは、同じレイヤー 2 ドメイン内にあります。



この例では、RHEL-host-A と RHEL-host-B は、ブリッジである **br0** を使用して、VXLAN 名が **vxlan10** である各ホストの仮想マシンの仮想ネットワークを接続します。この設定により、VXLAN は仮想マシンには表示されなくなり、仮想マシンに特別な設定は必要ありません。その後、別の仮想マシンを同じ仮想ネットワークに接続すると、仮想マシンは自動的に同じ仮想レイヤー 2 ドメインのメンバーになります。



重要

通常のレイヤー 2 トラフィックと同様、VXLAN のデータは暗号化されません。セキュリティー上の理由から、VPN 経由で VXLAN を使用するか、その他のタイプの暗号化接続を使用します。

10.1. VXLAN の利点

仮想拡張可能 LAN (VXLAN) の主な利点は、以下のとおりです。

- VXLAN は 24 ビット ID を使用します。そのため、最大 16,777,216 の分離されたネットワークを作成できます。たとえば、仮想 LAN (VLAN) は 4,096 の分離されたネットワークのみをサポートします。
- VXLAN は IP プロトコルを使用します。これにより、トラフィックをルーティングし、仮想的に実行するシステムを、同じレイヤー 2 ドメイン内の異なるネットワークと場所に置くことができます。

- ほとんどのトンネルプロトコルとは異なり、VXLAN はポイントツーポイントネットワークだけではありません。VXLAN は、他のエンドポイントの IP アドレスを動的に学習するか、静的に設定された転送エントリーを使用できます。
- 特定のネットワークカードは、UDP トンネル関連のオフロード機能に対応します。

関連情報

- **kernel-doc** パッケージにより提供されている **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vxlan.rst**

10.2. ホストでのイーサネットインターフェイスの設定

RHEL 仮想マシンホストをイーサネットに接続するには、ネットワーク接続プロファイルを作成し、IP 設定を設定して、プロファイルをアクティブにします。

両方の RHEL ホストでこの手順を実行し、IP アドレス設定を調整します。

前提条件

- ホストがイーサネットに接続されている。

手順

1. NetworkManager に新しいイーサネット接続プロファイルを追加します。

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

2. IPv4 を設定します。

```
# 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
```

   ネットワークが DHCP を使用する場合は、この手順をスキップします。

3. **Example** コネクションをアクティブにします。

```
# nmcli connection up Example
```

検証

1. デバイスおよび接続の状態を表示します。

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

2. リモートネットワークでホストに ping を実行して、IP 設定を確認します。

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

そのホストでネットワークを設定する前に、その他の仮想マシンホストに ping を実行することはできないことに注意してください。

関連情報

- **nm-settings(5)** man ページ

10.3. VXLAN が接続されたネットワークブリッジの作成

仮想拡張可能な LAN (VXLAN) を仮想マシンに表示しないようにするには、ホストでブリッジを作成し、VXLAN をブリッジに割り当てます。NetworkManager を使用して、ブリッジと VXLAN の両方を作成します。仮想マシンのトラフィックアクセスポイント (TAP) デバイス (通常はホスト上の **vnet***) をブリッジに追加することはありません。**libvirtd** は、仮想マシンの起動時に動的に追加します。

両方の RHEL ホストでこの手順を実行し、必要に応じて IP アドレスを調整します。

手順

1. ブリッジ **br0** を作成します。

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

このコマンドは、ブリッジデバイスに IPv4 アドレスおよび IPv6 アドレスを設定しません。これは、このブリッジがレイヤー 2 で機能するためです。

2. VXLAN インターフェイスを作成し、**br0** に割り当てます。

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

このコマンドは、次の設定を使用します。

- **id 10**:VXLAN 識別子を設定します。
- **local 198.51.100.2**:送信パケットの送信元 IP アドレスを設定します。
- **remote 203.0.113.1**:VXLAN デバイスフォワーディングデータベースで宛先リンク層アドレスが不明な場合に、送信パケットで使用するユニキャストまたはマルチキャストの IP アドレスを設定します。
- **master br0**:この VXLAN 接続を、**br0** 接続のポートとして作成するように設定します。
- **ipv4.method disabled** および **ipv6.method disabled**:ブリッジで IPv4 および IPv6 を無効にします。

初期設定では、NetworkManager は **8472** を宛先ポートとして使用します。宛先ポートが異なる場合は、追加で、**destination-port <port_number>** オプションをコマンドに渡します。

3. **br0** 接続プロファイルを有効にします。

```
# nmcli connection up br0
```

4. ローカルファイアウォールで、着信 UDP 接続用にポート **8472** を開くには、次のコマンドを実行します。

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

検証

- 転送テーブルを表示します。

```
# 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
...
```

関連情報

- **nm-settings(5)** man ページ

10.4. 既存のブリッジを使用した LIBVIRT での仮想ネットワークの作成

仮想マシンが、接続した仮想拡張可能 LAN (VXLAN) で **br0** ブリッジを使用できるようにするには、最初に、このブリッジを使用する **libvirtd** サービスに仮想ネットワークを追加します。

前提条件

- **libvirt** をインストールしている。
- **libvirtd** を起動して有効にしている。
- RHEL 上の VXLAN で **br0** デバイスーを設定している。

手順

1. 以下の内容で **~/vxlan10-bridge.xml** を作成します。

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

2. **~/vxlan10-bridge.xml** を使用して、**libvirt** に新しい仮想ネットワークを作成します。

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

3. **~/vxlan10-bridge.xml** を削除します。

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

4. **vxlan10-bridge** 仮想ネットワークを起動します。

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

5. **libvirtd** の起動時に自動的に起動するように **vxlan10-bridge** 仮想ネットワークを設定します。

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

検証

- 仮想ネットワークのリストを表示します。

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

関連情報

- **virsh(1)** man ページ

10.5. VXLAN を使用するように仮想マシンの設定

ホストで、接続されている仮想拡張 LAN (VXLAN) でブリッジデバイスを使用するように仮想マシンを設定するには、**vxlan10-bridge** 仮想ネットワークを使用する新しい仮想マシンを作成するか、このネットワークを使用する既存の仮想マシンの設定を更新します。

RHEL ホストでこの手順を実行します。

前提条件

- **libvirtd** で **vxlan10-bridge** 仮想ネットワークを設定している。

手順

- 新しい仮想マシンを作成し、**vxlan10-bridge** ネットワークを使用するように設定するには、仮想マシンの作成時に、**--network network:vxlan10-bridge** オプションを **virt-install** に渡します。

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

- 既存の仮想マシンのネットワーク設定を変更するには、次のコマンドを実行します。

  a. 仮想マシンのネットワークインターフェイスを、**vxlan10-bridge** 仮想ネットワークに接続します。

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

  b. 仮想マシンをシャットダウンして、再起動します。

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

検証

1. ホストの仮想マシンの仮想ネットワークインターフェイスを表示します。

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

2. **vxlan10-bridge** ブリッジに接続されているインターフェイスを表示します。

```
# 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
```

libvirtd は、ブリッジの設定を動的に更新することに注意してください。**vxlan10-bridge** ネットワークを使用する仮想マシンを起動すると、ホストの対応する **vnet*** デバイスがブリッジのポートとして表示されます。

3. アドレス解決プロトコル (ARP) 要求を使用して、仮想マシンが同じ VXLAN にあるかどうかを確認します。

   a. 同じ VXLAN で、2 つ以上の仮想マシンを起動します。

   b. 仮想マシンから別の仮想マシンに ARP 要求を送信します。

```
# 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))
```

   コマンドが応答を示す場合、仮想マシンは同じレイヤー 2 ドメイン、およびこの場合は同じ VXLAN にあります。

   arping ユーティリティーを使用するには、**iputils** をインストールします。

関連情報

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

第11章 WIFI 接続の管理

RHEL には、wifi ネットワークを設定して接続するための複数のユーティリティーとアプリケーションが用意されています。次に例を示します。

- **nmcli** ユーティリティーを使用して、コマンドラインで接続を設定する。
- **nmtui** アプリケーションを使用して、テキストベースのユーザーインターフェイスで接続を設定する。
- GNOME システムメニューを使用すると、設定を必要としない Wi-Fi ネットワークにすばやく接続する。
- **GNOME Settings** アプリケーションを使用して、GNOME アプリケーションで接続を設定する。
- **nm-connection-editor** アプリケーションを使用して、グラフィカルユーザーインターフェイスで接続を設定する。
- **network** RHEL システムロールを使用して、1つまたは複数のホストでの接続の設定を自動化する。

11.1. サポートされている WIFI セキュリティータイプ

wifi ネットワークがサポートするセキュリティータイプに応じて、多かれ少なかれ安全にデータを送信できます。

警告

暗号化を使用しない、または安全でない WEP または WPA 標準のみをサポートする wifi ネットワークには接続しないでください。

Red Hat Enterprise Linux 9 は、以下の wifi セキュリティータイプをサポートします。

- **None**:暗号化は無効になり、ネットワーク経由でプレーンテキスト形式でデータが転送されます。
- **Enhanced Open**:opportunistic wireless encryption (OWE) を使用すると、デバイスは一意のペアワイズマスターキー (PMK) をネゴシエートして、認証なしでワイヤレスネットワークの接続を暗号化します。
- **LEAP**:Cisco が開発した Lightweight Extensible Authentication Protocol は、拡張認証プロトコル (EAP) の独自バージョンです。
- **WPA & WPA2 Personal**:パーソナルモードでは、Wi-Fi Protected Access (WPA) および Wi-Fi Protected Access 2 (WPA2) 認証方法で事前共有キーが使用されます。
- **WPA & WPA2 Enterprise**:エンタープライズモードでは、WPA と WPA2 は EAP フレームワークを使用し、リモート認証ダイヤルインユーザーサービス (RADIUS) サーバーに対してユーザーを認証します。

- **WPA3 Personal**:Wi-Fi Protected Access 3 (WPA3) Personal は、辞書攻撃を防ぐために Pre-shared Key (PSK) の代わりに Simultaneous Authentication of Equals (SAE) を使用します。WPA3 では、Perfect Forward Secrecy (PFS) が使用されます。

11.2. NMCLI を使用した WIFI ネットワークへの接続

nmcli ユーティリティーを使用して、wifi ネットワークに接続できます。初めてネットワークに接続しようとすると、ユーティリティーは NetworkManager 接続プロファイルを自動的に作成します。ネットワークに静的 IP アドレスなどの追加設定が必要な場合は、プロファイルが自動的に作成された後にプロファイルを変更できます。

前提条件

- ホストに wifi デバイスがインストールされている。
- ハードウェアスイッチがある場合は、wifi デバイスが有効になっている。

手順

1. NetworkManager で wifi 無線が無効になっている場合は、この機能を有効にします。

```
# nmcli radio wifi on
```

2. オプション: 利用可能な wifi ネットワークを表示します。

```
# 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
```

サービスセット識別子 (**SSID**) 列には、ネットワークの名前が含まれています。列に **--** が表示されている場合、このネットワークのアクセスポイントは SSID をブロードキャストしていません。

3. wifi ネットワークに接続します。

```
# nmcli device wifi connect Office --ask
Password: wifi-password
```

対話的に入力するのではなく、コマンドでパスワードを設定する場合は、コマンドで **--ask** の代わりに **password wifi-password** オプションを使用します。

```
# nmcli device wifi connect Office wifi-password
```

ネットワークが静的 IP アドレスを必要とする場合、NetworkManager はこの時点で接続のアクティブ化に失敗することに注意してください。後の手順で IP アドレスを設定できます。

4. ネットワークに静的 IP アドレスが必要な場合:

   a. IPv4 アドレス設定を設定します。次に例を示します。

```
# 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. IPv6 アドレス設定を設定します。次に例を示します。

```
# 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. 接続を再度有効にします。

```
# nmcli connection up Office
```

検証

1. アクティブな接続を表示します。

```
# nmcli connection show --active
NAME    ID                                    TYPE  DEVICE
Office  2501eb7e-7b16-4dc6-97ef-7cc460139a58  wifi  wlp0s20f3
```

作成した wifi 接続が出力にリストされている場合、その接続はアクティブです。

2. ホスト名または IP アドレスに ping を実行します。

```
# ping -c 3 example.com
```

関連情報

- **nm-settings-nmcli (5)** man ページ

11.3. GNOME システムメニューを使用した WI-FI ネットワークへの接続

GNOME システムメニューを使用して、wifi ネットワークに接続できます。初めてネットワークに接続するとき、GNOME は NetworkManager 接続プロファイルを作成します。接続プロファイルを自動的に接続しないように設定した場合、GNOME システムメニューを用いて、既存の NetworkManager 接続プロファイルを使用して wifi ネットワークに手動で接続することもできます。

注記

GNOME システムメニューを使用して初めて wifi ネットワークへの接続を確立する場合、一定の制限があります。たとえば、IP アドレス設定を構成することはできません。この場合、最初に接続を設定します。

- **GNOME 設定** アプリケーションで
- **nm-connection-editor** アプリケーションで
- **nmcli** コマンドの使用

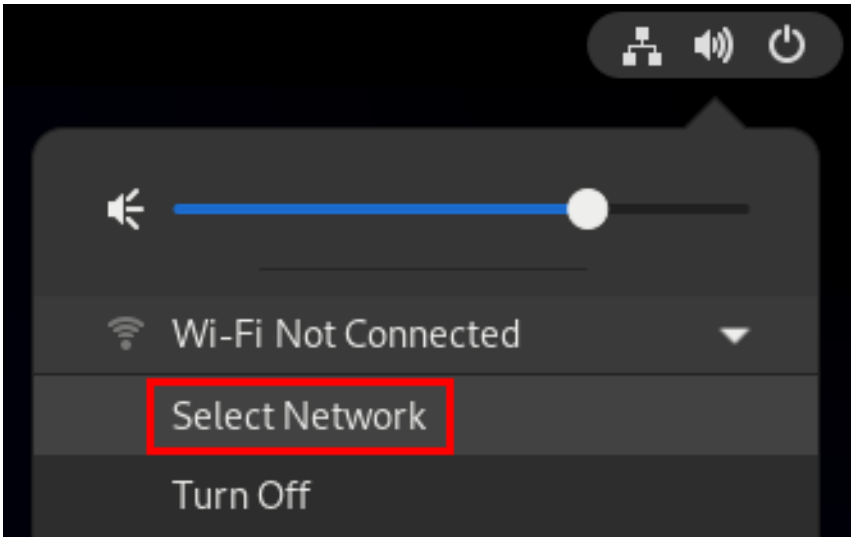
前提条件

- ホストに wifi デバイスがインストールされている。
- ハードウェアスイッチがある場合は、wifi デバイスが有効になっている。

手順

手順

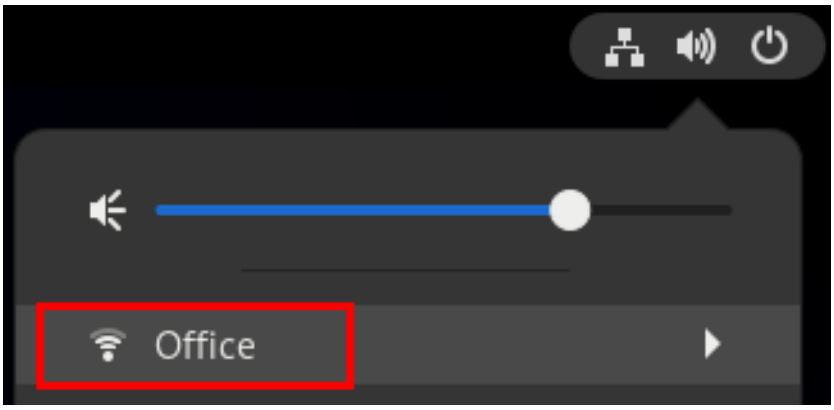
1. トップバーの右側にあるシステムメニューを開きます。
2. **Wi-Fi Not Connected** エントリーを展開します。
3. **Select Network** をクリックします。



4. 接続する wifi ネットワークを選択します。
5. **Connect** をクリックします。
6. このネットワークに初めて接続する場合は、ネットワークのパスワードを入力し、**Connect** をクリックします。

検証

1. トップバーの右側にあるシステムメニューを開き、wifi ネットワークが接続されていることを確認します。



ネットワークがリストに表示されていれば、接続されています。

2. ホスト名または IP アドレスに ping を実行します。

```
# ping -c 3 example.com
```

11.4. GNOME 設定アプリケーションを使用した WI-FI ネットワークへの接続

gnome-control-center という名前の **GNOME settings** アプリケーションを使用して、wifi ネットワークに接続し、接続を設定できます。初めてネットワークに接続するとき、GNOME は NetworkManager 接続プロファイルを作成します。

GNOME settings では、RHEL がサポートするすべての wifi ネットワークセキュリティータイプの wifi 接続を設定できます。

前提条件

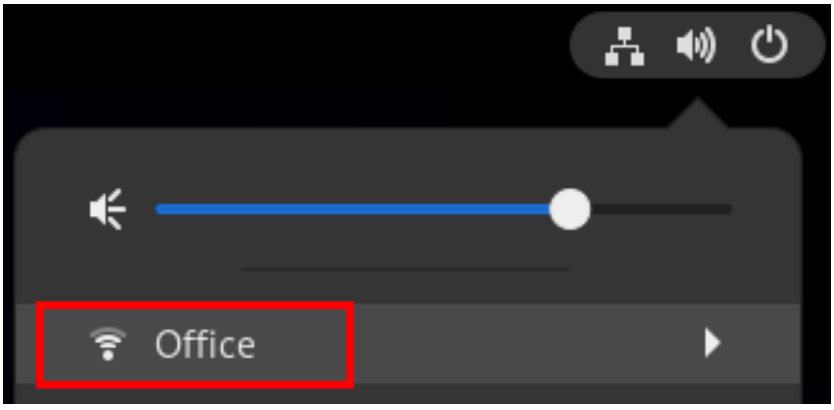
- ホストに wifi デバイスがインストールされている。
- ハードウェアスイッチがある場合は、wifi デバイスが有効になっている。

手順

1. **Super** キーを押し、**Wi-Fi** と入力して **Enter** を押します。
2. 接続したい wifi ネットワークの名前をクリックします。
3. ネットワークのパスワードを入力し、**Connect** をクリックします。
4. 静的 IP アドレスや WPA2 パーソナル以外のセキュリティータイプなど、ネットワークに追加の設定が必要な場合:
   a. ネットワーク名の横にある歯車のアイコンをクリックします。
   b. オプション: **Details** タブでネットワークプロファイルを設定して、自動的に接続しないようにします。
   この機能を無効にした場合は、**GNOME settings** や GNOME システムメニューなどを使用して、常に手動でネットワークに接続する必要があります。
   c. **IPv4** タブで IPv4 設定を設定し、**IPv6** タブで IPv6 設定を設定します。
   d. **Security** タブで、ネットワークの認証 (**WPA3 Personal** など) を選択し、パスワードを入力します。
   選択したセキュリティーに応じて、アプリケーションは追加のフィールドを表示します。それに応じてそれらを埋めます。詳しくは wifi ネットワークの管理者におたずねください。
   e. **Apply** をクリックします。

検証

1. トップバーの右側にあるシステムメニューを開き、wifi ネットワークが接続されていることを確認します。



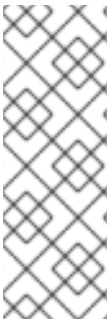
ネットワークがリストに表示されていれば、接続されています。

2. ホスト名または IP アドレスに ping を実行します。

```
# ping -c 3 example.com
```

11.5. NMTUI を使用した WIFI 接続の設定

nmtui アプリケーションは、NetworkManager 用のテキストベースのユーザーインターフェイスを提供します。**nmtui** を使用して Wi-Fi ネットワークに接続できます。



注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

手順

1. 接続に使用するネットワークデバイス名がわからない場合は、使用可能なデバイスを表示します。

```
# nmcli device status
DEVICE    TYPE    STATE          CONNECTION
wlp2s0    wifi    unavailable    --
...
```

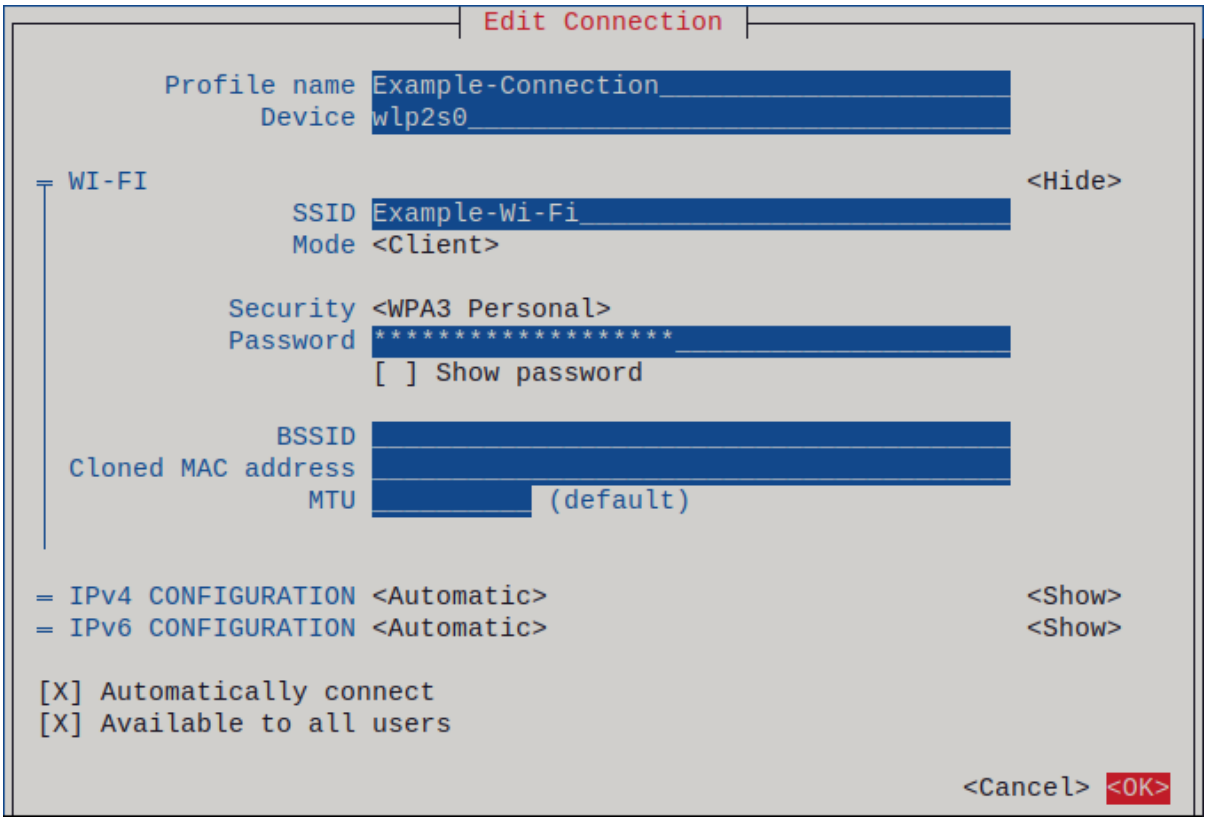
2. **nmtui** を開始します。

```
# nmtui
```

3. **Edit a connection** 選択し、**Enter** を押します。
4. **Add** ボタンを押します。
5. ネットワークタイプのリストから **Wi-Fi** を選択し、**Enter** を押します。
6. オプション: 作成する NetworkManager プロファイルの名前を入力します。
ホストに複数のプロファイルがある場合は、わかりやすい名前を付けると、プロファイルの目的を識別しやすくなります。
7. **Device** フィールドにネットワークデバイス名を入力します。
8. Wi-Fi ネットワークの名前である Service Set Identifier (SSID) を **SSID** フィールドに入力します。
9. **Mode** フィールドはデフォルトの **Client** のままにします。
10. **Security** フィールドを選択して **Enter** を押し、リストからネットワークの認証タイプを設定します。

選択した認証タイプに応じて、**nmtui** は異なるフィールドを表示します。

11. 認証タイプ関連のフィールドに入力します。
12. Wi-Fi ネットワークに静的 IP アドレスが必要な場合:
    a. プロトコルの横にある **Automatic** ボタンを押し、表示されたリストから **Manual** を選択します。
    b. 設定するプロトコルの横にある **Show** ボタンを押して、追加のフィールドを表示し、それらに入力します。
13. **OK** ボタンを押して、新しい接続を作成し、自動的にアクティブにします。



14. **Back** ボタンを押してメインメニューに戻ります。
15. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

1. アクティブな接続を表示します。

```
# nmcli connection show --active
NAME    ID                                    TYPE  DEVICE
Office  2501eb7e-7b16-4dc6-97ef-7cc460139a58  wifi  wlp0s20f3
```

作成した wifi 接続が出力にリストされている場合、その接続はアクティブです。

2. ホスト名または IP アドレスに ping を実行します。

```
# ping -c 3 example.com
```

11.6. NM-CONNECTION-EDITOR を使用した WIFI 接続の設定

nm-connection-editor アプリケーションを使用して、ワイヤレスネットワークの接続プロファイルを作成できます。このアプリケーションでは、RHEL がサポートするすべての wifi ネットワーク認証タイプを設定できます。

デフォルトでは、NetworkManager は接続プロファイルの自動接続機能を有効にし、保存されたネットワークが利用可能な場合は自動的に接続します。

前提条件

- ホストに wifi デバイスがインストールされている。
- ハードウェアスイッチがある場合は、wifi デバイスが有効になっている。

手順

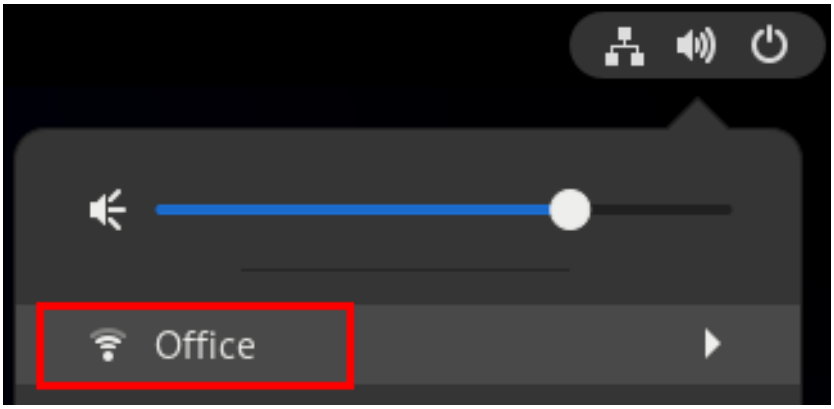
1. ターミナルを開き、次のコマンドを入力します。

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

2. **+** ボタンをクリックして、新しい接続を追加します。
3. **Wi-Fi** 接続タイプを選択し、**Create** をクリックします。
4. オプション: 接続プロファイルの名前を設定します。
5. オプション: **General** タブでネットワークプロファイルを設定して、自動的に接続しないようにします。
   この機能を無効にした場合は、**GNOME settings** や GNOME システムメニューなどを使用して、常に手動でネットワークに接続する必要があります。
6. **Wi-Fi** タブで、**SSID** フィールドにサービスセット識別子 (SSID) を入力します。
7. **Wi-Fi Security** タブで、ネットワークの認証タイプ (**WPA3 Personal** など) を選択し、パスワードを入力します。
   選択したセキュリティーに応じて、アプリケーションは追加のフィールドを表示します。それに応じてそれらを埋めます。詳しくは wifi ネットワークの管理者におたずねください。
8. **IPv4** タブで IPv4 設定を設定し、**IPv6** タブで IPv6 設定を設定します。
9. **Save** をクリックします。
10. **Network Connections** ウィンドウを閉じます。

検証

1. トップバーの右側にあるシステムメニューを開き、wifi ネットワークが接続されていることを確認します。



ネットワークがリストに表示されていれば、接続されています。

2. ホスト名または IP アドレスに ping を実行します。

```
# ping -c 3 example.com
```

11.7. NETWORK RHEL システムロールを使用した 802.1X ネットワーク認証による WI-FI 接続の設定

RHEL システムロールを使用すると、Wi-Fi 接続の作成を自動化できます。たとえば、Ansible Playbook を使用して、**wlp1s0** インターフェイスのワイヤレス接続プロファイルをリモートで追加できます。作成されたプロファイルは、802.1X 標準を使用して、wifi ネットワークに対してクライアントを認証します。Playbook は、DHCP を使用するように接続プロファイルを設定します。静的 IP 設定を設定するには、それに応じて **IP** ディクショナリーのパラメーターを調整します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理対象ノードに **wpa_supplicant** パッケージをインストールしている。
- DHCP は、管理対象ノードのネットワークで使用できる。
- TLS 認証に必要な以下のファイルがコントロールノードにある。
  - クライアントキーは、**/srv/data/client.key** ファイルに保存されます。
  - クライアント証明書は **/srv/data/client.crt** ファイルに保存されます。
  - CA 証明書は **/srv/data/ca.crt** ファイルに保存されます。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
```

```
- 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"
```

これらの設定では、**wlp1s0** インターフェイスの Wi-Fi 接続プロファイルを定義します。このプロファイルは、802.1X 標準を使用して、Wi-Fi ネットワークに対してクライアントを認証します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

11.8. NMCLI を使用した既存のプロファイルでの 802.1X ネットワーク認証による WI-FI 接続の設定

nmcli ユーティリティーを使用して、クライアントがネットワークに対して自己認証するように設定できます。たとえば、**wlp1s0** という名前の既存の NetworkManager wifi 接続プロファイルで、MSCHAPv2 (Microsoft Challenge-Handshake Authentication Protocol version 2) を使用する PEAP (Protected Extensible Authentication Protocol) 認証を設定します。

前提条件

- ネットワークには 802.1X ネットワーク認証が必要です。
- wifi 接続プロファイルが NetworkManager に存在し、有効な IP 設定があります。
- クライアントがオーセンティケーターの証明書を検証する必要がある場合は、認証局 (CA) 証明書を **/etc/pki/ca-trust/source/anchors/** ディレクトリーに保存する必要があります。
- **wpa_supplicant** パッケージがインストールされている。

手順

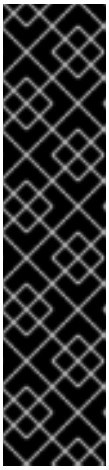
1. wifi セキュリティーモードを **wpa-eap** に設定し、Extensible Authentication Protocol (EAP) を **peap** に設定し、内部認証プロトコルを **mschapv2** に設定し、ユーザー名を設定します。

```
# nmcli connection modify wlp1s0 wireless-security.key-mgmt wpa-eap 802-1x.eap peap 802-1x.phase2-auth mschapv2 802-1x.identity user_name
```

1 つのコマンドで **wireless-security.key-mgmt** パラメーター、**802-1x.eap** パラメーター、**802-1x.phase2-auth** パラメーター、および **802-1x.identity** パラメーターを設定する必要があります。

2. 必要に応じて、パスワードを設定に保存します。

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



重要

デフォルトでは、NetworkManager はパスワードをプレーンテキストで **/etc/sysconfig/network-scripts/keys-connection_name** ファイルに保存します。このファイルは **root** ユーザーのみが読み取ることができます。ただし、設定ファイル内のプレーンテキストのパスワードは、セキュリティーリスクになる可能性があります。

セキュリティーを強化するには、**802-1x.password-flags** パラメーターを **0x1** に設定します。この設定では、GNOME デスクトップ環境または **nm-applet** が実行中のサーバーで、NetworkManager がこれらのサービスからパスワードを取得します。その他の場合は、NetworkManager によりパスワードの入力が求められます。

3. クライアントがオーセンティケーターの証明書を検証する必要がある場合は、接続プロファイルの **802-1x.ca-cert** パラメーターを CA 証明書のパスに設定します。

```
# nmcli connection modify wlp1s0 802-1x.ca-cert /etc/pki/ca-
trust/source/anchors/ca.crt
```

注記

セキュリティー上の理由から、Red Hat は、クライアントがオーセンティケーターの ID を検証できるように、オーセンティケーターの証明書を推奨しています。

4. 接続プロファイルをアクティベートします。

```
# nmcli connection up wlp1s0
```

検証

- ネットワーク認証が必要なネットワーク上のリソースにアクセスします。

関連情報

- wifi 接続の管理
- **nm-settings(5)** man ページ
- **nmcli(1)** man ページ

11.9. ワイヤレス規制ドメインの手動設定

RHEL では、**udev** ルールが **setregdomain** ユーティリティーを実行してワイヤレス規制ドメインを設定します。次に、ユーティリティーはこの情報をカーネルに提供します。

デフォルトでは、**setregdomain** は国コードを自動的に決定しようとします。これが失敗する場合は、ワイヤレス規制ドメインの設定が間違っている可能性があります。この問題を回避するには、国コードを手動で設定します。

重要

規制ドメインを手動で設定すると、自動検出が無効になります。そのため、後で別の国でコンピューターを使用すると、以前に設定された設定が正しくなくなる可能性があります。この場合、**/etc/sysconfig/regdomain** ファイルを削除して自動検出に戻すか、以下の手順を使用して規制ドメイン設定を手動で再度更新します。

手順

1. オプション: 現在の規制ドメイン設定を表示します。

```
# iw reg get
global
country US: DFS-FCC
...
```

2. 次の内容で **/etc/sysconfig/regdomain** ファイルを作成します。

```
COUNTRY=<country_code>
```

COUNTRY 変数を ISO 3166-1 alpha2 国コード (ドイツの場合は **DE**、アメリカ合衆国の場合は **US** など) に設定します。

3. 規制ドメインを設定します。

```
# setregdomain
```

検証

- 規制ドメインの設定を表示します。

```
# iw reg get
global
country DE: DFS-ETSI
...
```

関連情報

- **setregdomain(1)** man ページ
- **iw(8)** man ページ
- **regulatory.bin(5)** man ページ
- ISO 3166 国コード

第12章 RHEL を WPA2 または WPA3 パーソナルアクセスポイントとして設定する方法

Wi-Fi デバイスを備えたホストでは、NetworkManager を使用して、このホストをアクセスポイントとして設定できます。Wi-Fi Protected Access 2 (WPA2) および Wi-Fi Protected Access 3 (WPA3) Personal はセキュアな認証方法を提供します。ワイヤレスクライアントは事前共有キー (PSK) を使用してアクセスポイントに接続し、RHEL ホスト上およびネットワーク内のサービスを使用できます。

アクセスポイントを設定すると、NetworkManager は自動的に以下を行います。

- クライアントに DHCP および DNS サービスを提供するように **dnsmasq** サービスを設定します
- IP 転送を有効にします
- **nftables** ファイアウォールルールを追加して、wifi デバイスからのトラフィックをマスカレードし、IP 転送を設定します

前提条件

- Wi-Fi デバイスが、アクセスポイントモードでの実行をサポートしている
- Wi-Fi デバイスは使用していない
- ホストがインターネットにアクセスできる

手順

1. Wi-Fi デバイスを一覧表示して、アクセスポイントを提供するデバイスを特定します。

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

2. デバイスがアクセスポイントモードをサポートしていることを確認します。

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

Wi-Fi デバイスをアクセスポイントとして使用するには、デバイスがこの機能をサポートしている必要があります。

3. **dnsmasq** および **NetworkManager-wifi** パッケージをインストールします。

```
# dnf install dnsmasq NetworkManager-wifi
```

NetworkManager は **dnsmasq** サービスを使用して、アクセスポイントのクライアントに DHCP および DNS サービスを提供します。

4. アクセスポイントの初期設定を作成します。

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

このコマンドは、WPA2 および WPA3 Personal 認証を提供する **wlp0s20f3** デバイス上のアクセスポイントの接続プロファイルを作成します。ワイヤレスネットワークの名前である Service Set Identifier (SSID) は **Example-Hotspot** で、事前共有キーの **password** を使用します。

5. オプション: WPA3 のみをサポートするようにアクセスポイントを設定します。

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

6. デフォルトでは、NetworkManager は wifi デバイスに IP アドレス **10.42.0.1** を使用し、残りの **10.42.0.0/24** サブネットからの IP アドレスをクライアントに割り当てます。別のサブネットと IP アドレスを設定するには、次のように入力します。

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

設定した IP アドレス (この場合は **192.0.2.254**) は、NetworkManager が wifi デバイスに割り当てるものです。クライアントは、この IP アドレスをデフォルトゲートウェイおよび DNS サーバーとして使用します。

7. 接続プロファイルをアクティベートします。

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

検証

1. サーバーの場合:

   a. NetworkManager が **dnsmasq** サービスを開始し、そのサービスがポート 67 (DHCP) および 53 (DNS) でリッスンしていることを確認します。

```
# 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. **nftables** ルールセットを表示して、NetworkManager が **10.42.0.0/24** サブネットからのトラフィックの転送とマスカレードを有効にしていることを確認します。

```
# 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. Wi-Fi アダプターを備えたクライアントの場合:

   a. 利用可能なネットワークのリストを表示します。

```
# 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. **Example-Hotspot** ワイヤレスネットワークに接続します。Managing Wi-Fi connections を参照してください。

   c. リモートネットワークまたはインターネット上のホストに ping を実行し、接続が機能していることを確認します。

```
# ping -c 3 www.redhat.com
```

関連情報

- **nm-settings(5)** man ページ

第13章 MACSEC を使用した同じ物理ネットワーク内のレイヤー 2 トラフィックの暗号化

MACsec を使用して、2 つのデバイス間の通信を (ポイントツーポイントで) セキュリティー保護できます。たとえば、ブランチオフィスがメトロイーサネット接続を介してセントラルオフィスに接続されている場合、オフィスを接続する 2 つのホストで MACsec を設定して、セキュリティーを強化できます。

Media Access Control Security (MACsec) は、イーサーネットリンクで異なるトラフィックタイプを保護するレイヤー 2 プロトコルです。これには以下が含まれます。

- DHCP (Dynamic Host Configuration Protocol)
- アドレス解決プロトコル (ARP)
- インターネットプロトコルのバージョン 4 / 6 (**IPv4** / **IPv6**)
- TCP や UDP などの IP 経由のトラフィック

MACsec はデフォルトで、LAN 内のすべてのトラフィックを GCM-AES-128 アルゴリズムで暗号化および認証し、事前共有キーを使用して参加者ホスト間の接続を確立します。共有前の鍵を変更する場合は、MACsec を使用するネットワーク内のすべてのホストで NM 設定を更新する必要があります。

MACsec 接続は、親としてイーサネットネットワークカード、VLAN、トンネルデバイスなどのイーサネットデバイスを使用します。暗号化した接続のみを使用して他のホストと通信するように、MACsec デバイスでのみ IP 設定を指定するか、親デバイスに IP 設定を指定することもできます。後者の場合、親デバイスを使用して、暗号化されていない接続と暗号化された接続用の MACsec デバイスで他のホストと通信できます。

MACsec には特別なハードウェアは必要ありません。たとえば、ホストとスイッチの間のトラフィックのみを暗号化する場合を除き、任意のスイッチを使用できます。このシナリオでは、スイッチが MACsec もサポートする必要があります。

つまり、MACsec を設定する方法は 2 つあります。

- ホスト対ホスト
- 他のホストに切り替えるホスト

重要

MACsec は、同じ (物理または仮想) LAN のホスト間でのみ使用することができます。

13.1. NMCLI を使用した MACSEC 接続の設定

nmcli ツールを使用して、MACsec を使用するようにイーサーネットインターフェイスを設定できます。たとえば、イーサネット経由で接続された 2 つのホスト間に MACsec 接続を作成できます。

手順

1. MACsec を設定する最初のホストで:
   - 事前共有鍵の接続アソシエーション鍵 (CAK) と接続アソシエーション鍵名 (CKN) を作成します。

a. 16 バイトの 16 進 CAK を作成します。

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

b. 32 バイトの 16 進 CKN を作成します。

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. 両方のホストで、MACsec 接続を介して接続します。

3. MACsec 接続を作成します。

```
# 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
```

前の手順で生成された CAK および CKN を **macsec.mka-cak** および **macsec.mka-ckn** パラメーターで使用します。この値は、MACsec で保護されるネットワーク内のすべてのホストで同じである必要があります。

4. MACsec 接続で IP を設定します。

a. **IPv4** 設定を指定します。たとえば、静的 **IPv4** アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **macsec0** 接続に設定するには、以下のコマンドを実行します。

```
# 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. **IPv6** 設定を指定しますたとえば、静的 **IPv6** アドレス、ネットワークマスク、デフォルトゲートウェイ、および DNS サーバーを **macsec0** 接続に設定するには、以下のコマンドを実行します。

```
# 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. 接続をアクティベートします。

```
# nmcli connection up macsec0
```

検証

1. トラフィックが暗号化されていることを確認します。

```
# tcpdump -nn -i enp1s0
```

2. オプション: 暗号化されていないトラフィックを表示します。

```
# tcpdump -nn -i macsec0
```

3. MACsec の統計を表示します。

```
# ip macsec show
```

4. integrity-only (encrypt off) および encryption (encrypt on) の各タイプの保護に対して個々のカウンターを表示します。

```
# ip -s macsec show
```

13.2. NMSTATECTL を使用した MACSEC 接続の設定

nmstatectl ユーティリティーを宣言的に使用して、イーサネットインターフェイスが MACsec を使用するように設定できます。たとえば、YAML ファイルでは、ネットワークの望ましい状態を記述します。ネットワークでは、イーサネット経由で接続された 2 つのホスト間に MACsec 接続があることが想定されます。**nmstatectl** ユーティリティーは、YAML ファイルを解釈し、ホスト間に永続的かつ一貫したネットワーク設定をデプロイします。

リンク層 (Open Systems Interconnection (OSI) モデルのレイヤー 2 とも呼ばれます) での通信を保護するために MACsec セキュリティー標準を使用すると、主に次のような利点が得られます。

- レイヤー 2 で暗号化することで、レイヤー 7 で個々のサービスを暗号化する必要がなくなります。これにより、各ホストの各エンドポイントで多数の証明書を管理することに関連するオーバーヘッドが削減されます。
- ルーターやスイッチなどの直接接続されたネットワークデバイス間のポイントツーポイントセキュリティー。
- アプリケーションや上位レイヤープロトコルに変更を加える必要がなくなります。

前提条件

- 物理または仮想イーサネットネットワークインターフェイスコントローラー (NIC) がサーバーに設定されている。
- **nmstate** パッケージがインストールされている。

手順

1. MACsec を設定する最初のホストで、事前共有キー用の接続関連キー (CAK: connectivity association key) および接続関連キー名 (CKN: connectivity-association key name) を作成します。
   a. 16 バイトの 16 進 CAK を作成します。

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

   b. 32 バイトの 16 進 CKN を作成します。

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. MACsec 接続を介して接続するホストの両方で、次の手順を実行します。

a. 次の設定を含む YAML ファイル (例: **create-macsec-connection.yml**) を作成します。

```
---
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-interface: macsec0
    next-hop-address: 192.0.2.2
    table-id: 254
  - destination: 192.0.2.2/32
    next-hop-interface: macsec0
    next-hop-address: 0.0.0.0
    table-id: 254
dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
interfaces:
- name: macsec0
  type: macsec
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 32
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
  macsec:
    encrypt: true
    base-iface: enp0s1
    mka-cak: 50b71a8ef0bd5751ea76de6d6c98c03a
    mka-ckn: f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
    port: 0
    validation: strict
    send-sci: true
```

b. 前の手順で生成された CAK および CKN を **mka-cak** および **mka-ckn** パラメーターで使用します。この値は、MACsec で保護されるネットワーク内のすべてのホストで同じである必要があります。

c. オプション: 同じ YAML 設定ファイルで、次の設定も指定できます。

- 静的 IPv4 アドレス: **192.0.2.1** (サブネットマスクが **/32**)
- 静的 IPv6 アドレス: **2001:db8:1::1** (サブネットマスクが **/64**)
- IPv4 デフォルトゲートウェイ - **192.0.2.2**
- IPv4 DNS サーバー - **192.0.2.200**

- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

3. 設定をシステムに適用します。

```
# nmstatectl apply create-macsec-connection.yml
```

検証

1. 現在の状態を YAML 形式で表示します。

```
# **nmstatectl show macsec0
```

2. トラフィックが暗号化されていることを確認します。

```
# tcpdump -nn -i enp0s1
```

3. オプション: 暗号化されていないトラフィックを表示します。

```
# tcpdump -nn -i macsec0
```

4. MACsec の統計を表示します。

```
# ip macsec show
```

5. integrity-only (encrypt off) および encryption (encrypt on) の各タイプの保護に対して個々のカウンターを表示します。

```
# ip -s macsec show
```

関連情報

- MACsec: a different solution to encrypt network traffic

13.3. 関連情報

- MACsec: a different solution to encrypt network traffic ブログ

第14章 IPVLAN の使用

IPVLAN は、仮想ネットワークデバイス用のドライバーで、コンテナー環境でホストネットワークにアクセスするのに使用できます。IPVLAN は外部ネットワークに対し、ホストネットワーク内で作成された IPVLAN デバイスの数に関わらず、MAC アドレスを 1 つ公開します。つまり、ユーザーは複数コンテナーに複数の IPVLAN デバイスを持つことができますが、対応するスイッチは MAC アドレスを 1 つ読み込むということです。IPVLAN ドライバーは、ローカルスイッチで管理できる MAC アドレスの数に制限がある場合に役立ちます。

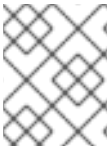
14.1. IPVLAN モード

IPVLAN では、次のモードが使用できます。

- **L2 モード**
  IPVLAN の **L2 モード** では、仮想デバイスは アドレス解決プロトコル (ARP) リクエストを受信して応答します。**netfilter** フレームワークは、仮想デバイスを所有するコンテナー内でのみ動作します。**netfilter** チェーンは、コンテナー化したトラッフィクにあるデフォルトの名前空間では実行されません。**L2 モード**を使用すると、パフォーマンスは高くなりますが、ネットワークトラフィックの制御性は低下します。

- **L3 モード**
  L3 モードでは、仮想デバイスは **L3** 以上のトラフィックのみを処理します。仮想デバイスは ARP リクエストに応答せず、関連するピアの IPVLAN IP アドレスは、隣接エントリーをユーザーが手動で設定する必要があります。関連するコンテナーの送信トラフィックはデフォルトの名前空間の **netfilter** の POSTROUTING および OUTPUT チェーンに到達する一方、ingress トラフィックは **L2 モード** と同様にスレッド化されます。**L3 モード** を使用すると、制御性は高くなりますが、ネットワークトラフィックのパフォーマンスは低下します。

- **L3S モード**
  L3S モード では、仮想デバイスは **L3 モード** と同様の処理をしますが、関連するコンテナーの egress トラフィックと ingress トラフィックの両方がデフォルトの名前空間の **netfilter** チェーンに到達する点が異なります。**L3S モード** は、**L3 モード** と同様の動作をしますが、ネットワークの制御が強化されます。



注記

IPVLAN 仮想デバイスは、**L3** モードおよび **L3S** モードでは、ブロードキャストトラフィックおよびマルチキャストトラフィックを受信しません。

14.2. IPVLAN および MACVLAN の比較

以下の表は、MACVLAN と IPVLAN の主な相違点を示しています。

MACVLAN	IPVLAN
各 MACVLAN デバイスに対して、MAC アドレスを使用します。 スイッチが MAC テーブルに保存できる MAC アドレスの最大数に達すると、接続が失われる可能性があることに注意してください。	IPVLAN デバイスの数を制限しない MAC アドレスを 1 つ使用します。

MACVLAN	IPVLAN
グローバル名前空間の netfilter ルールは、子名前空間の MACVLAN デバイスとの間のトラフィックに影響を与えることはできません。	**L3 モード** および **L3S モード** の IPVLAN デバイスとの間のトラフィックを制御できます。

IPVLAN と MACVLAN はどちらも、いかなるレベルのカプセル化も必要としません。

14.3. IPROUTE2 を使用した IPVLAN デバイスの作成および設定

この手順では、**iproute2** を使用して IPVLAN デバイスを設定する方法を説明します。

手順

1. IPVLAN デバイスを作成するには、次のコマンドを実行します。

```
# ip link add link real_NIC_device name IPVLAN_device type ipvlan mode l2
```

ネットワークインターフェイスコントローラー (NIC) は、コンピューターをネットワークに接続するハードウェアコンポーネントです。

例14.1 IPVLAN デバイスの作成

```
# 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. **IPv4** アドレスまたは **IPv6** アドレスをインターフェイスに割り当てるには、次のコマンドを実行します。

```
# ip addr add dev IPVLAN_device IP_address/subnet_mask_prefix
```

3. **L3 モード** または **L3S モード** の IPVLAN デバイスを設定する場合は、以下の設定を行います。

   a. リモートホストのリモートピアのネイバー設定を行います。

   ```
   # ip neigh add dev peer_device IPVLAN_device_IP_address lladdr MAC_address
   ```

   MAC_address は、IPVLAN デバイスのベースである実際の NIC の MAC アドレスになります。

   b. **L3 モード** の IPVLAN デバイスを設定する場合は、次のコマンドを実行します。

   ```
   # ip route add dev <real_NIC_device> <peer_IP_address/32>
   ```

   L3S モード の場合は、次のコマンドを実行します。

```
# ip route add dev real_NIC_device peer_IP_address/32
```

IP アドレスは、リモートピアのアドレスを使用します。

4. IPVLAN デバイスをアクティブに設定するには、次のコマンドを実行します。

```
# ip link set dev IPVLAN_device up
```

5. IPVLAN デバイスがアクティブであることを確認するには、リモートホストで次のコマンドを実行します。

```
# ping IP_address
```

IP_address には、IPVLAN デバイスの IP アドレスを使用します。

第15章 特定のデバイスを無視するように NETWORKMANAGER の設定

デフォルトでは、NetworkManager はすべてのデバイスを管理します。特定のデバイスを無視するには、NetworkManager を **unmanaged** として設定できます。

15.1. NMCLI を使用したループバックインターフェイスの設定

デフォルトでは、NetworkManager はループバック (**lo**) インターフェイスを管理しません。**lo** インターフェイスの接続プロファイルを作成した後、NetworkManager を使用してこのデバイスを設定できます。例としては次のようなものがあります。

- **lo** インターフェイスに追加の IP アドレスを割り当てる
- DNS アドレスを定義する
- **lo** インターフェイスの最大伝送単位 (MTU) サイズを変更する

手順

1. タイプ **loopback** の新しい接続を作成します。

```
# nmcli connection add con-name example-loopback type loopback
```

2. カスタム接続設定を設定します。次に例を示します。

   a. 追加の IP アドレスをインターフェイスに割り当てるには、次のように入力します。

```
# nmcli connection modify example-loopback +ipv4.addresses 192.0.2.1/24
```

注記

NetworkManager は、再起動後も持続する IP アドレス **127.0.0.1** および **::1** を常に割り当てることで、**lo** インターフェイスを管理します。**127.0.0.1** および **::1** をオーバーライドすることはできません。ただし、追加の IP アドレスをインターフェイスに割り当てることができます。

   b. カスタムの最大伝送単位 (MTU) を設定するには、次のように入力します。

```
# nmcli con mod example-loopback loopback.mtu 16384
```

   c. DNS サーバーに IP アドレスを設定するには、次のように入力します。

```
# nmcli connection modify example-loopback ipv4.dns 192.0.2.0
```

   ループバック接続プロファイルで DNS サーバーを設定すると、このエントリーが **/etc/resolv.conf** ファイルで常に使用可能になります。DNS サーバーのエントリーは、ホストが異なるネットワーク間をローミングするかどうかには関係ありません。

3. 接続をアクティベートします。

```
# nmcli connection up example-loopback
```

検証

1. **lo** インターフェイスの設定を表示します。

```
# 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. DNS アドレスを確認します。

```
# cat /etc/resolv.conf

...
nameserver 192.0.2.0
...
```

15.2. NETWORKMANAGER でデバイスをマネージド外として永続的に設定

インターフェイス名、MAC アドレス、デバイスタイプなどのいくつかの基準に基づいてデバイスを **unmanaged** として永続的に設定できます。

ネットワークデバイスを一時的に **unmanaged** として設定する場合は、Temporarily configuring a device as unmanaged in NetworkManager を参照してください。

手順

1. オプション: デバイスの一覧を表示して、**unmanaged** に設定するデバイスまたは MAC アドレス を特定します。

```
# 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. 以下の内容で **/etc/NetworkManager/conf.d/99-unmanaged-devices.conf** ファイルを作成します。

- 特定のインターフェイスを管理対象外として設定するには、以下を追加します。

```
[keyfile]
unmanaged-devices=interface-name:enp1s0
```

- 特定の MAC アドレスを管理対象外として設定するには、以下を追加します。

```
[keyfile]
unmanaged-devices=mac:52:54:00:74:79:56
```

- 特定のタイプのすべてのデバイスを管理対象外として設定するには、以下を追加します。

```
[keyfile]
unmanaged-devices=type:ethernet
```

- 複数のデバイスを管理対象外に設定するには、**unmanaged-devices** パラメーターのエントリーをセミコロンで区切ります。以下に例を示します。

```
[keyfile]
unmanaged-devices=interface-name:enp1s0;interface-name:enp7s0
```

3. **NetworkManager** サービスを再読み込みします。

```
# systemctl reload NetworkManager
```

検証

- デバイスのリストを表示します。

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

enp1s0 デバイスの横にある **マネージド外** 状態は、NetworkManager がこのデバイスを管理していないことを示しています。

トラブルシューティング

- デバイスが **unmanaged** として表示されない場合は、NetworkManager 設定を表示します。

```
# NetworkManager --print-config
...
[keyfile]
unmanaged-devices=interface-name:enp1s0
...
```

指定した設定と出力が一致しない場合は、より優先度が高い設定ファイルによって設定がオーバーライドされていないことを確認してください。NetworkManager が複数の設定ファイルをマージする方法の詳細は、man ページの **NetworkManager.conf(5)** を参照してください。

15.3. NETWORKMANAGER でデバイスをマネージド外として一時的に設定

デバイスを一時的に **unmanaged** として設定できます。

この方法は、たとえば、テスト目的で使用します。ネットワークデバイスを **unmanaged** に応じて永続的に設定するには、Permanently configuring a device as unmanaged in NetworkManager を参照してください。

手順

1. オプション: デバイスのリストを表示して、**unmanaged** に設定するデバイスを特定します。

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

2. **enp1s0** デバイスを **unmanaged** の状態に設定します。

```
# nmcli device set enp1s0 managed no
```

検証

- デバイスのリストを表示します。

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

enp1s0 デバイスの横にある **マネージド外** 状態は、NetworkManager がこのデバイスを管理していないことを示しています。

関連情報

- **NetworkManager.conf(5)** man ページ

第16章 ダミーインターフェイスの作成

Red Hat Enterprise Linux ユーザーは、デバッグおよびテストの目的でダミーネットワークインターフェイスを作成および使用できます。ダミーインターフェイスは、実際には送信せずにパケットをルーティングするデバイスを提供します。NetworkManager が管理する追加のループバックのようなデバイスを作成し、非アクティブな SLIP (Serial Line Internet Protocol) アドレスをローカルプログラムの実アドレスのようにすることができます。

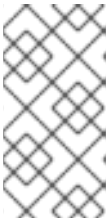
16.1. NMCLI を使用して IPV4 アドレスと IPV6 アドレスの両方を使用したダミーインターフェイスの作成

IPv4 アドレスや IPv6 アドレスなどのさまざまな設定でダミーインターフェイスを作成できます。ダミーインターフェイスを作成すると、NetworkManager により自動的にデフォルトの **public firewalld** ゾーンに割り当てられます。

手順

- 静的 IPv4 および IPv6 アドレスを使用して、**dummy0** という名前のダミーインターフェイスを作成します。

```
# 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
```



注記

IPv4 および IPv6 アドレスなしでダミーインターフェイスを設定するには、**ipv4.method** および **ipv6.method** パラメーターの両方を **disabled** に設定します。それ以外の場合は、IP 自動設定が失敗し、NetworkManager が接続を無効にしてデバイスを削除します。

検証

- 接続プロファイルを一覧表示します。

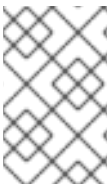
```
# nmcli connection show
NAME           UUID                                  TYPE     DEVICE
dummy-dummy0   aaf6eb56-73e5-4746-9037-eed42caa8a65  dummy    dummy0
```

関連情報

- **nm-settings(5)** man ページ

第17章 NETWORKMANAGER で特定接続の IPV6 の無効化

NetworkManager を使用してネットワークインターフェイスを管理するシステムでは、ネットワークが IPv4 のみを使用している場合は、IPv6 プロトコルを無効にできます。**IPv6** を無効にすると、NetworkManager はカーネルに対応する **sysctl** 値を自動的に設定します。



注記

カーネルの設定項目またはカーネルブートパラメーターを使用して IPv6 を無効にする場合は、システム設定に追加で配慮が必要です。詳細は、ナレッジベースの記事 How do I disable or enable the IPv6 protocol in RHEL? を参照してください。

17.1. NMCLI を使用した接続で IPV6 の無効化

nmcli ユーティリティーを使用して、コマンドラインで **IPv6** プロトコルを無効にすることができます。

前提条件

- システムは、NetworkManager を使用してネットワークインターフェイスを管理します。

手順

1. 必要に応じて、ネットワーク接続のリストを表示します。

```
# nmcli connection show
NAME    UUID                                  TYPE      DEVICE
Example 7a7e0151-9c18-4e6f-89ee-65bb2d64d365  ethernet  enp1s0
...
```

2. 接続の **ipv6.method** パラメーターを **disabled** に設定します。

```
# nmcli connection modify Example ipv6.method "disabled"
```

3. ネットワーク接続が再起動します。

```
# nmcli connection up Example
```

検証

1. デバイスの IP 設定を表示します。

```
# 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
```

inet6 エントリーが表示されない場合は、デバイスで **IPv6** が無効になります。

2. **/proc/sys/net/ipv6/conf/enp1s0/disable_ipv6** ファイルに値 **1** が含まれていることを確認します。

```
# cat /proc/sys/net/ipv6/conf/enp1s0/disable_ipv6
1
```

値が **1** の場合は、デバイスに対して **IPv6** が無効になります。

第18章 ホスト名の変更

システムのホスト名は、システム自体の名前になります。RHEL のインストール時に名前を設定し、後で変更できます。

18.1. NMCLI を使用したホスト名の変更

nmcli ユーティリティーを使用して、システムのホスト名を更新できます。その他のユーティリティーは、静的または永続的なホスト名などの別の用語を使用する可能性があることに注意してください。

手順

1. オプション: 現在のホスト名設定を表示します。

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

2. 新しいホスト名を設定します。

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

3. NetworkManager は、**systemd-hostnamed** を自動的に再起動して、新しい名前をアクティブにします。変更を有効にするには、ホストを再起動します。

```
# reboot
```

あるいは、そのホスト名を使用するサービスがわかっている場合は、次のようにします。

a. サービスの起動時にホスト名のみを読み取るすべてのサービスを再起動します。

```
# systemctl restart <service_name>
```

b. 変更を反映するには、アクティブなシェルユーザーを再ログインする必要があります。

検証

- ホスト名を表示します。

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

18.2. HOSTNAMECTL を使用したホスト名の変更

hostnamectl ユーティリティーを使用してホスト名を更新できます。デフォルトでは、このユーティリティーは以下のホスト名タイプを設定します。

- 静的ホスト名:**/etc/hostname** ファイルに保存されます。通常、サービスはこの名前をホスト名として使用します。
- Pretty ホスト名:**Proxy server in data center A** などのわかりやすい名前。
- 一時的なホスト名:通常、ネットワーク設定から受け取るフォールバック値。

手順

1. オプション: 現在のホスト名設定を表示します。

```
# hostnamectl status --static
old-hostname.example.com
```

2. 新しいホスト名を設定します。

```
# hostnamectl set-hostname new-hostname.example.com
```

このコマンドは、static、pretty、および transient のホスト名を新しい値に設定します。特定のタイプのみを設定するには、**--static** オプション、**--pretty** オプション、または **--transient** オプションをコマンドに渡します。

3. **hostnamectl** ユーティリティーは、**systemd-hostnamed** を自動的に再起動して、新しい名前をアクティブにします。変更を有効にするには、ホストを再起動します。

```
# reboot
```

あるいは、そのホスト名を使用するサービスがわかっている場合は、次のようにします。

a. サービスの起動時にホスト名のみを読み取るすべてのサービスを再起動します。

```
# systemctl restart <service_name>
```

b. 変更を反映するには、アクティブなシェルユーザーを再ログインする必要があります。

検証

- ホスト名を表示します。

```
# hostnamectl status --static
new-hostname.example.com
```

関連情報

- **hostnamectl(1)**
- **systemd-hostnamed.service(8)**

第19章 NETWORKMANAGER の DHCP の設定

NetworkManager は、DHCP に関連するさまざまな設定オプションを提供します。たとえば、ビルトイン DHCP クライアント (デフォルト) または外部クライアントを使用するように NetworkManager を設定したり、個々のプロファイルの DHCP 設定に影響を与えることができます。

19.1. NETWORKMANAGER の DHCP クライアントの変更

デフォルトでは、NetworkManager は内部 DHCP クライアントを使用します。ただし、ビルトインクライアントが提供しない機能を備えた DHCP クライアントが必要な場合は、代わりに **dhclient** を使用するように NetworkManager を設定できます。

RHEL は **dhcpcd** を提供しないため、NetworkManager はこのクライアントを使用できないことに注意してください。

手順

1. 次のコンテンツで **/etc/NetworkManager/conf.d/dhcp-client.conf** ファイルを作成します。

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

   dhcp パラメーターを **internal** (デフォルト) または **dhclient** に設定できます。

2. **dhcp** パラメーターを **dhclient** に設定した場合は、**dhcp-client** パッケージをインストールします。

   ```
   # dnf install dhcp-client
   ```

3. NetworkManager を再起動します。

   ```
   # systemctl restart NetworkManager
   ```

   再起動すると、すべてのネットワーク接続が一時的に中断されることに注意してください。

検証

- **/var/log/messages** ログファイルで、次のようなエントリーを検索します。

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

  このログエントリーは、NetworkManager が DHCP クライアントとして **dhclient** を使用していることを確認します。

関連情報

- **NetworkManager.conf(5)** man ページ

19.2. NETWORKMANAGER 接続の DHCP 動作の設定

DHCP (Dynamic Host Configuration Protocol) クライアントは、クライアントがネットワークに接続するたびに、動的 IP アドレスと対応する設定情報を DHCP サーバーに要求します。

DHCP サーバーから IP アドレスを取得するように接続を設定すると、NetworkManager は DHCP サーバーから IP アドレスを要求します。デフォルトでは、クライアントはこのリクエストが完了するまで 45 秒待機します。dhcp クライアントは、**DHCP** 接続が開始する際に、**DHCP** サーバーに IP アドレスを要求します。

前提条件

- DHCP を使用する接続がホストに設定されている。

手順

1. **ipv4.dhcp-timeout** および **ipv6.dhcp-timeout** プロパティーを設定します。たとえば、両方のオプションを **30** 秒に設定するには、次のコマンドを実行します。

```
# nmcli connection modify connection_name ipv4.dhcp-timeout 30 ipv6.dhcp-timeout 30
```

   パラメーターを **infinity** に設定すると、成功するまで NetworkManager が IP アドレスのリクエストおよび更新を停止しないようにします。

2. オプション: タイムアウト前に NetworkManager が IPv4 アドレスを受信しない場合にこの動作を設定します。

```
# nmcli connection modify connection_name ipv4.may-fail value
```

   ipv4.may-fail オプションを以下のように設定します。

   - **はい**、接続の状態は IPv6 設定により異なります。
     - IPv6 設定が有効になり、成功すると、NetworkManager は IPv6 接続をアクティブにし、IPv4 接続のアクティブ化を試みなくなります。
     - IPv6 設定が無効であるか設定されていないと、接続は失敗します。
   - **いいえ**、接続は非アクティブになります。この場合は、以下のようになります。
     - 接続の **autoconnect** プロパティーが有効になっている場合、NetworkManager は、**autoconnect-retries** プロパティーに設定された回数だけ、接続のアクティベーションを再試行します。デフォルト値は **4** です。
     - それでも接続が DHCP アドレスを取得できないと、自動アクティベーションは失敗します。5 分後に自動接続プロセスが再開され、DHCP サーバーから IP アドレスを取得するようになりました。

3. オプション: 必要に応じて、タイムアウト前に NetworkManager が IPv6 アドレスを受信しない場合にこの動作を設定します。

```
# nmcli connection modify connection_name ipv6.may-fail value
```

関連情報

- **nm-settings(5)** man ページ

第20章 NETWORKMANAGER で DISPATCHER スクリプトを使用して DHCLIENT の終了フックを実行する

NetworkManager の dispatcher スクリプトを使用して、**dhclient** の終了フックを実行できます。

20.1. NETWORKMANAGER の DISPATCHER スクリプトの概念

NetworkManager-dispatcher サービスは、ネットワークイベントが発生した場合に、ユーザーが提供したスクリプトをアルファベット順に実行します。通常、これらのスクリプトはシェルスクリプトですが、任意の実行可能スクリプトまたはアプリケーションにすることができます。たとえば、dispatcher スクリプトを使用して、NetworkManager では管理できないネットワーク関連の設定を調整できます。

dispatcher スクリプトは、以下のディレクトリーに保存できます。

- **/etc/NetworkManager/dispatcher.d/**:**root** ユーザーが変更できるディスパッチャースクリプトの全般的な場所。
- **/usr/lib/NetworkManager/dispatcher.d/**:事前にデプロイされた不変のディスパッチャースクリプト用。

セキュリティー上の理由から、**NetworkManager-dispatcher** では、以下の条件が満たされた場合にのみスクリプトを実行します。

- このスクリプトは、**root** ユーザーが所有します。
- このスクリプトは、**root** でのみ読み取りと書き込みが可能です。
- **setuid** ビットはスクリプトに設定されていません。

NetworkManager-dispatcher サービスは、2 つの引数を指定して、それぞれのスクリプトを実行します。

1. 操作が発生したデバイスのインターフェイス名。
2. インターフェイスがアクティブになったときの動作 (**up** など)。

NetworkManager(8) の man ページの **Dispatcher scripts** セクションには、スクリプトで使用できるアクションと環境変数の概要が記載されています。

NetworkManager-dispatcher サービスは、一度に 1 つのスクリプトを実行しますが、NetworkManager のメインプロセスとは非同期に実行します。スクリプトがキューに入れられている場合、後のイベントによってスクリプトが廃止された場合でも、サービスは常にスクリプトを実行することに注意してください。ただし、**NetworkManager-dispatcher** サービスは、以前のスクリプトの終了を待たずに、**/etc/NetworkManager/dispatcher.d/no-wait.d/** 内のファイルを参照するシンボリックリンクであるスクリプトを即座に、そして並行して実行します。

関連情報

- **NetworkManager(8)** man ページ

20.2. DHCLIENT の終了フックを実行する NETWORKMANAGER の DISPATCHER スクリプトの作成

DHCP サーバーが IPv4 アドレスを割り当てまたは更新すると、NetworkManager は **/etc/dhcp/dhclient-exit-hooks.d/** ディレクトリーに保存されている dispatcher スクリプトを実行できます。この dispatcher スクリプトは、**dhclient** の終了フックなどを実行できます。

前提条件

- **dhclient** の終了フックは、**/etc/dhcp/dhclient-exit-hooks.d/** ディレクトリーに保存されます。

手順

1. 以下の内容で **/etc/NetworkManager/dispatcher.d/12-dhclient-down** ファイルを作成します。

```
#!/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. **root** ユーザーをファイルの所有者として設定します。

```
# chown root:root /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

3. 権限を設定して、root ユーザーのみが実行できるようにします。

```
# chmod 0700 /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

4. SELinux コンテキストを復元します。

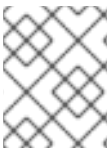
```
# restorecon /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

関連情報

- **NetworkManager(8)** man ページ

第21章 /ETC/RESOLV.CONF ファイルの手動設定

デフォルトでは、NetworkManager は、アクティブな NetworkManager 接続プロファイルの DNS 設定を使用して **/etc/resolv.conf** ファイルを動的に更新します。ただし、この動作を無効にし、**/etc/resolv.conf** で DNS 設定を手動で設定できます。



注記

または、**/etc/resolv.conf** で特定の DNS サーバーの順序が必要な場合は、DNS サーバーの順序の設定 を参照してください。

21.1. NETWORKMANAGER 設定で DNS 処理の無効化

デフォルトでは、NetworkManager は **/etc/resolv.conf** ファイルで DNS 設定を管理し、DNS サーバーの順序を設定できます。または、**/etc/resolv.conf** で DNS 設定を手動で設定する場合は、NetworkManager で DNS 処理を無効にできます。

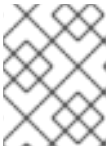
手順

1. root ユーザーとして、テキストエディターを使用して、以下の内容で **/etc/NetworkManager/conf.d/90-dns-none.conf** ファイルを作成します。

```
[main]
dns=none
```

2. **NetworkManager** サービスを再読み込みします。

```
# systemctl reload NetworkManager
```



注記

サービスを再読み込みすると、NetworkManager は **/etc/resolv.conf** ファイルを更新しなくなります。ただし、ファイルの最後の内容は保持されます。

3. 必要に応じて、混乱を避けるために、**NetworkManager により生成された** コメントを **/etc/resolv.conf** から削除します。

検証

1. **/etc/resolv.conf** ファイルを編集し、設定を手動で更新します。
2. **NetworkManager** サービスを再読み込みします。

```
# systemctl reload NetworkManager
```

3. **/etc/resolv.conf** ファイルを表示します。

```
# cat /etc/resolv.conf
```

DNS 処理を無効にできた場合、NetworkManager は手動で設定した設定を上書きしませんでした。

トラブルシューティング

- NetworkManager 設定を表示して、優先度の高い他の設定ファイルが設定をオーバーライドしていないことを確認します。

```
# NetworkManager --print-config
...
dns=none
...
```

関連情報

- **NetworkManager.conf(5)** man ページ
- NetworkManager を使用した DNS サーバーの順序の設定

21.2. /ETC/RESOLV.CONF を、DNS 設定を手動で設定するシンボリックリンクに置き換え

デフォルトでは、NetworkManager は **/etc/resolv.conf** ファイルで DNS 設定を管理し、DNS サーバーの順序を設定できます。または、**/etc/resolv.conf** で DNS 設定を手動で設定する場合は、NetworkManager で DNS 処理を無効にできます。たとえば、**/etc/resolv.conf** がシンボリックリンクの場合、NetworkManager は DNS 設定を自動的に更新しません。

前提条件

- NetworkManager **rc-manager** 設定オプションは、**ファイル** に設定されていません。検証には、**NetworkManager --print-config** コマンドを使用します。

手順

1. **/etc/resolv.conf.manually-configured** などのファイルを作成し、お使いの環境の DNS 設定を追加します。元の **/etc/resolv.conf** と同じパラメーターと構文を使用します。
2. **/etc/resolv.conf** ファイルを削除します。

```
# rm /etc/resolv.conf
```

3. **/etc/resolv.conf.manually-configured** を参照する **/etc/resolv.conf** という名前のシンボリックリンクを作成します。

```
# ln -s /etc/resolv.conf.manually-configured /etc/resolv.conf
```

関連情報

- **resolv.conf (5)** man ページ
- **NetworkManager.conf(5)** man ページ
- NetworkManager を使用した DNS サーバーの順序の設定

第22章 DNS サーバーの順序の設定

ほとんどのアプリケーションは、**glibc** ライブラリーの **getaddrinfo()** 関数を使用して DNS 要求を解決します。デフォルトでは、**glibc** はすべての DNS 要求を、**/etc/resolv.conf** ファイルで指定された最初の DNS サーバーに送信します。このサーバーが応答しない場合、RHEL は、このファイルに指定されている次のサーバーを使用します。NetworkManager を使用すると、**etc/resolv.conf** 内の DNS サーバーの順序に影響を与えることができます。

22.1. NETWORKMANAGER が /ETC/RESOLV.CONF で DNS サーバーを順序付ける方法

NetworkManager は、以下のルールに基づいて **/etc/resolv.conf** ファイルの DNS サーバーの順序を付けます。

- 接続プロファイルが 1 つしか存在しない場合、NetworkManager は、その接続で指定された IPv4 および IPv6 の DNS サーバーの順序を使用します。
- 複数の接続プロファイルがアクティベートされると、NetworkManager は DNS の優先度の値に基づいて DNS サーバーを順序付けます。DNS の優先度を設定すると、NetworkManager の動作は、**dns** パラメーターに設定した値によって異なります。このパラメーターは、**/etc/NetworkManager/NetworkManager.conf** ファイルの **[main]** セクションで設定できます。
  - **dns=default** または **dns** パラメーターが設定されていないと、以下のようになります。NetworkManager は、各接続の **ipv4.dns-priority** パラメーターおよび **ipv6.dns-priority** パラメーターに基づいて、複数の接続から DNS サーバーを順序付けます。

    値を指定しない場合、または **ipv4.dns-priority** および **ipv6.dns-priority** を **0** に設定すると、NetworkManager はグローバルのデフォルト値を使用します。DNS 優先度パラメーターのデフォルト値 を参照してください。
  - **dns=dnsmasq** または **dns=systemd-resolved**:
    この設定のいずれかを使用すると、NetworkManager は **dnsmasq** の **127.0.0.1** に設定するか、**127.0.0.53** を **nameserver** エントリーとして **/etc/resolv.conf** ファイルに設定します。

    dnsmasq サービスおよび **systemd-resolved** サービスの両方で、NetworkManager 接続に設定された検索ドメインのクエリーをその接続で指定された DNS サーバーに転送し、その他のドメインへのクエリーをデフォルトのルートを持つ接続に転送します。複数の接続に同じ検索ドメインが設定されている場合は、**dnsmasq** および **systemd-resolved** が、このドメインのクエリーを、優先度の値が最も低い接続に設定された DNS サーバーへ転送します。

DNS 優先度パラメーターのデフォルト値
NetworkManager は、接続に以下のデフォルト値を使用します。

- VPN 接続の場合は **50**
- 他の接続の場合は **100**

有効な DNS 優先度の値:
グローバルのデフォルトおよび接続固有の **ipv4.dns-priority** パラメーターおよび **ipv6.dns-priority** パラメーターの両方を **-2147483647** から **2147483647** までの値に設定できます。

- 値が小さいほど優先度が高くなります。

- 負の値は、値が大きい他の設定を除外する特別な効果があります。たとえば、優先度が負の値の接続が 1 つでも存在する場合は、NetworkManager が、優先度が最も低い接続プロファイルで指定された DNS サーバーのみを使用します。
- 複数の接続の DNS の優先度が同じ場合、NetworkManager は以下の順番で DNS の優先順位を決定します。
  a. VPN 接続。
  b. アクティブなデフォルトルートとの接続。アクティブなデフォルトルートは、メトリックスが最も低いデフォルトルートです。

関連情報

- **nm-settings(5)** man ページ
- 異なるドメインでの各種 DNS サーバーの使用

22.2. NETWORKMANAGER 全体でデフォルトの DNS サーバー優先度の値の設定

NetworkManager は、接続に以下の DNS 優先度のデフォルト値を使用します。

- VPN 接続の場合は **50**
- 他の接続の場合は **100**

これらのシステム全体のデフォルトは、IPv4 接続および IPv6 接続のカスタムデフォルト値で上書きできます。

手順

1. **/etc/NetworkManager/NetworkManager.conf** ファイルを編集します。
   a. **[connection]** セクションが存在しない場合は追加します。

   ```
   [connection]
   ```

   b. **[connection]** セクションにカスタムのデフォルト値を追加します。たとえば、IPv4 と IPv6 の両方で新しいデフォルトを **200** に設定するには、以下を追加します。

   ```
   ipv4.dns-priority=200
   ipv6.dns-priority=200
   ```

   パラメーターは、**-2147483647** から **2147483647** までの値に設定できます。パラメーターを **0** に設定すると、組み込みのデフォルト (VPN 接続の場合は **50**、他の接続の場合は **100**) が有効になります。
2. **NetworkManager** サービスを再読み込みします。

   ```
   # systemctl reload NetworkManager
   ```

関連情報

- **NetworkManager.conf(5)** man ページ

22.3. NETWORKMANAGER 接続の DNS 優先度の設定

特定の DNS サーバーの順序が必要な場合は、接続プロファイルに優先度の値を設定できます。NetworkManager はこれらの値を使用して、サービスが **/etc/resolv.conf** ファイルを作成または更新する際にサーバーを順序付けます。

DNS 優先度の設定は、異なる DNS サーバーが設定された複数の接続がある場合にのみ有効であることに注意してください。複数の DNS サーバーが設定された接続が 1 つしかない場合は、接続プロファイルで DNS サーバーを優先順に手動で設定します。

前提条件

- システムに NetworkManager の接続が複数設定されている。
- システムで、**/etc/NetworkManager/NetworkManager.conf** ファイルに **dns** パラメーターが設定されていないか、そのパラメーターが **default** に設定されている。

手順

1. 必要に応じて、利用可能な接続を表示します。

```
# 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. **ipv4.dns-priority** パラメーターおよび **ipv6.dns-priority** パラメーターを設定します。たとえば、**Example_con_1** 接続に対して、両方のパラメーターを **10** に設定するには、次のコマンドを実行します。

```
# nmcli connection modify Example_con_1 ipv4.dns-priority 10 ipv6.dns-priority 10
```

3. 必要に応じて、他のコネクションに対しても 1 つ前の手順を繰り返します。
4. 更新した接続を再度アクティブにします。

```
# nmcli connection up Example_con_1
```

検証

- **/etc/resolv.conf** ファイルの内容を表示して、DNS サーバーの順序が正しいことを確認します。

```
# cat /etc/resolv.conf
```

第23章 異なるドメインでの各種 DNS サーバーの使用

デフォルトでは、Red Hat Enterprise Linux (RHEL) は、すべての DNS リクエストを、**/etc/resolv.conf** ファイルで指定されている最初の DNS サーバーに送信します。このサーバーが応答しない場合、RHEL は、このファイルに指定されている次のサーバーを使用します。ある DNS サーバーがすべてのドメインを解決できない環境では、管理者は、特定のドメインの DNS 要求を選択した DNS サーバーに送信するように RHEL を設定できます。

たとえば、サーバーを仮想プライベートネットワーク (VPN) に接続し、VPN 内のホストが **example.com** ドメインを使用するとします。この場合、次の方法で DNS クエリーを処理するように RHEL を設定できます。

- **example.com** の DNS 要求のみを VPN ネットワーク内の DNS サーバーに送信します。
- 他のすべての要求は、デフォルトゲートウェイを使用して接続プロファイルで設定されている DNS サーバーに送信します。

23.1. NETWORKMANAGER で DNSMASQ を使用して、特定のドメインの DNS リクエストを選択した DNS サーバーに送信する

dnsmasq のインスタンスを開始するように NetworkManager を設定できます。次に、この DNS キャッシュサーバーは、**loopback** デバイスのポート **53** をリッスンします。したがって、このサービスはローカルシステムからのみ到達でき、ネットワークからは到達できません。

この設定では、NetworkManager は **nameserver 127.0.0.1** エントリーを **/etc/resolv.conf** ファイルに追加し、**dnsmasq** は DNS 要求を NetworkManager 接続プロファイルで指定された対応する DNS サーバーに動的にルーティングします。

前提条件

- システムに NetworkManager の接続が複数設定されている。
- DNS サーバーおよび検索ドメインは、特定のドメインを解決する NetworkManager 接続プロファイルで設定されます。
  たとえば、VPN 接続で指定された DNS サーバーが **example.com** ドメインのクエリーを解決するようにするには、VPN 接続プロファイルに以下の設定が含まれている必要があります。
  - **example.com** を解決できる DNS サーバー
  - **ipv4.dns-search** および **ipv6.dns-search** パラメーターで **example.com** に設定された検索ドメイン
- **dnsmasq** サービスが実行されていないか、**localhost** とは異なるインターフェイスでリッスンするように設定されています。

手順

1. **dnsmasq** パッケージをインストールします。

```
# dnf install dnsmasq
```

2. **/etc/NetworkManager/NetworkManager.conf** ファイルを編集し、**[main]** セクションに以下のエントリーを設定します。

```
dns=dnsmasq
```

3. **NetworkManager** サービスを再読み込みします。

```
# systemctl reload NetworkManager
```

検証

1. **NetworkManager** ユニットの **systemd** ジャーナルで、サービスが別の DNS サーバーを使用しているドメインを検索します。

```
# journalctl -xeu NetworkManager
...
Jun 02 13:30:17 client_hostname dnsmasq[5298]: using nameserver 198.51.100.7#53 for
domain example.com
...
```

2. **tcpdump** パケットスニファを使用して、DNS 要求の正しいルートを確認します。

   a. **tcpdump** パッケージをインストールします。

```
# dnf install tcpdump
```

   b. 1 つのターミナルで **tcpdump** を起動し、すべてのインターフェイスで DNS トラフィックを取得します。

```
# tcpdump -i any port 53
```

   c. 別のターミナルで、例外が存在するドメインと別のドメインのホスト名を解決します。次に例を示します。

```
# host -t A www.example.com
# host -t A www.redhat.com
```

   d. **tcpdump** 出力で、Red Hat Enterprise Linux が **example.com** ドメインの DNS クエリーのみを指定された DNS サーバーに、対応するインターフェイスを通じて送信していることを確認します。

```
...
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 は、**www.example.com** の DNS クエリーを **198.51.100.7** の DNS サーバーに送信し、**www.redhat.com** のクエリーを **192.0.2.1** に送信します。

トラブルシューティング

1. **/etc/resolv.conf** ファイルの **nameserver** エントリーが **127.0.0.1** を指していることを確認します。

```
# cat /etc/resolv.conf
nameserver 127.0.0.1
```

エントリーがない場合は、**/etc/NetworkManager/NetworkManager.conf** ファイルの **dns** パラメーターを確認します。

2. **dnsmasq** サービスが **loopback** デバイスのポート **53** でリッスンしていることを確認します。

```
# 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))
```

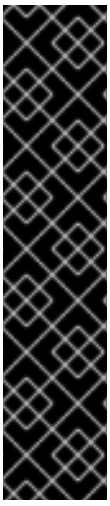
サービスが **127.0.0.1:53** をリッスンしていない場合は、**NetworkManager** ユニットのジャーナルエントリーを確認します。

```
# journalctl -u NetworkManager
```

23.2. NETWORKMANAGER で SYSTEMD-RESOLVED を使用して、特定のドメインの DNS 要求を選択した DNS サーバーに送信する

NetworkManager を設定して、**systemd-resolved** のインスタンスを開始することができます。次に、この DNS スタブリゾルバーは、IP アドレス **127.0.0.53** のポート **53** でリッスンします。したがって、このスタブリゾルバーはローカルシステムからのみ到達でき、ネットワークからは到達できません。

この設定では、NetworkManager は **nameserver 127.0.0.53** エントリーを **/etc/resolv.conf** ファイルに追加し、**systemd-resolved** は、NetworkManager 接続プロファイルで指定された対応する DNS サーバーに DNS 要求を動的にルーティングします。



重要

systemd-resolved サービスは、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

サポートされるソリューションについては、Using dnsmasq in NetworkManager to send DNS requests for a specific domain to a selected DNS server を参照してください。

前提条件

- システムに NetworkManager の接続が複数設定されている。
- DNS サーバーおよび検索ドメインは、特定のドメインを解決する NetworkManager 接続プロファイルで設定されます。
  たとえば、VPN 接続で指定された DNS サーバーが **example.com** ドメインのクエリーを解決するようにするには、VPN 接続プロファイルに以下の設定が含まれている必要があります。
  - **example.com** を解決できる DNS サーバー

- **ipv4.dns-search** および **ipv6.dns-search** パラメーターで **example.com** に設定された検索ドメイン

手順

1. **systemd-resolved** サービスを有効にして起動します。

```
# systemctl --now enable systemd-resolved
```

2. **/etc/NetworkManager/NetworkManager.conf** ファイルを編集し、**[main]** セクションに以下のエントリーを設定します。

```
dns=systemd-resolved
```

3. **NetworkManager** サービスを再読み込みします。

```
# systemctl reload NetworkManager
```

検証

1. **systemd-resolved** が使用する DNS サーバーと、サービスが別の DNS サーバーを使用するドメインを表示します。

```
# 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
```

この出力では、**systemd-resolved** が **example.com** ドメインに異なる DNS サーバーを使用していることを確認します。

2. **tcpdump** パケットスニファを使用して、DNS 要求の正しいルートを確認します。

a. **tcpdump** パッケージをインストールします。

```
# dnf install tcpdump
```

b. 1 つのターミナルで **tcpdump** を起動し、すべてのインターフェイスで DNS トラフィックを取得します。

```
# tcpdump -i any port 53
```

c. 別のターミナルで、例外が存在するドメインと別のドメインのホスト名を解決します。次に例を示します。

```
# host -t A www.example.com
# host -t A www.redhat.com
```

d. **tcpdump** 出力で、Red Hat Enterprise Linux が **example.com** ドメインの DNS クエリーのみを指定された DNS サーバーに、対応するインターフェイスを通じて送信していることを確認します。

```
...
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 は、**www.example.com** の DNS クエリーを **198.51.100.7** の DNS サーバーに送信し、**www.redhat.com** のクエリーを **192.0.2.1** に送信します。

トラブルシューティング

1. **/etc/resolv.conf** ファイルの **nameserver** エントリーが **127.0.0.53** を指していることを確認します。

```
# cat /etc/resolv.conf
nameserver 127.0.0.53
```

エントリーがない場合は、**/etc/NetworkManager/NetworkManager.conf** ファイルの **dns** パラメーターを確認します。

2. **systemd-resolved** サービスがローカルの IP アドレス **127.0.0.53** の **53** ポートでリッスンしていることを確認します。

```
# 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))
```

サービスが **127.0.0.53:53** をリッスンしない場合は、**systemd-resolved** サービスが実行されているかどうかを確認します。

第24章 デフォルトのゲートウェイ設定の管理

デフォルトゲートウェイは、他のルートがパケットの宛先と一致する場合にネットワークパケットを転送するルーターです。ローカルネットワークでは、通常、デフォルトゲートウェイは、インターネットの近くの1ホップのホストです。

24.1. NMCLI を使用した既存の接続でデフォルトのゲートウェイ設定

ほとんどの場合、管理者は、nmcli を使用したイーサネット接続の設定 などの説明に従って接続を作成する場合のデフォルトのゲートウェイを設定します。

ほとんどの場合、管理者は、接続を作成する場合のデフォルトのゲートウェイを設定します。ただし、**nmcli** ユーティリティーを使用して、以前に作成した接続でデフォルトのゲートウェイ設定を設定したり、更新したりできます。

前提条件

- デフォルトゲートウェイが設定されている接続で、静的 IP アドレスを少なくとも1つ設定している。
- 物理コンソールにログインしている場合は、十分な権限を有している。それ以外の場合は、**root** 権限が必要になります。

手順

1. デフォルトゲートウェイの IP アドレスを設定します。
   たとえば、**example** 接続のデフォルトゲートウェイの IPv4 アドレスを **192.0.2.1** に設定するには、次のコマンドを実行します。

   ```
   # nmcli connection modify example ipv4.gateway "192.0.2.1"
   ```

   たとえば、**example** 接続のデフォルトゲートウェイの IPv6 アドレスを **2001:db8:1::1** に設定するには、次のコマンドを実行します。

   ```
   # nmcli connection modify example ipv6.gateway "2001:db8:1::1"
   ```

2. ネットワーク接続を再起動して、変更を有効にします。たとえば、コマンドラインで **example** 接続を再起動するには、次のコマンドを実行します。

   ```
   # nmcli connection up example
   ```

   警告

   このネットワーク接続を現在使用しているすべての接続が、再起動時に一時的に中断されます。

3. 必要に応じて、ルートがアクティブであることを確認します。
   IPv4 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

IPv6 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

24.2. NMCLI インタラクティブモードを使用した既存の接続でのデフォルトゲートウェイ設定

ほとんどの場合、管理者は、nmcli インタラクティブエディターを使用したイーサネット接続の設定 などの説明に従って、接続を作成する場合のデフォルトのゲートウェイを設定します。

ほとんどの場合、管理者は、接続を作成する場合のデフォルトのゲートウェイを設定します。ただし、**nmcli** ユーティリティーのインタラクティブモードを使用して、以前に作成した接続でデフォルトのゲートウェイを設定したり、更新したりすることもできます。

前提条件

- デフォルトゲートウェイが設定されている接続で、静的 IP アドレスを少なくとも 1 つ設定している。
- 物理コンソールにログインしている場合は、十分な権限を有している。それ以外の場合は、**root** 権限が必要になります。

手順

1. 必要な接続に対して **nmcli** インタラクティブモードを開きます。たとえば、**example** 接続の **nmcli** インタラクティブモードを開くには、次のコマンドを実行します。

```
# nmcli connection edit example
```

2. デフォルトのゲートウェイを設定します。
たとえば、**example** 接続のデフォルトゲートウェイの IPv4 アドレスを **192.0.2.1** に設定するには、次のコマンドを実行します。

```
nmcli> set ipv4.gateway 192.0.2.1
```

たとえば、**example** 接続のデフォルトゲートウェイの IPv6 アドレスを **2001:db8:1::1** に設定するには、次のコマンドを実行します。

```
nmcli> set ipv6.gateway 2001:db8:1::1
```

3. 必要に応じて、デフォルトゲートウェイが正しく設定されていることを確認します。

```
nmcli> print
...
ipv4.gateway:                   192.0.2.1
...
ipv6.gateway:                   2001:db8:1::1
...
```

4. 設定を保存します。

```
nmcli> save persistent
```

5. ネットワーク接続を再起動して、変更を有効にします。

```
nmcli> activate example
```

警告

このネットワーク接続を現在使用しているすべての接続が、再起動時に一時的に中断されます。

6. **nmcli** インタラクティブモードを終了します。

```
nmcli> quit
```

7. 必要に応じて、ルートがアクティブであることを確認します。
IPv4 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

IPv6 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

関連情報

- nmcli インタラクティブエディターを使用したイーサネット接続の設定

24.3. NM-CONNECTION-EDITOR を使用した既存の接続でのデフォルトゲートウェイ設定

ほとんどの場合、管理者は、接続を作成する場合のデフォルトのゲートウェイを設定します。ただし、**nm-connection-editor** アプリケーションを使用して、以前に作成した接続でデフォルトのゲートウェイを設定したり、更新したりすることもできます。

前提条件

- デフォルトゲートウェイが設定されている接続で、静的 IP アドレスを少なくとも 1 つ設定している。

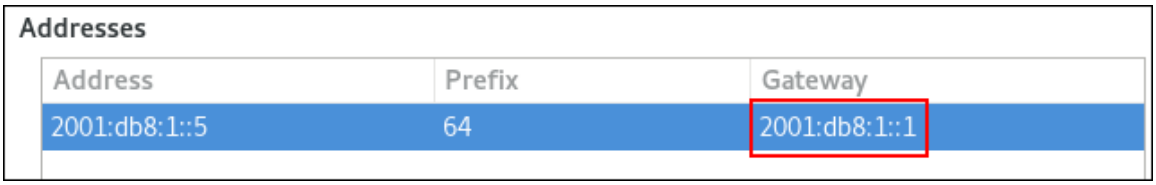
手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

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

2. 変更する接続を選択し、歯車のアイコンをクリックして、既存の接続を編集します。
3. IPv4 デフォルトゲートウェイを設定します。たとえば、その接続のデフォルトゲートウェイの IPv4 アドレスを **192.0.2.1** に設定します。
   a. **IPv4 Settings** タブを開きます。
   b. そのゲートウェイのアドレスが含まれる IP アドレスの範囲の隣の **gateway** フィールドにアドレスを入力します。

4. IPv6 デフォルトゲートウェイを設定します。たとえば、接続のデフォルトゲートウェイの IPv6 アドレスを **2001:db8:1::1** に設定するには、以下を行います。
   a. **IPv6** タブを開きます。
   b. そのゲートウェイのアドレスが含まれる IP アドレスの範囲の隣の **gateway** フィールドにアドレスを入力します。



5. **OK** をクリックします。
6. **Save** をクリックします。
7. ネットワーク接続を再起動して、変更を有効にします。たとえば、コマンドラインで **example** 接続を再起動するには、次のコマンドを実行します。

```
# nmcli connection up example
```

警告

このネットワーク接続を現在使用しているすべての接続が、再起動時に一時的に中断されます。

8. 必要に応じて、ルートがアクティブであることを確認します。
   IPv4 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

IPv6 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
# ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

関連情報

- nm-connection-editor を使用したイーサネット接続の設定

24.4. CONTROL-CENTER を使用した既存の接続でのデフォルトゲートウェイ設定

ほとんどの場合、管理者は、接続を作成する場合のデフォルトのゲートウェイを設定します。ただし、**control-center** アプリケーションを使用して、以前に作成した接続でデフォルトのゲートウェイを設定したり、更新したりできます。

前提条件

- デフォルトゲートウェイが設定されている接続で、静的 IP アドレスを少なくとも 1 つ設定している。
- **control-center** アプリケーションで、接続のネットワーク設定を開いている。

手順

1. IPv4 デフォルトゲートウェイを設定します。たとえば、その接続のデフォルトゲートウェイの IPv4 アドレスを **192.0.2.1** に設定します。
   a. **IPv4** タブを開きます。
   b. そのゲートウェイのアドレスが含まれる IP アドレスの範囲の隣の **gateway** フィールドにアドレスを入力します。

2. IPv6 デフォルトゲートウェイを設定します。たとえば、接続のデフォルトゲートウェイの IPv6 アドレスを **2001:db8:1::1** に設定するには、以下を行います。
   a. **IPv6** タブを開きます。
   b. そのゲートウェイのアドレスが含まれる IP アドレスの範囲の隣の **gateway** フィールドにアドレスを入力します。

3. **Apply** をクリックします。

4. **Network** ウィンドウに戻り、接続のボタンを **Off** に切り替えてから **On** に戻して、接続を無効にして再度有効にし、変更を適用します。

警告

このネットワーク接続を現在使用しているすべての接続が、再起動時に一時的に中断されます。

5. 必要に応じて、ルートがアクティブであることを確認します。
IPv4 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

IPv6 デフォルトゲートウェイを表示するには、次のコマンドを実行します。

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

関連情報

- control-center によるイーサネット接続の設定

24.5. NMSTATECTL を使用した既存の接続でのデフォルトゲートウェイ設定

nmstatectl ユーティリティーを使用して、Nmstate API を介してデフォルトゲートウェイを設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

前提条件

- デフォルトゲートウェイが設定されている接続で、静的 IP アドレスを少なくとも 1 つ設定している。
- **enp1s0** インターフェイスが設定され、デフォルトゲートウェイの IP アドレスがこのインターフェイスの IP 設定のサブネット内にある。
- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイル (例: ~/**set-default-gateway.yml**) を作成します。

```
---
routes:
  config:
```

```
- destination: 0.0.0.0/0
  next-hop-address: 192.0.2.1
  next-hop-interface: enp1s0
```

これらの設定では、**192.0.2.1** をデフォルトゲートウェイとして定義します。デフォルトゲートウェイは **enp1s0** インターフェイス経由で到達可能です。

2. 設定をシステムに適用します。

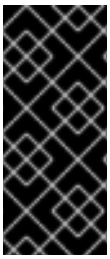
```
# nmstatectl apply ~/set-default-gateway.yml
```

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

24.6. NETWORK RHEL システムロールを使用して既存の接続にデフォルトゲートウェイを設定する

network RHEL システムロールを使用して、デフォルトゲートウェイを設定できます。



重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

この手順では、すでに存在するかどうかに応じて、以下の設定で **enp1s0** 接続プロファイルを作成または更新します。

- 静的 IPv4 アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **198.51.100.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **198.51.100.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。

- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

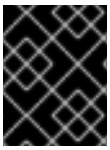
24.7. NETWORKMANAGER が複数のデフォルトゲートウェイを管理する方法

フォールバック上の理由で特定の状況では、ホストに複数のデフォルトゲートウェイを設定します。ただし、非同期ルーティングの問題を回避するために、同じプロトコルの各デフォルトゲートウェイには別のメトリック値が必要です。RHEL は、最も低いメトリックセットを持つデフォルトゲートウェイへ

の接続のみを使用することに注意してください。

以下のコマンドを使用して、接続の IPv4 ゲートウェイと IPv6 ゲートウェイの両方にメトリックを設定できます。

```
# nmcli connection modify connection-name ipv4.route-metric value ipv6.route-metric value
```



重要

ルーティングの問題を回避するために、複数の接続プロファイルで同じプロトコルに同じメトリック値を設定しないでください。

メトリック値なしでデフォルトのゲートウェイを設定すると、NetworkManager は、インターフェイスタイプに基づいてメトリック値を自動的に設定します。このため、NetworkManager は、アクティブな最初の接続に、このネットワークタイプのデフォルト値を割り当て、そのネットワークタイプがアクティベートされる順序で、同じタイプの他の接続にインクリメントした値を設定します。たとえば、デフォルトゲートウェイを持つ 2 つのイーサネット接続が存在する場合、NetworkManager は、ルートに **100** のメトリックを、最初にアクティブにしている接続のデフォルトゲートウェイに設定します。2 つ目の接続では、NetworkManager は **101** を設定します。

以下は、よく使用されるネットワークタイプと、そのデフォルトのメトリックの概要です。

connection.type	デフォルトのメトリック値
VPN	50
イーサネット	100
MACsec	125
Infiniband	150
bond=	300
team=	350
VLAN	400
ブリッジ	425
TUN	450
Wi-Fi	600
IP トンネル	675

関連情報

- 代替ルートを定義するポリシーベースのルーティングの設定

- Multipath TCP の使用

24.8. 特定のプロファイルでのデフォルトゲートウェイの指定を防ぐための NETWORKMANAGER の設定

NetworkManager が特定のプロファイルを使用してデフォルトゲートウェイを指定しないようにすることができます。デフォルトゲートウェイに接続されていない接続プロファイルには、以下の手順に従います。

前提条件

- デフォルトゲートウェイに接続されていない接続の NetworkManager 接続プロファイルが存在する。

手順

1. 接続で動的 IP 設定を使用する場合は、NetworkManager が、IPv4 および IPv6 接続のデフォルトルートとして接続を使用しないように設定します。

```
# nmcli connection modify connection_name ipv4.never-default yes ipv6.never-default yes
```

ipv4.never-default および **ipv6.never-default** を **yes** に設定すると、対応するプロトコルのデフォルトのゲートウェイ IP アドレスが、接続プロファイルから削除されることに注意してください。

2. 接続をアクティベートします。

```
# nmcli connection up connection_name
```

検証

- **ip -4 route** コマンドおよび **ip -6 route** コマンドを使用して、RHEL が、IPv4 プロトコルおよび IPv6 プロトコルのデフォルトルートにネットワークインターフェイスを使用しないことを確認します。

24.9. 複数のデフォルトゲートウェイによる予期しないルーティング動作の修正

マルチパス TCP を使用する場合など、ホストで複数のデフォルトゲートウェイが必要なシナリオはそれほどありません。多くの場合、ルーティングの動作や非同期ルーティングの問題を回避するために、1 つのデフォルトゲートウェイのみを設定します。

注記

異なるインターネットプロバイダーにトラフィックをルーティングするには、複数のデフォルトゲートウェイの代わりにポリシーベースのルーティングを使用します。

前提条件

- ホストは NetworkManager を使用してネットワーク接続を管理します。これはデフォルトです。

- ホストには複数のネットワークインターフェイスがある。
- ホストには複数のデフォルトゲートウェイが設定されている。

手順

1. ルーティングテーブルを表示します。
   - IPv4 の場合は、次のコマンドを実行します。

     ```
     # 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
     ...
     ```

   - IPv6 の場合は、次のコマンドを実行します。

     ```
     # 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
     ...
     ```

   default で開始するエントリーはデフォルトのルートを示します。**dev** の横に表示されるこれらのエントリーのインターフェイス名を書き留めます。

2. 以下のコマンドを使用して、前の手順で特定したインターフェイスを使用する NetworkManager 接続を表示します。

   ```
   # 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
   ```

   この例では、**Corporate-LAN** と **Internet-Provider** という名前のプロファイルにはデフォルトのゲートウェイが設定されています。これは、ローカルネットワークでは、通常、インターネット 1 ホップ近いホストがデフォルトゲートウェイであるため、この手順の残りの部分では、**Corporate-LAN** のデフォルトゲートウェイが正しくないことを想定するためです。

3. NetworkManager が、IPv4 および IPv6 接続のデフォルトルートとして **Corporate-LAN** 接続を使用しないように設定します。

   ```
   # nmcli connection modify Corporate-LAN ipv4.never-default yes ipv6.never-default yes
   ```

   ipv4.never-default および **ipv6.never-default** を **yes** に設定すると、対応するプロトコルのデフォルトのゲートウェイ IP アドレスが、接続プロファイルから削除されることに注意してください。

4. **Corporate-LAN** 接続をアクティブにします。

```
# nmcli connection up Corporate-LAN
```

検証

- IPv4 および IPv6 ルーティングテーブルを表示し、プロトコルごとに 1 つのデフォルトゲートウェイのみが利用可能であることを確認します。
  - IPv4 の場合は、次のコマンドを実行します。

```
# ip -4 route
default via 192.0.2.1 dev enp1s0 proto static metric 101
...
```

  - IPv6 の場合は、次のコマンドを実行します。

```
# ip -6 route
default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
...
```

関連情報

- 代替ルートを定義するポリシーベースのルーティングの設定
- Multipath TCP の使用

第25章 静的ルートの設定

ルーティングにより、相互に接続されたネットワーク間でトラフィックを送受信できるようになります。大規模な環境では、管理者は通常、ルーターが他のルーターについて動的に学習できるようにサービスを設定します。小規模な環境では、管理者は多くの場合、静的ルートを設定して、トラフィックが1つのネットワークから次のネットワークに確実に到達できるようにします。

次の条件がすべて当てはまる場合、複数のネットワーク間で機能する通信を実現するには、静的ルートが必要です。

- トラフィックは複数のネットワークを通過する必要があります。
- デフォルトゲートウェイを通過する排他的なトラフィックフローは十分ではありません。

「静的ルートを必要とするネットワークの例」 では、スタティックルートを設定しない場合のシナリオと、異なるネットワーク間でトラフィックがどのように流れるかについて説明します。

25.1. 静的ルートを必要とするネットワークの例

すべての IP ネットワークが 1 つのルーターを介して直接接続されているわけではないため、この例では静的ルートが必要です。スタティックルートがないと、一部のネットワークは相互に通信できません。さらに、一部のネットワークからのトラフィックは一方向にしか流れません。

注記

この例のネットワークトポロジーは人為的なものであり、静的ルーティングの概念を説明するためにのみ使用されています。これは、実稼働環境で推奨されるトポロジーではありません。

この例のすべてのネットワーク間で通信を機能させるには、Raleigh (**198.51.100.0/24**) への静的ルートを設定し、次のホップ Router 2 (**203.0.113.10**) を設定します。ネクストホップの IP アドレスは、データセンターネットワークのルーター 2 のものです (**203.0.113.0/24**)。

スタティックルートは次のように設定できます。

- 設定を簡素化するには、この静的ルートをルーター1だけに設定します。ただし、データセンター (**203.0.113.0/24**) からのホストがトラフィックを Raleigh (**198.51.100.0/24**) に送信するため、常にルーター1を経由してルーター 2 に送信されるため、ルーター1のトラフィックが増加します。
- より複雑な設定の場合、データセンター (**203.0.113.0/24**) 内のすべてのホストでこの静的ルートを設定します。このサブネット内のすべてのホストは、Raleigh (**198.51.100.0/24**) に近いルーター 2 (**203.0.113.10**) にトラフィックを直接送信します。

どのネットワーク間でトラフィックが流れるかどうかの詳細については、図の下の説明を参照してください。

必要な静的経路が設定されていないときに、通信がうまくいく場合とうまくいかない場合を以下に示します。

- ベルリンネットワークのホスト (**192.0.2.0/24**):
  - 直接接続されているため、同じサブネット内の他のホストと通信できます。
  - Router 1 はベルリンネットワーク (**192.0.2.0/24**) 内にあり、インターネットにつながるデフォルトゲートウェイがあるため、インターネットと通信できます。
  - ルーター 1 はベルリン (**192.0.2.0/24**) とデータセンター (**203.0.113.0/24**) ネットワークの両方にインターフェイスを持っているため、データセンターネットワーク (**203.0.113.0/24**) と通信できます。
  - ローリーネットワーク (**198.51.100.0/24**) と通信できません。これは、ルーター 1 がこのネットワークにインターフェイスを持たないためです。したがって、Router 1 はトラフィックを独自のデフォルトゲートウェイ (インターネット) に送信します。
- データセンターネットワーク内のホスト (**203.0.113.0/24**):
  - 直接接続されているため、同じサブネット内の他のホストと通信できます。
  - デフォルトゲートウェイがルーター 1 に設定されているため、インターネットと通信できます。ルーター 1 には、データセンター (**203.0.113.0/24**) とインターネットの両方のネットワークにインターフェイスがあります。

- デフォルトゲートウェイがルーター1に設定されているため、ベルリンネットワーク (**192.0.2.0/24**) と通信でき、ルーター1にはデータセンター (**203.0.113.0/24**) とベルリン (**192.0.2.0/24**) の両方にインターフェイスがあります。) ネットワーク。

- Raleigh ネットワーク (**198.51.100.0/24**) と通信できません。これは、データセンターネットワークがこのネットワークにインターフェイスを持たないためです。したがって、データセンター (**203.0.113.0/24**) 内のホストは、トラフィックをデフォルトゲートウェイ (ルーター1) に送信します。ルーター1も Raleigh ネットワーク (**198.51.100.0/24**) にインターフェイスを持たないため、ルーター1はこのトラフィックを独自のデフォルトゲートウェイ (インターネット) に送信します。

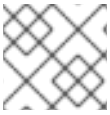
- Raleigh ネットワーク内のホスト (**198.51.100.0/24**):

  - 直接接続されているため、同じサブネット内の他のホストと通信できます。

  - インターネット上のホストと通信できません。デフォルトゲートウェイの設定により、ルーター2はトラフィックをルーター1に送信します。ルーター1の実際の動作は、リバースパスフィルター (**rp_filter**) システム制御 (**sysctl**) の設定によって異なります。RHEL のデフォルトでは、Router 1は送信トラフィックをインターネットにルーティングする代わりにドロップします。ただし、設定された動作に関係なく、スタティックルートがないと通信できません。

  - データセンターネットワーク (**203.0.113.0/24**) と通信できません。デフォルトゲートウェイの設定により、発信トラフィックはルーター2を経由して宛先に到達します。ただし、データセンターネットワーク (**203.0.113.0/24**) 内のホストがデフォルトゲートウェイ (ルーター1) に応答を送信するため、パケットへの応答は送信者に届きません。次に、Router 1がトラフィックをインターネットに送信します。

  - ベルリンのネットワーク (**192.0.2.0/24**) と通信できません。デフォルトゲートウェイの設定により、ルーター2はトラフィックをルーター1に送信します。ルーター1の実際の動作は、**rp_filter sysctl** 設定によって異なります。RHEL のデフォルトでは、Router 1は発信トラフィックを Berlin ネットワーク (**192.0.2.0/24**) に送信する代わりにドロップします。ただし、設定された動作に関係なく、スタティックルートがないと通信できません。



注記

静的ルートの設定に加え、両方のルーターで IP 転送を有効にする必要があります。

関連情報

- Why cannot a server be pinged if net.ipv4.conf.all.rp_filter is set on the server?
- Enabling IP forwarding

25.2. NMCLI コマンドを使用して、静的ルートを設定する方法

静的ルートを設定するには、次の構文で **nmcli** ユーティリティーを使用します。

```
$ nmcli connection modify connection_name ipv4.routes "ip[/prefix] [next_hop] [metric] [attribute=value] [attribute=value] ..."
```

このコマンドは、次のルート属性に対応します。

- **cwnd=n**:パケット数で定義された輻輳ウィンドウ (CWND) サイズを設定します。
- **lock-cwnd=true|false**:カーネルが CWND 値を更新できるかどうかを定義します。

- **lock-mtu=true|false**:カーネルが MTU をパス MTU ディスカバリーに更新できるかどうかを定義します。
- **lock-window=true|false**:カーネルが TCP パケットの最大ウィンドウサイズを更新できるかどうかを定義します。
- **mtu=n**:宛先へのパスに沿って使用する最大転送単位 (MTU) を設定します。
- **onlink=true|false**:ネクストホップがどのインターフェイス接頭辞とも一致しない場合でも、このリンクに直接接続されるかどうかを定義します。
- **scope=n**:IPv4 ルートの場合、この属性は、ルート 接頭辞によってカバーされる宛先の範囲を設定します。値を整数 (0〜255) として設定します。
- **src=address**:ルート接頭辞の対象となる宛先にトラフィックを送信するときに優先する送信元アドレスを設定します。
- **table=table_id**:ルートを追加するテーブルの ID を設定します。このパラメーターを省略すると、NetworkManager は **main** テーブルを使用します。
- **tos=n**:サービスのタイプ (TOS) キーを設定します。値を整数 (0〜255) として設定します。
- **type=value**:ルートタイプを設定します。NetworkManager は、**unicast**、**local**、**blackhole**、**unreachable**、**prevent**、および **throw** ルートタイプをサポートします。デフォルトは **unicast** です。
- **window=n**:これらの宛先にアドバタイズする TCP の最大ウィンドウサイズをバイト単位で設定します。

ipv4.routes サブコマンドを使用する場合は、**nmcli** が、このパラメーターの現在の設定をすべて上書きします。

ルートを追加するには:

```
$ nmcli connection modify connection_name +ipv4.routes "<route>"
```

同様に、特定のルートを削除するには:

```
$ nmcli connection modify connection_name -ipv4.routes "<route>"
```

25.3. NMCLI を使用した静的ルートの設定

nmcli connection modify コマンドを使用して、既存の NetworkManager 接続プロファイルに静的ルートを追加できます。

以下の手順では、以下の経路を設定します。

- リモート **198.51.100.0/24** ネットワークへの IPv4 ルート。IP アドレス **192.0.2.10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。
- リモート **2001:db8:2::/64** ネットワークへの IPv6 ルート。IP アドレス **2001:db8:1::10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。

前提条件

- **example** の接続プロファイルが存在し、このホストがゲートウェイと同じ IP サブネットになるように設定されています。

手順

1. **example** の接続プロファイルに静的 IPv4 ルートを追加します。

```
# nmcli connection modify example +ipv4.routes "198.51.100.0/24 192.0.2.10"
```

1 回で複数のルートを設定するには、個々のルートをコンマで区切ってコマンドに渡す必要があります。たとえば、ルートを **198.51.100.0/24** および **203.0.113.0/24** のネットワークに追加して、両方のルートが **192.0.2.10** ゲートウェイを通るには、以下のコマンドを実行します。

```
# nmcli connection modify example +ipv4.routes "198.51.100.0/24 192.0.2.10,
203.0.113.0/24 192.0.2.10"
```

2. **example** の接続プロファイルに静的 IPv6 ルートを追加します。

```
# nmcli connection modify example +ipv6.routes "2001:db8:2::/64 2001:db8:1::10"
```

3. 接続を再度有効にします。

```
# nmcli connection up example
```

検証

1. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. IPv6 ルートを表示します。

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

関連情報

- **nmcli(1)** man ページ
- **nm-settings-nmcli (5)** man ページ

25.4. NMTUI を使用した静的ルートの設定

nmtui アプリケーションは、NetworkManager 用のテキストベースのユーザーインターフェイスを提供します。**nmtui** を使用して、グラフィカルインターフェイスを使用せずにホスト上で静的ルートを設定できます。

たとえば、以下の手順では **198.51.100.1** で実行しているゲートウェイを使用する **192.0.2.0/24** ネットワークに経路を追加します。これは、既存の接続プロファイルから到達可能です。

注記

nmtui で 以下を行います。

- カーソルキーを使用してナビゲートします。
- ボタンを選択して **Enter** を押します。
- **Space** を使用して、チェックボックスを選択および選択解除します。

前提条件

- ネットワークが設定されている。
- 静的ルートのゲートウェイが、インターフェイスで直接到達できる。
- 物理コンソールにログインしている場合は、十分な権限を有している。それ以外の場合は、コマンドに root 権限が必要になります。

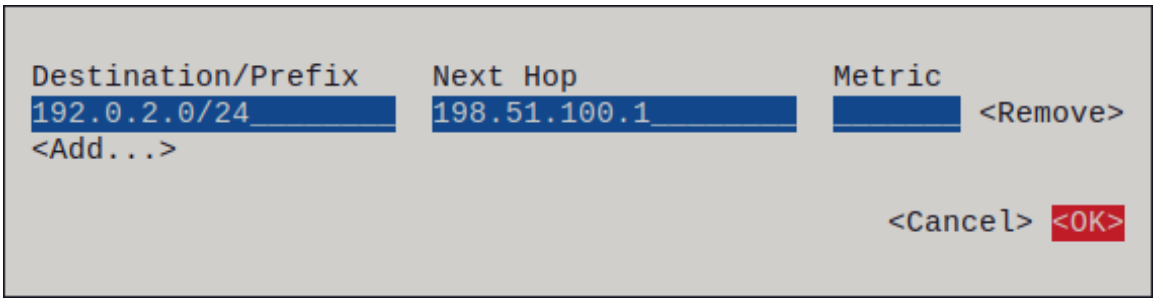
手順

1. **nmtui** を開始します。

```
# nmtui
```

2. **Edit a connection** 選択し、**Enter** を押します。
3. 宛先ネットワークへのネクストホップに到達できる接続プロファイルを選択し、**Enter** を押します。
4. IPv4 ルートまたは IPv6 ルートに応じて、プロトコルの設定エリアの横にある **Show** ボタンを押します。
5. **Routing** の横にある **Edit** ボタンを押します。これにより、静的ルートを設定する新しいウィンドウが開きます。
   a. **Add** ボタンを押して、次のように入力します。
      - Classless Inter-Domain Routing (CIDR) 形式の接頭辞を含む宛先ネットワーク
      - ネクストホップの IP アドレス
      - 同じネットワークに複数のルートを追加し、効率によってルートに優先順位を付けたい場合のメトリック値
   b. 追加するルートごとに前の手順を繰り返し、この接続プロファイルを介して到達できます。
   c. **OK** ボタンを押して、接続設定のウィンドウに戻ります。

図25.1 メトリックのない静的ルートの例



6. **OK** ボタンを押して **nmtui** メインメニューに戻ります。
7. **Activate a connection** を選択し、**Enter** を押します。
8. 編集した接続プロファイルを選択し、**Enter** キーを 2 回押して非アクティブ化し、再度アクティブ化します。

重要

再アクティブ化する接続プロファイルを使用する SSH などのリモート接続で **nmtui** を実行する場合は、この手順をスキップしてください。この場合は、**nmtui** で非アクティブ化すると、接続が切断されるため、再度アクティブ化することはできません。この問題を回避するには、**nmcli connection connection_profile_name up** コマンドを使用して、前述のシナリオで接続を再アクティブ化します。

9. **Back** ボタンを押してメインメニューに戻ります。
10. **Quit** を選択し、**Enter** キーを押して **nmtui** アプリケーションを閉じます。

検証

- ルートがアクティブであることを確認します。

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

25.5. CONTROL-CENTER を使用した静的ルートの設定

GNOME で **control-center** を使用して、ネットワーク接続の設定に静的ルートを追加します。

以下の手順では、以下の経路を設定します。

- リモート **198.51.100.0/24** ネットワークへの IPv4 ルート。対応するゲートウェイの IP アドレスは **192.0.2.10** です。
- リモート **2001:db8:2::/64** ネットワークへの IPv6 ルート。対応するゲートウェイの IP アドレスは **2001:db8:1::10** です。

前提条件

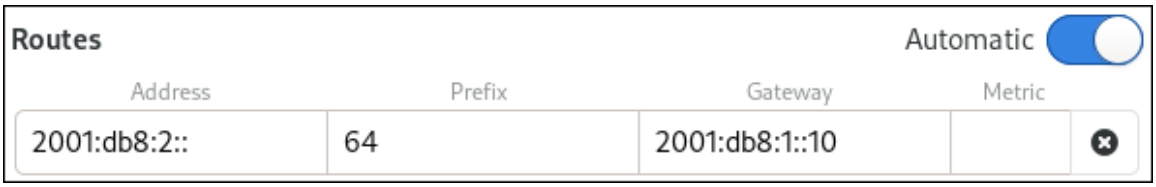
- ネットワークが設定されている。

- このホストは、ゲートウェイと同じ IP サブネットにあります。
- **control-center** アプリケーションで接続のネットワーク設定が開いている。nm-connection-editor を使用したイーサネット接続の設定 を参照してください。

手順

1. **IPv4** タブで:
   a. オプション: **IPv4** タブの **Routes** セクションの **On** ボタンをクリックして自動ルートを無効にし、静的ルートのみを使用します。自動ルートが有効になっている場合は、Red Hat Enterprise Linux が静的ルートと、DHCP サーバーから受け取ったルートを使用します。
   b. IPv4 ルートのアドレス、ネットマスク、ゲートウェイ、およびオプションでメトリック値を入力します。

2. **IPv6** タブで:
   a. オプション: **IPv4** タブの **Routes** セクションの **On** ボタンをクリックして自動ルートを無効にし、静的ルートのみを使用します。
   b. IPv6 ルートのアドレス、ネットマスク、ゲートウェイ、およびオプションでメトリック値を入力します。



3. **Apply** をクリックします。
4. **Network** ウィンドウに戻り、接続のボタンを **Off** に切り替えてから **On** に戻して、接続を無効にして再度有効にし、変更を適用します。

警告

接続を再起動すると、そのインターフェイスの接続が一時的に中断します。

検証

1. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. IPv6 ルートを表示します。

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

25.6. NM-CONNECTION-EDITOR を使用した静的ルートの設定

nm-connection-editor アプリケーションを使用して、ネットワーク接続の設定に静的ルートを追加できます。

以下の手順では、以下の経路を設定します。

- リモート **198.51.100.0/24** ネットワークへの IPv4 ルート。IP アドレス **192.0.2.10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。
- リモート **2001:db8:2::/64** ネットワークへの IPv6 ルート。IP アドレス **2001:db8:1::10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。

前提条件

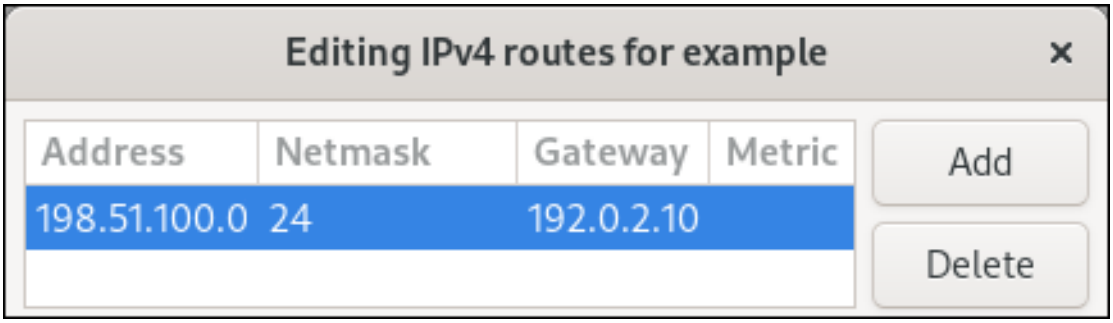
- ネットワークが設定されている。
- このホストは、ゲートウェイと同じ IP サブネットにあります。

手順

1. ターミナルを開き、**nm-connection-editor** と入力します。

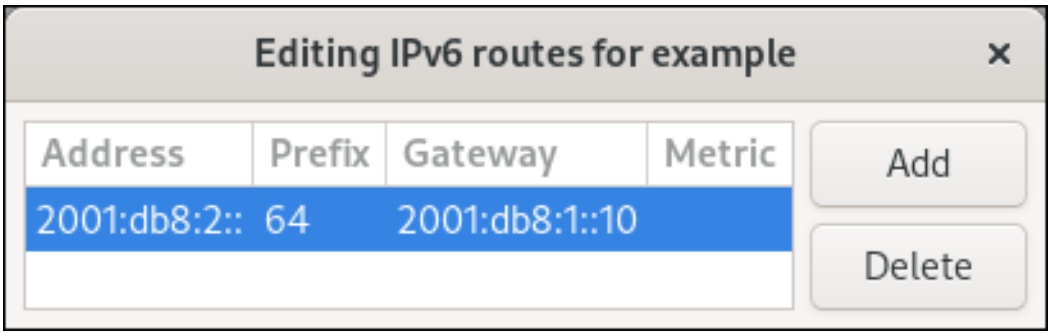
```
$ nm-connection-editor
```

2. **example** 接続プロファイルを選択し、歯車アイコンをクリックして、既存の接続を変更します。
3. **IPv4 Settings** タブで、以下を行います。
   a. **Routes** ボタンをクリックします。
   b. **Add** ボタンをクリックして、アドレス、ネットマスク、ゲートウェイを入力します。必要に応じてメトリック値を入力します。



   c. **OK** をクリックします。
4. **IPv6 Settings** タブで、以下を行います。
   a. **Routes** ボタンをクリックします。

b. **Add** ボタンをクリックして、アドレス、ネットマスク、ゲートウェイを入力します。必要に応じてメトリック値を入力します。



c. **OK** をクリックします。

5. **Save** をクリックします。

6. ネットワーク接続を再起動して、変更を有効にします。たとえば、コマンドラインで **example** 接続を再起動するには、次のコマンドを実行します。

```
# nmcli connection up example
```

検証

1. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. IPv6 ルートを表示します。

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

25.7. NMCLI 対話モードを使用した静的ルートの設定

nmcli ユーティリティーのインタラクティブモードを使用して、ネットワーク接続の設定に静的ルートを追加できます。

以下の手順では、以下の経路を設定します。

- リモート **198.51.100.0/24** ネットワークへの IPv4 ルート。IP アドレス **192.0.2.10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。
- リモート **2001:db8:2::/64** ネットワークへの IPv6 ルート。IP アドレス **2001:db8:1::10** を持つ対応するゲートウェイは、**example** の接続を介して到達可能です。

前提条件

- **example** の接続プロファイルが存在し、このホストがゲートウェイと同じ IP サブネットになるように設定されています。

手順

1. **example** 接続の **nmcli** インタラクティブモードを開きます。

```
# nmcli connection edit example
```

2. 静的 IPv4 ルートを追加します。

```
nmcli> set ipv4.routes 198.51.100.0/24 192.0.2.10
```

3. 静的 IPv6 ルートを追加します。

```
nmcli> set ipv6.routes 2001:db8:2::/64 2001:db8:1::10
```

4. 必要に応じて、ルートが設定に正しく追加されたことを確認します。

```
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 属性には、転送するネットワークと、ゲートウェイの **nh** 属性 (次のホップ) が表示されます。

5. 設定を保存します。

```
nmcli> save persistent
```

6. ネットワーク接続が再起動します。

```
nmcli> activate example
```

7. **nmcli** インタラクティブモードを終了します。

```
nmcli> quit
```

検証

1. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. IPv6 ルートを表示します。

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

関連情報

- **nmcli(1)** man ページ
- **nm-settings-nmcli (5)** man ページ

25.8. NMSTATECTL を使用した静的ルートの設定

nmstatectl ユーティリティーを使用して、Nmstate API を介して静的ルートを設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

前提条件

- **enp1s0** ネットワークインターフェイスが設定され、ゲートウェイと同じ IP サブネット内にあります。
- **nmstate** パッケージがインストールされている。

手順

1. 以下の内容を含む YAML ファイルを作成します (例: **~/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
```

これらの設定では、次の静的ルートを定義します。

- リモート **198.51.100.0/24** ネットワークへの IPv4 ルート。IP アドレス **192.0.2.10** の対応するゲートウェイは、**enp1s0** インターフェイスを介して到達できます。
- リモート **2001:db8:2::/64** ネットワークへの IPv6 ルート。IP アドレス **2001:db8:1::10** の対応するゲートウェイは、**enp1s0** インターフェイスを介して到達できます。

2. 設定をシステムに適用します。

```
# nmstatectl apply ~/add-static-route-to-enp1s0.yml
```

検証

1. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp1s0
```

2. IPv6 ルートを表示します。

```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp1s0 metric 1024 pref medium
```

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

25.9. NETWORK RHEL システムロールを使用した静的ルートの設定

network RHEL システムロールを使用して、静的ルートを設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

この手順では、すでに存在するかどうかに応じて、以下の設定で **enp7s0** 接続プロファイルを作成または更新します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- 静的ルート:
  - **198.51.100.0/24** のゲートウェイ **192.0.2.10**
  - **2001:db8:2::/64** とゲートウェイ **2001:db8:1::10**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 管理対象ノードで以下を行います。
   a. IPv4 ルートを表示します。

```
# ip -4 route
...
198.51.100.0/24 via 192.0.2.10 dev enp7s0
```

b. IPv6 ルートを表示します。

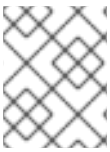
```
# ip -6 route
...
2001:db8:2::/64 via 2001:db8:1::10 dev enp7s0 metric 1024 pref medium
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第26章 代替ルートを定義するポリシーベースのルーティングの設定

デフォルトでは、RHEL のカーネルは、ルーティングテーブルを使用して宛先アドレスに基づいてネットワークパケットを転送する場所を決定します。ポリシーベースのルーティングにより、複雑なルーティングシナリオを設定できます。たとえば、送信元アドレス、パケットメタデータ、プロトコルなどのさまざまな基準に基づいてパケットをルーティングできます。



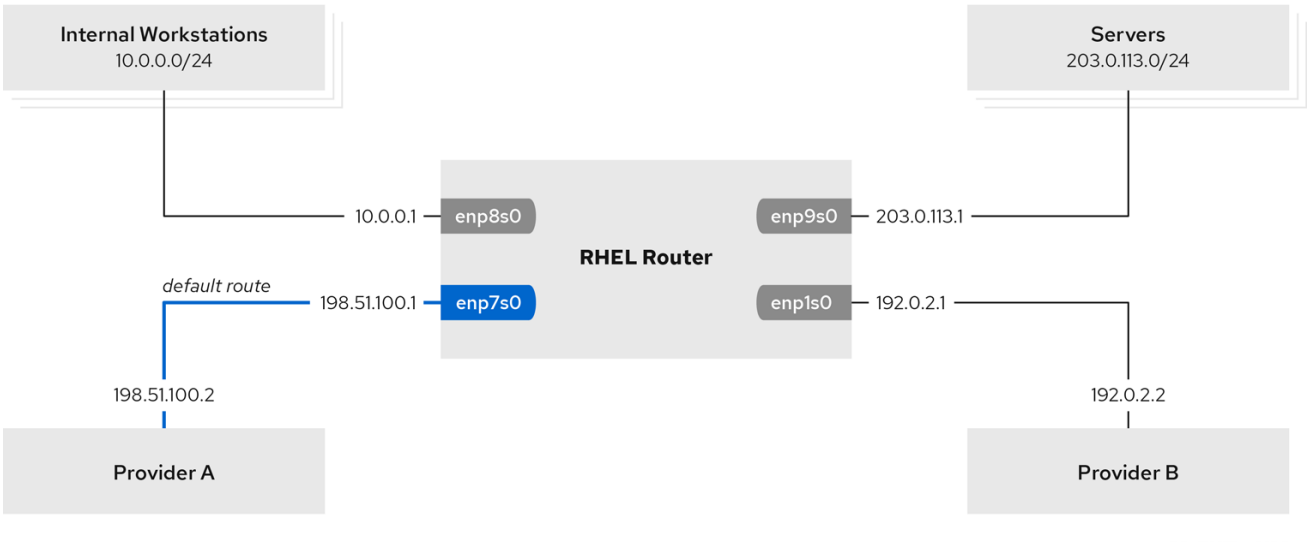
注記

NetworkManager を使用するシステムでは、**nmcli** ユーティリティーのみがルーティングルールの設定と、ルートの特定テーブルへの割り当てをサポートします。

26.1. NMCLI を使用した特定のサブネットから異なるデフォルトゲートウェイへのトラフィックのルーティング

ポリシーベースのルーティングを使用して、特定のサブネットからのトラフィックに対して別のデフォルトゲートウェイを設定できます。たとえば、デフォルトルートを使用して、すべてのトラフィックをインターネットプロバイダー A にデフォルトでルーティングするルーターとして RHEL を設定できます。ただし、内部ワークステーションサブネットから受信したトラフィックはプロバイダー B にルーティングされます。

この手順では、次のネットワークトポロジーを想定しています。



前提条件

- システムは、**NetworkManager** を使用して、ネットワークを設定します (これがデフォルトです)。
- この手順で設定する RHEL ルーターには、4 つのネットワークインターフェイスがあります。
  - **enp7s0** インターフェイスはプロバイダー A のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **198.51.100.2** で、ネットワークは **/30** ネットワークマスクを使用します。

- **enp1s0** インターフェイスはプロバイダー B のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **192.0.2.2** で、ネットワークは **/30** ネットワークマスクを使用します。
- **enp8s0** インターフェイスは、内部ワークステーションで **10.0.0.0/24** サブネットに接続されています。
- **enp9s0** インターフェイスは、会社のサーバーで **203.0.113.0/24** サブネットに接続されています。

- 内部ワークステーションのサブネット内のホストは、デフォルトゲートウェイとして **10.0.0.1** を使用します。この手順では、この IP アドレスをルーターの **enp8s0** ネットワークインターフェイスに割り当てます。
- サーバーサブネット内のホストは、デフォルトゲートウェイとして **203.0.113.1** を使用します。この手順では、この IP アドレスをルーターの **enp9s0** ネットワークインターフェイスに割り当てます。
- デフォルトでは、**firewalld** サービスは有効でアクティブになっています。

手順

1. プロバイダー A へのネットワークインターフェイスを設定します。

```
# 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 コマンドでは、NetworkManager 接続プロファイルが作成されます。このコマンドでは次のオプションを使用します。

- **type ethernet**:接続タイプがイーサネットであることを定義します。
- **con-name connection_name**:プロファイルの名前を設定します。混乱を避けるために、わかりやすい名前を使用してください。
- **ifname network_device**:ネットワークインターフェイスを設定します。
- **ipv4.method manual**:静的 IP アドレスを設定できるようにします。
- **ipv4.addresses IP_address/subnet_mask**:IPv4 アドレスおよびサブネットマスクを設定します。
- **ipv4.gateway IP_address**:デフォルトのゲートウェイアドレスを設定します。
- **ipv4.dns IP_of_DNS_server**:DNS サーバーの IPv4 アドレスを設定します。
- **connection.zone firewalld_zone**:定義した **firewalld** ゾーンにネットワークインターフェイスを割り当てます。**firewalld** は、**外部** ゾーンに割り当てられたマスカレードインターフェイスを自動的に有効にすることに注意してください。

2. プロバイダー B へのネットワークインターフェイスを設定します。

```
# 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
```

このコマンドは、デフォルトゲートウェイを設定する **ipv4.gateway** の代わりに、**ipv4.routes** パラメーターを使用します。これは、この接続のデフォルトゲートウェイを、デフォルトのルーティングテーブル (**5000**) に割り当てるために必要です。NetworkManager は、接続がアクティブになると、この新しいルーティングテーブルを自動的に作成します。

3. 内部ワークステーションサブネットへのネットワークインターフェイスを設定します。

```
# 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
```

このコマンドは、**ipv4.routes** パラメーターを使用して、ID が **5000** のルーティングテーブルに静的ルートを追加します。**10.0.0.0/24** サブネットのこの静的ルートは、ローカルネットワークインターフェイスの IP を使用してプロバイダー B (**192.0.2.1**) を次のホップとして使用します。

また、このコマンドでは **ipv4.routing-rules** パラメーターを使用して、優先度 **5** のルーティングルールを追加します。このルーティングルールは、トラフィックを **10.0.0.0/24** サブネットからテーブル **5000** へルーティングします。値が小さいほど優先度が高くなります。

ipv4.routing-rules パラメーターの構文は **ip rule add** コマンドと同じですが、**ipv4.routing-rules** は常に優先度を指定する必要があります。

4. サーバーサブネットへのネットワークインターフェイスを設定します。

```
# nmcli connection add type ethernet con-name Servers ifname enp9s0 ipv4.method manual ipv4.addresses 203.0.113.1/24 connection.zone trusted
```

検証

1. 内部ワークステーションサブネットの RHEL ホストで、以下を行います。

   a. **traceroute** パッケージをインストールします。

   ```
   # dnf install traceroute
   ```

   b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

   ```
   # 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
    ...
   ```

   コマンドの出力には、ルーターがプロバイダー B のネットワークである **192.0.2.1** 経由でパケットを送信することが表示されます。

2. サーバーのサブネットの RHEL ホストで、以下を行います。

   a. **traceroute** パッケージをインストールします。

   ```
   # dnf install traceroute
   ```

b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

```
# 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
 ...
```

コマンドの出力には、ルーターがプロバイダー A のネットワークである **198.51.100.2** 経由でパケットを送信することが表示されます。

トラブルシューティングの手順

RHEL ルーターで以下を行います。

1. ルールのリストを表示します。

```
# 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
```

デフォルトでは、RHEL には、**local** テーブル、**main** テーブル、および **default** テーブルのルールが含まれます。

2. テーブル **5000** のルートを表示します。

```
# 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. インターフェイスとファイアウォールゾーンを表示します。

```
# firewall-cmd --get-active-zones
external
  interfaces: enp1s0 enp7s0
trusted
  interfaces: enp8s0 enp9s0
```

4. **external** ゾーンでマスカレードが有効になっていることを確認します。

```
# firewall-cmd --info-zone=external
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

関連情報

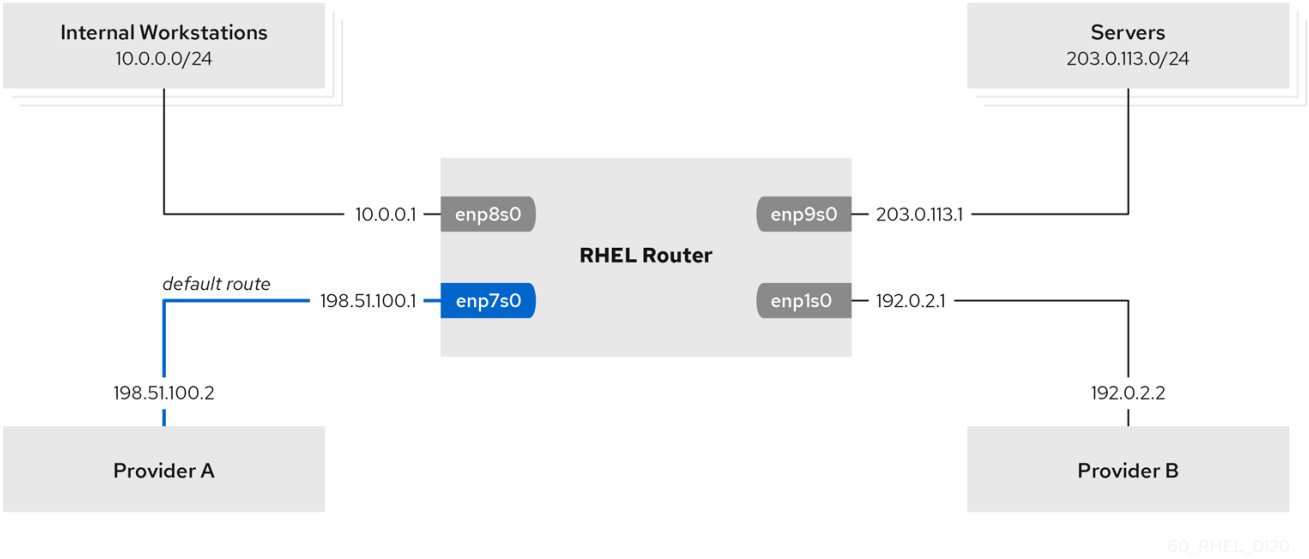
- **nm-settings(5)** man ページ
- **nmcli(1)** man ページ
- Is it possible to set up Policy Based Routing with NetworkManager in RHEL?

26.2. NETWORK RHEL システムロールを使用した特定のサブネットから別のデフォルトゲートウェイへのトラフィックのルーティング

ポリシーベースのルーティングを使用して、特定のサブネットからのトラフィックに対して別のデフォルトゲートウェイを設定できます。たとえば、デフォルトルートを使用して、すべてのトラフィックをインターネットプロバイダー A にデフォルトでルーティングするルーターとして RHEL を設定できます。ただし、内部ワークステーションサブネットから受信したトラフィックはプロバイダー B にルーティングされます。

ポリシーベースのルーティングをリモートで複数のノードに設定するには、**network** RHEL システムロールを使用できます。

この手順では、次のネットワークトポロジーを想定しています。



前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- 管理対象ノードは、**NetworkManager** および **firewalld** サービスを使用します。
- 設定する管理対象ノードには、次の 4 つのネットワークインターフェイスがあります。
  - **enp7s0** インターフェイスはプロバイダー A のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **198.51.100.2** で、ネットワークは **/30** ネットワークマスクを使用します。

  - **enp1s0** インターフェイスはプロバイダー B のネットワークに接続されます。プロバイダーのネットワークのゲートウェイ IP は **192.0.2.2** で、ネットワークは **/30** ネットワークマスクを使用します。
  - **enp8s0** インターフェイスは、内部ワークステーションで **10.0.0.0/24** サブネットに接続されています。
  - **enp9s0** インターフェイスは、会社のサーバーで **203.0.113.0/24** サブネットに接続されています。

- 内部ワークステーションのサブネット内のホストは、デフォルトゲートウェイとして **10.0.0.1** を使用します。この手順では、この IP アドレスをルーターの **enp8s0** ネットワークインターフェイスに割り当てます。
- サーバーサブネット内のホストは、デフォルトゲートウェイとして **203.0.113.1** を使用します。この手順では、この IP アドレスをルーターの **enp9s0** ネットワークインターフェイスに割り当てます。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

検証

1. 内部ワークステーションサブネットの RHEL ホストで、以下を行います。

   a. **traceroute** パッケージをインストールします。

   ```
   # dnf install traceroute
   ```

   b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

   ```
   # 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
 ...
```

コマンドの出力には、ルーターがプロバイダー B のネットワークである **192.0.2.1** 経由でパケットを送信することが表示されます。

2. サーバーのサブネットの RHEL ホストで、以下を行います。

   a. **traceroute** パッケージをインストールします。

   ```
   # dnf install traceroute
   ```

   b. **traceroute** ユーティリティーを使用して、インターネット上のホストへのルートを表示します。

   ```
   # 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
    ...
   ```

   コマンドの出力には、ルーターがプロバイダー A のネットワークである **198.51.100.2** 経由でパケットを送信することが表示されます。

3. RHEL システムロールを使用して設定した RHEL ルーターで、次の手順を実行します。

   a. ルールのリストを表示します。

   ```
   # 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
   ```

   デフォルトでは、RHEL には、**local** テーブル、**main** テーブル、および **default** テーブルのルールが含まれます。

   b. テーブル **5000** のルートを表示します。

   ```
   # 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. インターフェイスとファイアウォールゾーンを表示します。

   ```
   # firewall-cmd --get-active-zones
   external
     interfaces: enp1s0 enp7s0
   trusted
     interfaces: enp8s0 enp9s0
   ```

   d. **external** ゾーンでマスカレードが有効になっていることを確認します。

   ```
   # firewall-cmd --info-zone=external
   ```

```
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0 enp7s0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  ...
```

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第27章 異なるインターフェイスでの同じ IP アドレスの再利用

VRF (Virtual Routing and Forwarding) を使用すると、管理者は、同じホストで複数のルーティングテーブルを同時に使用できます。このため、VRF はレイヤー 3 でネットワークをパーティションで区切ります。これにより、管理者は、VRF ドメインごとに個別の独立したルートテーブルを使用してトラフィックを分離できるようになります。この技術は、レイヤー 2 でネットワークのパーティションを作成する仮想 LAN (VLAN) に類似しており、ここではオペレーティングシステムが異なる VLAN タグを使用して、同じ物理メディアを共有するトラフィックを分離させます。

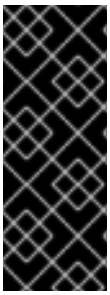
レイヤー 2 のパーティションにある VRF の利点は、関与するピアの数に対して、ルーティングが適切にスケーリングすることです。

Red Hat Enterprise Linux は、各 VRF ドメインに仮想 **vrt** デバイスを使用し、既存のネットワークデバイスを VRF デバイスに追加して、VRF ドメインにルートを含めます。元のデバイスに接続していたアドレスとルートは、VRF ドメイン内に移動します。

各 VRF ドメインが互いに分離しているることに注意してください。

27.1. 別のインターフェイスで同じ IP アドレスを永続的に再利用する

VRF (Virtual Routing and Forwarding) 機能を使用して、1 台のサーバーの異なるインターフェイスで同じ IP アドレスを永続的に使用できます。



重要

同じ IP アドレスを再利用しながら、リモートのピアが VRF インターフェイスの両方に接続するようにするには、ネットワークインターフェイスが異なるブロードキャストドメインに属する必要があります。ネットワークのブロードキャストドメインは、ノードのいずれかによって送信されたブロードキャストトラフィックを受信するノードセットです。ほとんどの設定では、同じスイッチに接続されているすべてのノードが、同じブロードキャストドメインに属するようになります。

前提条件

- **root** ユーザーとしてログインしている。
- ネットワークインターフェイスが設定されていない。

手順

1. 最初の VRF デバイスを作成して設定します。
   a. VRF デバイスの接続を作成し、ルーティングテーブルに割り当てます。たとえば、ルーティングテーブル **1001** に割り当てられた **vrf0** という名前の VRF デバイスを作成するには、次のコマンドを実行します。

      ```
      # nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1001 ipv4.method disabled ipv6.method disabled
      ```

   b. **vrf0** デバイスを有効にします。

      ```
      # nmcli connection up vrf0
      ```

c. 上記で作成した VRF にネットワークデバイスを割り当てます。たとえば、イーサネットデバイス **enp1s0** を **vrf0** VRF デバイスに追加し、IP アドレスとサブネットマスクを **enp1s0** に割り当てるには、次のコマンドを実行します。

```
# nmcli connection add type ethernet con-name vrf.enp1s0 ifname enp1s0 master vrf0 ipv4.method manual ipv4.address 192.0.2.1/24
```

d. **vrf.enp1s0** 接続をアクティベートします。

```
# nmcli connection up vrf.enp1s0
```

2. 次の VRF デバイスを作成して設定します。

a. VRF デバイスを作成し、ルーティングテーブルに割り当てます。たとえば、ルーティングテーブル **1002** に割り当てられた **vrf1** という名前の VRF デバイスを作成するには、次のコマンドを実行します。

```
# nmcli connection add type vrf ifname vrf1 con-name vrf1 table 1002 ipv4.method disabled ipv6.method disabled
```

b. **vrf1** デバイスをアクティベートします。

```
# nmcli connection up vrf1
```

c. 上記で作成した VRF にネットワークデバイスを割り当てます。たとえば、イーサネットデバイス **enp7s0** を **vrf1** VRF デバイスに追加し、IP アドレスとサブネットマスクを **enp7s0** に割り当てるには、次のコマンドを実行します。

```
# nmcli connection add type ethernet con-name vrf.enp7s0 ifname enp7s0 master vrf1 ipv4.method manual ipv4.address 192.0.2.1/24
```

d. **vrf.enp7s0** デバイスをアクティベートします。

```
# nmcli connection up vrf.enp7s0
```

27.2. 複数のインターフェイスで同じ IP アドレスを一時的に再利用

VRF (Virtual Routing and Forwarding) 機能を使用して、1 台のサーバーの異なるインターフェイスで同じ IP アドレスを一時的に使用できます。この手順は、システムの再起動後に設定が一時的で失われてしまうため、テスト目的にのみ使用します。

重要

同じ IP アドレスを再利用しながら、リモートのピアが VRF インターフェイスの両方に接続するようにするには、ネットワークインターフェイスが異なるブロードキャストドメインに属する必要があります。ネットワークのブロードキャストドメインは、ノードのいずれかによって送信されたブロードキャストトラフィックを受信するノードセットです。ほとんどの設定では、同じスイッチに接続されているすべてのノードが、同じブロードキャストドメインに属するようになります。

前提条件

- **root** ユーザーとしてログインしている。

- ネットワークインターフェイスが設定されていない。

手順

1. 最初の VRF デバイスを作成して設定します。

   a. VRF デバイスを作成し、ルーティングテーブルに割り当てます。たとえば、**1001** ルーティングテーブルに割り当てられた **blue** という名前の VRF デバイスを作成するには、次のコマンドを実行します。

   ```
   # ip link add dev blue type vrf table 1001
   ```

   b. **blue** デバイスを有効にします。

   ```
   # ip link set dev blue up
   ```

   c. VRF デバイスにネットワークデバイスを割り当てます。たとえば、イーサネットデバイス **enp1s0** を、VRF デバイス **blue** に追加するには、次のコマンドを実行します。

   ```
   # ip link set dev enp1s0 master blue
   ```

   d. **enp1s0** デバイスを有効にします。

   ```
   # ip link set dev enp1s0 up
   ```

   e. IP アドレスとサブネットマスクを **enp1s0** デバイスに割り当てます。たとえば、**192.0.2.1/24** に設定するには、以下を実行します。

   ```
   # ip addr add dev enp1s0 192.0.2.1/24
   ```

2. 次の VRF デバイスを作成して設定します。

   a. VRF デバイスを作成し、ルーティングテーブルに割り当てます。たとえば、ルーティングテーブル **1002** に割り当てられた **red** という名前の VRF デバイスを作成するには、次のコマンドを実行します。

   ```
   # ip link add dev red type vrf table 1002
   ```

   b. **red** デバイスを有効にします。

   ```
   # ip link set dev red up
   ```

   c. VRF デバイスにネットワークデバイスを割り当てます。たとえば、イーサネットデバイス **enp7s0** を、VRF デバイス **red** に追加するには、次のコマンドを実行します。

   ```
   # ip link set dev enp7s0 master red
   ```

   d. **enp7s0** デバイスを有効にします。

   ```
   # ip link set dev enp7s0 up
   ```

e. VRF ドメイン **blue** の **enp1s0** に使用したものと同じ IP アドレスとサブネットマスクを **enp7s0** デバイスに割り当てます。

```
# ip addr add dev enp7s0 192.0.2.1/24
```

3. 必要に応じて、上記のとおりに、VRF デバイスをさらに作成します。

27.3. 関連情報

- **kernel-doc** パッケージの **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/networking/vrf.txt**

第28章 分離された VRF ネットワーク内でのサービスの開始

VRF (Virtual Routing and Forwarding) を使用すると、オペレーティングシステムのメインのルーティングテーブルとは異なるルーティングテーブルを使用して、分離したネットワークを作成できます。その後、サービスとアプリケーションを起動して、そのルーティングテーブルで定義されたネットワークにのみアクセスできるようにできます。

28.1. VRF デバイスの設定

VRF (Virtual Routing and Forwarding) を使用するには、VRF デバイスを作成し、物理ネットワークインターフェイスまたは仮想ネットワークインターフェイスを割り当て、そのデバイスにルーティング情報を提供します。

警告

リモートでロックアウトを防ぐには、ローカルコンソール、または VRF デバイスに割り当てないネットワークインターフェイスを介してリモートでこの手順を実行します。

前提条件

- ローカルでログインしているか、VRF デバイスに割り当てるネットワークインターフェイスとは異なるネットワークインターフェイスを使用している。

手順

1. 同じ名前の仮想デバイスで **vrf0** 接続を作成し、これをルーティングテーブル **1000** に割り当てます。

```
# nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1000 ipv4.method disabled ipv6.method disabled
```

2. **enp1s0** デバイスーを **vrf0** 接続に追加し、IP 設定を設定します。

```
# nmcli connection add type ethernet con-name enp1s0 ifname enp1s0 master vrf0 ipv4.method manual ipv4.address 192.0.2.1/24 ipv4.gateway 192.0.2.254
```

このコマンドは、**enp1s0** 接続を、**vrf0** 接続のポートとして作成します。この設定により、ルーティング情報は、**vrf0** デバイスに関連付けられているルーティングテーブル **1000** に自動的に割り当てられます。

3. 分離したネットワークで静的ルートが必要な場合は、以下のコマンドを実行します。

   a. 静的ルートを追加します。

```
# nmcli connection modify enp1s0 +ipv4.routes "198.51.100.0/24 192.0.2.2"
```

192.0.2.2 をルーターとして使用する **198.51.100.0/24** ネットワークにルートを追加します。

b. 接続をアクティベートします。

```
# nmcli connection up enp1s0
```

検証

1. **vrf0** に関連付けられている機器の IP 設定を表示します。

```
# ip -br addr show vrf vrf0
enp1s0    UP    192.0.2.1/24
```

2. VRF デバイスと、その関連ルーティングテーブルを表示します。

```
# ip vrf show
Name              Table
----------------------
vrf0               1000
```

3. メインのルーティングテーブルを表示します。

```
# ip route show
default via 203.0.113.0/24 dev enp7s0 proto static metric 100
```

メインルーティングテーブルには、**enp1s0** デバイスまたは **192.0.2.1/24** サブネットに関連付けられたルートは記載されていません。

4. ルーティングテーブルの **1000** を表示します。

```
# 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 エントリーは、このルーティングテーブルを使用するサービスでは、**192.0.2.254** をデフォルトゲートウェイとして使用し、メインルーティングテーブルのデフォルトゲートウェイは使用しないことを示しています。

5. **vrf0** に関連付けられたネットワークで **traceroute** ユーティリティーを実行し、ユーティリティーがテーブル **1000** からのルートを使用することを確認します。

```
# 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
...
```

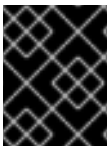
最初のホップは、ルーティングテーブル **1000** に割り当てられるデフォルトゲートウェイで、システムのメインルーティングテーブルのデフォルトゲートウェイではありません。

関連情報

- **ip-vrf(8)** の man ページ

28.2. 分離された VRF ネットワーク内でのサービスの開始

Apache HTTP Server などのサービスを、分離された仮想ルーティングおよび転送 (VRF) ネットワーク内で開始するように設定できます。



重要

サービスは、同じ VRF ネットワーク内にあるローカル IP アドレスにのみバインドできます。

前提条件

- **vrf0** デバイスを設定している。
- Apache HTTP Server が、**vrf0** デバイスに関連付けられたインターフェイスに割り当てられた IP アドレスのみをリッスンするように設定している。

手順

1. **httpd** systemd サービスの内容を表示します。

```
# systemctl cat httpd
...
[Service]
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
...
```

分離された VRF ネットワーク内で同じコマンドを実行するには、後続の手順で **ExecStart** パラメーターの内容を確認する必要があります。

2. **/etc/systemd/system/httpd.service.d/** ディレクトリーを作成します。

```
# mkdir /etc/systemd/system/httpd.service.d/
```

3. 以下の内容で **/etc/systemd/system/httpd.service.d/override.conf** ファイルを作成します。

```
[Service]
ExecStart=
ExecStart=/usr/sbin/ip vrf exec vrf0 /usr/sbin/httpd $OPTIONS -DFOREGROUND
```

ExecStart パラメーターを上書きするには、まず設定を解除してから、以下のように新しい値を設定する必要があります。

4. systemd を再ロードします。

```
# systemctl daemon-reload
```

5. **httpd** サービスを再起動します。

```
# systemctl restart httpd
```

検証

1. **httpd** プロセスのプロセス ID (PID) を表示します。

```
# pidof -c httpd
1904 ...
```

2. PID の VRF アソシエーションを表示します。以下に例を示します。

```
# ip vrf identify 1904
vrf0
```

3. **vrf0** デバイスに関連付けられているすべての PID を表示します。

```
# ip vrf pids vrf0
1904  httpd
...
```

関連情報

- **ip-vrf(8)** の man ページ

第29章 NETWORKMANAGER 接続プロファイルでの ETHTOOL 設定の実行

NetworkManager は、特定のネットワークドライバー設定とハードウェア設定を永続的に設定できます。**ethtool** ユーティリティーを使用してこれらの設定を管理する場合と比較して、これには再起動後に設定が失われないという利点があります。

NetworkManager 接続プロファイルでは、次の **ethtool** 設定を行うことができます。

オフロード機能

ネットワークインターフェイスコントローラーは、TCP オフロードエンジン (TOE) を使用して、特定の操作の処理をネットワークコントローラーにオフロードできます。これにより、ネットワークのスループットが向上します。

割り込み結合設定

割り込み結合を使用すると、システムはネットワークパケットを収集し、複数のパケットに対して割り込みを 1 つ生成します。これにより、1 つのハードウェア割り込みでカーネルに送信されたデータ量が増大し、割り込み負荷が減り、スループットを最大化します。

リングバッファー

これらのバッファーは、送受信ネットワークパケットを保存します。高いパケットドロップ率を下げるためにリングバッファーを増やすことができます。

29.1. NMCLI を使用した ETHTOOL オフロード機能の設定

NetworkManager を使用して、接続プロファイルで **ethtool** オフロード機能を有効または無効にすることができます。

手順

1. たとえば、RX オフロード機能を有効にし、**enp1s0** 接続プロファイルで TX オフロードを無効にするには、次のコマンドを実行します。

```
# nmcli con modify enp1s0 ethtool.feature-rx on ethtool.feature-tx off
```

このコマンドは、RX オフロードを明示的に有効にし、TX オフロードを無効にします。

2. 以前に有効または無効にしたオフロード機能の設定を削除するには、機能のパラメーターを null 値に設定します。たとえば、TX オフロードの設定を削除するには、次のコマンドを実行します。

```
# nmcli con modify enp1s0 ethtool.feature-tx ""
```

3. ネットワークプロファイルを再度アクティブにします。

```
# nmcli connection up enp1s0
```

検証

- **ethtool -k** コマンドを使用して、ネットワークデバイスの現在のオフロード機能を表示します。

```
# ethtool -k network_device
```

関連情報

- **nm-settings-nmcli (5)** man ページ

29.2. NETWORK RHEL システムロールを使用した ETHTOOL オフロード機能の設定

network RHEL システムロールを使用して、NetworkManager 接続の **ethtool** 機能を設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 **IPv4** アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 **IPv6** アドレス - **2001:db8:1::1** と **/64** サブネットマスク
- **IPv4** デフォルトゲートウェイ - **198.51.100.254**
- **IPv6** デフォルトゲートウェイ - **2001:db8:1::fffe**
- **IPv4** DNS サーバー - **198.51.100.200**
- **IPv6** DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **ethtool** 機能:
  - 汎用受信オフロード (GRO): 無効
  - Generic segmentation offload(GSO): 有効化
  - TX stream control transmission protocol (SCTP) segmentation: 無効

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

29.3. NMCLI を使用した ETHTOOL COALESCE の設定

NetworkManager を使用して、接続プロファイルに **ethtool** coalesce を設定できます。

手順

1. たとえば、**enp1s0** 接続プロファイルで受信パケットの最大数を **128** に設定するには、次のコマンドを実行します。

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames 128
```

2. coalesce 設定を削除するには、null 値に設定します。たとえば、**ethtool.coalesce-rx-frames** 設定を削除するには、次のコマンドを実行します。

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames ""
```

3. ネットワークプロファイルを再度アクティブにするには、以下を実行します。

```
# nmcli connection up enp1s0
```

検証

1. **ethtool -c** コマンドを使用して、ネットワークデバイスの現在のオフロード機能を表示します。

```
# ethtool -c network_device
```

関連情報

- **nm-settings-nmcli (5)** man ページ

29.4. NETWORK RHEL システムロールを使用した ETHTOOL COALESCE の設定

network RHEL システムロールを使用して、NetworkManager 接続の **ethtool** coalesce を設定できます。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 IPv4 アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **198.51.100.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **198.51.100.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **ethtool** coalesce の設定:
  - RX フレーム:**128**
  - TX フレーム:**128**

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

29.5. NMCLI を使用して、高いパケットドロップ率を減らすためにリングバッファーサイズを増やす

パケットドロップ率が原因でアプリケーションがデータの損失、タイムアウト、またはその他の問題を報告する場合は、イーサネットデバイスのリングバッファーのサイズを増やします。

受信リングバッファーは、デバイスドライバーとネットワークインターフェイスコントローラー (NIC) の間で共有されます。カードは、送信 (TX) および受信 (RX) リングバッファーを割り当てます。名前が示すように、リングバッファーは循環バッファーであり、オーバーフローによって既存のデータが上書きされます。NIC からカーネルにデータを移動するには、ハードウェア割り込みと、SoftIRQ とも呼ばれるソフトウェア割り込みの 2 つの方法があります。

カーネルは RX リングバッファーを使用して、デバイスドライバーが着信パケットを処理できるようになるまで着信パケットを格納します。デバイスドライバーは、通常は SoftIRQ を使用して RX リングをドレインします。これにより、着信パケットは **sk_buff** または **skb** と呼ばれるカーネルデータ構造に配置され、カーネルを経由して関連するソケットを所有するアプリケーションまでの移動を開始します。

カーネルは TX リングバッファーを使用して、ネットワークに送信する必要がある発信パケットを保持します。これらのリングバッファーはスタックの一番下にあり、パケットドロップが発生する重要なポイントであり、ネットワークパフォーマンスに悪影響を及ぼします。

手順

1. インターフェイスのパケットドロップ統計を表示します。

```
# ethtool -S enp1s0
    ...
    rx_queue_0_drops: 97326
    rx_queue_1_drops: 63783
    ...
```

コマンドの出力は、ネットワークカードとドライバーに依存することに注意してください。

discard または **drop** カウンターの値が高い場合は、カーネルがパケットを処理できるよりも速く、使用可能なバッファーがいっぱいになることを示します。リングバッファーを増やすと、このような損失を回避できます。

2. 最大リングバッファーサイズを表示します。

```
# 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
```

Pre-set maximums セクションの値が **Current hardware settings** セクションよりも高い場合は、次の手順で設定を変更できます。

3. このインターフェイスを使用する NetworkManager 接続プロファイルを特定します。

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

4. 接続プロファイルを更新し、リングバッファーを増やします。

   - RX リングバッファーを増やすには、次のように入力します。

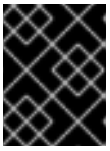
```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

   - TX リングバッファーを増やすには、次のように入力します。

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

5. NetworkManager 接続をリロードします。

```
# nmcli connection up Example-Connection
```



重要

NIC が使用するドライバーによっては、リングバッファーを変更すると、ネットワーク接続が短時間中断されることがあります。

関連情報

- ifconfig および ip コマンドがパケットドロップを報告する
- 0.05% のパケットドロップ率について心配する必要がありますか?
- **ethtool(8)** man ページ

29.6. NETWORK RHEL システムロールを使用して、高いパケットドロップ率を減らすためにリングバッファーサイズを増やす

パケットドロップ率が原因でアプリケーションがデータの損失、タイムアウト、またはその他の問題を報告する場合は、イーサネットデバイスのリングバッファーのサイズを増やします。

リングバッファーは循環バッファーであり、オーバーフローによって既存のデータが上書きされます。ネットワークカードは、送信 (TX) および受信 (RX) リングバッファーを割り当てます。受信リングバッファーは、デバイスドライバーとネットワークインターフェイスコントローラー (NIC) の間で共有され

ます。データは、ハードウェア割り込みまたは SoftIRQ とも呼ばれるソフトウェア割り込みによって NIC からカーネルに移動できます。

カーネルは RX リングバッファーを使用して、デバイスドライバーが着信パケットを処理できるようになるまで着信パケットを格納します。デバイスドライバーは、通常は SoftIRQ を使用して RX リングをドレインします。これにより、着信パケットは **sk_buff** または **skb** と呼ばれるカーネルデータ構造に配置され、カーネルを経由して関連するソケットを所有するアプリケーションまでの移動を開始します。

カーネルは TX リングバッファーを使用して、ネットワークに送信する必要がある発信パケットを保持します。これらのリングバッファーはスタックの一番下にあり、パケットドロップが発生する重要なポイントであり、ネットワークパフォーマンスに悪影響を及ぼします。

重要

network RHEL システムロールを使用するプレイの実行時に、プレイで指定した値と設定値が一致しない場合、当該ロールは同じ名前の既存の接続プロファイルをオーバーライドします。これらの値がデフォルトにリセットされないようにするには、IP 設定などの設定がすでに存在する場合でも、ネットワーク接続プロファイルの設定全体をプレイで必ず指定してください。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- デバイスがサポートする最大リングバッファーサイズを把握している。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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
```

この Playbook は、**enp1s0** 接続プロファイルを次の設定で作成します。プロファイルがすでに存在する場合は、次の設定に更新します。

- 静的 **IPv4** アドレス - **/24** サブネットマスクを持つ **198.51.100.20**
- 静的 **IPv6** アドレス - **2001:db8:1::1** と **/64** サブネットマスク
- **IPv4** デフォルトゲートウェイ - **198.51.100.254**
- **IPv6** デフォルトゲートウェイ - **2001:db8:1::fffe**
- **IPv4** DNS サーバー - **198.51.100.200**
- **IPv6** DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- リングバッファーエントリーの最大数:
  - 受信 (RX): 4096
  - 送信 (TX): 4096

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第30章 NETWORKMANAGER のデバッグの概要

すべてのドメインまたは特定のドメインのログレベルを増やすと、NetworkManager が実行する操作の詳細をログに記録するのに役立ちます。この情報を使用して問題のトラブルシューティングを行うことができます。NetworkManager は、ロギング情報を生成するさまざまなレベルとドメインを提供します。**/etc/NetworkManager/NetworkManager.conf** ファイルは、NetworkManager の主な設定ファイルです。ログはジャーナルに保存されます。

30.1. NETWORKMANAGER の REAPPLY メソッドの概要

NetworkManager サービスは、プロファイルを使用してデバイスの接続設定を管理します。Desktop Bus (D-Bus) API は、これらの接続設定を作成、変更、削除できます。プロファイルに変更があった場合、D-Bus API は既存の設定を変更された接続設定に複製します。複製はされるものの、変更された設定に変更は適用されません。これを反映するには、接続の既存の設定を再度アクティブにするか、**reapply()** メソッドを使用します。

reapply() メソッドには次の機能があります。

1. ネットワークインターフェイスの非アクティブ化または再起動を行わずに、変更された接続設定を更新します。
2. 変更された接続設定から保留中の変更を削除します。**NetworkManager** は、手動による変更を元に戻さないため、デバイスを再設定して外部パラメーターまたは手動パラメーターを元に戻すことができます。
3. 既存の接続設定とは異なる、変更された接続設定を作成します。

また、**reapply()** メソッドは次の属性をサポートします。

- **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**

関連情報

- **nm-settings-nmcli (5)** man ページ

30.2. NETWORKMANAGER ログレベルの設定

デフォルトでは、すべてのログドメインは **INFO** ログレベルを記録します。デバッグログを収集する前にレート制限を無効にします。帯域制限により、**systemd-journald** は、短時間にメッセージが多すぎる場合にメッセージを破棄します。これは、ログレベルが **TRACE** の場合に発生する可能性があります。

この手順では、レート制限を無効にし、すべての (ALL) ドメインのデバッグログの記録を有効にします。

手順

1. レート制限を無効にするには、**/etc/systemd/journald.conf** ファイルを編集し、**[Journal]** セクションの **RateLimitBurst** パラメーターのコメントを解除し、その値を **0** に設定します。

```
RateLimitBurst=0
```

2. **systemd-journald** サービスを再起動します。

```
# systemctl restart systemd-journald
```

3. 以下の内容で **/etc/NetworkManager/conf.d/95-nm-debug.conf** ファイルを作成します。

```
[logging]
domains=ALL:TRACE
```

domains パラメーターには、複数のコンマ区切りの **domain:level** ペアを含めることができます。

4. NetworkManager サービスを再読み込みします。

```
# systemctl restart NetworkManager
```

検証

- **systemd** ジャーナルにクエリーを実行して、**NetworkManager** ユニットのジャーナルエントリーを表示します。

```
# 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. NMCLI を使用して、ランタイム時にログレベルを一時的に設定

nmcli を使用すると、ランタイム時にログレベルを変更できます。ただし、Red Hat は、設定ファイルを使用してデバッグを有効にし、NetworkManager を再起動することを推奨します。**.conf** ファイルを使用してデバッグの **levels** および **domains** を更新すると、ブートの問題をデバッグし、初期状態からすべてのログをキャプチャーできます。

手順

1. オプション: 現在のログ設定を表示します。

```
# 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. ログレベルおよびドメインを変更するには、以下のオプションを使用します。

   - すべてのドメインのログレベルを同じ **LEVEL** に設定するには、次のコマンドを実行します。

     ```
     # nmcli general logging level LEVEL domains ALL
     ```

   - 特定のドメインのレベルを変更するには、以下を入力します。

     ```
     # nmcli general logging level LEVEL domains DOMAINS
     ```

     このコマンドを使用してログレベルを更新すると、他のすべてのドメインのログが無効になることに注意してください。

   - 特定のドメインのレベルを変更し、他のすべてのドメインのレベルを保持するには、次のコマンドを実行します。

     ```
     # nmcli general logging level KEEP domains DOMAIN:LEVEL,DOMAIN:LEVEL
     ```

30.4. NETWORKMANAGER ログの表示

トラブルシューティング用の NetworkManager ログを表示できます。

手順

- ログを表示するには、以下を入力します。

  ```
  # journalctl -u NetworkManager -b
  ```

関連情報

- **NetworkManager.conf(5)** man ページ
- **journalctl(1)** の man ページ

30.5. デバッグレベルおよびドメイン

levels および **domains** パラメーターを使用して、NetworkManager のデバッグを管理できます。レベルは詳細レベルを定義しますが、ドメインは特定の重大度 (**level**) でログを記録するメッセージのカテゴリーを定義します。

ログレベル	説明
OFF	NetworkManager に関するメッセージをログに記録しません。
ERR	重大なエラーのみのログ
WARN	操作を反映できる警告をログに記録します。
INFO	状態および操作の追跡に役立つさまざまな情報メッセージをログに記録します。

ログレベル	説明
DEBUG	デバッグの目的で詳細なログを有効にします。
TRACE	**DEBUG** レベルよりも多くの詳細ロギングを有効にします。

後続のレベルでは、以前のレベルのすべてのメッセージをログに記録することに注意してください。たとえば、ログレベルを **INFO** に設定すると、**ERR** および **WARN** ログレベルに含まれるメッセージをログに記録します。

関連情報

- **NetworkManager.conf(5)** man ページ

第31章 LLDP を使用したネットワーク設定の問題のデバッグ

Link Layer Discovery Protocol (LLDP) を使用して、トポロジー内のネットワーク設定の問題をデバッグできます。つまり、LLDP は、他のホストまたはルーターやスイッチとの設定の不整合を報告できます。

31.1. LLDP 情報を使用した誤った VLAN 設定のデバッグ

特定の VLAN を使用するようにスイッチポートを設定し、ホストがこれらの VLAN パケットを受信しない場合は、Link Layer Discovery Protocol (LLDP) を使用して問題をデバッグできます。パケットを受信しないホストでこの手順を実行します。

前提条件

- **nmstate** パッケージがインストールされている。
- スイッチは LLDP をサポートしています。
- LLDP は隣接デバイスで有効になっています。

手順

1. 次のコンテンツで **~/enable-LLDP-enp1s0.yml** ファイルを作成します。

```
interfaces:
  - name: enp1s0
    type: ethernet
    lldp:
      enabled: true
```

2. **~/enable-LLDP-enp1s0.yml** ファイルを使用して、インターフェイス **enp1s0** で LLDP を有効にします。

```
# nmstatectl apply ~/enable-LLDP-enp1s0.yml
```

3. LLDP 情報を表示します。

```
# nmstatectl show enp1s0
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: false
    dhcp: false
  ipv6:
    enabled: false
    autoconf: false
    dhcp: false
  lldp:
    enabled: true
    neighbors:
    - - type: 5
        system-name: Summit300-48
      - type: 6
```

```
      system-description: Summit300-48 - Version 7.4e.1 (Build 5)
        05/27/05 04:53:11
    - type: 7
      system-capabilities:
      - MAC Bridge component
      - Router
    - type: 1
      _description: MAC address
      chassis-id: 00:01:30:F9:AD:A0
      chassis-id-type: 4
    - type: 2
      _description: Interface name
      port-id: 1/1
      port-id-type: 5
    - type: 127
      ieee-802-1-vlans:
      - name: v2-0488-03-0505
        vid: 488
      oui: 00:80:c2
      subtype: 3
    - type: 127
      ieee-802-3-mac-phy-conf:
        autoneg: true
        operational-mau-type: 16
        pmd-autoneg-cap: 27648
      oui: 00:12:0f
      subtype: 1
    - type: 127
      ieee-802-1-ppvids:
      - 0
      oui: 00:80:c2
      subtype: 2
    - type: 8
      management-addresses:
      - address: 00:01:30:F9:AD:A0
        address-subtype: MAC
        interface-number: 1001
        interface-number-subtype: 2
    - type: 127
      ieee-802-3-max-frame-size: 1522
      oui: 00:12:0f
      subtype: 4
  mac-address: 82:75:BE:6F:8C:7A
  mtu: 1500
```

4. 出力を確認して、想定される設定と一致していることを確認します。たとえば、スイッチに接続されているインターフェイスの LLDP 情報は、このホストが接続されているスイッチポートが VLAN ID **448** を使用していることを示しています。

```
- type: 127
      ieee-802-1-vlans:
      - name: v2-0488-03-0505
        vid: 488
```

enp1s0 インターフェイスのネットワーク設定で異なる VLANID を使用している場合は、それに応じて変更してください。

関連情報

VLAN タグの設定

第32章 LINUX トラフィックの制御

Linux は、パケットの送信を管理および操作するためのツールを提供します。Linux Traffic Control (TC) サブシステムは、ネットワークトラフィックの規制、分類、成熟、およびスケジューリングに役立ちます。また、TC はフィルターとアクションを使用して分類中にパケットコンテンツをマスリングします。TC サブシステムは、TC アーキテクチャーの基本要素であるキューイング規則(**qdisc**)を使用してこれを実現します。

スケジューリングメカニズムは、異なるキューに入るか、終了する前にパケットを設定または再編成します。最も一般的なスケジューラーは First-In-First-Out (FIFO) スケジューラーです。**qdiscs** 操作は、**tc** ユーティリティーを使用して一時的に、NetworkManager を使用して永続的に実行できます。

Red Hat Enterprise Linux では、デフォルトのキューの規則をさまざまな方法で設定して、ネットワークインターフェイスのトラフィックを管理できます。

32.1. キュー規則の概要

グルーピング規則 (**qdiscs**) は、ネットワークインターフェイスによるトラフィックのスケジューリング、後でキューに役に立ちます。**qdisc** には 2 つの操作があります。

- パケットを後送信用にキューに入れるできるようにするキュー要求。
- キューに置かれたパケットのいずれかを即時に送信できるように要求を解除します。

各 **qdisc** には、**ハンドル** と呼ばれる 16 ビットの 16 進数の識別番号があり、**1:** や **abcd:** などのコロンが付けられています。この番号は **qdisc** メジャー番号と呼ばれます。**qdisc** にクラスがある場合、識別子はマイナー番号 (**<major>:<minor>**) の前にメジャー番号を持つ 2 つの数字のペア (**abcd:1**) として形成されます。マイナー番号の番号設定スキームは、**qdisc** タイプによって異なります。1 つ目のクラスには ID **<major>:1**、2 つ目の **<major>:2** などが含まれる場合があります。一部の **qdiscs** では、クラスの作成時にクラスマイナー番号を任意に設定することができます。

分類的な qdiscs

ネットワークインターフェイスへのパケット転送には、さまざまな **qdiscs** があり、そのタイプの qdiscs が存在します。root、親、または子クラスを使用して **qdiscs** を設定できます。子を割り当て可能なポイントはクラスと呼ばれます。**qdisc** のクラスは柔軟性があり、常に複数の子クラス、または 1 つの子 **qdisc** を含めることができます。これは、クラスフルな **qdisc** 自体を含むクラスに対して禁止がないため、複雑なトラフィック制御シナリオが容易になります。
分類的な **qdiscs** はパケットを格納しません。代わりに、**qdisc** 固有の基準に従って、子のいずれかに対してキューをキューに入れ、デキューします。最終的にこの再帰パケットが渡される場所は、パケットが格納される場所 (またはデキューの場合はから取得) となります。

クラスレス qdiscs

一部の **qdiscs** には子クラスがなく、クラスレス **qdiscs** と呼ばれます。クラスレス **qdiscs** は、クラスフル **qdiscs** と比較してカスタマイズが少なくなります。通常、インターフェイスに割り当てるだけで十分です。

関連情報

- **tc(8)** の man ページ
- **tc-actions(8)** の man ページ

32.2. 接続追跡の概要

Netfilter フレームワークは、ファイアウォールで外部ネットワークからのパケットをフィルタリングします。パケットが到着すると、**Netfilter** は接続追跡エントリーを割り当てます。接続追跡は、接続を追跡し、それらの接続内のパケットフローを識別する、論理ネットワーク用の Linux カーネルネットワーキング機能です。この機能は、すべてのパケットをフィルタリングして分析し、接続ステータスを保存するために接続追跡テーブルをセットアップし、識別されたパケットに基づいて接続ステータスを更新します。たとえば、FTP 接続の場合、**Netfilter** は、接続追跡エントリーを割り当てて、FTP 接続のすべてのパケットが同じように機能するようにします。接続追跡エントリーは、**Netfilter** マークを格納し、新しいパケットタプルが既存のエントリーにマッピングするメモリーテーブル内の接続状態情報を追跡します。パケットタプルが既存のエントリーにマップされない場合、パケットは、同じ接続のパケットをグループ化する新しい接続追跡エントリーを追加します。

ネットワークインターフェイス上のトラフィックを制御および分析できます。**tc** トラフィックコントローラーユーティリティーは、**qdisc** 規則を使用してネットワーク内のパケットスケジューラーを設定します。**qdisc** カーネルで設定されたキューイング規則は、パケットをインターフェイスのキューに入れます。**qdisc** を使用することにより、カーネルはネットワークインターフェイスがトラフィックを送信する前にすべてのトラフィックを取得します。また、同じ接続に属するパケットの帯域幅レートを制限するには、**tc qdisc** コマンドを使用します。

接続追跡マークからさまざまなフィールドにデータを取り出すには、**ctinfo** モジュールと **connmark** 機能を備えた **tc** ユーティリティーを使用します。パケットマーク情報を格納するために、**ctinfo** モジュールは、**Netfilter** マークと接続状態情報をソケットバッファー (**skb**) マークメタデータフィールドにコピーします。

物理メディア上でパケットを送信すると、パケットのすべてのメタデータが削除されます。パケットがメタデータを失う前に、**ctinfo** モジュールは、**Netfilter** マーク値をパケットの **IP** フィールドにある Diffserv コードポイント (DSCP) の特定の値にマッピングしてコピーします。

関連情報

- **tc (8)** および **tc-ctinfo (8)** の man ページ

32.3. TC ユーティリティーを使用したネットワークインターフェイスの QDISC の検査

デフォルトでは、Red Hat Enterprise Linux システムは **fq_codel qdisc** を使用します。**tc** ユーティリティーを使用して **qdisc** カウンターを検査できます。

手順

1. オプション: 現在の **qdisc** を表示します。

```
# tc qdisc show dev enp0s1
```

2. 現在の **qdisc** カウンターを検査します。

```
# 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**: すべてのキューが満杯であるため、パケットがドロップされる回数
- **overlimits**: 設定されたリンク容量が一杯になる回数

- **sent**: デキューの数

32.4. デフォルトの QDISC の更新

現在の **qdisc** でネットワークパケットの損失を確認する場合は、ネットワーク要件に基づいて **qdisc** を変更できます。

手順

1. 現在のデフォルト **qdisc** を表示します。

```
# sysctl -a | grep qdisc
net.core.default_qdisc = fq_codel
```

2. 現在のイーサネット接続の **qdisc** を表示します。

```
# tc -s qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval
100.0ms memory_limit 32Mb ecn
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
maxpacket 0 drop_overlimit 0 new_flow_count 0 ecn_mark 0
new_flows_len 0 old_flows_len 0
```

3. 既存の **qdisc** を更新します。

```
# sysctl -w net.core.default_qdisc=pfifo_fast
```

4. 変更を適用するには、ネットワークドライバーを再読み込みします。

```
# modprobe -r NETWORKDRIVERNAME
# modprobe NETWORKDRIVERNAME
```

5. ネットワークインターフェイスを起動します。

```
# ip link set enp0s1 up
```

検証

- イーサネット接続の **qdisc** を表示します。

```
# 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
....
```

関連情報

- How to set **sysctl** variables on Red Hat Enterprise Linux

32.5. TC ユーティリティーを使用してネットワークインターフェイスの現在の QDISC を一時的に設定する手順

デフォルトの qdisc を変更せずに、現在の **qdisc** を更新できます。

手順

1. オプション: 現在の **qdisc** を表示します。

```
# tc -s qdisc show dev enp0s1
```

2. 現在の **qdisc** を更新します。

```
# tc qdisc replace dev enp0s1 root htb
```

検証

- 更新された現在の **qdisc** を表示します。

```
# 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. NETWORKMANAGER を使用してネットワークインターフェイスの現在の QDISK を永続的に設定する

NetworkManager 接続の現在の **qdisc** 値を更新できます。

手順

1. オプション: 現在の **qdisc** を表示します。

```
# tc qdisc show dev enp0s1
  qdisc fq_codel 0: root refcnt 2
```

2. 現在の **qdisc** を更新します。

```
# nmcli connection modify enp0s1 tc.qdiscs 'root pfifo_fast'
```

3. オプション: 存在する **qdisc** に別の **qdisc** を追加するには、**+tc.qdisc** を使用します。

```
# nmcli connection modify enp0s1 +tc.qdisc 'ingress handle ffff:'
```

4. 変更を有効にします。

```
# nmcli connection up enp0s1
```

検証

- ネットワークインターフェイスの現在の **qdisc** を表示します。

```
# 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 ----------------
```

関連情報

- **nm-settings(5)** man ページ

32.7. TC-CTINFO ユーティリティーを使用したパケットのレート制限の設定

レート制限を使用すると、ネットワークトラフィックを制限し、ネットワーク内のリソースの枯渇を防ぐことができます。レート制限を使用すると、特定の時間枠内での反復的なパケット要求を制限することで、サーバーの負荷を軽減することもできます。さらに、**tc-ctinfo** ユーティリティーを使用して、カーネルでトラフィック制御を設定することにより、帯域幅レートを管理できます。

接続追跡エントリーには、**Netfilter** マークと接続情報が格納されます。ルーターがファイアウォールからパケットを転送するとき、ルーターはパケットから接続追跡エントリーを削除または変更します。接続追跡情報 (**ctinfo**) モジュールは、接続追跡マークからさまざまなフィールドにデータを取り出します。このカーネルモジュールは、**Netfilter** マークを、ソケットバッファー (**skb**) マークメタデータフィールドにコピーすることで保存します。

前提条件

- **iperf3** ユーティリティーがサーバーとクライアントにインストールされている。

手順

1. サーバーで次の手順を実行します。
   a. ネットワークインターフェイスに仮想リンクを追加します。

      ```
      # ip link add name ifb4eth0 numtxqueues 48 numrxqueues 48 type ifb
      ```

      このコマンドには次のパラメーターがあります。

      name ifb4eth0
      新しい仮想デバイスインターフェイスを設定します。

      numtxqueues 48
      送信キューの数を設定します。

      numrxqueues 48
      受信キューの数を設定します。

      type ifb
      新しいデバイスのタイプを設定します。

   b. インターフェイスの状態を変更します。

      ```
      # ip link set dev ifb4eth0 up
      ```

   c. 物理ネットワークインターフェイスに **qdisc** 属性を追加し、それを受信トラフィックに適用します。

```
# tc qdisc add dev enp1s0 handle ffff: ingress
```

handle ffff: オプションでは、**handle** パラメーターはデフォルト値としてメジャー番号 **ffff:** を **enp1s0** 物理ネットワークインターフェイス上のクラスフル **qdisc** に割り当てます。ここで、**qdisc** は、トラフィック制御を分析するためのキューイング規則パラメーターになります。

d. **IP** プロトコルの物理インターフェイスにフィルターを追加して、パケットを分類します。

```
# tc filter add dev enp1s0 parent ffff: protocol ip u32 match u32 0 0 action ctinfo cpmark 100 action mirred egress redirect dev ifb4eth0
```

このコマンドには次の属性があります。

parent ffff:

親 **qdisc** のメジャー番号 **ffff:** を設定します。

u32 match u32 0 0

パターン **u32** の IP ヘッダーと一致 (**match**) するように、**u32** フィルターを設定します。最初の **0** は、IP ヘッダーの 2 番目のバイトを表し、もう 1 つの **0** は、一致させるビットがどれであるかをフィルターに指示するマスク照合です。

action ctinfo

接続追跡マークからさまざまなフィールドにデータを取り出すアクションを設定します。

cpmark 100

接続追跡マーク (connmark) **100** をパケットの IP ヘッダーフィールドにコピーします。

action mirred egress redirect dev ifb4eth0

actionmirred を設定して、受信パケットを宛先インターフェイス **ifb4eth0** にリダイレクトします。

e. クラスフル **qdisc** をインターフェイスに追加します。

```
# tc qdisc add dev ifb4eth0 root handle 1: htb default 1000
```

このコマンドは、メジャー番号 **1** を root **qdisc** に設定し、マイナー ID **1000** のクラスフル **qdisc** を持つ **htb** 階層トークンバケットを使用します。

f. インターフェイス上のトラフィックを 1 Mbit/s に制限し、上限を 2 Mbit/s にします。

```
# tc class add dev ifb4eth0 parent 1:1 classid 1:100 htb ceil 2mbit rate 1mbit prio 100
```

このコマンドには次のパラメーターがあります。

parent 1:1

classid を **1**、**root** を **1** として **parent** を設定します。

classid 1:100

classid を **1:100** に設定します。ここで、**1** は親 **qdisc** の数で、**100** は親 **qdisc** のクラスの数です。

htb ceil 2mbit

htb のクラスフルな **qdisc** では、**ceil** レート制限として **2 Mbit/s** の上限帯域幅が許可されます。

g. クラスレス **qdisc** の Stochastic Fairness Queuing (**sfq**) を、**60** 秒の時間間隔でインターフェイスに適用して、キューアルゴリズムの摂動を軽減します。

```
# tc qdisc add dev ifb4eth0 parent 1:100 sfq perturb 60
```

h. ファイアウォールマーク (**fw**) フィルターをインターフェイスに追加します。

```
# tc filter add dev ifb4eth0 parent 1:0 protocol ip prio 100 handle 100 fw classid 1:100
```

i. 接続マーク (**CONNMARK**) からパケットのメタマークを復元します。

```
# nft add rule ip mangle PREROUTING counter meta mark set ct mark
```

このコマンドでは、**nft** ユーティリティーにはチェーンルール仕様 **PREROUTING** を含む **mangle** テーブルがあり、ルーティング前に受信パケットを変更してパケットマークを **CONNMARK** に置き換えます。

j. **nft** テーブルとチェーンが存在しない場合は、以下のようにテーブルを作成してチェーンルールを追加します。

```
# nft add table ip mangle
# nft add chain ip mangle PREROUTING {type filter hook prerouting priority mangle \;}
```

k. 指定された宛先アドレス **192.0.2.3** で受信した **tcp** パケットにメタマークを設定します。

```
# nft add rule ip mangle PREROUTING ip daddr 192.0.2.3 counter meta mark set 0x64
```

l. パケットマークを接続マークに保存します。

```
# nft add rule ip mangle PREROUTING counter ct mark set mark
```

m. **-s** パラメーターを使用して、**iperf3** ユーティリティーをシステム上のサーバーとして実行すると、サーバーはクライアント接続の応答を待ちます。

```
# iperf3 -s
```

2. クライアント上で、**iperf3** をクライアントとして実行し、IP アドレス **192.0.2.3** で定期的な HTTP 要求と応答のタイムスタンプをリッスンするサーバーに接続します。

```
# iperf3 -c 192.0.2.3 -t TCP_STREAM | tee rate
```

192.0.2.3 はサーバーの IP アドレスで、**192.0.2.4** はクライアントの IP アドレスです。

3. **Ctrl**+**C** を押して、サーバー上の **iperf3** ユーティリティーを終了します。

```
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. **Ctrl**+**C** を押して、クライアント上の **iperf3** ユーティリティーを終了します。

```
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 Mbits/sec 0   76.4 KBytes
[5]   1.00-2.00   sec   223 KBytes  1.83 Mbits/sec 0   82.0 KBytes
...
[ID] Interval        Transfer  Bitrate      Retr
[5]   0.00-14.00  sec  3.92 MBytes  2.35 Mbits/sec   32     sender
[5]   0.00-14.00  sec  0.00 Bytes  0.00 bits/sec            receiver

iperf3: error - the server has terminated
```

検証

1. インターフェイス上の **htb** クラスと **sfq** クラスのパケット数に関する統計情報を表示します。

```
# 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. **mirred** アクションと **ctinfo** アクションのパケット数の統計情報を表示します。

```
# 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. **htb** レートリミッターとその設定の統計情報を表示します。

```
# 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
```

関連情報

- **tc(8)** および **tc-ctinfo(8)** の man ページ
- **nft(8)** の man ページ

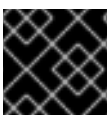
32.8. RHEL で利用できる QDISCS

各 **qdisc** は、ネットワーク関連の固有の問題に対応します。以下は、RHEL で利用可能な **qdiscs** のリストです。以下の **qdisc** を使用して、ネットワーク要件に基づいてネットワークトラフィックを形成できます。

表32.1 RHEL で利用可能なスケジューラー

qdisc 名	以下に含まれる	オフロードサポート
非同期転送モード (ATM)	**kernel-modules-extra**	
クラスベースのキューイング	**kernel-modules-extra**	
クレジットカードベースのシェーパー	**kernel-modules-extra**	はい
応答フローを選択するおよび Keep、応答しないフロー (CHOKE) の場合は CHOose および Kill	**kernel-modules-extra**	
Controlled Delay (CoDel)	**kernel-core**	
不足ラウンドロビン (DRR)	**kernel-modules-extra**	
Differentiated Services marker (DSMARK)	**kernel-modules-extra**	
Enhanced Transmission Selection (ETS)	**kernel-modules-extra**	はい

qdisc 名	以下に含まれる	オフロードサポート
Fair Queue (FQ)	**kernel-core**	
FQ_CODel (Fair Queuing Controlled Delay)	**kernel-core**	
GRED (Generalized Random Early Detection)	**kernel-modules-extra**	
階層化されたサービス曲線 (HSFC)	**kernel-core**	
負荷の高い永続フィルター (HHF)	**kernel-core**	
階層型トークンバケット (HTB)	**kernel-core**	
INGRESS	**kernel-core**	はい
MQPRIO (Multi Queue Priority)	**kernel-modules-extra**	はい
マルチキュー (MULTIQ)	**kernel-modules-extra**	はい
ネットワークエミュレーター (NETEM)	**kernel-modules-extra**	
Proportional Integral-controller Enhanced (PIE)	**kernel-core**	
PLUG	**kernel-core**	
Quick Fair Queueing (QFQ)	**kernel-modules-extra**	
ランダム初期値検出 (RED)	**kernel-modules-extra**	はい
SFB (Stochastic Fair Blue)	**kernel-modules-extra**	
SFQ (Stochastic Fairness Queueing)	**kernel-core**	
トークンバケットフィルター (TBF)	**kernel-core**	はい
TEQL (Trivial Link Equalizer)	**kernel-modules-extra**	



重要

qdisc オフロードには、NIC でハードウェアとドライバーのサポートが必要です。

関連情報

- **tc(8)** の man ページ

第33章 ファイルシステムに保存されている証明書で 802.1X 標準を使用したネットワークへの RHEL クライアントの認証

管理者は、IEEE 802.1X 標準に基づいてポートベースのネットワークアクセス制御 (NAC) を使用して、承認されていない LAN および Wi-Fi クライアントからネットワークを保護します。クライアントがそのようなネットワークに接続できるようにするには、このクライアントで 802.1X 認証を設定する必要があります。

33.1. NMCLI を使用した既存のイーサネット接続での 802.1X ネットワーク認証の設定

nmcli ユーティリティーを使用して、コマンドラインで 802.1X ネットワーク認証によるイーサネット接続を設定できます。

前提条件

- ネットワークは 802.1X ネットワーク認証をサポートしている。
- イーサネット接続プロファイルが NetworkManager に存在し、有効な IP 設定があります。
- TLS 認証に必要な以下のファイルがクライアントにある。
  - クライアント鍵が保存されているのは **/etc/pki/tls/private/client.key** ファイルで、そのファイルは所有されており、**root** ユーザーのみが読み取り可能です。
  - クライアント証明書は **/etc/pki/tls/certs/client.crt** に保存されます。
  - 認証局 (CA) 証明書は、**/etc/pki/tls/certs/ca.crt** ファイルに保存されています。
- **wpa_supplicant** パッケージがインストールされている。

手順

1. EAP (Extensible Authentication Protocol) を **tls** に設定し、クライアント証明書およびキーファイルへのパスを設定します。

```
# 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
```

   1 つのコマンドで、**802-1x.eap** パラメーター、**802-1x.client-cert** パラメーター、および **802-1x.private-key** パラメーターを設定する必要があります。

2. CA 証明書のパスを設定します。

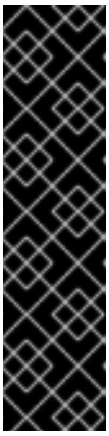
```
# nmcli connection modify enp1s0 802-1x.ca-cert /etc/pki/tls/certs/ca.crt
```

3. 証明書で使用するユーザーの ID を設定します。

```
# nmcli connection modify enp1s0 802-1x.identity user@example.com
```

4. 必要に応じて、パスワードを設定に保存します。

```
# nmcli connection modify enp1s0 802-1x.private-key-password password
```



重要

デフォルトでは、NetworkManager は、パスワードを、**/etc/sysconfig/network-scripts/keys-connection_name** ファイルにクリアテキストで保存します。これは、**root** ユーザーのみが読み取れるようにします。ただし、設定ファイルのクリアテキストパスワードはセキュリティーリスクとなる可能性があります。

セキュリティーを強化するには、**802-1x.password-flags** パラメーターを **0x1** に設定します。この設定では、GNOME デスクトップ環境または **nm-applet** が実行中のサーバーで、NetworkManager がこれらのサービスからパスワードを取得します。その他の場合は、NetworkManager によりパスワードの入力が求められます。

5. 接続プロファイルをアクティベートします。

```
# nmcli connection up enp1s0
```

検証

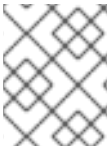
- ネットワーク認証が必要なネットワーク上のリソースにアクセスします。

関連情報

- イーサネット接続の設定
- **nm-settings(5)** man ページ
- **nmcli(1)** man ページ

33.2. NMSTATECTL を使用した 802.1X ネットワーク認証による静的イーサネット接続の設定

nmstatectl ユーティリティーを使用して、Nmstate API を介して、802.1X ネットワーク認証によるイーサネット接続を設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。



注記

nmstate ライブラリーは、**TLS** Extensible Authentication Protocol (EAP) 方式のみをサポートします。

前提条件

- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理ノードは NetworkManager を使用している。
- TLS 認証に必要な以下のファイルがクライアントにある。
  - クライアント鍵が保存されているのは **/etc/pki/tls/private/client.key** ファイルで、そのファイルは所有されており、**root** ユーザーのみが読み取り可能です。
  - クライアント証明書は **/etc/pki/tls/certs/client.crt** に保存されます。

- 認証局 (CA) 証明書は、**/etc/pki/tls/certs/ca.crt** ファイルに保存されています。

手順

1. 以下の内容を含む YAML ファイル (例: **~/create-ethernet-profile.yml**) を作成します。

```
---
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
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**

- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**
- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **TLS** EAP プロトコルを使用した 802.1X ネットワーク認証

2. 設定をシステムに適用します。

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

検証

- ネットワーク認証が必要なネットワーク上のリソースにアクセスします。

33.3. NETWORK RHEL システムロールを使用した 802.1X ネットワーク認証による静的イーサネット接続の設定

network RHEL システムロールを使用して、802.1X ネットワーク認証によるイーサネット接続をリモートで設定できます。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理対象ノードは NetworkManager を使用します。
- TLS 認証に必要な以下のファイルがコントロールノードにある。
  - クライアントキーは、**/srv/data/client.key** ファイルに保存されます。
  - クライアント証明書は **/srv/data/client.crt** ファイルに保存されます。
  - 認証局 (CA) 証明書は、**/srv/data/ca.crt** ファイルに保存されます。

手順

1. 次の内容を含む Playbook ファイル (例: **~/playbook.yml**) を作成します。

```
---
- 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
```

これらの設定では、次の設定を使用して **enp1s0** デバイスのイーサネット接続プロファイルを定義します。

- 静的 IPv4 アドレス: サブネットマスクが **/24** の **192.0.2.1**
- 静的 IPv6 アドレス - **2001:db8:1::1** (**/64** サブネットマスクあり)
- IPv4 デフォルトゲートウェイ - **192.0.2.254**
- IPv6 デフォルトゲートウェイ - **2001:db8:1::fffe**
- IPv4 DNS サーバー - **192.0.2.200**

- IPv6 DNS サーバー - **2001:db8:1::ffbb**
- DNS 検索ドメイン - **example.com**
- **TLS** Extensible Authentication Protocol (EAP) を使用した 802.1X ネットワーク認証

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

33.4. NETWORK RHEL システムロールを使用した 802.1X ネットワーク認証による WI-FI 接続の設定

RHEL システムロールを使用すると、Wi-Fi 接続の作成を自動化できます。たとえば、Ansible Playbook を使用して、**wlp1s0** インターフェイスのワイヤレス接続プロファイルをリモートで追加できます。作成されたプロファイルは、802.1X 標準を使用して、wifi ネットワークに対してクライアントを認証します。Playbook は、DHCP を使用するように接続プロファイルを設定します。静的 IP 設定を設定するには、それに応じて **IP** ディクショナリーのパラメーターを調整します。

前提条件

- 制御ノードと管理ノードを準備している
- 管理対象ノードで Playbook を実行できるユーザーとしてコントロールノードにログインしている。
- 管理対象ノードへの接続に使用するアカウントに、そのノードに対する **sudo** 権限がある。
- ネットワークは 802.1X ネットワーク認証をサポートしている。
- 管理対象ノードに **wpa_supplicant** パッケージをインストールしている。
- DHCP は、管理対象ノードのネットワークで使用できる。
- TLS 認証に必要な以下のファイルがコントロールノードにある。
  - クライアントキーは、**/srv/data/client.key** ファイルに保存されます。
  - クライアント証明書は **/srv/data/client.crt** ファイルに保存されます。
  - CA 証明書は **/srv/data/ca.crt** ファイルに保存されます。

手順

1. 次の内容を含む Playbook ファイル (例: ~/**playbook.yml**) を作成します。

```
---
- 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"
```

これらの設定では、**wlp1s0** インターフェイスの Wi-Fi 接続プロファイルを定義します。このプロファイルは、802.1X 標準を使用して、Wi-Fi ネットワークに対してクライアントを認証します。接続では、DHCP サーバーと IPv6 ステートレスアドレス自動設定 (SLAAC) から、IPv4 アドレス、IPv6 アドレス、デフォルトゲートウェイ、ルート、DNS サーバー、および検索ドメインを取得します。

2. Playbook の構文を検証します。

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

このコマンドは構文を検証するだけであり、有効だが不適切な設定から保護するものではないことに注意してください。

3. Playbook を実行します。

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

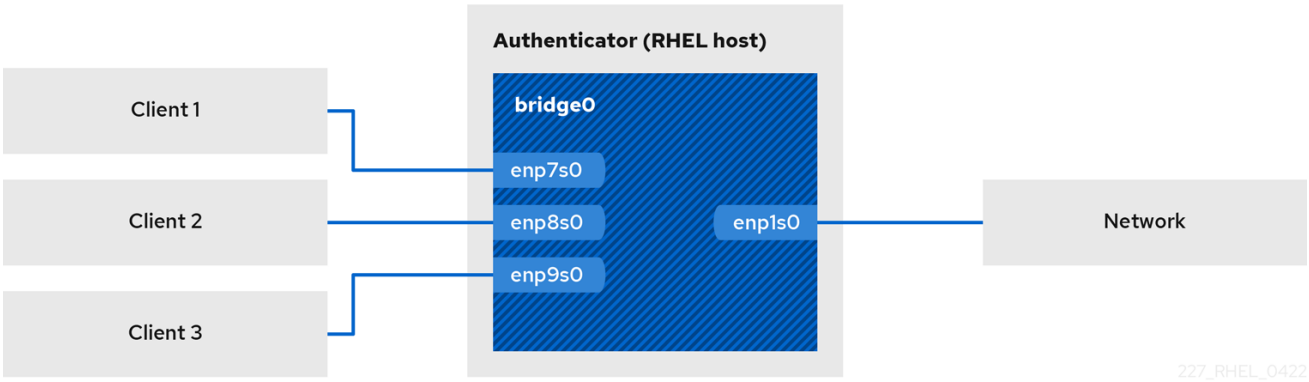
関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

第34章 FREERADIUS バックエンドで HOSTAPD を使用して LAN クライアント用の 802.1X ネットワーク認証サービスをセットアップする

IEEE 802.1X 標準は、許可されていないクライアントからネットワークを保護するためのセキュアな認証および承認方法を定義しています。**hostapd** サービスと FreeRADIUS を使用すると、ネットワークにネットワークアクセス制御 (NAC) を提供できます。

本書では、RHEL ホストは、さまざまなクライアントを既存のネットワークに接続するためのブリッジとして機能します。ただし、RHEL ホストは、認証されたクライアントのみにネットワークへのアクセスを許可します。



34.1. 前提条件

- FreeRADIUS のクリーンインストール。
  freeradius パッケージがすでにインストールされている場合は、**/etc/raddb/** ディレクトリーを削除し、アンインストールしてから、パッケージを再度インストールします。**/etc/raddb/** ディレクトリー内の権限とシンボリックリンクが異なるため、**dnf reinstall** コマンドを使用してパッケージを再インストールしないでください。

34.2. オーセンティケーターにブリッジを設定する

ネットワークブリッジは、MAC アドレスのテーブルに基づいてホストとネットワーク間のトラフィックを転送するリンク層デバイスです。RHEL を 802.1X オーセンティケーターとして設定する場合は、認証を実行するインターフェイスと LAN インターフェイスの両方をブリッジに追加します。

前提条件

- サーバーには複数のイーサネットインターフェイスがあります。

手順

1. ブリッジインターフェイスを作成します。

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

2. イーサネットインターフェイスをブリッジに割り当てます。

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

```
enp1s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port2 ifname
enp7s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port3 ifname
enp8s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port4 ifname
enp9s0 master br0
```

3. ブリッジが拡張認証プロトコル over LAN (EAPOL) パケットを転送できるようにします。

```
# nmcli connection modify br0 group-forward-mask 8
```

4. ポートを自動的にアクティブ化するように接続を設定します。

```
# nmcli connection modify br0 connection.autoconnect-slaves 1
```

5. 接続をアクティベートします。

```
# nmcli connection up br0
```

検証

1. 特定のブリッジのポートであるイーサネットデバイスのリンクステータスを表示します。

```
# 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. EAPOL パケットの転送が **br0** デバイスで有効になっているかどうかを確認します。

```
# cat /sys/class/net/br0/bridge/group_fwd_mask
0x8
```

コマンドが **0x8** を返す場合、転送が有効になります。

関連情報

- **nm-settings(5)** man ページ

34.3. FREERADIUS による証明書の要件

セキュアな FreeRADIUS サービスを利用するには、さまざまな目的で TLS 証明書が必要です。

- サーバーへの暗号化された接続用の TLS サーバー証明書。信頼済み認証局 (CA) を使用して証明書を発行します。
  サーバー証明書には、**TLS Web Server Authentication** に設定された拡張鍵用途 (EKU) フィールドが必要です。

- 拡張認証プロトコルトランスポート層セキュリティー (EAP-TLS) のために同じ CA によって発行されたクライアント証明書。EAP-TLS は証明書ベースの認証を提供し、デフォルトで有効になっています。

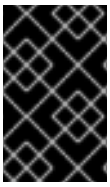
クライアント証明書では、EKU フィールドを **TLS Web Client Authentication** に設定する必要があります。

警告

接続をセキュリティー保護するには、会社の CA を使用するか、独自の CA を作成して FreeRADIUS の証明書を発行します。パブリック CA を使用する場合は、パブリック CA がユーザーを認証し、EAP-TLS のクライアント証明書を発行できるようにします。

34.4. テスト目的で FREERADIUS サーバーに一連の証明書を作成する

テストの目的で、**freeradius** パッケージはスクリプトと設定ファイルを **/etc/raddb/certs/** ディレクトリーにインストールして、独自の認証局 (CA) を作成し、証明書を発行します。



重要

デフォルト設定を使用する場合、これらのスクリプトによって生成された証明書は 60 日後に期限切れになり、キーは安全でないパスワード (何でも) を使用します。ただし、CA、サーバー、およびクライアントの設定をカスタマイズできます。

手順を実行すると、本書の後半で必要となる次のファイルが作成されます。

- **/etc/raddb/certs/ca.pem**:CA 証明書 (CA certificate)
- **/etc/raddb/certs/server.key**:サーバー証明書の秘密鍵
- **/etc/raddb/certs/server.pem**:サーバー証明書
- **/etc/raddb/certs/client.key**:クライアント証明書の秘密鍵
- **/etc/raddb/certs/client.pem**:クライアント証明書

前提条件

- **freeradius** パッケージをインストールしました。

手順

1. **/etc/raddb/certs/** ディレクトリーに移動します。

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

2. オプション: **/etc/raddb/certs/ca.cnf** ファイルで CA 設定をカスタマイズします。

```
...
[ 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. オプション: **/etc/raddb/certs/server.cnf** ファイルでサーバー設定をカスタマイズします::

```
...
[ 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. オプション: **/etc/raddb/certs/client.cnf** ファイルでクライアント設定をカスタマイズします::

```
...
[ 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. 証明書を作成します。

```
# make all
```

6. **/etc/raddb/certs/server.pem** ファイルのグループを **radiusd** に変更します。

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

関連情報

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

34.5. ネットワーククライアントを安全に認証するための FREERADIUS の設定 (EAP 使用)

FreeRADIUS は、拡張認証プロトコル (EAP) のさまざまな方法をサポートしています。ただし、セキュアなネットワークの場合は、以下のセキュアな EAP 認証方法のみをサポートするように FreeRADIUS を設定します。

- EAP-TLS (Transport Layer Security) は、セキュアな TLS 接続を使用し、証明書を使用したクライアントの認証を行います。EAP-TLS を使用するには、各ネットワーククライアントの TLS クライアント証明書とサーバーのサーバー証明書が必要です。同じ認証局 (CA) が証明書を発行している必要があることに注意してください。使用する CA によって発行されたすべてのクライアント証明書は FreeRADIUS サーバーに対して認証できるため、常に独自の CA を使用して証明書を作成してください。

- Extensible Authentication Protocol - Tunneled Transport Layer Security (EAP-TTLS) は、セキュアな TLS 接続を使用し、パスワード認証プロトコル (PAP) やチャレンジハンドシェイク認証プロトコル (CHAP) などのメカニズムを使用してクライアントを認証します。EAP-TTLS を使用するには、TLS サーバー証明書が必要です。

- EAP-PEAP (保護された拡張認証プロトコル) は、トンネルを設定するための外部認証プロトコルとしてセキュアな TLS 接続を使用します。オーセンティケーターは、RADIUS サーバーの証明書を認証します。その後、サプリカントは、Microsoft チャレンジハンドシェイク認証プロトコルバージョン 2 (MS-CHAPv2) またはその他の方法を使用して、暗号化されたトンネルを介して認証します。

注記

デフォルトの FreeRADIUS 設定ファイルはドキュメントとして機能し、すべてのパラメーターとディレクティブを記述します。特定の機能を無効にする場合は、設定ファイルの対応する部分を削除するのではなく、コメントアウトしてください。これにより、設定ファイルと含まれているドキュメントの構造を保持できます。

前提条件

- **freeradius** パッケージをインストールしました。

- **/etc/raddb/** ディレクトリー内の設定ファイルは変更されておらず、**freeradius** パッケージによって提供されています。

- サーバーには次のファイルがあります。

  - FreeRADIUS ホストの TLS 秘密鍵: **/etc/raddb/certs/server.key**

  - FreeRADIUS ホストの TLS サーバー証明書: **/etc/raddb/certs/server.pem**

- TLS CA 証明書: **/etc/raddb/certs/ca.pem**

ファイルを別の場所に保存する場合、またはファイルの名前が異なる場合は、それに応じて **/etc/raddb/mods-available/eap** ファイルの **private_key_file**、**certificate_file**、および **ca_file** パラメーターを設定します。

手順

1. Diffie-Hellman (DH) パラメーターを持つ **/etc/raddb/certs/dh** が存在しない場合は、作成します。たとえば、2048 ビットの素数を持つ DH ファイルを作成するには、次のように入力します。

```
# openssl dhparam -out /etc/raddb/certs/dh 2048
```

セキュリティー上の理由から、素数が 2048 ビット未満の DH ファイルは使用しないでください。ビット数によっては、ファイルの作成に数分かかる場合があります。

2. TLS 秘密鍵、サーバー証明書、CA 証明書、および DH パラメーターを使用したファイルにセキュアな権限を設定します。

```
# 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. **/etc/raddb/mods-available/eap** ファイルを編集します。

   a. **private_key_password** パラメーターで秘密鍵のパスワードを設定します。

```
eap {
    ...
    tls-config tls-common {
        ...
        private_key_password = key_password
        ...
    }
}
```

   b. 環境に応じて、**eap** ディレクティブの **default_eap_type** パラメーターを、使用するプライマリー EAP タイプに設定します。

```
eap {
    ...
    default_eap_type = ttls
    ...
}
```

   セキュアな環境では、**ttls**、**tls**、または **peap** のみを使用してください。

   c. 安全でない EAP-MD5 認証方式を無効にするには、**md5** ディレクティブをコメントアウトします。

```
eap {
    ...
    # md5 {
```

```
    # }
    ...
}
```

デフォルトの設定ファイルでは、他の安全でない EAP 認証方法がデフォルトでコメントアウトされていることに注意してください。

4. **/etc/raddb/sites-available/default** ファイルを編集し、**eap** 以外のすべての認証方法をコメントアウトします。

```
authenticate {
    ...
    # Auth-Type PAP {
    #     pap
    # }

    # Auth-Type CHAP {
    #     chap
    # }

    # Auth-Type MS-CHAP {
    #     mschap
    # }

    # mschap

    # digest
    ...
}
```

これにより、EAP のみが有効になり、プレーンテキスト認証方式が無効になります。

5. **/etc/raddb/clients.conf** ファイルを編集します。

   a. **localhost** および **localhost_ipv6** クライアントディレクティブでセキュアなパスワードを設定します。

```
client localhost {
    ipaddr = 127.0.0.1
    ...
    secret = client_password
    ...
}

client localhost_ipv6 {
    ipv6addr = ::1
    secret = client_password
}
```

   b. リモートホスト上のネットワークオーセンティケーターなどの RADIUS クライアントが FreeRADIUS サービスにアクセスできる必要がある場合は、それらに対応するクライアントディレクティブを追加します。

```
client hostapd.example.org {
    ipaddr = 192.0.2.2/32
    secret = client_password
```

```
}
```

ipaddr パラメーターは IPv4 および IPv6 アドレスを受け入れ、オプションのクラスレスドメイン間ルーティング (CIDR) 表記を使用して範囲を指定できます。ただし、このパラメーターに設定できる値は 1 つだけです。たとえば、IPv4 および IPv6 アドレスへのアクセスを許可するには、2 つのクライアントディレクティブを追加します。

ホスト名や IP 範囲が使用される場所を説明する単語など、クライアントディレクティブのわかりやすい名前を使用します。

6. EAP-TTLS または EAP-PEAP を使用する場合は、ユーザーを **/etc/raddb/users** ファイルに追加します。

```
example_user       Cleartext-Password := "user_password"
```

証明書ベースの認証 (EAP-TLS) を使用する必要があるユーザーの場合、エントリーを追加しないでください。

7. 設定ファイルを確認します。

```
# radiusd -XC
...
Configuration appears to be OK
```

8. **radiusd** サービスを有効にして開始します。

```
# systemctl enable --now radiusd
```

検証

- FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TTLS 認証のテスト
- FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TLS 認証のテスト

トラブルシューティング

1. **radiusd** サービスを停止します。

```
# systemctl stop radiusd
```

2. デバッグモードでサービスを開始します。

```
# radiusd -X
...
Ready to process requests
```

3. **Verification** セクションで参照されているように、FreeRADIUS ホストで認証テストを実行します。

次のステップ

- 不要になった認証方法や使用しないその他の機能を無効にします。

34.6. 有線ネットワークでのオーセンティケーターとしての HOSTAPD の設定

ホストアクセスポイントデーモン (**hostapd**) サービスは、有線ネットワークでオーセンティケーターとして機能し、802.1X 認証を提供できます。このため、**hostapd** サービスには、クライアントを認証する RADIUS サーバーが必要です。

hostapd サービスは、統合された RADIUS サーバーを提供します。ただし、統合 RADIUS サーバーはテスト目的でのみ使用してください。実稼働環境では、さまざまな認証方法やアクセス制御などの追加機能をサポートする FreeRADIUS サーバーを使用します。

重要

hostapd サービスはトラフィックプレーンと相互作用しません。このサービスは、オーセンティケーターとしてのみ機能します。たとえば、**hostapd** 制御インターフェイスを使用するスクリプトまたはサービスを使用して、認証イベントの結果に基づいてトラフィックを許可または拒否します。

前提条件

- **hostapd** パッケージをインストールしました。
- FreeRADIUS サーバーが設定され、クライアントを認証する準備が整いました。

手順

1. 次のコンテンツで **/etc/hostapd/hostapd.conf** ファイルを作成します。

```
# General settings of hostapd
# ============================

# Control interface settings
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel

# Enable logging for all modules
logger_syslog=-1
logger_stdout=-1

# Log level
logger_syslog_level=2
logger_stdout_level=2


# Wired 802.1X authentication
# ============================

# Driver interface type
driver=wired

# Enable IEEE 802.1X authorization
ieee8021x=1

# Use port access entry (PAE) group address
# (01:80:c2:00:00:03) when sending EAPOL frames
```

```
use_pae_group_addr=1


# Network interface for authentication requests
interface=br0


# RADIUS client configuration
# ============================

# Local IP address used as NAS-IP-Address
own_ip_addr=192.0.2.2

# Unique NAS-Identifier within scope of RADIUS server
nas_identifier=hostapd.example.org

# RADIUS authentication server
auth_server_addr=192.0.2.1
auth_server_port=1812
auth_server_shared_secret=client_password

# RADIUS accounting server
acct_server_addr=192.0.2.1
acct_server_port=1813
acct_server_shared_secret=client_password
```

この設定で使用されるパラメーターの詳細は、**/usr/share/doc/hostapd/hostapd.conf** サンプル設定ファイルの説明を参照してください。

2. **hostapd** サービスを有効にして開始します。

```
# systemctl enable --now hostapd
```

検証

- 参照:
  - FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TTLS 認証のテスト
  - FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TLS 認証のテスト

トラブルシューティング

1. **hostapd** サービスを停止します。

```
# systemctl stop hostapd
```

2. デバッグモードでサービスを開始します。

```
# hostapd -d /etc/hostapd/hostapd.conf
```

3. **Verification** セクションで参照されているように、FreeRADIUS ホストで認証テストを実行します。

関連情報

関連情報

- **hostapd.conf(5)** man page
- **/usr/share/doc/hostapd/hostapd.conf** ファイル

34.7. FREERADIUS サーバーまたはオーセンティケーターに対する EAP-TTLS 認証のテスト

Extensible Authentication Protocol - Tunneled Transport Layer Security (EAP-TTLS) を使用した認証が、期待どおりに機能するかテストするには、次の手順を実行します。

- FreeRADIUS サーバーをセットアップした後
- **hostapd** サービスを 802.1X ネットワーク認証のオーセンティケーターとして設定した後。

この手順で使用されるテストユーティリティーの出力は、EAP 通信に関する追加情報を提供し、問題のデバッグに役立ちます。

前提条件

- 認証する場合:
  - FreeRADIUS サーバー:
    - **hostapd** パッケージによって提供される **eapol_test** ユーティリティーがインストールされます。
    - この手順を実行するクライアントは、FreeRADIUS サーバーのクライアントデータベースで承認されています。
  - 同じ名前のパッケージによって提供されるオーセンティケーター、**wpa_supplicant** ユーティリティーがインストールされます。
- 認証局 (CA) 証明書を **/etc/pki/tls/certs/ca.pem** ファイルに保存しました。

手順

1. 次のコンテンツで **/etc/wpa_supplicant/wpa_supplicant-TTLS.conf** ファイルを作成します。

```
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. 認証するには:

   - FreeRADIUS サーバーには、次のように入力します。

     ```
     # 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 オプションは FreeRADIUS サーバーの IP アドレスを定義し、 **-s** オプションは FreeRADIUS サーバーのクライアント設定でコマンドを実行するホストのパスワードを指定します。

   - オーセンティケーター。次のように入力します。

     ```
     # wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -D wired -i enp0s31f6
     ...
     enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
     ...
     ```

     -i オプションは、**wpa_supplicant** が LAN(EAPOL) パケットを介して拡張認証プロトコルを送信するネットワークインターフェイス名を指定します。

     デバッグ情報の詳細は、コマンドに **-d** オプションを渡してください。

関連情報

- **/usr/share/doc/wpa_supplicant/wpa_supplicant.conf** ファイル

34.8. FREERADIUS サーバーまたはオーセンティケーターに対する EAP-TLS 認証のテスト

Extensible Authentication Protocol - Transport Layer Security (EAP-TLS) を使用した認証が、期待どおりに機能するかテストするには、次の手順を実行します。

- FreeRADIUS サーバーをセットアップした後
- **hostapd** サービスを 802.1X ネットワーク認証のオーセンティケーターとして設定した後。

この手順で使用されるテストユーティリティーの出力は、EAP 通信に関する追加情報を提供し、問題のデバッグに役立ちます。

前提条件

- 認証する場合:

- FreeRADIUS サーバー:
  - **hostapd** パッケージによって提供される **eapol_test** ユーティリティーがインストールされます。
  - この手順を実行するクライアントは、FreeRADIUS サーバーのクライアントデータベースで承認されています。
- 同じ名前のパッケージによって提供されるオーセンティケーター、**wpa_supplicant** ユーティリティーがインストールされます。

- 認証局 (CA) 証明書を **/etc/pki/tls/certs/ca.pem** ファイルに保存しました。
- クライアント証明書を発行した CA は、FreeRADIUS サーバーのサーバー証明書を発行した CA と同じです。
- クライアント証明書を **/etc/pki/tls/certs/client.pem** ファイルに保存しました。
- クライアントの秘密鍵を **/etc/pki/tls/private/client.key** に保存しました

手順

1. 次のコンテンツで **/etc/wpa_supplicant/wpa_supplicant-TLS.conf** ファイルを作成します。

```
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. 認証するには:
   - FreeRADIUS サーバーには、次のように入力します。

```
# 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 オプションは FreeRADIUS サーバーの IP アドレスを定義し、 **-s** オプションは FreeRADIUS サーバーのクライアント設定でコマンドを実行するホストのパスワードを指定します。

- オーセンティケーター。次のように入力します。

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TLS.conf -D wired -i enp0s31f6
...
enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
```

-i オプションは、**wpa_supplicant** が LAN(EAPOL) パケットを介して拡張認証プロトコルを送信するネットワークインターフェイス名を指定します。

デバッグ情報の詳細は、コマンドに **-d** オプションを渡してください。

関連情報

- **/usr/share/doc/wpa_supplicant/wpa_supplicant.conf** ファイル

34.9. HOSTAPD 認証イベントに基づくトラフィックのブロックと許可

hostapd サービスはトラフィックプレーンと相互作用しません。このサービスは、オーセンティケーターとしてのみ機能します。ただし、認証イベントの結果に基づいてトラフィックを許可および拒否するスクリプトを作成できます。

重要

この手順はサポートされておらず、エンタープライズ対応のソリューションではありません。**hostapd_cli** によって取得されたイベントを評価することにより、トラフィックをブロックまたは許可する方法のみを示しています。

802-1x-tr-mgmt systemd サービスが開始すると、RHEL は LAN(EAPOL) パケットを介した拡張認証プロトコルを除く **hostapd** のリッスンポート上のすべてのトラフィックをブロックし、**hostapd_cli** ユーティリティーを使用して **hostapd** 制御インターフェイスに接続します。次に、**/usr/local/bin/802-1x-tr-mgmt** スクリプトがイベントを評価します。**hostapd_cli** が受信するさまざまなイベントに応じて、スクリプトは MAC アドレスのトラフィックを許可またはブロックします。**802-1x-tr-mgmt** サービスが停止すると、すべてのトラフィックが自動的に再度許可されることに注意してください。

hostapd サーバーでこの手順を実行します。

前提条件

- **hostapd** サービスが設定され、サービスはクライアントを認証する準備ができています。

手順

1. 次のコンテンツで **/usr/local/bin/802-1x-tr-mgmt** ファイルを作成します。

```
#!/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 ibrname "br0" jump accesscontrol
        }
}
EOF
    echo "802-1x-tr-mgmt Blocking all traffic through br0. Traffic for given host will be allowed
after 802.1x authentication"

elif [ "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. 次のコンテンツで **/etc/systemd/system/802-1x-tr-mgmt@.service** サービスファイルを作成します。

```
[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. systemd を再ロードします。

```
# systemctl daemon-reload
```

4. **hostapd** がリッスンしているインターフェイス名で **802-1x-tr-mgmt** サービスを有効にして開始します。

```
# systemctl enable --now 802-1x-tr-mgmt@br0.service
```

検証

- ネットワークに対してクライアントで認証します。参照:
  - FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TTLS 認証のテスト
  - FreeRADIUS サーバーまたはオーセンティケーターに対する EAP-TLS 認証のテスト

関連情報

- **systemd.service(5)** man ページ

第35章 MULTIPATH TCP の使用

Transmission Control Protocol (TCP) は、インターネットを介したデータの信頼できる配信を保証し、ネットワーク負荷に応じて帯域幅を自動的に調整します。マルチパス TCP (MPTCP) は、元の TCP プロトコル (シングルパス) のエクステンションです。MPTCP は、トランスポート接続が複数のパスで同時に動作することを可能にし、ユーザーエンドポイントデバイスにネットワーク接続の冗長性をもたらします。

35.1. MPTCP について

マルチパス TCP (MPTCP) プロトコルを使用すると、接続エンドポイント間で複数のパスを同時に使用できます。プロトコル設計により、接続の安定性が向上し、シングルパス TCP と比較して他の利点ももたらされます。

注記

MPTCP 用語では、リンクはパスと見なされます。

以下に、MPTCP を使用する利点の一部を示します。

- これにより、接続が複数のネットワークインターフェイスを同時に使用できるようになります。
- 接続がリンク速度にバインドされている場合は、複数のリンクを使用すると、接続スループットが向上します。接続が CPU にバインドされている場合は、複数のリンクを使用すると接続が遅くなることに注意してください。
- これは、リンク障害に対する耐障害性を高めます。

MPTCP の詳細については、**関連情報** を確認することを強く推奨します。

関連情報

- Understanding Multipath TCP:High availability for endpoints and the networking highway of the future
- RFC8684:TCP Extensions for Multipath Operation with Multiple Addresses

35.2. MPTCP サポートを有効にするための RHEL の準備

デフォルトでは、RHEL で MPTCP サポートが無効になっています。この機能に対応するアプリケーションを使用できるように、MPTCP を有効にします。また、アプリケーションにデフォルトで TCP ソケットがある場合は、MPTCP ソケットを強制的に使用するように、ユーザー空間アプリケーションを設定する必要があります。

前提条件

以下のパッケージがインストールされている。

- **iperf3**
- **mptcpd**
- **systemtap**

手順

1. カーネルで MPTCP ソケットを有効にします。

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. **iperf3** サーバーを起動し、TCP ソケットの代わりに MPTCP ソケットを強制的に作成する場合は、次のコマンドを実行します。

```
# mptcpize run iperf3 -s

Server listening on 5201
```

3. クライアントをサーバーに接続し、TCP ソケットの代わりに MPTCP ソケットを強制的に作成する場合は、次のコマンドを実行します。

```
# mptcpize iperf3 -c 127.0.0.1 -t 3
```

4. 接続が確立されたら、**ss** 出力を確認し、サブフロー固有のステータスを確認します。

```
# 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
advmss: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. MPTCP カウンターを確認します。

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX          2                  0.0
MPTcpExtMPCapableSYNTX          2                  0.0
MPTcpExtMPCapableSYNACKRX       2                  0.0
MPTcpExtMPCapableACKRX          2                  0.0
```

関連情報

- **tcp(7)** man ページ
- **mptcpize(8)** man ページ

35.3. IPROUTE2 を使用した MPTCP アプリケーションの複数パスの一時的な設定と有効化

各 MPTCP 接続は、プレーンな TCP と似た 1 つのサブフローを使用します。MPTCP を活用するには、各 MPTCP 接続のサブフローの最大数に上限を指定します。次に、追加のエンドポイントを設定して、それらのサブフローを作成します。

重要

この手順の設定は、マシンを再起動すると保持されません。

MPTCP は現在、同じソケットの IPv6 エンドポイントと IPv4 エンドポイントの組み合わせに対応していません。同じアドレスファミリーに属するエンドポイントを使用します。

前提条件

- **mptcpd** がインストールされている。
- **iperf3** がインストールされている。
- サーバーネットワークインターフェイスの設定:
  - enp4s0:**192.0.2.1/24**
  - enp1s0:**198.51.100.1/24**
- クライアントネットワークインターフェイスの設定:
  - enp4s0f0:**192.0.2.2/24**
  - enp4s0f1:**198.51.100.2/24**

手順

1. サーバーによって提供される追加のリモートアドレスを最大 1 つ受け入れるようにクライアントを設定します。

```
# ip mptcp limits set add_addr_accepted 1
```

2. IP アドレス **198.51.100.1** を、サーバー上の新しい MPTCP エンドポイントとして追加します。

```
# ip mptcp endpoint add 198.51.100.1 dev enp1s0 signal
```

signal オプションは、スリーウェイハンドシェイクの後に **ADD_ADDR** パケットが送信されるようにします。

3. **iperf3** サーバーを起動し、TCP ソケットの代わりに MPTCP ソケットを強制的に作成する場合は、次のコマンドを実行します。

```
# mptcpize run iperf3 -s

Server listening on 5201
```

4. クライアントをサーバーに接続し、TCP ソケットの代わりに MPTCP ソケットを強制的に作成する場合は、次のコマンドを実行します。

```
# mptcpize iperf3 -c 192.0.2.1 -t 3
```

検証

1. 接続が確立されたことを確認します。

```
# ss -nti '( sport :5201 )'
```

2. 接続および IP アドレス制限を確認します。

```
# ip mptcp limit show
```

3. 新たに追加されたエンドポイントを確認します。

```
# ip mptcp endpoint show
```

4. サーバーで **nstat MPTcp*** コマンドを使用して MPTCP カウンターを確認します。

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX          2                  0.0
MPTcpExtMPCapableACKRX          2                  0.0
MPTcpExtMPJoinSynRx             2                  0.0
MPTcpExtMPJoinAckRx             2                  0.0
MPTcpExtEchoAdd                 2                  0.0
```

関連情報

- **ip-mptcp(8)** man ページ
- **mptcpize(8)** man ページ

35.4. MPTCP アプリケーションの複数パスの永続的な設定

nmcli コマンドを使用してマルチパス TCP (MPTCP) を設定し、ソースシステムと宛先システムの間に複数のサブフローを永続的に確立できます。サブフローは、さまざまなリソース、宛先へのさまざまなルート、さまざまなネットワークを使用できます。たとえばイーサネット、セルラー、wifi などです。その結果、接続が組み合わされ、ネットワークの回復力とスループットが向上します。

ここで使用した例では、サーバーは次のネットワークインターフェイスを使用します。

- enp4s0:**192.0.2.1/24**
- enp1s0:**198.51.100.1/24**
- enp7s0:**192.0.2.3/24**

ここで使用した例では、クライアントは次のネットワークインターフェイスを使用します。

- enp4s0f0:**192.0.2.2/24**
- enp4s0f1:**198.51.100.2/24**
- enp6s0:**192.0.2.5/24**

前提条件

前提条件

- 関連するインターフェイスでデフォルトゲートウェイを設定している

手順

1. カーネルで MPTCP ソケットを有効にします。

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. オプション: RHEL カーネルのサブフロー制限のデフォルトは 2 です。不足する場合、以下を実行します。

   a. 次の内容で **/etc/systemd/system/set_mptcp_limit.service** ファイルを作成します。

```
[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
```

   ワンショット ユニットは、各ブートプロセス中にネットワーク (**network.target**) が動作した後に **ip mptcp limits set subflows 3** コマンドを実行します。

   ip mptcp limits set subflows 3 コマンドは、各接続の **追加** サブフローの最大数を設定するため、合計が 4 になります。追加サブフローは最大 3 つです。

   b. **set_mptcp_limit** サービスを有効にします。

```
# systemctl enable --now set_mptcp_limit
```

3. 接続のアグリゲーションに使用するすべての接続プロファイルで MPTCP を有効にします。

```
# nmcli connection modify <profile_name> connection.mptcp-flags
signal,subflow,also-without-default-route
```

   connection.mptcp-flags パラメーターは、MPTCP エンドポイントと IP アドレスフラグを設定します。MPTCP が NetworkManager 接続プロファイルで有効になっている場合、設定により、関連するネットワークインターフェイスの IP アドレスが MPTCP エンドポイントとして設定されます。

   デフォルトでは、デフォルトゲートウェイがない場合、NetworkManager は MPTCP フラグを IP アドレスに追加しません。そのチェックをバイパスしたい場合は、**also-without-default-route** フラグも使用する必要があります。

検証

1. MPTCP カーネルパラメーターが有効になっていることを確認します。

```
# sysctl net.mptcp.enabled
net.mptcp.enabled = 1
```

2. デフォルトでは不足する場合に備えて、サブフロー制限を適切に設定していることを確認します。

```
# ip mptcp limit show
add_addr_accepted 2 subflows 3
```

3. アドレスごとの MPTCP 設定が正しく設定されていることを確認します。

```
# 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
...
```

関連情報

- **nm-settings-nmcli(5)**
- **ip-mptcp(8)**
- 「MPTCP について」
- Understanding Multipath TCP:High availability for endpoints and the networking highway of the future
- RFC8684:TCP Extensions for Multipath Operation with Multiple Addresses
- Using Multipath TCP to better survive outages and increase bandwidth

35.5. MPTCP サブフローのモニタリング

マルチパス TCP (MPTCP) ソケットのライフサイクルは複雑になる可能性があります。メインの MPTCP ソケットが作成され、MPTCP パスが検証され、1 つ以上のサブフローが作成され、最終的に削除されます。最後に、MPTCP ソケットが終了します。

MPTCP プロトコルを使用すると、**iproute** パッケージで提供される **ip** ユーティリティーを使用して、ソケットおよびサブフローの作成と削除に関連する MPTCP 固有のイベントをモニタリングできます。このユーティリティーは、**netlink** インターフェイスを使用して MPTCP イベントをモニターします。

この手順は、MPTCP イベントをモニターする方法を示しています。そのために、MPTCP サーバーアプリケーションをシミュレートし、クライアントがこのサービスに接続します。この例に関係するクライアントは、次のインターフェイスと IP アドレスを使用します。

- サーバー**192.0.2.1**
- クライアント (イーサネット接続):**192.0.2.2**
- クライアント (WiFi 接続):**192.0.2.3**

この例を単純化するために、すべてのインターフェイスは同じサブネット内にあります。これは必須ではありません。ただし、ルーティングが正しく設定されており、クライアントが両方のインターフェイスを介してサーバーに到達できることが重要です。

前提条件

- イーサネットと WiFi を備えたラップトップなど、2 つのネットワークインターフェイスを備えた RHEL クライアント
- クライアントは両方のインターフェイスを介してサーバーに接続できます
- RHEL サーバー
- クライアントとサーバーの両方が RHEL9.0 以降を実行しています
- クライアントとサーバーの両方に **mptcpd** パッケージをインストールしました

手順

1. クライアントとサーバーの両方で、接続ごとの追加のサブフロー制限を **1** に設定します。

```
# ip mptcp limits set add_addr_accepted 0 subflows 1
```

2. サーバーで、MPTCP サーバーアプリケーションをシミュレートするには、TCP ソケットの代わりに強制された MPTCP ソケットを使用してリッスンモードで **netcat** (**nc**) を開始します。

```
# mptcpize run nc -l -k -p 12345
```

-k オプションを指定すると、**nc** は、最初に受け入れられた接続の後でリスナーを閉じません。これは、サブフローのモニタリングを示すために必要です。

3. クライアント上:

   a. メトリックが最も低いインターフェイスを特定します。

```
# 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 インターフェイスのメトリックは、**wlp1s0** よりも低くなります。したがって、RHEL はデフォルトで **enp1s0** を使用します。

   b. 最初のターミナルで、モニタリングを開始します。

```
# ip mptcp monitor
```

   c. 2 番目のターミナルで、サーバーへの MPTCP 接続を開始します。

```
# mptcpize run nc 192.0.2.1 12345
```

   RHEL は、**enp1s0** インターフェイスとそれに関連する IP アドレスをこの接続のソースとして使用します。

   モニタリングターミナルで、**ip mptcp monitor** コマンドが次のログを記録するようになりました。

```
[      CREATED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2 daddr4=192.0.2.1
sport=36444 dport=12345
```

トークンは MPTCP ソケットを一意の ID として識別し、後で同じソケットで MPTCP イベントを相互に関連付けることができます。

d. サーバーへの **nc** 接続が実行されているターミナルで、**Enter** を押します。この最初のデータパケットは、接続を完全に確立します。データが送信されていない限り、接続は確立されないことに注意してください。
モニタリングターミナルで、**ip mptcp monitor** が次のログを記録するようになりました。

```
[   ESTABLISHED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2
daddr4=192.0.2.1 sport=36444 dport=12345
```

e. オプション: サーバーのポート **12345** への接続を表示します。

```
# ss -taunp | grep ":12345"
tcp ESTAB  0  0         192.0.2.2:36444 192.0.2.1:12345
```

この時点で、サーバーへの接続は 1 つだけ確立されています。

f. 3 番目のターミナルで、別のエンドポイントを作成します。

```
# ip mptcp endpoint add dev wlp1s0 192.0.2.3 subflow
```

このコマンドは、クライアントの WiFi インターフェイスの名前と IP アドレスを設定します。

モニタリングターミナルで、**ip mptcp monitor** が次のログを記録するようになりました。

```
[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 フィールドには、新しいサブフローのローカルアドレス ID が表示され、接続でネットワークアドレス変換 (NAT) が使用されている場合でも、このサブフローが識別されます。**saddr4** フィールドは、**ip mptcp endpoint add** コマンドからのエンドポイントの IP アドレスと一致します。

g. オプション: サーバーのポート **12345** への接続を表示します。

```
# 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
```

このコマンドは、2 つの接続を表示します。

- ソースアドレス **192.0.2.2** との接続は、以前に確立した最初の MPTCP サブフローに対応します。
- 送信元アドレスが **192.0.2.3** の **wlp1s0** インターフェイスを介したサブフローからの接続。

h. 3 番目のターミナルで、エンドポイントを削除します。

```
# ip mptcp endpoint delete id 2
```

ip mptcp monitor 出力の **locid** フィールドの ID を使用するか、**ip mptcp endpoint show** コマンドを使用してエンドポイント ID を取得します。

モニタリングターミナルで、**ip mptcp monitor** が次のログを記録するようになりました。

```
[     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. **nc** クライアントを備えた最初のターミナルで、**Ctrl**+**C** を押してセッションを終了します。
モニタリングターミナルで、**ip mptcp monitor** が次のログを記録するようになりました。

```
[       CLOSED] token=63c070d2
```

関連情報

- **ip-mptcp(1)** man page
- NetworkManager が複数のデフォルトゲートウェイを管理する方法

35.6. カーネルでの MULTIPATH TCP の無効化

カーネルの MPTCP オプションを明示的に無効にできます。

手順

- **mptcp.enabled** オプションを無効にします。

```
# echo "net.mptcp.enabled=0" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

検証

- カーネルで **mptcp.enabled** が無効になっているかどうかを確認します。

```
# sysctl -a | grep mptcp.enabled
net.mptcp.enabled = 0
```

第36章 MPTCPD サービスの管理

このセクションでは、**mptcpd** サービスの基本的な管理について説明します。**mptcpd** パッケージは、**TCP** 環境で **mptcp** プロトコルをオンにする **mptcpize** ツールを提供します。

36.1. MPTCPD の設定

mptcpd サービスは、**mptcp** エンドポイントを設定するためのインストルメントを提供する **mptcp** プロトコルのコンポーネントです。**mptcpd** サービスは、デフォルトでアドレスごとにサブフローエンドポイントを作成します。エンドポイントリストは、実行中のホストでの IP アドレスの変更に応じて動的に更新されます。**mptcpd** サービスは、エンドポイントのリストを自動的に作成します。**ip** ユーティリティーを使用する代わりに複数のパスを有効にします。

前提条件

- インストールされた **mptcpd** パッケージ

手順

1. 次のコマンドを使用して、カーネルで **mptcp.enabled** オプションを有効にします。

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. **mptcpd** サービスを開始します。

```
# systemctl start mptcp.service
```

3. エンドポイントの作成を確認します。

```
# ip mptcp endpoint
```

4. **mptcpd** サービスを停止するには、次のコマンドを使用します。

```
# systemctl stop mptcp.service
```

5. **mptcpd** サービスを手動で設定するには、**/etc/mptcpd/mptcpd.conf** 設定ファイルを変更します。

mptcpd サービスが作成するエンドポイントは、ホストがシャットダウンするまで続くことに注意してください。

関連情報

- **mptcpd(8)** の man ページ。

36.2. MPTCPIZE ツールを使用したアプリケーションの管理

mptcpize ツールを使用して、アプリケーションとサービスを管理します。

以下の手順は、**mptcpize** ツールを使用して **TCP** 環境でアプリケーションを管理する方法を示しています。

仮定すると、有効な **MPTCP** ソケットを使用して **iperf3** ユーティリティーを実行する必要があります。以下の手順でこの目標を達成できます。

前提条件

- **mptcpd** がインストールされている。
- **iperf3** がインストールされている。

手順

- **MPTCP** ソケットを有効にして **iperf3** ユーティリティーを開始します。

```
# mptcpize run iperf3 -s &
```

36.3. MPTCPIZE ユーティリティーを使用したサービスの MPTCP ソケットの有効化

次の一連のコマンドは、**mptcpize** ツールを使用してサービスを管理する方法を示しています。サービスの **mptcp** ソケットを有効または無効にできます。

仮定すると、**nginx** サービスの **mptcp** ソケットを管理する必要があります。以下の手順でこの目標を達成できます。

前提条件

- **mptcpd** がインストールされている。
- **nginx** パッケージがインストールされています

手順

1. サービスの **MPTCP** ソケットを有効にします。

```
# mptcpize enable nginx
```

2. サービスの **MPTCP** ソケットを無効にします。

```
# mptcpize disable nginx
```

3. サービスを再起動して、変更を有効にします。

```
# systemctl restart nginx
```

第37章 キーファイル形式の NETWORKMANAGER 接続プロファイル

デフォルトでは、Red Hat Enterprise Linux 9 以降の NetworkManager は、接続プロファイルをキーファイル形式で保存します。非推奨の **ifcfg** 形式とは異なり、キーファイル形式は NetworkManager が提供するすべての接続設定をサポートします。

37.1. NETWORKMANAGER プロファイルのキーファイル形式

キーファイルの形式は INI 形式に似ています。たとえば、次はキーファイル形式のイーサネット接続プロファイルです。

```
[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
```

各セクションは、**nm-settings(5)** および **nm-settings-keyfile(5)** man ページで説明されているように、NetworkManager の設定名に対応します。セクションの各 key-value-pair は、man ページの settings 仕様に記載されているプロパティーのいずれかになります。

NetworkManager キーファイルのほとんどの変数には、1対1のマッピングがあります。つまり、NetworkManager プロパティーは、同じ名前と形式の変数としてキーファイルに保存されます。ただし、主にキーファイルの構文を読みやすくするために例外があります。この例外の一覧は、**nm-settings-keyfile(5)** man ページを参照してください。

重要

接続プロファイルには秘密鍵やパスフレーズなどの機密情報が含まれる可能性があるため、セキュリティー上の理由から、NetworkManager は **root** ユーザーが所有し、**root** のみが読み書きできる設定ファイルのみを使用します。

接続プロファイルの目的に応じて、次のいずれかのディレクトリーに保存します。

- **/etc/NetworkManager/system-connections/**:永続プロファイルの場所。NetworkManager API を使用して、永続プロファイルを変更すると、NetworkManager は、このディレクトリーにファイルを書き込み、上書きします。
- **/run/NetworkManager/system-connections/**:システムの再起動時に自動的に削除される一時プロファイル。
- **/usr/lib/NetworkManager/system-connections/**:事前にデプロイされた不変プロファイル用。NetworkManager の API を使用してこのようなプロファイルを編集すると、NetworkManager はこのプロファイルを永続ストレージまたは一時ストレージのいずれかにコピーします。

NetworkManager は、ディスクからプロファイルを自動的に再読み込みしません。キーファイル形式で接続プロファイルを作成または更新する場合は、**nmcli connection reload** コマンドを使用して、変更を NetworkManager に通知します。

37.2. NMCLI を使用したオフラインモードでのキーファイル接続プロファイルの作成

Red Hat は、**nmcli**、**network** RHEL システムロール、または **nmstate** API などの NetworkManager ユーティリティーを使用して NetworkManager 接続を管理し、設定ファイルを作成および更新することを推奨しています。ただし、**nmcli --offline connection add** コマンドを使用して、オフラインモードでキーファイル形式のさまざまな接続プロファイルを作成することもできます。

オフラインモードでは、**nmcli** が **NetworkManager** サービスなしで動作し、標準出力を介してキーファイル接続プロファイルを生成することが保証されます。この機能は、次の場合に役立ちます。

- どこかに事前に展開する必要がある接続プロファイルを作成する場合。たとえば、コンテナーイメージ内、または RPM パッケージとして作成する場合。
- **NetworkManager** サービスが利用できない環境で接続プロファイルを作成する場合。たとえば、**chroot** ユーティリティーを使用する場合。または、Kickstart **%post** スクリプトを使用してインストールする RHEL システムのネットワーク設定を作成または変更する場合。

次の接続プロファイルタイプを作成できます。

- 静的イーサネット接続
- 動的イーサネット接続
- ネットワークボンド
- ネットワークブリッジ
- VLAN またはサポートされているあらゆる種類の接続

手順

1. キーファイル形式で新しい接続プロファイルを作成します。たとえば、DHCP を使用しないイーサネットデバイスの接続プロファイルの場合は、同様の **nmcli** コマンドを実行します。

```
# 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
```

注記

con-name キーで指定した接続名は、生成されたプロファイルの **id** 変数に保存されます。後で **nmcli** コマンドを使用してこの接続を管理する場合は、次のように接続を指定します。

- **id** 変数を省略しない場合は、**Example-Connection** などの接続名を使用します。
- **id** 変数を省略する場合は、**output** のように **.nmconnection** 接尾辞のないファイル名を使用します。

2. 設定ファイルにパーミッションを設定して、**root** ユーザーのみが読み取りおよび更新できるようにします。

```
# chmod 600 /etc/NetworkManager/system-connections/output.nmconnection
# chown root:root /etc/NetworkManager/system-connections/output.nmconnection
```

3. **NetworkManager** サービスを開始します。

```
# systemctl start NetworkManager.service
```

4. プロファイルの **autoconnect** 変数を **false** に設定した場合は、コネクションをアクティブにします。

```
# nmcli connection up Example-Connection
```

検証

1. **NetworkManager** サービスが実行されていることを確認します。

```
# systemctl status NetworkManager.service
● NetworkManager.service - Network Manager
   Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; vendor preset:
enabled)
   Active: active (running) since Wed 2022-08-03 13:08:32 CEST; 1min 40s ago
...
```

2. NetworkManager が設定ファイルからプロファイルを読み込めることを確認します。

```
# 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
...
```

新しく作成された接続が出力に表示されない場合は、使用したキーファイルのパーミッションと構文が正しいことを確認してください。

3. 接続プロファイルを表示します。

```
# 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
...
```

関連情報

- **nmcli(1)**

- **nm-settings-keyfile(5)**
- NetworkManager プロファイルのキーファイル形式
- nmcli を使用したイーサネット接続の設定
- nmcli を使用した VLAN タグ付けの設定
- nmcli を使用したネットワークブリッジの設定
- nmcli を使用したネットワークボンディングの設定

37.3. キーファイル形式での NETWORKMANAGER プロファイルの手動作成

NetworkManager 接続プロファイルは、キーファイル形式で手動で作成できます。

注記

設定ファイルを手動で作成または更新すると、予期しないネットワーク設定や、機能しないネットワーク設定が発生する可能性があります。代わりに、オフラインモードで **nmcli** を使用できます。nmcli を使用したオフラインモードでのキーファイル接続プロファイルの作成 を参照してください。

手順

1. Ethernet などのハードウェアインターフェイスのプロファイルを作成する場合は、このインターフェイスの MAC アドレスを表示します。

```
# 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. 接続プロファイルを作成します。たとえば、DHCP を使用するイーサネットデバイスの接続プロファイルを作成する場合は、次の内容で **/etc/NetworkManager/system-connections/example.nmconnection** ファイルを作成します。

```
[connection]
id=example_connection
type=ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[ethernet]
mac-address=00:53:00:8f:fa:66
```

注記

ファイル名には、**.nmconnection** の接尾辞を付けた任意のファイル名を使用できます。ただし、後で **nmcli** コマンドを使用して接続を管理する場合は、この接続を参照する際に、**id** に設定した接続名を使用する必要があります。**id** を省略する場合は、**.nmconnection** を使用せずにファイルネームを使用して、このコネクションを参照してください。

3. 設定ファイルにパーミッションを設定して、**root** のユーザーのみが読み取りおよび更新できるようにします。

```
# chown root:root /etc/NetworkManager/system-connections/example.nmconnection
# chmod 600 /etc/NetworkManager/system-connections/example.nmconnection
```

4. 接続プロファイルを再読み込みします。

```
# nmcli connection reload
```

5. NetworkManager が設定ファイルからプロファイルを読み込んでいることを確認します。

```
# nmcli -f NAME,UUID,FILENAME connection
NAME               UUID                                  FILENAME
example-connection  86da2486-068d-4d05-9ac7-957ec118afba
/etc/NetworkManager/system-connections/example.nmconnection
...
```

このコマンドで、新しく追加した接続が表示されない場合は、ファイルの権限と、ファイルで使用した構文が正しいことを確認します。

6. プロファイルの **autoconnect** 変数を **false** に設定した場合は、コネクションをアクティブにします。

```
# nmcli connection up example_connection
```

検証

1. 接続プロファイルを表示します。

```
# nmcli connection show example_connection
```

関連情報

- **nm-settings-keyfile(5)**

37.4. IFCFG およびキーファイル形式でのプロファイルを使用したインターフェイスの名前変更における違い

provider または **lan** などのカスタムネットワークインターフェイス名を定義して、インターフェイス名をよりわかりやすいものにすることができます。この場合、**udev** サービスはインターフェイスの名前を変更します。名前変更プロセスは、**ifcfg** またはキーファイル形式で接続プロファイルを使用するかどうかによって異なる動作をします。

ifcfg 形式でプロファイルを使用する場合のインターフェイスの名前変更プロセス

1. **/usr/lib/udev/rules.d/60-net.rules udev** ルールは、**/lib/udev/rename_device** ヘルパーユーティリティーを呼び出します。
2. ヘルパーユーティリティーは、**/etc/sysconfig/network-scripts/ifcfg-*** ファイルの **HWADDR** パラメーターを検索します。
3. 変数に設定した値がインターフェイスの MAC アドレスに一致すると、ヘルパーユーティリティーは、インターフェイスの名前を、ファイルの **DEVICE** パラメーターに設定した名前に変更します。

キーファイル形式でプロファイルを使用する場合のインターフェイスの名前変更プロセス

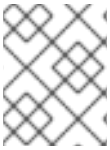
1. インターフェイスの名前を変更する systemd リンクファイル または udev ルール を作成します。
2. NetworkManager 接続プロファイルの **interface-name** プロパティーで、カスタムインターフェイス名を使用します。

関連情報

- udev デバイスマネージャーによるネットワークインターフェイスの名前変更の仕組み
- udev ルールを使用したユーザー定義のネットワークインターフェイス名の設定
- systemd リンクファイルを使用したユーザー定義のネットワークインターフェイス名の設定

37.5. IFCFG からキーファイル形式への NETWORKMANAGER プロファイルの移行

非推奨の **ifcfg** 形式で引き続き接続プロファイルを使用する場合は、それらをキーファイル形式に変換できます。



注記

ifcfg ファイルに **NM_CONTROLLED=no** 設定が含まれる場合、NetworkManager は、このプロファイルを制御しないため、移行プロセスはそれを無視します。

前提条件

- **/etc/sysconfig/network-scripts/** ディレクトリーに **ifcfg** 形式の接続プロファイルがある。
- 接続プロファイルに、**provider** や **lan** などのカスタムデバイス名に設定されている **DEVICE** 変数が含まれている場合は、カスタムデバイス名ごとに systemd リンクファイル または udev ルール を作成している。

手順

- 接続プロファイルを移行します。

```
# nmcli connection migrate
Connection 'enp1s0' (43ed18ab-f0c4-4934-af3d-2b3333948e45) successfully migrated.
Connection 'enp2s0' (883333e8-1b87-4947-8ceb-1f8812a80a9b) successfully migrated.
```

```
...
```

検証

- 必要に応じて、すべての接続プロファイルが正常に移行されたことを確認できます。

```
# 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
...
```

関連情報

- **nm-settings-keyfile(5)**
- **nm-settings-ifcfg-rh(5)**
- udev デバイスマネージャーによるネットワークインターフェイスの名前変更の仕組み

第38章 SYSTEMD ネットワークターゲットおよびサービス

NetworkManager は、システムの起動時にネットワークを設定します。ただし、root ディレクトリーが iSCSI デバイスに保存されている場合など、リモートルート (/) で起動すると、RHEL が起動する前に、ネットワーク設定が初期 RAM ディスク (**initrd**) に適用されます。たとえば、**rd.neednet=1** を使用してカーネルコマンドラインでネットワーク設定を指定すると、リモートファイルシステムのマウントに設定を指定すると、ネットワーク設定が **initrd** に適用されます。

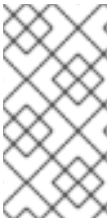
RHEL は、ネットワーク設定を適用する間に、**network** および **network-online** ターゲットと **NetworkManager-wait-online** サービスを使用します。また、これらのサービスを動的にリロードできない場合には、ネットワークが完全に利用可能になった後に **systemd** サービスを起動するように設定できます。

38.1. SYSTEMD ターゲット NETWORK と NETWORK-ONLINE の違い

Systemd は、ターゲットユニット **network** および **network-online** を維持します。**NetworkManager-wait-online.service** などの特殊ユニットは、**WantedBy=network-online.target** パラメーターおよび **Before=network-online.target** パラメーターを持ちます。有効にすると、このようなユニットは **network-online.target** で開始し、一部の形式のネットワーク接続が確立されるまでターゲットに到達させるよう遅延します。ネットワークが接続されるまで、**network-online** ターゲットが遅延します。

network-online ターゲットはサービスを開始します。これにより、実行の遅延が大幅に増加します。Systemd は、このターゲットユニットの **Wants** パラメーターおよび **After** パラメーターの依存関係を、**$network** ファシリティーを参照する Linux Standard Base (LSB) ヘッダーを持つすべての System V(SysV) **init** スクリプトサービスユニットに自動的に追加します。LSB ヘッダーは、**init** スクリプトのメタデータです。これを使用して依存関係を指定できます。これは **systemd** ターゲットに似ています。

network ターゲットは、起動プロセスの実行を大幅に遅らせません。**network** ターゲットに到達すると、ネットワークの設定を行うサービスが開始していることになります。ただし、ネットワークデバイスが設定されているわけではありません。このターゲットは、システムのシャットダウン時に重要です。たとえば、起動中に **network** ターゲットの後に順序付けされたサービスがあると、この依存関係はシャットダウン中に元に戻されます。サービスが停止するまで、ネットワークは切断されません。リモートネットワークファイルシステムのすべてのマウントユニットは、**network-online** ターゲットユニットを自動的に起動し、その後に自身を置きます。



注記

network-online ターゲットユニットは、システムの起動時にのみ役に立ちます。システムの起動が完了すると、このターゲットがネットワークのオンライン状態を追跡しなくなります。したがって、**network-online** を使用してネットワーク接続を監視することはできません。このターゲットは、1回限りのシステム起動の概念を提供します。

38.2. NETWORKMANAGER-WAIT-ONLINE の概要

NetworkManager-wait-online サービスは、ネットワークを設定するタイムアウトで待機します。このネットワーク設定には、イーサネットデバイスへのプラグイン、Wi-Fi デバイスのスキャンなどが含まれます。NetworkManager は、自動的に起動するように設定された適切なプロファイルを自動的にアクティブにします。DHCP のタイムアウトや同様のイベントによる自動アクティベーションプロセスが失敗しても、NetworkManager が長時間ビジー状態を維持される可能性があります。設定によっては、NetworkManager は同じプロファイルまたは別のプロファイルのアクティブ化を再試行します。

起動が完了すると、すべてのプロファイルが非接続状態であるか、正常にアクティベートされます。プロファイルを自動接続するように設定できます。以下は、タイムアウトを設定したり、接続がアクティブとみなされるタイミングを定義するいくつかのパラメーター例です。

- **connection.wait-device-timeout** - ドライバーがデバイスを検出するためのタイムアウトを設定します。
- **ipv4.may-fail** および **ipv6.may-fail** - 1 つの IP アドレスファミリーの準備ができている状態でアクティベーションを設定します。または、特定のアドレスファミリーが設定を完了しているかどうかを設定します。
- **ipv4.gateway-ping-timeout** - アクティベーションを遅延します。

関連情報

- **nm-settings(5)** man ページ

38.3. ネットワークの開始後に SYSTEMD サービスが起動する設定

Red Hat Enterprise Linux は、**systemd** サービスファイルを **/usr/lib/systemd/system/** ディレクトリーにインストールします。以下の手順では、**/etc/systemd/system/service_name.service.d/** にあるサービスファイル用のドロップインスニペットを作成し、**/usr/lib/systemd/system/** にあるサービスファイルとともに、ネットワークがオンラインになった後に特定の **サービス** を開始するために使用します。ドロップインスニペットの設定が、**/usr/lib/systemd/system/** 内のサービスファイルにある値と重複する場合は、優先度が高くなります。

手順

1. エディターでサービスファイルを開くには、次のコマンドを実行します。

```
# systemctl edit service_name
```

2. 以下を入力し、変更を保存します。

```
[Unit]
After=network-online.target
```

3. **systemd** サービスを再読み込みします。

```
# systemctl daemon-reload
```

第39章 NMSTATE の概要

nmstate は宣言型のネットワークマネージャー API です。**nmstate** パッケージは、RHEL の NetworkManager を管理するために、 **libnmstate** Python ライブラリー、およびコマンドラインユーティリティー **nmstatectl** を提供します。Nmstate を使用する場合、YAML または JSON 形式の命令を使用して想定されるネットワーク状態を記述します。

Nmstate には多くの利点があります。たとえば、以下のようになります。

- 安定性と拡張可能なインターフェイスを提供して RHEL ネットワーク機能を管理する。
- ホストおよびクラスターレベルでのアトミックおよびトランザクション操作をサポートする。
- ほとんどのプロパティーの部分編集をサポートし、この手順で指定されていない既存の設定を保持する。
- 管理者が独自のプラグインを使用できるようにプラグインサポートを提供する。

39.1. PYTHON アプリケーションでの LIBNMSTATE ライブラリーの使用

libnmstate Python ライブラリーを使用すると、開発者は独自のアプリケーションで Nmstate を使用できます。

ライブラリーを使用するには、ソースコードにインポートします。

```
import libnmstate
```

このライブラリーを使用するには、**nmstate** パッケージをインストールする必要があることに注意してください。

例39.1 libnmstate ライブラリーを使用したネットワーク状態のクエリー

以下の Python コードは、**libnmstate** ライブラリーをインポートし、利用可能なネットワークインターフェイスとその状態を表示します。

```
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. NMSTATECTL を使用した現在のネットワーク設定の更新

nmstatectl ユーティリティーを使用して、1つまたはすべてのインターフェイスの現在のネットワーク設定をファイルに保存できます。このファイルを使用して、以下を行うことができます。

- 設定を変更し、同じシステムに適用します。
- 別のホストにファイルをコピーし、同じまたは変更された設定でホストを設定します。

たとえば、**enp1s0** インターフェイスの設定をファイルにエクスポートして、設定を変更し、その設定をホストに適用することができます。

前提条件

- **nmstate** パッケージがインストールされている。

手順

1. **enp1s0** インターフェイスの設定を ~/**network-config.yml** ファイルにエクスポートします。

```
# nmstatectl show enp1s0 > ~/network-config.yml
```

このコマンドにより、**enp1s0** の設定が YAML 形式で保存されます。JSON 形式で出力を保存するには、**--json** オプションをコマンドに渡します。

インターフェイス名を指定しない場合、**nmstatectl** はすべてのインターフェイスの設定をエクスポートします。

2. テキストエディターで ~/**network-config.yml** ファイルを変更して、設定を更新します。

3. ~/**network-config.yml** ファイルからの設定を適用します。

```
# nmstatectl apply ~/network-config.yml
```

JSON 形式で設定をエクスポートしている場合は、**--json** オプションをコマンドに渡します。

39.3. NMSTATE SYSTEMD サービス

nmstate systemd サービスを設定することで、Red Hat Enterprise Linux システムの起動時に新しいネットワーク設定を自動的に適用できます。

nmstate パッケージをインストールすると、Nmstate 命令を含む ***.yml** ファイルを /**etc**/**nmstate**/ ディレクトリーに保存できます。**nmstate** サービスは、次回の再起動時、またはサービスを手動で再起動したときに、ファイルを自動的に適用します。Nmstate はファイルを正常に適用した後、サービスが同じファイルを再度処理しないように、ファイルの **.yml** 接尾辞の名前を **.applied** に変更します。

nmstate サービスは、**oneshot systemd** サービスです。したがって、**systemd** は、システムの起動時とサービスを手動で再起動したときにのみ、これを実行します。

注記

デフォルトでは、**nmstate** サービスは無効になっています。これを有効にするには **systemctl enable nmstate** コマンドを使用します。その後、**systemd** は、システムが起動するたびにこのサービスを実行します。

39.4. NETWORK RHEL システムロールのネットワーク状態

network RHEL システムロールは、Playbook でデバイスを設定するための状態設定をサポートしています。これには、**network_state** 変数の後に状態設定を使用します。

Playbook で **network_state** 変数を使用する利点:

- 状態設定で宣言型の方法を使用すると、インターフェイスを設定でき、NetworkManager はこれらのインターフェイスのプロファイルをバックグラウンドで作成します。
- **network_state** 変数を使用すると、変更が必要なオプションを指定できます。他のすべてのオプションはそのまま残ります。ただし、**network_connections** 変数を使用して、ネットワーク接続プロファイルを変更するには、すべての設定を指定する必要があります。

たとえば、動的 IP アドレス設定でイーサネット接続を作成するには、Playbook で次の **vars** ブロックを使用します。

状態設定を含む Playbook	通常の Playbook

vars:
  network_state:
  interfaces:
  - name: enp7s0
  type: ethernet
  state: up
  ipv4:
  enabled: true
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  ipv6:
  enabled: true
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  autoconf: true
  dhcp: true
	vars:
	  network_connections:
	  - name: enp7s0
	  interface_name: enp7s0
	  type: ethernet
	  autoconnect: yes
	  ip:
	  dhcp4: yes
	  auto6: yes
	  state: up

たとえば、上記のように作成した動的 IP アドレス設定の接続ステータスのみを変更するには、Playbook で次の **vars** ブロックを使用します。

状態設定を含む Playbook	通常の Playbook

vars:
  network_state:
  interfaces:
  - name: enp7s0
  type: ethernet
  state: down
	vars:
	  network_connections:
	  - name: enp7s0
	  interface_name: enp7s0
	  type: ethernet
	  autoconnect: yes
	  ip:
	  dhcp4: yes
	  auto6: yes
	  state: down

関連情報

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** ファイル
- **/usr/share/doc/rhel-system-roles/network/** ディレクトリー

39.5. 関連情報

- **/usr/share/doc/nmstate/README.md**
- **/usr/share/doc/nmstate/examples/**

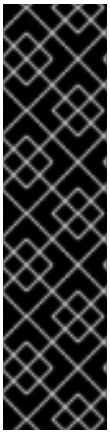
第40章 ネットワークパケットのキャプチャー

ネットワークの問題と通信をデバッグするには、ネットワークパケットをキャプチャーできます。以下のセクションでは、ネットワークパケットのキャプチャーに関する手順と追加情報を提供します。

40.1. XDP プログラムがドロップしたパケットを含むネットワークパケットをキャプチャーするために XDPDUMP を使用

xdpdump ユーティリティーは、ネットワークパケットをキャプチャーします。**tcpdump** ユーティリティーとは異なり、**xdpdump** はこのタスクに extended Berkeley Packet Filter(eBPF) プログラムを使用します。これにより、**xdpdump** は Express Data Path (XDP) プログラムによりドロップされたパケットをキャプチャーできます。**tcpdump** などのユーザー空間ユーティリティーは、この削除されたパッケージや、XDP プログラムによって変更された元のパケットをキャプチャーできません。

xdpdump を使用して、インターフェイスにすでに割り当てられている XDP プログラムをデバッグすることができます。したがって、ユーティリティーは、XDP プログラムを起動し、終了する前にパケットをキャプチャーできます。後者の場合、**xdpdump** は XDP アクションもキャプチャーします。デフォルトでは、**xdpdump** は XDP プログラムのエントリーで着信パケットをキャプチャーします。



重要

AMD および Intel 64 ビット以外のアーキテクチャーでは、**xdpdump** ユーティリティーはテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

xdpdump には、パケットフィルターまたはデコード機能がないことに注意してください。ただし、パケットのデコードに **tcpdump** と組み合わせて使用できます。

前提条件

- XDP プログラムをサポートするネットワークドライバー。
- XDP プログラムが **enp1s0** インターフェイスに読み込まれている。プログラムが読み込まれていない場合は、**xdpdump** が後方互換性として **tcpdump** と同様にパケットをキャプチャーします。

手順

1. **enp1s0** インターフェイスでパケットをキャプチャーして、**/root/capture.pcap** ファイルに書き込むには、次のコマンドを実行します。

```
# xdpdump -i enp1s0 -w /root/capture.pcap
```

2. パケットの取得を停止するには、**Ctrl+C** を押します。

関連情報

- **xdpdump(8)** の man ページ
- 開発者であり、**xdpdump** のソースコードに関心がある場合は、Red Hat カスタマーポータルから対応するソース RPM (SRPM) をダウンロードしてインストールします。

40.2. 関連情報

- How to capture network packets with tcpdump?

第41章 RHEL 9 の EBPF ネットワーク機能について

eBPF (extended Berkeley Packet Filter) は、カーネル領域でのコード実行を可能にするカーネル内の仮想マシンです。このコードは、限られた一連の関数にのみアクセスできる制限付きサンドボックス環境で実行されます。

ネットワークでは、eBPF を使用してカーネルパケット処理を補完したり、置き換えることができます。使用するフックに応じて、eBPF プログラムには以下のような記述があります。

- パケットデータおよびメタデータへの読み取りおよび書き込みアクセス
- ソケットとルートを検索できる
- ソケットオプションを設定できる
- パケットをリダイレクト可能

41.1. RHEL 9 におけるネットワーク EBPF 機能の概要

eBPF (extended Berkeley Paket Filter) ネットワークプログラムは、RHEL の以下のフックに割り当てることができます。

- eXpress Data Path (XDP):カーネルネットワークスタックが受信したパケットを処理する前に、このパケットへの早期アクセスを提供します。
- direct-action フラグを持つ **tc** eBPF 分類子:ingress および egress での強力なパケット処理を提供します。
- Control Groups version 2 (cgroup v2):コントロールグループ内のプログラムが実行するソケットベースの操作のフィルタリングおよび上書きを有効にします。
- ソケットフィルタリング:ソケットから受信したパケットのフィルタリングを有効にします。この機能は、従来の Berkeley Packet Filter (cBPF) でも利用できますが、eBPF プログラムに対応するために拡張されました。
- ストリームパーサー:個別のメッセージへのストリームの分散、フィルタリング、ソケットへのリダイレクトを有効にします。
- **SO_REUSEPORT** ソケットの選択:**reuseport** ソケットグループから受信したソケットをプログラム可能な選択を提供します。
- Flow dissector:特定の状況でカーネルがパケットヘッダーを解析する方法をオーバーライドします。
- TCP 輻輳制御コールバック:カスタム TCP 輻輳制御アルゴリズムの実装を有効にします。
- カプセル化によるルート:カスタムのトンネルカプセル化の作成を有効にします。

XDP

BPF_PROG_TYPE_XDP タイプのプログラムはネットワークインターフェイスに割り当てることができます。次にカーネルは、カーネルネットワークスタックが処理を開始する前に受信したパケットでプログラムを実行します。これにより、高速パケットドロップなど、特定の状況で高速なパケット転送が可能になり、負荷分散シナリオにおいて DDoS (Distributed Denial of Service) 攻撃や高速パケットリダイレクトを防ぐことができます。

さまざまな形式のパケット監視やサンプリングに XDP を使用することもできます。カーネルは、XDP プログラムはパケットを変更し、カーネルネットワークスタックへのさらなる処理を可能にします。

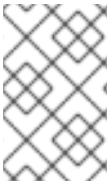
以下の XDP モードを使用できます。

- ネイティブ (ドライバー) XDP:カーネルは、パケット受信時に最速の可能点からプログラムを実行します。この時点で、カーネルはパケットを解析しなかったため、カーネルが提供するメタデータは利用できません。このモードでは、ネットワークインターフェイスドライバーが XDP をサポートしている必要がありますが、すべてのドライバーがこのネイティブモードをサポートするわけではありません。

- ジェネリック XDP:カーネルネットワークスタックは、処理の初期段階で XDP プログラムを実行します。この時点で、カーネルデータ構造が割り当てられ、パケットを事前に処理しています。パケットをドロップまたはリダイレクトする必要がある場合は、ネイティブモードと比較して大きなオーバーヘッドが必要になります。ただし、汎用モードはネットワークインターフェイスドライバーのサポートを必要とせず、すべてのネットワークインターフェイスで機能します。

- オフロード XDP:カーネルは、ホストの CPU 上ではなく、ネットワークインターフェイスで XDP プログラムを実行します。これには特定のハードウェアが必要で、特定の eBPF 機能のみがこのモードで使用できることに注意してください。

RHEL で、**libxdp** ライブラリーを使用してすべての XDP プログラムを読み込みます。このライブラリーは、XDP のシステム制御を可能にします。



注記

現在、XDP プログラムにはシステム設定に制限があります。たとえば、受信側インターフェイスで特定のハードウェアオフロード機能を無効にする必要があります。また、ネイティブモードをサポートするすべてのドライバーで利用可能なわけではありません。

RHEL 9 では、**libxdp** ライブラリーを使用してプログラムをカーネルにロードする場合にのみ、Red Hat は XDP 機能をサポートします。

AF_XDP

指定した **AF_XDP** ソケットにパケットをフィルターしてリダイレクトする XDP プログラムを使用すると、**AF_XDP** プロトコルファミリーから 1 つ以上のソケットを使用して、カーネルからユーザー空間にパケットを速やかにコピーできます。

トラフィック制御

Traffic Control (**tc**) サブシステムは、以下のタイプの eBPF プログラムを提供します。

- **BPF_PROG_TYPE_SCHED_CLS**

- **BPF_PROG_TYPE_SCHED_ACT**

これらのタイプを使用すると、カスタム **tc** 分類子と **tc** アクションを eBPF に記述できます。これは、**tc** エコシステムの一部とともに、強力なパケット処理機能を提供します。また、複数のコンテナーネットワークオーケストレーションソリューションの中核となります。

多くの場合、direct-action フラグと同様に、eBPF 分類子は、同じ eBPF プログラムから直接アクションを実行できます。**clsact** Queueing Discipline (**qdisc**) は、Ingress 側でこれを有効にするように設計されています。

flow dissector の eBPF プログラムは、**flower** などのその他の **qdiscs** や **tc** 分類子の操作に影響を与える可能性があることに注意してください。

ソケットフィルター
複数のユーティリティーは、ソケットで受信されるパケットのフィルタリングに、従来の Berkeley Packet Filter (cBPF) を使用または使用しています。たとえば、**tcpdump** ユーティリティーを使用すると、ユーザーは、どの **tcpdump** を cBPF コードに変換するか、式を指定できます。

cBPF の代替として、カーネルは、同じ目的で **BPF_PROG_TYPE_SOCKET_FILTER** タイプの eBPF プログラムを許可します。

コントロールグループ
RHEL では、cgroup に割り当てられる eBPF プログラムを複数使用できます。カーネルは、指定の cgroup のプログラムが操作を実行する際に、これらのプログラムを実行します。cgroups バージョン 2 のみを使用できます。

RHEL では、以下のネットワーク関連の cgroup eBPF プログラムが利用できます。

- **BPF_PROG_TYPE_SOCK_OPS**:カーネルは、さまざまな TCP イベントでこのプログラムを呼び出します。プログラムは、カスタム TCP ヘッダーオプションなどを含め、カーネル TCP スタックの動作を調整できます。

- **BPF_PROG_TYPE_CGROUP_SOCK_ADDR**:**connect**、**bind**、**sendto**、**recvmsg**、**getpeername**、および **getsockname** の操作時に呼び出されます。このプログラムは、IP アドレスとポートを変更できます。これは、ソケットベースのネットワークアドレス変換 (NAT) を eBPF に実装する場合に便利です。

- **BPF_PROG_TYPE_CGROUP_SOCKOPT**:カーネルは、**setsockopt** および **getsockopt** 操作時にこのプログラムを呼び出して、オプションの変更を可能にします。

- **BPF_PROG_TYPE_CGROUP_SOCK**:カーネルは、ソケットの作成時、ソケットの開放時、アドレスのバインド時にこのプログラムを呼び出します。これらのプログラムを使用して操作を許可または拒否するか、統計のソケット作成の検査のみを行います。

- **BPF_PROG_TYPE_CGROUP_SKB**:このプログラムは ingress および egress の個別のパケットをフィルターし、パケットを受信または拒否できます。

- **BPF_PROG_TYPE_CGROUP_SYSCTL**:このプログラムはシステム制御 (**sysctl**) へのアクセスをフィルタリングできます。

ストリームパーサー
ストリームパーサーは、特別な eBPF マップに追加されるソケットのグループで動作します。次に、eBPF プログラムは、カーネルがこれらのソケットで受信または送信するパケットを処理します。

RHEL では、以下のストリームパーサー eBPF プログラムを利用できます。

- **BPF_PROG_TYPE_SK_SKB**:eBPF プログラムは、ソケットから受信したパケットを個別のメッセージに解析したり、それらのメッセージをドロップしたり、グループ内の別のソケットに送信するようにカーネルに指示します。

- **BPF_PROG_TYPE_SK_MSG**:このプログラムは egress メッセージをフィルタリングします。eBPF プログラムは、パケットを個別のメッセージを解析し、そのパケットを承認または拒否します。

SO_REUSEPORT ソケットの選択
このソケットオプションを使用することで、複数のソケットを同じ IP アドレスとポートにバインドできます。eBPF がない場合、カーネルは接続ハッシュに基づいて受信ソケットを選択します。**BPF_PROG_TYPE_SK_REUSEPORT** プログラムを使用すると、受信ソケットの選択が完全にプログラム可能になります。

Flow dissector
プロトコルの完全なデコードを待たずにカーネルがパケットヘッダーを処理する必要がある場合、これらは **破棄されます**。たとえば、これは、**tc** サブシステム、ボンディングのルーティング、またはパケットのハッシュを計算する際に発生します。この場合、カーネルはパケットヘッダーを解析し、パケットヘッダーからの情報を使用して内部構造を埋めます。この内部解析は、**BPF_PROG_TYPE_FLOW_DISSECTOR** プログラムを使用して置き換えることができます。RHEL の eBPF では、TCP および UDP を IPv4 および IPv6 上でのみ破棄できます。

TCP 輻輳制御
struct tcp_congestion_oops コールバックを実装する **BPF_PROG_TYPE_STRUCT_OPS** プログラムのグループを使用して、カスタム TCP 輻輳制御アルゴリズムを作成できます。この方法を実装するアルゴリズムは、ビルトインのカーネルアルゴリズムとともにシステムで利用できます。

カプセル化によるルート
以下のいずれかの eBPF プログラムタイプは、トンネルのカプセル化属性として、ルーティングテーブルのルートに割り当てることができます。

- **BPF_PROG_TYPE_LWT_IN**
- **BPF_PROG_TYPE_LWT_OUT**
- **BPF_PROG_TYPE_LWT_XMIT**

このような eBPF プログラムの機能は特定のトンネル設定に限定され、汎用のカプセル化またはデシリアライズソリューションの作成はできません。

ソケットルックアップ
bind システムコールの制限を回避するには、**BPF_PROG_TYPE_SK_LOOKUP** タイプの eBPF プログラムを使用します。このようなプログラムは、新しい受信 TCP 接続のリスニングソケットまたは UDP パケットの非接続ソケットを選択できます。

41.2. RHEL 9 におけるネットワークカードごとの XDP 機能の概要

以下は、XDP 対応ネットワークカードと、それらで使用できる XDP 機能の概要です。

ネットワークカード	ドライバー	ベーシック	リダイレクト	ターゲット	HW オフロード	Zero-copy	Large MTU
Amazon Elastic Network Adapter	**ena**	はい	はい	はい [a]	いいえ	いいえ	いいえ
aQuania AQtion イーサネットカード	**atlantic**	はい	はい	いいえ	いいえ	いいえ	いいえ
Broadcom NetXtreme-C/E 10/25/40/50 gigabit Ethernet	**bnxt_en**	はい	はい	はい [a]	いいえ	いいえ	はい
Cavium Thunder Virtual function	**nicvf**	はい	いいえ	いいえ	いいえ	いいえ	いいえ

ネットワークカード	ドライバー	ベーシック	リダイレクト	ターゲット	HW オフロード	Zero-copy	Large MTU
Google Virtual NIC (gVNIC) のサポート	**gve**	はい	はい	はい	いいえ	はい	いいえ
Intel® 10GbE PCI Express Virtual Function Ethernet	**ixgbevf**	はい	いいえ	いいえ	いいえ	いいえ	いいえ
Intel® 10GbE PCI Express adapters	**ixgbe**	はい	はい	はい [a]	いいえ	はい	はい [b]
Intel® Ethernet Connection E800 Series	**ice**	はい	はい	はい [a]	いいえ	はい	はい
Intel® Ethernet Controller I225-LM/I225-V family	**igc**	はい	はい	はい	いいえ	はい	はい [b]
Intel® PCI Express Gigabit adapters	**igb**	はい	はい	はい [a]	いいえ	いいえ	はい [b]
Intel® Ethernet Controller XL710 Family	**i40e**	はい	はい	はい [a] [c]	いいえ	はい	いいえ
Marvell OcteonTX2	**rvu_nicpf**	はい	はい	必須 [a] [c]	いいえ	いいえ	いいえ
Mellanox 5th generation network adapters (ConnectX series)	**mlx5_core**	はい	はい	はい [c]	いいえ	はい	はい
Mellanox Technologies 1/10/40Gbit Ethernet	**mlx4_en**	はい	はい	いいえ	いいえ	いいえ	いいえ
Microsoft Azure Network Adapter	**mana**	はい	はい	はい	いいえ	いいえ	いいえ
Microsoft Hyper-V virtual network	**hv_netvsc**	はい	はい	はい	いいえ	いいえ	いいえ
Netronome® NFP4000/NFP6000 NIC [d]	**nfp**	はい	いいえ	いいえ	はい	はい	いいえ
QEMU Virtio network	**virtio_net**	はい	はい	はい [a]	いいえ	いいえ	はい

ネットワークカード	ドライバー	ベーシック	リダイレクト	ターゲット	HW オフロード	Zero-copy	Large MTU
QLogic QED 25/40/100Gb Ethernet NIC	**qede**	はい	はい	はい	いいえ	いいえ	いいえ
STMicroelectronics Multi-Gigabit Ethernet	**stmmac**	はい	はい	はい	いいえ	はい	いいえ
Solarflare SFC9000/SFC9100/EF100-family	**sfc**	はい	はい	はい [c]	いいえ	いいえ	いいえ
Universal TUN/TAP device	**tun**	はい	はい	はい	いいえ	いいえ	いいえ
Virtual ethernet pair device	**veth**	はい	はい	はい	いいえ	いいえ	はい
VMware VMXNET3 イーサネットドライバー	**vmxnet3**	はい	はい	必須 [a] [c]	いいえ	いいえ	いいえ
Xen 準仮想ネットワークデバイス	**xen-netfront**	はい	はい	はい	いいえ	いいえ	いいえ

[a] XDP プログラムがインターフェイスで読み込まれている場合にのみします。

[b] 送信側のみ。XDP 経由で大きなパケットを受信することはできません。

[c] 最大の CPU インデックス以上の XDP TX キューを複数割り当てる必要があります。

[d] リストされている機能の一部は、Netronome® NFP3800 NIC では使用できません。

説明:

- Basic:基本的な戻りコード**DROP**、**PASS**、**ABORTED**、および **TX** に対応します。
- Redirect:**XDP_REDIRECT** の戻りコードをサポートします。
- Target:**XDP_REDIRECT** の戻りコードのターゲットにすることができます。
- HW offload:XDP ハードウェアオフロードをサポートします。
- Zero-copy:**AF_XDP** プロトコルファミリーの zero-copy モードをサポートします。
- Large MTU:ページサイズより大きいパケットをサポートします。

第42章 BPF コンパイラーコレクションを使用したネットワークトレース

BPF コンパイラーコレクション (BCC) は、eBPF (extended Berkeley Packet Filter) プログラムの作成を容易にするライブラリーです。eBPF プログラムの主なユーティリティーは、オーバーヘッドやセキュリティー上の問題が発生することなく、オペレーティングシステムのパフォーマンスおよびネットワークパフォーマンスを分析することです。

BCC により、ユーザーは eBPF の技術詳細を把握する必要がなくなり、事前に作成した eBPF プログラムを含む **bcc-tools** パッケージなど、多くの標準スタートポイントを利用できます。

注記

eBPF プログラムは、ディスク I/O、TCP 接続、プロセス作成などのイベントでトリガーされます。プログラムがカーネルのセーフ仮想マシンで実行するため、カーネルがクラッシュしたり、ループしたり、応答しなくなることはあまりありません。

42.1. BCC-TOOLS パッケージのインストール

bcc-tools パッケージをインストールします。これにより、依存関係として BPF Compiler Collection (BCC) ライブラリーもインストールされます。

手順

1. **bcc-tools** をインストールします。

```
# dnf install bcc-tools
```

   BCC ツールは、**/usr/share/bcc/tools/** ディレクトリーにインストールされます。

2. 必要に応じて、ツールを検証します。

```
# 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
...
```

   上記のリストにある **doc** ディレクトリーには、各ツールのドキュメントが含まれます。

42.2. カーネルの受け入れキューに追加された TCP 接続の表示

カーネルは、TCP 3 方向ハンドシェイクで **ACK** パケットを受け取ると、カーネルは接続の状態が **ESTABLISHED** に変更された後に **SYN** キューから **accept** キューに移動します。そのため、正常な TCP 接続だけがこのキューに表示されます。

tcpaccept ユーティリティーは、eBPF 機能を使用して、カーネルが **accept** キューに追加するすべて

の接続を表示します。このユーティリティーは、パケットをキャプチャーしてフィルタリングする代わりにカーネルの **accept()** 関数を追跡するため、軽量です。たとえば、一般的なトラブルシューティングには **tcpaccept** を使用して、サーバーが許可した新しい接続を表示します。

手順

1. 次のコマンドを実行して、カーネルの **許可** キューの追跡を開始します。

```
# /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
...
```

   カーネルが接続を受け入れるたびに、**tcpaccept** は接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpaccept(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpaccept_example.txt** ファイル

42.3. 発信 TCP 接続試行の追跡

tcpconnect ユーティリティーは、eBPF 機能を使用して発信 TCP 接続の試行を追跡します。ユーティリティーの出力には、失敗した接続も含まれます。

tcpconnect ユーティリティーは、パケットを取得してフィルタリングするのではなく、カーネルの **connect()** 関数などを追跡するため、軽量です。

手順

1. 以下のコマンドを入力し、すべての発信接続を表示する追跡プロセスを開始します。

```
# /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
...
```

   カーネルが発信接続を処理するたびに、**tcpconnect** は、接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpconnect(8)** man ページ
- **/usr/share/bcc/tools/doc/tcpconnect_example.txt** ファイル

42.4. 発信 TCP 接続のレイテンシーの測定

TCP 接続のレイテンシーは、接続を確立するのにかかった時間です。通常、これには、アプリケーションのランタイムではなく、カーネル TCP/IP 処理およびネットワークのラウンドトリップタイムが含まれます。

tcpconnlat ユーティリティーは、eBPF 機能を使用して、送信した **SYN** パケットと受信した応答パケットの時間を測定します。

手順

1. 発信接続のレイテンシーの測定を開始します。

```
# /usr/share/bcc/tools/tcpconnlat
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
...
```

   カーネルが発信接続を処理するたびに、**tcpconnlat** は、カーネルが応答パケットを受信すると接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpconnlat(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpconnlat_example.txt** ファイル

42.5. カーネルによって破棄された TCP パケットおよびセグメントの詳細の表示

tcpdrop ユーティリティーを使用すると、管理者はカーネルによって破棄された TCP パケットおよびセグメントの詳細を表示できます。このユーティリティーを使用して、リモートシステムがタイマーベースの再送信を送信する可能性がある破棄されたパケットの高レートをデバッグします。ドロップされたパケットおよびセグメントの高レートは、サーバーのパフォーマンスに影響を与える可能性があります。

リソース集約型のパケットを取得およびフィルタリングする代わりに、**tcpdrop** ユーティリティーは eBPF 機能を使用してカーネルから直接情報を取得します。

手順

1. 以下のコマンドを入力して、破棄された TCP パケットおよびセグメントの詳細表示を開始します。

```
# /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'
 ...
```

カーネルが TCP パケットとセグメントを破棄するたびに、**tcpdrop** は、破棄されたパッケージにつながるカーネルスタックトレースを含む接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpdrop(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpdrop_example.txt** ファイル

42.6. TCP セッションのトレース

tcplife ユーティリティーは eBPF を使用して、開いて閉じる TCP セッションを追跡し、出力を 1 行で出力してそれぞれを要約します。管理者は **tcplife** を使用して、接続と転送されたトラフィック量を特定できます。

たとえば、ポート **22** (SSH)への接続を表示して、以下の情報を取得できます。

- ローカルプロセス ID (PID)
- ローカルプロセス名
- ローカルの IP アドレスおよびポート番号
- リモートの IP アドレスおよびポート番号
- 受信および送信トラフィックの量 (KB 単位)
- 接続がアクティブであった時間 (ミリ秒単位)

手順

1. 次のコマンドを実行して、ローカルポート **22** への接続の追跡を開始します。

```
/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
...
```

接続が閉じられるたびに、**tcplife** は接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcplife(8)** の man ページ

- **/usr/share/bcc/tools/doc/tcplife_example.txt** ファイル

42.7. TCP 再送信の追跡

tcpretrans ユーティリティーは、ローカルおよびリモート IP アドレスおよびポート番号、再送信時の TCP 状態などの TCP 再送信の詳細を表示します。

このユーティリティーは eBPF 機能を使用するため、オーバーヘッドが非常に低くなります。

手順

1. 以下のコマンドを使用して、TCP 再送信の詳細を表示します。

```
# /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
...
```

   カーネルが TCP 再送信関数を呼び出すたびに、**tcpretrans** は、接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpretrans(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpretrans_example.txt** ファイル

42.8. TCP 状態変更情報の表示

TCP セッション時に、TCP の状態が変わります。**tcpstates** ユーティリティーは、eBPF 関数を使用してこれらの状態の変更を追跡し、各状態の期間を含む詳細を出力します。たとえば、**tcpstates** を使用して、接続の初期化に時間がかかりすぎるかどうかを特定します。

手順

1. 以下のコマンドを使用して、TCP 状態変更の追跡を開始します。

```
# /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
...
```

接続の状態が変更されるたびに、**tcpstates** は、更新された接続の詳細を含む新しい行を表示します。

複数の接続が状態を同時に変更する場合は、最初の列 (**SKADDR**) のソケットアドレスを使用して、同じ接続に属するエントリーを判断します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpstates(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpstates_example.txt** ファイル

42.9. 特定のサブネットに送信された TCP トラフィックの要約および集計

tcpsubnet ユーティリティーは、ローカルホストがサブネットに送信する IPv4 TCP トラフィックを要約し、固定の間隔で出力を表示します。このユーティリティーは、eBPF 機能を使用してデータを収集および要約して、オーバーヘッドを削減します。

デフォルトでは、**tcpsubnet** は以下のサブネットのトラフィックを要約します。

- **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**

最後のサブネット (**0.0.0.0/0**) は catch-all オプションであることに注意してください。**tcpsubnet** ユーティリティーは、この catch-all エントリーの最初の 4 つとは異なるサブネットのトラフィックをすべてカウントします。

192.0.2.0/24 および **198.51.100.0/24** サブネットのトラフィックをカウントするには、以下の手順に従います。他のサブネットへのトラフィックは **0.0.0.0/0** catch-all subnet entry で追跡されます。

手順

1. **192.0.2.0/24**、**198.51.100.0/24**、および他のサブネットに送信するトラフィック量の監視を開始します。

```
# /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
...
```

このコマンドは、指定したサブネットのトラフィックを 1 秒ごとに 1 回ずつバイト単位で表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcpsubnet(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcpsubnet.txt** ファイル

42.10. IP アドレスとポートによるネットワークスループットの表示

tcptop ユーティリティーは、ホストがキロバイト単位で送受信する TCP トラフィックを表示します。レポートは自動的に更新され、アクティブな TCP 接続のみが含まれます。このユーティリティーは eBPF 機能を使用するため、オーバーヘッドは非常に低くなります。

手順

1. 送受信トラフィックを監視するには、次のコマンドを実行します。

```
# /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
...
```

コマンドの出力には、アクティブな TCP 接続のみが含まれます。ローカルシステムまたはリモートシステムが接続を閉じると、接続が出力に表示されなくなります。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcptop(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcptop.txt** ファイル

42.11. 確立された TCP 接続の追跡

tcptracer ユーティリティーは、TCP 接続を接続、許可、および閉じるカーネル機能を追跡します。このユーティリティーは eBPF 機能を使用するため、オーバーヘッドが非常に低くなります。

手順

1. 次のコマンドを実行して、トレースプロセスを開始します。

```
# /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
...
```

カーネルが接続を開始し、受け入れ、または閉じるたびに、**tcptracer** は、接続の詳細を表示します。

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **tcptracer(8)** の man ページ
- **/usr/share/bcc/tools/doc/tcptracer_example.txt** ファイル

42.12. IPV4 および IPV6 リッスン試行の追跡

solisten ユーティリティーは、すべての IPv4 および IPv6 のリッスン試行を追跡します。最終的に失敗したり、接続を許可しないリスニングプログラムなど、リッスン試行を追跡します。このユーティリティーは、プログラムが TCP 接続をリッスンする場合にカーネルが呼び出される関数を追跡します。

手順

1. 次のコマンドを実行して、リッスンする TCP 試行をすべて表示するトレースプロセスを開始します。

```
# /usr/share/bcc/tools/solisten
PID    COMM         PROTO       BACKLOG     PORT     ADDR
3643   nc           TCPv4       1           4242     0.0.0.0
3659   nc           TCPv6       1           4242     2001:db8:1::1
4221   redis-server   TCPv6       128         6379     ::
4221   redis-server   TCPv4       128         6379     0.0.0.0
....
```

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **solisten(9)** の man ページ
- **/usr/share/bcc/tools/doc/solisten_example.txt** ファイル

42.13. ソフト割り込みのサービス時間の要約

softirqs ユーティリティーは、ソフト割り込み (ソフト IRQ) に費やした時間を要約し、この時間を合計またはヒストグラムのディストリビューションとして表示します。このユーティリティーは、安定したトレースメカニズムであるカーネルトレースポイント **irq:softirq_enter** および **irq:softirq_exit** を使用します。

手順

1. 以下のコマンドを実行して、**soft irq** イベント時間を追跡します。

```
# /usr/share/bcc/tools/softirqs
```

```
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ          TOTAL_usecs
tasklet                  166
block                   9152
net_rx                 12829
rcu                    53140
sched                 182360
timer                 306256
```

2. **Ctrl**+**C** を押して、追跡プロセスを停止します。

関連情報

- **softirqs(8)** の man ページ
- **/usr/share/bcc/tools/doc/softirqs_example.txt** ファイル
- **mpstat(1)** の man ページ

42.14. ネットワークインターフェイス上のパケットサイズとパケット数のまとめ

netqtop ユーティリティーは、特定のネットワークインターフェイスの各ネットワークキュー上の受信 (RX) パケットと送信 (TX) パケットの属性に関する統計情報を表示します。統計情報には次のものが含まれます。

- 1秒あたりのバイト数 (BPS)
- 1秒あたりのパケット数 (PPS)
- 平均パケットサイズ
- 総パケット数

これらの統計情報を生成するために、**netqtop** は、送信パケット **net_dev_start_xmit** および受信パケット **netif_receive_skb** のイベントを実行するカーネル関数をトレースします。

手順

1. **2** 秒間のバイトサイズの範囲内に含まれるパケット数を表示します。

```
# /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. **Ctrl**+**C** を押して **netqtop** を停止します。

関連情報

- **netqtop(8)** man ページ
- **/usr/share/bcc/tools/doc/netqtop_example.txt**

42.15. 関連情報

- **/usr/share/doc/bcc/README.md**

第43章 すべての MAC アドレスからのトラフィックを受け入れるようにネットワークデバイスを設定

ネットワークデバイスは通常、コントローラーが受信するようにプログラムされているパケットを傍受して読み取ります。ネットワークデバイスを設定して、仮想スイッチまたはポートグループレベルのすべての MAC アドレスからのトラフィックを受け入れることができます。

このネットワークモードを使用すると、以下を行うことができます。

- ネットワーク接続の問題診断
- セキュリティー上の理由から、ネットワークアクティビティーの監視
- ネットワーク内のプライベートデータイントラントまたは侵入傍受

InfiniBand を除くあらゆる種類のネットワークデバイスに対してこのモードを有効にできます。

43.1. 全トラフィックを受け入れるようなデバイスの一時設定

ip ユーティリティーを使用して、MAC アドレスに関係なく、すべてのトラフィックを受け入れるようにネットワークデバイスを一時的に設定できます。

手順

1. オプション: ネットワークインターフェイスを表示して、すべてのトラフィックを受信するインターフェイスを識別します。

```
# 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. デバイスを変更して、このプロパティーを有効または無効にします。
   - **enp1s0** の **accept-all-mac-addresses** モードを有効にするには、以下のコマンドを実行します。

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

   - **enp1s0** の **accept-all-mac-address** モードを有効にするには、以下のコマンドを実行します。

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

検証

- **accept-all-mac-addresses** モードが有効になっていることを確認します。

```
# 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
```

機器の説明の **PROMISC** フラグは、モードが有効であることを示しています。

43.2. NMCLI を使用して、すべてのトラフィックを受け入れるようにネットワークデバイスを永続的に設定

nmcli ユーティリティーを使用して、MAC アドレスに関係なく、すべてのトラフィックを受け入れるようにネットワークデバイスを永続的に設定できます。

手順

1. オプション: ネットワークインターフェイスを表示して、すべてのトラフィックを受信するインターフェイスを識別します。

```
# 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
...
```

   接続がない場合は、新しい接続を作成できます。

2. ネットワークデバイスを変更して、このプロパティーを有効または無効にします。

   - **enp1s0** の **ethernet.accept-all-mac-addresses** モードを有効にするには、以下のコマンドを実行します。

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

   - **enp1s0** の **accept-all-mac-address** モードを有効にするには、以下のコマンドを実行します。

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

3. 変更を適用し、接続を再度アクティブにします。

```
# nmcli connection up enp1s0
```

検証

- **ethernet.accept-all-mac-addresses** モードが有効になっていることを確認します。

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

   この **802-3-ethernet.accept-all-mac-addresses: true** は、モードが有効であることを示しています。

43.3. NMSTATECTL を使用して全トラフィックを受け入れるようにネットワークデバイスを永続的に設定する手順

nmstatectl ユーティリティーを使用して、Nmstate API を介して、MAC アドレスに関係なくすべての

トラフィックを受け入れるようにデバイスを設定します。Nmstate API は、設定を行った後、結果が設定ファイルと一致することを確認します。何らかの障害が発生した場合には、**nmstatectl** は自動的に変更をロールバックし、システムが不正な状態のままにならないようにします。

前提条件

- **nmstate** パッケージがインストールされている。
- デバイスの設定に使用した **enp1s0.yml** ファイルが利用できます。

手順

1. **enp1s0** 接続の既存の **enp1s0.yml** ファイルを編集し、以下の内容を追加します。

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

これらの設定では、**enp1s0** デバイスがすべてのトラフィックを受け入れるように設定します。

2. ネットワーク設定を適用します。

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

検証

- **802-3-ethernet.accept-all-mac-addresses** モードが有効になっていることを確認します。

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

この **802-3-ethernet.accept-all-mac-addresses: true** は、モードが有効であることを示しています。

関連情報

- **nmstatectl(8)** の man ページ
- **/usr/share/doc/nmstate/examples/** directory

第44章 NMCLI を使用したネットワークインターフェイスのミラーリング

ネットワーク管理者は、ポートミラーリングを使用して、あるネットワークデバイスから別のネットワークデバイスに通信中の受信および送信トラフィックを複製できます。インターフェイスのトラフィックのミラーリングは、次の状況で役に立ちます。

- ネットワークの問題をデバッグしてネットワークフローを調整する
- ネットワークトラフィックを検査および分析する
- 侵入を検出する

前提条件

- ネットワークトラフィックをミラーリングするネットワークインターフェイス。

手順

1. ネットワークトラフィックをミラーリングするネットワーク接続プロファイルを追加します。

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

2. **10:** handle で egress (送信) トラフィックについて、 **prio qdisc** を **enp1s0** に割り当てます。

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

   子なしでアタッチされた **prio qdisc** を使用すると、フィルターをアタッチできます。

3. **ffff:** ハンドルを使用して、イングレストラフィックの **qdisc** を追加します。

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

4. 次のフィルターを追加して、入力および出力 **qdiscs** のパケットを照合し、それらを **enp7s0** にミラーリングします。

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

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

   matchall フィルターは、すべてのパケットを照合し、**mirred** アクションではパケットを宛先にリダイレクトします。

5. 接続をアクティベートします。

```
# nmcli connection up enp1s0
```

検証

1. **tcpdump** ユーティリティーをインストールします。

```
# dnf install tcpdump
```

2. ターゲットデバイス (**enp7s0**) でミラーリングされたトラフィックを表示します。

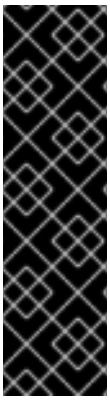
```
# tcpdump -i enp7s0
```

関連情報

- **tcpdump** を使用してネットワークパケットをキャプチャする方法

第45章 NMSTATE-AUTOCONF を使用した LLDP を使用したネットワーク状態の自動設定

ネットワークデバイスは、LLDP (Link Layer Discovery Protocol) を使用して、LAN でその ID、機能、およびネイバーを通知できます。**nmstate-autoconf** ユーティリティーは、この情報を使用してローカルネットワークインターフェイスを自動的に設定できます。



重要

nmstate-autoconf ユーティリティーは、テクノロジープレビューとしてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能では、最新の製品機能をいち早く提供します。これにより、お客様は開発段階で機能をテストし、フィードバックを提供できます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

45.1. NMSTATE-AUTOCONF を使用したネットワークインターフェイスの自動設定

nmstate-autoconf ユーティリティーは、LLDP を使用して、スイッチに接続されているインターフェイスの VLAN 設定を識別し、ローカルデバイスを設定します。

この手順では、以下のシナリオで、スイッチが LLDP を使用して VLAN 設定をブロードキャストすることを前提としています。

- RHEL サーバーの **enp1s0** および **enp2s0** インターフェイスは、VLAN ID **100** および VLAN 名 **prod-net** で設定されたスイッチポートに接続されています。
- RHEL サーバーの **enp3s0** インターフェイスは、VLAN ID **200** および VLAN 名 **mgmt-net** で設定されたスイッチポートに接続されています。

nmstate-autoconf ユーティリティーは、この情報を使用して、サーバーに以下のインターフェイスを作成します。

- **bond100** - **enp1s0** と **enp2s0** がポートとして使用されるボンディングインターフェイス
- **prod-net** - **bond100** 上の VLAN インターフェイスと VLAN ID **100**
- **mgmt-net** - **enp3s0** 上の VLAN インターフェイスと VLAN ID **200**

LLDP が同じ VLAN ID をブロードキャストする別のスイッチポートに複数のネットワークインターフェイスを接続する場合、**nmstate-autoconf** はこのインターフェイスでボンディングを作成し、さらにその上に共通 VLAN ID を設定します。

前提条件

- **nmstate** パッケージがインストールされている。
- ネットワークスイッチで LLDP が有効になっている。
- イーサネットインターフェイスが稼働している。

手順

1. イーサネットインターフェイスで LLDP を有効にします。

   a. 以下の内容で、~/**enable-lldp.yml** などのファイルを作成します。

```
interfaces:
  - name: enp1s0
    type: ethernet
    lldp:
      enabled: true
  - name: enp2s0
    type: ethernet
    lldp:
      enabled: true
  - name: enp3s0
    type: ethernet
    lldp:
      enabled: true
```

   b. 設定をシステムに適用します。

```
# nmstatectl apply ~/enable-lldp.yml
```

2. LLDP を使用してネットワークインターフェイスを設定します。

   a. 必要に応じて、ドライランを起動して、**nmstate-autoconf** が生成する YAML 設定を表示し、確認します。

```
# 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. **nmstate-autoconf** を使用して、LLDP から受信した情報に基づいて設定を生成し、その設定をシステムに適用します。

```
# nmstate-autoconf enp1s0,enp2s0,enp3s0
```

次のステップ

- ネットワークに、インターフェイスに IP 設定を提供する DHCP サーバーがない場合は、手動で設定します。詳細は、以下を参照してください。
  - イーサネット接続の設定
  - ネットワークボンディングの設定

検証

1. 各インターフェイスの設定を表示します。

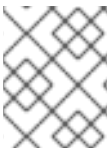
```
# nmstatectl show <interface_name>
```

関連情報

- **nmstate-autoconf(8)** の man ページ

第46章 802.3 リンク設定

オートネゴシエーションは、IEEE 802.3u ファストイーサネットプロトコルの機能です。これは、リンク経由で情報交換を行うために、速度、デュプレックスモード、およびフロー制御の最適なパフォーマンスを提供するデバイスポートを対象としています。オートネゴシエーションプロトコルを使用すると、イーサネット経由でデータ転送のパフォーマンスが最適化されます。



注記

オートネゴシエーションのパフォーマンスを最大限に活用するには、リンクの両側で同じ設定を使用します。

46.1. NMCLI ユーティリティーを使用した 802.3 リンクの設定

イーサネット接続の 802.3 リンクを設定するには、次の設定パラメーターを変更します。

- **802-3-ethernet.auto-negotiate**
- **802-3-ethernet.speed**
- **802-3-ethernet.duplex**

手順

1. 接続の現在の設定を表示します。

```
# nmcli connection show Example-connection
...
802-3-ethernet.speed:  0
802-3-ethernet.duplex: --
802-3-ethernet.auto-negotiate: no
...
```

問題が発生した場合にパラメーターをリセットする必要がある場合は、これらの値を使用できます。

2. 速度とデュプレックスリンクの設定を行います。

```
# nmcli connection modify Example-connection 802-3-ethernet.auto-negotiate yes 802-3-ethernet.speed 10000 802-3-ethernet.duplex full
```

このコマンドは、オートネゴシエーションを有効にし、接続の速度を **10000** Mbit フルデュプレックスに設定します。

3. 接続を再度アクティベートします。

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

検証

- **ethtool** ユーティリティーを使用して、イーサネットインターフェイス **enp1s0** の値を確認します。

```
# ethtool enp1s0
```

```
Settings for enp1s0:
...
Speed: 10000 Mb/s
Duplex: Full
Auto-negotiation: on
...
Link detected: yes
```

関連情報

- **nm-settings(5)** man ページ

第47章 DPDK の使用

データプレーン開発キット (DPDK) は、ユーザー空間でのパケット処理を高速化するためのライブラリーとネットワークドライバーを提供します。

管理者は、たとえば仮想マシンで、SR-IOV (Single Root I/O Virtualization) を使用して、レイテンシーを減らして I/O スループットを増やします。

注記

Red Hat は、実験的な DPDK API に対応していません。

47.1. DPDK パッケージのインストール

DPDK を使用するには、**dpdk** パッケージをインストールします。

手順

- **dnf** ユーティリティーを使用して、**dpdk** パッケージをインストールします。

```
# dnf install dpdk
```

47.2. 関連情報

- Network Adapter Fast Datapath Feature Support Matrix

第48章 TIPC の使用

Cluster Domain Sockets とも呼ばれる TIPC (Trans-process Communication) は、クラスター全体の操作の IPC (Inter-process Communication) サービスです。

高可用性環境および動的クラスター環境で実行されているアプリケーションには、特別なニーズがあります。クラスター内のノード数は異なる可能性があります。また、ルーターに障害が発生する可能性があります。負荷分散についての考慮事項により、クラスター内の異なるノードに機能が移行する可能性があります。TIPC は、アプリケーション開発者がこのような状況に対応する作業を最小限に抑え、適切かつ最適方法で処理される機会を最大化します。さらに、TIPC は TCP などの一般的なプロトコルよりも効率的で耐障害性のある通信を提供します。

48.1. TIPC のアーキテクチャー

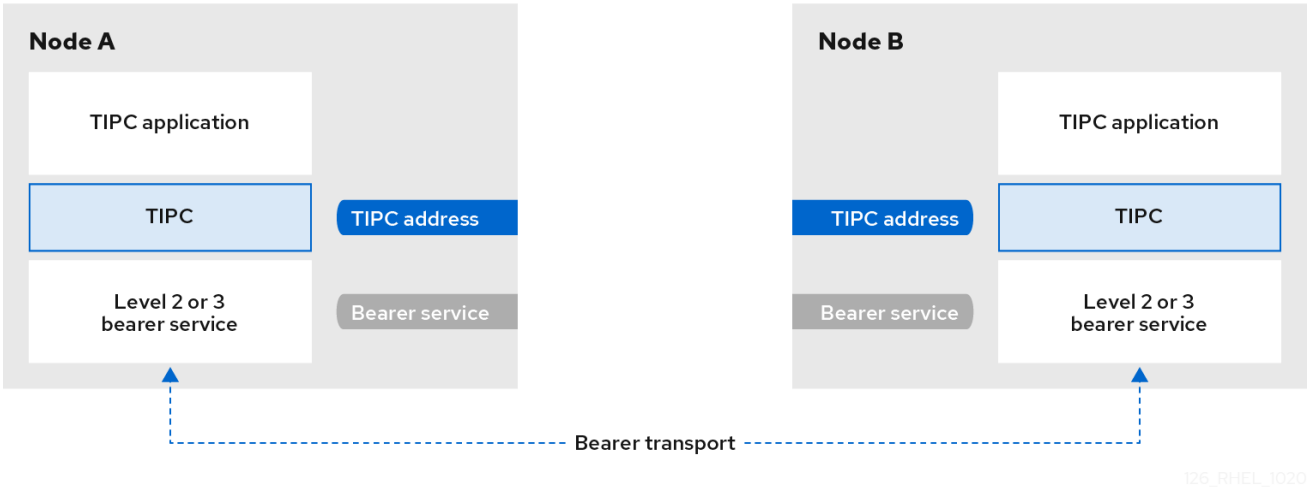
TIPC は、TIPC とパケットトランスポートサービス (**bearer**) を使用してアプリケーション間のレイヤーで、トランスポート、ネットワーク、およびシグナル側のリンク層を結び付けます。しかし、TIPC は異なるトランスポートプロトコルをベアラーとして使用することができるため、たとえば TCP 接続は TIPC シグナルリンクのベアラーとして機能できます。

TIPC は以下のベアラーをサポートします。

- イーサネット
- Infiniband
- UDP プロトコル

TIPC は、すべての TIPC 通信のエンドポイントである TIPC ポート間で、信頼できるメッセージの転送を提供します。

以下は TIPC アーキテクチャーの図です。



48.2. システムの起動時の TIPC モジュールの読み込み

TIPC プロトコルを使用するには、**tipc** カーネルモジュールをロードする必要があります。システムの起動時にこのカーネルモジュールを自動的にロードするように Red Hat Enterprise Linux を設定できます。

手順

1. 以下の内容で **/etc/modules-load.d/tipc.conf** ファイルを作成します。

```
tipc
```

2. **systemd-modules-load** サービスを再起動して、システムを再起動せずにモジュールを読み込みます。

```
# systemctl start systemd-modules-load
```

検証

1. 以下のコマンドを使用して、RHEL が **tipc** モジュールをロードしていることを確認します。

```
# lsmod | grep tipc
tipc    311296  0
```

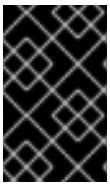
このコマンドに、**tipc** モジュールのエントリーが表示されない場合は、RHEL がそのモジュールの読み込みに失敗しました。

関連情報

- **modules-load.d(5)** の man ページ

48.3. TIPC ネットワークの作成

TIPC ネットワークを作成するには、TIPC ネットワークに参加する各ホストでこの手順を実行します。



重要

コマンドは、TIPC ネットワークを一時的に設定します。ノードに TIPC を永続的に設定するには、スクリプトでこの手順のコマンドを使用し、RHEL がシステムの起動時にそのスクリプトを実行するように設定します。

前提条件

- **tipc** モジュールがロードされている。詳細については、システム起動時の tipc モジュールのロードを参照してください。

手順

1. オプション: UUID またはノードのホスト名などの一意のノード ID を設定します。

```
# tipc node set identity host_name
```

アイデンティティーには、最大 16 文字と数字で設定される一意の文字列を使用できます。

この手順の後に ID を設定または変更することはできません。

2. ベアラーを追加します。たとえば、イーサネットを media として、**enp0s1** デバイスを物理ベアラーデバイスとして使用するには、次のコマンドを実行します。

```
# tipc bearer enable media eth device enp1s0
```

3. オプション: 冗長性とパフォーマンスを向上させるには、前の手順でコマンドを使用してさらにベアラーをアタッチします。最高 3 つのベアラーを設定できますが、同じメディアでは 2 つ以上のビギナーを設定することができます。

4. TIPC ネットワークに参加する必要のある各ノードで直前の手順を繰り返します。

検証

1. クラスターメンバーのリンクステータスを表示します。

```
# tipc link list
broadcast-link: up
5254006b74be:enp1s0-525400df55d1:enp1s0: up
```

この出力は、ノード **5254006b74be** のベアラー **enp1s0** とノード **525400df55d1** のベアラー **enp1s0** 間の接続が **up** になっていることを示します。

2. TIPC 公開テーブルを表示します。

```
# 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
```

- サービスタイプ **0** の 2 つのエントリーは、2 つのノードがこのクラスターのメンバーであることを示しています。
- サービスタイプ **1** のエントリーは、組み込みのトポロジーサービス追跡サービスを表します。
- サービスタイプ **2** のエントリーには、発行したノードから表示されるリンクが表示されます。範囲の上限 **3741353223** は、ピアエンドポイントのアドレス (ノード ID に基づく一意の 32 ビットハッシュ値) を 10 進数の形式で表します。

関連情報

- **tipc-bearer(8)** の man ページ
- **tipc-namespace(8)** の man ページ

48.4. 関連情報

- Red Hat は、他のベアラーレベルのプロトコルを使用して、トランスポートメディアに基づいてノード間の通信を暗号化することを推奨します。以下に例を示します。
  - MACSec:Using MACsec to encrypt layer 2 traffic を参照してください。
  - IPsec:IPsec を使用した VPN の設定 を参照してください。
- TIPC の使用例の例として、**git clone git://git.code.sf.net/p/tipc/tipcutils** コマンドを使用してアップストリームの GIT リポジトリーのクローンを作成します。このリポジトリーには、デモ

のソースコードと TIPC 機能を使用するプログラムが同梱されています。このリポジトリーは Red Hat では提供していないことに注意してください。

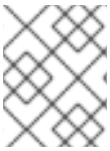
- **kernel-doc** パッケージにより提供される **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/output/networking/tipc.html**

第49章 NM-CLOUD-SETUP を使用してパブリッククラウドのネットワークインターフェイスを自動的に設定する

通常、仮想マシン (VM) には、DHCP によって設定可能なインターフェイスが 1 つだけあります。しかし、DHCP は、インターフェイス、IP サブネット、IP アドレスなど、複数のネットワークエンティティーを使用して仮想マシンを設定することはできません。また、仮想マシンインスタンスの実行中は設定を適用できません。この実行時設定の問題を解決するために、**nm-cloud-setup** ユーティリティーはクラウドサービスプロバイダーのメタデータサーバーから設定情報を自動的に取得し、ホストのネットワーク設定を更新します。このユーティリティーは、複数のネットワークインターフェイス、複数の IP アドレス、または 1 つのインターフェイスの IP サブネットを自動的に取得し、実行中の仮想マシンインスタンスのネットワークを再設定するのに役立ちます。

49.1. NM-CLOUD-SETUP の設定と事前デプロイ

パブリッククラウドでネットワークインターフェイスを有効にして設定するには、**nm-cloud-setup** をタイマーおよびサービスとして実行します。



注記

Red Hat Enterprise Linux On Demand および AWS ゴールデンイメージでは、**nm-cloud-setup** がすでに有効になっており、アクションは不要です。

前提条件

- ネットワーク接続が存在します。
- 接続は DHCP を使用します。
  デフォルトでは、NetworkManager は DHCP を使用する接続プロファイルを作成します。**/etc/NetworkManager/NetworkManager.conf** で **no-auto-default** パラメーターを設定したためにプロファイルが作成されなかった場合は、この初期接続を手動で作成します。

手順

1. **nm-cloud-setup** パッケージをインストールします。

```
# dnf install NetworkManager-cloud-setup
```

2. **nm-cloud-setup** サービスのスナップインファイルを作成して実行します。
   a. 次のコマンドを使用して、スナップインファイルの編集を開始します。

```
# systemctl edit nm-cloud-setup.service
```

   設定を有効にするには、サービスを明示的に開始するか、システムを再起動することが重要です。

   b. **systemd** スナップインファイルを使用して、**nm-cloud-setup** でクラウドプロバイダーを設定します。たとえば、Amazon EC2 を使用するには、次のように入力します。

```
[Service]
Environment=NM_CLOUD_SETUP_EC2=yes
```

   次の環境変数を設定して、クラウドが使用できるようにすることができます。

- **NM_CLOUD_SETUP_AZURE** for Microsoft Azure
- **NM_CLOUD_SETUP_EC2** for Amazon EC2 (AWS)
- **NM_CLOUD_SETUP_GCP** for Google Cloud Platform(GCP)
- **NM_CLOUD_SETUP_ALIYUN** for Alibaba Cloud (Aliyun)

c. ファイルを保存して、エディターを終了します。

3. **systemd** 設定をリロードします。

```
# systemctl daemon-reload
```

4. **nm-cloud-setup** サービスを有効にして開始します。

```
# systemctl enable --now nm-cloud-setup.service
```

5. **nm-cloud-setup** タイマーを有効にして開始します。

```
# systemctl enable --now nm-cloud-setup.timer
```

関連情報

- **nm-cloud-setup(8)** の man ページ
- イーサネット接続の設定

49.2. RHEL EC2 インスタンスにおける IMDSV2 と NM-CLOUD-SETUP のロールについて

Amazon EC2 のインスタンスメタデータサービス (IMDS) を使用すると、実行中の Red Hat Enterprise Linux (RHEL) EC2 インスタンスのインスタンスメタデータにアクセスする権限を管理できます。RHEL EC2 インスタンスは、セッション指向の方式である IMDS バージョン 2 (IMDSv2) を使用します。**nm-cloud-setup** ユーティリティーを使用すると、管理者はネットワークを再設定し、実行中の RHEL EC2 インスタンスの設定を自動的に更新できます。**nm-cloud-setup** ユーティリティーは、ユーザーの介入なしで IMDSv2 トークンを使用して IMDSv2 API 呼び出しを処理します。

- IMDS は、リンクローカルアドレス **169.254.169.254** で実行され、RHEL EC2 インスタンス上のネイティブアプリケーションへのアクセスを提供します。
- アプリケーションおよびユーザー用の各 RHEL EC2 インスタンスに IMDSv2 を指定して設定すると、IMDSv1 にはアクセスできなくなります。
- IMDSv2 を使用することにより、RHEL EC2 インスタンスは、IAM ロールを介してアクセス可能な状態を維持しながら、IAM ロールを使用せずにメタデータを維持します。
- RHEL EC2 インスタンスが起動すると、**nm-cloud-setup** ユーティリティーが自動的に実行され、RHEL EC2 インスタンス API を使用するための EC2 インスタンス API アクセストークンが取得されます。

注記

IMDSv2 トークンを HTTP ヘッダーとして使用して EC2 環境の詳細を確認してください。

関連情報

- **nm-cloud-setup(8)** の man ページ