OpenShift Container Platform 4.18

Kubernetes NMState

OpenShift Container Platform で Kubernetes NMState を使用してノードのネットワーク状態と設定を監視および更新する

Last Updated: 2025-12-08

OpenShift Container Platform 4.18 Kubernetes NMState

OpenShift Container Platform で Kubernetes NMState を使用してノードのネットワーク状態と設定を監視および更新する

Legal Notice

Abstract

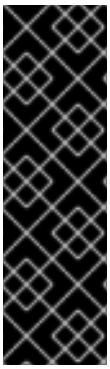
このドキュメントでは、Kubernetes NMState を使用して、OpenShift Container Platform のノードネットワーク設定を監視、更新、およびトラブルシューティングする方法を説明します。

Table of Contents

第1章 ノードのネットワーク状態と設定の監視と更新

Kubernetes NMState Operator をインストールした後、Operator を使用して、クラスターのノードのネットワーク状態とネットワーク設定を監視および更新できます。

NMState Operator のインストール方法の詳細は、Kubernetes NMState Operator を参照してください。



重要

既存の **br-ex** ブリッジ、OVN-Kubernetes が管理する Open vSwitch ブリッジ、または任意のインターフェイス、ボンディング、VLAN などを、**br-ex** ブリッジに変更することはできません。ただし、カスタマイズした br-ex ブリッジを設定することはできます。

詳細は、**インストーラーでプロビジョニングされるクラスターのベアメタルへのデプロイ**、または **ユーザーがプロビジョニングしたクラスターをベアメタルにインストール** ドキュメントの「カスタマイズされた br-ex ブリッジを含むマニフェストオブジェクトの作成」を参照してください。

1.1. CLI を使用したノードのネットワーク状態の表示

ノードのネットワーク状態は、クラスター内のすべてのノードのネットワーク設定です。**NodeNetworkState** オブジェクトはクラスター内のすべてのノードにあります。このオブジェクトは定期的に更新され、ノードのネットワークの状態を取得します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. クラスターのすべての **NodeNetworkState** オブジェクトをリスト表示します。

```
$ oc get nns
```

2. **NodeNetworkState** オブジェクトを検査して、そのノードにネットワークを表示します。この例の出力は、明確にするために編集されています。

```
$ oc get nns node01 -o yaml
```

出力例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  name: node01 (1)
status:
  currentState: (2)
    dns-resolver:
# ...
    interfaces:
# ...
    route-rules:
```

```
# ...
    routes:
# ...
  lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
```

1. **NodeNetworkState** オブジェクトの名前はノードから取られています。
2. **currentState** には、DNS、インターフェイス、およびルートを含む、ノードの完全なネットワーク設定が含まれます。
3. 最後に成功した更新のタイムスタンプ。これは、ノードが到達可能であり、レポートの鮮度の評価に使用できる限り定期的に更新されます。

1.2. WEB コンソールからノードのネットワーク状態 (NNS) を表示する

管理者は、OpenShift Container Platform Web コンソールを使用して、**NodeNetworkState** リソースとネットワークインターフェイスを観察し、ネットワークの詳細にアクセスできます。

手順

1. **Networking → NodeNetworkState** に移動します。
   NodeNetworkState ページでは、**NodeNetworkState** リソースと、ノード上に作成された対応するインターフェイスのリストを表示できます。**Interface state**、**Interface type**、および **IP** に基づく **フィルター**、または **Name** または **Label** の条件に基づく検索バーを使用して、表示される **NodeNetworkState** リソースを絞り込むことができます。
2. **NodeNetworkState** リソースに関する詳細情報にアクセスするには、**Name** 列にリストされている **NodeNetworkState** リソース名をクリックします。
3. **NodeNetworkState** リソースの **Network Details** セクションを展開して表示するには、大なり記号 (>) をクリックします。あるいは、**Network interface** 列の下の各インターフェイスタイプをクリックして、ネットワークの詳細を表示することもできます。

1.2.1. NNS トポロジーのグラフィック表現を表示する

クラスター内のノードネットワークの設定をわかりやすくするために、図で表示できます。NNS トポロジーダイアグラムには、すべてのノードコンポーネント (ネットワークインターフェイスコントローラー、ブリッジ、ボンディング、VLAN)、それらのプロパティーと設定、およびノード間の接続が表示されます。

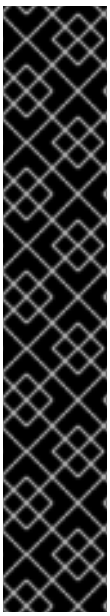
クラスターのトポロジービューを開くには、次の手順を実行します。

1. Web コンソールの **Administrator** ビューで、**Networking → NodeNetworkState** に移動します。
2. ページの右上隅にある **Topology** アイコンをクリックします。
   NNS トポロジー図が開きます。コンポーネントの各グループは単一のノードを表します。
   - ノードの設定とプロパティーを表示するには、ノードの境界内をクリックします。
   - 特定のコンポーネント (インターフェイスやブリッジなど) の機能または YAML ファイルを表示するには、コンポーネントのアイコンをクリックします。

- アクティブなコンポーネントのアイコンは緑の枠線で示され、非接続コンポーネントのアイコンは赤の枠線で示されます。

1.3. NODENETWORKCONFIGURATIONPOLICY マニフェストファイル

NodeNetworkConfigurationPolicy (NNCP) マニフェストファイルは、Kubernetes NMState Operator が OpenShift Container Platform クラスター内に存在するノードのネットワークを設定するために使用するポリシーを定義します。



重要

複数の NNCP CR をノードに適用する場合は、ポリシー名の英数字順のソートに基づき、論理的な順序で NNCP を作成する必要があります。Kubernetes NMState Operator は、Operator が CR をノードに即座に適用できるように、新しく作成された NNCP CR を継続的にチェックします。例として、次の論理的な順序の問題を考えてみましょう。

1. **eth1.1000** などの VLAN ポートをリッスンするブリッジインターフェイスを定義するために、NNCP 1 を作成します。
2. VLAN インターフェイスを定義するために NNCP 2 を作成し、このインターフェイスのポート (**eth1.1000** など) を指定します。
3. NNCP 2 をノードに適用する前に、NNCP 1 を適用します。

ポート **eth1.1000** が存在しないため、ノードでノード接続の問題が発生します。その結果、クラスターで障害が発生します。

ノードのネットワークポリシーをノードに適用すると、Kubernetes NMState Operator によって、ノードのネットワークポリシーの詳細に従ってノードのネットワークが設定されます。

警告

次のインターフェイス名は予約されており、NMstate 設定では使用できません。

- **br-ext**
- **br-int**
- **br-local**
- **br-nexthop**
- **br0**
- **ext-vxlan**
- **ext**
- **genev_sys_***
- **int**
- **k8s-***
- **ovn-k8s-***
- **patch-br-***
- **tun0**
- **vxlan_sys_***

OpenShift CLI (**oc**) または OpenShift Container Platform Web コンソールを使用して NNCP を作成できます。インストール後のタスクとして、NNCP を作成したり、既存の NNCP を編集したりできます。

注記

NNCP を作成する前に、「各種インターフェイスのポリシー設定例」のドキュメントを一読してください。

NNCP を削除する必要がある場合は、**oc delete nncp** コマンドを使用して削除アクションを完了できます。ただし、このコマンドではブリッジインターフェイスなどのオブジェクトは削除されません。

ノードにインターフェイスを追加したノードネットワークポリシーを削除しても、そのノード上のポリシー設定は変更されません。同様に、インターフェイスを削除してもポリシーは削除されません。これは、Pod またはノードが再起動されるたびに、Kubernetes NMState Operator が、削除されたインターフェイスを再度追加するためです。

NNCP、ノードネットワークポリシー、およびインターフェイスを実際に削除するには、通常、次の操作が必要です。

1. NNCP を編集し、ファイルからインターフェイスの詳細を削除します。ファイルから **name**、**state**、**type** パラメーターは削除しないでください。
2. NNCP の **interfaces.state** セクションに **state: absent** を追加します。
3. **oc apply -f <nncp_file_name>** を実行します。Kubernetes NMState Operator がクラスター内の各ノードにノードネットワークポリシーを適用すると、各ノードに存在するすべてのインターフェイスが **absent** とマークされます。
4. **oc delete nncp** を実行して NNCP を削除します。

1.4. 関連情報

- 異なるインターフェイスのポリシー設定の例
- ノードからインターフェイスの削除

1.5. WEB コンソールからのポリシーの管理

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用して、ノードからのインターフェイスの追加または削除など、ノードネットワーク設定を更新できます。Web コンソールからポリシーを管理するには、**Networking** メニューの **NodeNetworkConfigurationPolicy** ページで作成されたポリシーのリストにアクセスします。このページでは、ポリシーを作成、更新、監視、削除できます。

1.5.1. ポリシーステータスの監視

NodeNetworkConfigurationPolicy ページからポリシーのステータスを監視できます。このページには、クラスター内に作成されたすべてのポリシーが次の列を含む表形式で表示されます。

名前
作成されたポリシーの名前。

一致したノード
ポリシーが適用されるノードの数。これは、ノードセレクターに基づくノードのサブセット、またはクラスター上のすべてのノードのいずれかになります。

ノードのネットワーク状態
一致したノードの実行状態。制定状態をクリックすると、ステータスの詳細情報が表示されます。

目的のポリシーを見つけるには、**Filter** オプションを使用するか、検索オプションを使用して、制定状態に基づいてリストをフィルターできます。

1.5.2. ポリシーの作成

Web コンソールでフォームまたは YAML を使用してポリシーを作成できます。

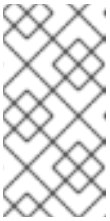
手順

1. **Networking → NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで **Create** をクリックし、**From Form** オプションを選択します。
既存のポリシーがない場合は、**Create NodeNetworkConfigurationPolicy** をクリックして、フォームを使用してポリシーを作成することもできます。

注記

YAML を使用してポリシーを作成するには、**Create** をクリックし、**With YAML** オプションを選択します。次の手順は、フォームを使用してポリシーを作成する場合にのみ適用されます。

3. オプション: **Apply this NodeNetworkConfigurationPolicy only to specific subsets of nodes using the node selector** チェックボックスをオンにして、ポリシーを適用する必要があるノードを指定します。
4. **Policy name** フィールドにポリシー名を入力します。
5. オプション: **Description** フィールドにポリシーの説明を入力します。
6. オプション: **Policy Interface(s)** セクションでは、編集可能なフィールドにプリセット値が設定されたブリッジインターフェイスがデフォルトで追加されます。次の手順を実行して値を編集します。
    a. **Interface name** フィールドにインターフェイスの名前を入力します。
    b. **Network state** ドロップダウンからネットワーク状態を選択します。デフォルトで選択されている値は **Up** です。
    c. **Type** ドロップダウンからインターフェイスのタイプを選択します。使用可能な値は、**Bridge**、**Bonding**、および **Ethernet** です。デフォルトで選択されている値は **Bridge** です。



注記

フォームを使用した VLAN インターフェイスの追加はサポートされていません。VLAN インターフェイスを追加するには、YAML を使用してポリシーを作成する必要があります。ポリシーを追加すると、フォームを使用してポリシーを編集できません。

    d. オプション: IP 設定セクションで、**IPv4** チェックボックスをオンにしてインターフェイスに IPv4 アドレスを割り当て、IP アドレス割り当ての詳細を設定します。
        i. **IP address** をクリックしてインターフェイスを静的 IP アドレスで設定するか、**DHCP** をクリックして IP アドレスを自動割り当てします。
        ii. **IP address** オプションを選択した場合は、**IPV4 address** フィールドに IPv4 アドレスを、**Prefix length** フィールドに接頭辞長を入力します。
        DHCP オプションを選択した場合は、無効にするオプションのチェックを外します。使用可能なオプションは、**Auto-DNS**、**Auto-routes**、および **Auto-gateway** です。すべてのオプションがデフォルトで選択されています。
    e. オプション: **Port** フィールドにポート番号を入力します。
    f. オプション: STP を有効にするには、**Enable STP** チェックボックスをオンにします。
    g. オプション: ポリシーにインターフェイスを追加するには、**Add another interface to the policy** をクリックします。

    h. オプション: ポリシーからインターフェイスを削除するには、インターフェイスの横にある

h. オプション: ポリシーからインターフェイスを削除するには、インターフェイスの横にある

アイコン をクリックします。

注記

または、ページ上部の **Edit YAML** をクリックして、YAML を使用してフォームの編集を続けることもできます。

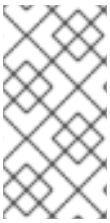
7. **Create** をクリックしてポリシーの作成を完了します。

1.6. ポリシーの更新

1.6.1. フォームを使用してポリシーを更新する

手順

1. **Networking → NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、編集するポリシーの横にあるアイコン  を選択し、**Edit** をクリックします。
3. 更新するフィールドを編集します。
4. **Save** をクリックします。



注記

フォームを使用した VLAN インターフェイスの追加はサポートされていません。VLAN インターフェイスを追加するには、YAML を使用してポリシーを作成する必要があります。ポリシーを追加すると、フォームを使用してポリシーを編集することはできません。

1.6.2. YAML を使用したポリシーの更新

手順

1. **Networking → NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、編集するポリシーの **Name** 列の下にあるポリシー名をクリックします。
3. **YAML** タブをクリックし、YAML を編集します。

4. **Save** をクリックします。

1.6.3. ポリシーの削除

手順

1. **Networking → NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、削除するポリシーの横にあるアイコン を選択し、**Delete** をクリックします。
3. ポップアップウィンドウで、削除を確認するポリシー名を入力し、**Delete** をクリックします。

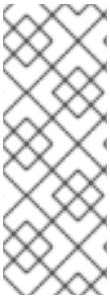
1.7. CLI を使用したポリシーの管理

1.7.1. ノード上でのインターフェイスの作成

NodeNetworkConfigurationPolicy (NNCP) マニフェストをクラスターに適用して、クラスター内のノードにインターフェイスを作成します。マニフェストには、インターフェイスの要求された設定の詳細が含まれます。

デフォルトでは、マニフェストはクラスター内のすべてのノードに適用されます。インターフェイスを特定ノードに追加するには、ノードセレクターの **spec: nodeSelector** パラメーターおよび適切な **<key>:<value>** を追加します。

複数の nmstate 対応ノードを同時に設定できます。この設定は、並列のノードの 50% に適用されます。このストラテジーでは、ネットワーク接続に障害が発生した場合にクラスター全体が使用できなくなるのを回避します。クラスターの特定の部分にポリシー設定を並行して適用するには、**NodeNetworkConfigurationPolicy** マニフェスト設定ファイルで **maxUnavailable** パラメーターを使用します。



注記

ノードが 2 つある場合に、**maxUnavailable** パラメーターを **50%** に設定した NNCP マニフェストをこれらのノードに適用すると、1 つのノードに NNCP 設定が反映されます。その後、**maxUnavailable** パラメーターを **50%** に設定した追加の NNCP マニフェストファイルを導入すると、この NCCP は最初の NNCP とは別に適用されます。つまり、2 つの NNCP マニフェストによってノードに不適切な設定が適用されると、クラスターの半分を確実に機能させることができなくなります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **NodeNetworkConfigurationPolicy** マニフェストを作成します。以下の例は、すべてのワーカーノードで Linux ブリッジを設定し、DNS リゾルバーを設定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
```

```
metadata:
  name: br1-eth1-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3 (4)
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port (5)
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
          auto-dns: false
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1
    dns-resolver: (6)
      config:
        search:
        - example.com
        - example.org
        server:
        - 8.8.8.8
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

(3) この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。

(4) オプション: ポリシー設定を同時に適用できる nmstate 対応ノードの最大数を指定します。このパラメーターは、**"10%"** などのパーセンテージ値 (文字列)、または **3** などの絶対値 (数値) のいずれかに設定できます。

(5) オプション: インターフェイスの人間が判読できる説明。

(6) オプション:DNS サーバーの検索およびサーバー設定を指定します。

2. ノードのネットワークポリシーを作成します。

```
$ oc apply -f br1-eth1-policy.yaml (1)
```

(1) ノードネットワーク設定ポリシーマニフェストのファイル名。

1.8. 関連情報

- 同じポリシーで複数のインターフェイスを作成する例
- ポリシーの各種 IP の管理方法の例

1.8.1. ノード上でのノードネットワークポリシー更新の確認

ノードネットワークポリシーを適用する際に、**NodeNetworkConfigurationEnactment** オブジェクトがクラスター内のすべてのノードに作成されます。ノードネットワーク設定の enactment (実行) は、そのノードでのポリシーの実行ステータスを表す読み取り専用オブジェクトです。ポリシーがノードに適用されない場合、そのノードの enactment (実行) にはトラブルシューティングのためのトレースバックが含まれます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. ポリシーがクラスターに適用されていることを確認するには、ポリシーとそのステータスをリスト表示します。

```
$ oc get nncp
```

2. オプション: ポリシーの設定に想定されている以上の時間がかかる場合は、特定のポリシーの要求される状態とステータスの状態を検査できます。

```
$ oc get nncp <policy> -o yaml
```

3. オプション: ポリシーのすべてのノード上での設定に想定されている以上の時間がかかる場合は、クラスターの enactment (実行) のステータスをリスト表示できます。

```
$ oc get nnce
```

4. オプション: 特定の enactment (実行) の設定 (失敗した設定のエラーレポートを含む) を表示するには、以下を実行します。

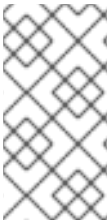
```
$ oc get nnce <node>.<policy> -o yaml
```

1.8.2. ノードからインターフェイスの削除

NodeNetworkConfigurationPolicy オブジェクトを編集し、インターフェイスの **state** を **absent** に設定して、クラスターの 1 つ以上のノードからインターフェイスを削除できます。

ノードからインターフェイスを削除しても、ノードのネットワーク設定は以前の状態に自動的に復元されません。以前の状態に復元する場合、そのノードのネットワーク設定をポリシーで定義する必要があります。

ブリッジまたはボンディングインターフェイスを削除すると、そのブリッジまたはボンディングインターフェイスに以前に接続されたか、それらの下位にあるノード NIC は **down** の状態になり、到達できなくなります。接続が失われないようにするには、同じポリシーでノード NIC を設定し、ステータスを **up** にし、DHCP または静的 IP アドレスのいずれかになるようにします。



注記

インターフェイスを追加したポリシーを削除しても、ノード上のポリシーの設定は変更されません。**NodeNetworkConfigurationPolicy** はクラスターのオブジェクトですが、このオブジェクトは要求される設定のみを表します。同様に、インターフェイスを削除してもポリシーは削除されません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. インターフェイスの作成に使用する **NodeNetworkConfigurationPolicy** マニフェストを更新します。以下の例は Linux ブリッジを削除し、接続が失われないように DHCP で **eth1** NIC を設定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
    - name: br1
      type: linux-bridge
      state: absent 4
    - name: eth1 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。
4. 状態を **absent** に変更すると、インターフェイスが削除されます。
5. ブリッジインターフェイスから接続が解除されるインターフェイスの名前。
6. インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。
7. インターフェイスの要求された状態。
8.

オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。

9 この例では **ipv4** を有効にします。

2. ノード上でポリシーを更新し、インターフェイスを削除します。

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

1 ポリシーマニフェストのファイル名。

1.9. 異なるインターフェイスのポリシー設定の例

ポリシーをノードに適用する場合は、さまざまな **NodeNetworkConfigurationPolicy** (NNCP) マニフェスト設定例を読む前に、クラスターが最適なパフォーマンス状態で実行されるように次の要素を考慮してください。

- 複数の NNCP CR をノードに適用する場合は、ポリシー名の英数字順のソートに基づき、論理的な順序で NNCP を作成する必要があります。Kubernetes NMState Operator は、Operator が CR をノードに即座に適用できるように、新しく作成された NNCP CR を継続的にチェックします。
- 多数のノードにポリシーを適用する必要があるが、すべてのノードに対して 1 つの NNCP のみを作成する場合、Kubernetes NMState Operator は各ノードにポリシーを順番に適用します。クラスターの設定ファイルの **maxUnavailable** パラメーターを使用して、ターゲットノードに対するポリシー適用の速度と範囲を設定できます。パラメーターのパーセンテージ値を低く設定すると、ポリシー適用を受信しているノードのごく一部に障害が影響する場合に、クラスター全体の障害が発生するリスクを軽減できます。
- 2 つの NNCP マニフェストで **maxUnavailable** パラメーターを **50%** に設定すると、ポリシー設定がクラスター内のノードの 100% に適用されます。
- ノードが再起動すると、Kubernetes NMState Operator はノードにポリシーを適用する順序を制御できなくなります。Kubernetes NMState Operator は、相互依存するポリシーを順番に適用する場合があります。その結果、ネットワークオブジェクトがデグレード状態になることがあります。
- 関連するすべてのネットワーク設定を 1 つのポリシーで指定することを検討してください。

1.9.1. 例: イーサネットインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノードにイーサネットインターフェイスを作成します。

以下の YAML ファイルは、イーサネットインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy 1
spec:
  nodeSelector: 2
```

```
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1 4
      description: Configuring eth1 on node01 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では、**hostname** ノードセレクターを使用します。
4. インターフェイスの名前。
5. オプション: 人間が判読できるインターフェイスの説明。
6. インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。
7. 作成後のインターフェイスの要求された状態。
8. オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
9. この例では **ipv4** を有効にします。

1.9.2. 例: Linux ブリッジインターフェイスノードネットワーク設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に Linux ブリッジインターフェイスを作成します。

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

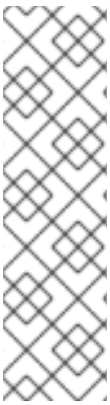
```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: br1 4
        description: Linux bridge with eth1 as a port 5
        type: linux-bridge 6
        state: up 7
```

```
    ipv4:
      dhcp: true 8
      enabled: true 9
    bridge:
      options:
        stp:
          enabled: false 10
      port:
        - name: eth1 11
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では、**hostname** ノードセレクターを使用します。
4. インターフェイスの名前。
5. オプション: 人間が判読できるインターフェイスの説明。
6. インターフェイスのタイプ。この例では、ブリッジを作成します。
7. 作成後のインターフェイスの要求された状態。
8. オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
9. この例では **ipv4** を有効にします。
10. この例では **stp** を無効にします。
11. ブリッジが接続されるノードの NIC。

1.9.3. 例: VLAN インターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に VLAN インターフェイスを作成します。



注記

1 つの **NodeNetworkConfigurationPolicy** マニフェストで、ノードの VLAN インターフェイスに関連するすべての設定を定義します。たとえば、同じ **NodeNetworkConfigurationPolicy** マニフェストで、ノードの VLAN インターフェイスと VLAN インターフェイスの関連ルートを定義します。

ノードが再起動すると、Kubernetes NMState Operator はポリシーを適用する順序を制御できません。したがって、関連するネットワーク設定に別々のポリシーを使用すると、Kubernetes NMState Operator がこれらのポリシーを順番に適用するため、ネットワークオブジェクトがデグレード状態になる可能性があります。

以下の YAML ファイルは、VLAN インターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
    - name: eth1.102 (4)
      description: VLAN using eth1 (5)
      type: vlan (6)
      state: up (7)
      vlan:
        base-iface: eth1 (8)
        id: 102 (9)
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

(3) この例では、**hostname** ノードセレクターを使用します。

(4) インターフェイスの名前。ベアメタルにデプロイする場合、**<interface_name>.<vlan_number>** VLAN 形式のみがサポートされます。

(5) オプション: 人間が判読できるインターフェイスの説明。

(6) インターフェイスのタイプ。以下の例では VLAN を作成します。

(7) 作成後のインターフェイスの要求された状態。

(8) VLAN が接続されているノードの NIC。

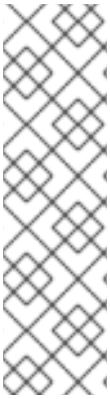
(9) VLAN タグ。

関連情報

- SR-IOV ネットワークデバイスの設定
- ハードウェアオフロードの設定

1.9.4. 例: ボンドインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してノード上にボンドインターフェイスを作成します。



注記

OpenShift Container Platform は以下の bond モードのみをサポートします。

- **active-backup**
- **balance-xor**
- **802.3ad**

その他のボンディングモードはサポートされていません。

balance-xor および **802.3ad** ボンディングモードの場合は、スイッチ設定で "EtherChannel" または同様のポートグループを設定する必要があります。これら 2 つのモードでは、インターフェイスを通過するトラフィックの送信元と送信先に応じて、追加の負荷分散設定も必要になります。**active-backup** ボンディングモードでは、スイッチの設定は必要ありません。その他のボンディングモードはサポートされていません。

以下の YAML ファイルは、ボンドインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
    - name: bond0 (4)
      description: Bond with ports eth1 and eth2 (5)
      type: bond (6)
      state: up (7)
      ipv4:
        dhcp: true (8)
        enabled: true (9)
      link-aggregation:
        mode: active-backup (10)
        options:
          miimon: '140' (11)
        port: (12)
        - eth1
        - eth2
      mtu: 1450 (13)
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

(3) この例では、**hostname** ノードセレクターを使用します。

4 インターフェイスの名前。

5 オプション: 人間が判読できるインターフェイスの説明。

6 インターフェイスのタイプ。この例では、ボンドを作成します。

7 作成後のインターフェイスの要求された状態。

8 オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。

9 この例では **ipv4** を有効にします。

10 ボンドのドライバーモード。この例では、**active backup** を使用します。

11 オプション: この例では、miimon を使用して 140ms ごとにボンドリンクを検査します。

12 ボンド内の下位ノードの NIC。

13 オプション: ボンドの Maximum Transmission Unit (MTU)指定がない場合、この値はデフォルトで **1500** に設定されます。

1.9.5. 例：セカンダリーボンディングインターフェイスのソース負荷分散(SLB)

Open vSwitch (OVS)の **balance-slb** モードをセカンダリーボンドインターフェイスで実行して、プライマリーネットワークからトラフィックを分離することができます。トラフィック管理用のこの設定は、クラスターに高可用性およびトラフィックのセグメント化機能を提供します。

以下の YAML ファイルは、セカンダリーボンディングインターフェイスのマニフェストの例を示しています。このファイルにはサンプル値が含まれていますが、これは独自の情報に置き換える必要があります。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond1-policy 1
spec:
  nodeSelector:
    kubernetes.io/hostname: <node01>
  desiredState:
    interfaces:
    - name: br1
      type: ovs-interface 2
      state: up
      copy-mac-from: enp2s0 3
      ipv4:
        enabled: true
        dhcp: true
    - name: br1
      type: ovs-bridge
      state: up
      ipv4:
        enabled: false
        dhcp: false
      bridge:
```

```
        port:
          - name: br1 4
          - name: bond0
            link-aggregation:
              mode: balance-slb 5
              port: 6
                - name: enp2s0
                - name: enp3s0
      - name: enp2s0
        type: ethernet
        state: up
        ipv4:
          enabled: false
        ipv6:
          enabled: false
      - name: enp3s0
        type: ethernet
        state: up
        ipv4:
          enabled: false
        ipv6:
          enabled: false
# ...
```

1. ポリシーの名前。
2. Open vSwitch (OVS)が管理する仮想インターフェイスを指定します。
3. 仮想インターフェイスが、物理ネットワークインターフェイスコントローラー(NIC)、**enp2s0** の MAC アドレスを継承するようにします。
4. OVS ブリッジの名前。**br0** インターフェイス名は予約されているため、NMstate 設定ではその名前を使用できません。
5. ボンディングモードを SLB に設定して、トラフィックの分散がソース MAC アドレスと VLAN に基づいて行われるようにします。
6. セカンダリーボンディングインターフェイスに含まれる物理インターフェイスを一覧表示します。

1.9.6. 例: 同じノードネットワーク設定ポリシーでの複数のインターフェイス

同じノードネットワーク設定ポリシーで複数のインターフェイスを作成できます。これらのインターフェイスは相互に参照でき、単一のポリシーマニフェストを使用してネットワーク設定をビルドし、デプロイできます。

以下の YAML ファイルの例では、2 つの NIC 間に **bond10** という名前のボンドと、ボンドに接続する **bond10.103** という名前の VLAN を作成します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond-vlan 1
spec:
  nodeSelector: 2
```

```
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond10 4
      description: Bonding eth2 and eth3 5
      type: bond 6
      state: up 7
      link-aggregation:
        mode: balance-xor 8
        options:
          miimon: '140' 9
        port: 10
        - eth2
        - eth3
    - name: bond10.103 11
      description: vlan using bond10 12
      type: vlan 13
      state: up 14
      vlan:
        base-iface: bond10 15
        id: 103 16
      ipv4:
        dhcp: true 17
        enabled: true 18
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では、**hostname** ノードセレクターを使用します。
4. 11 インターフェイスの名前。
5. 12 オプション: 人間が判読できるインターフェイスの説明。
6. 13 インターフェイスのタイプ。
7. 14 作成後のインターフェイスの要求された状態。
8. ボンドのドライバーモード。
9. オプション: この例では、miimon を使用して 140ms ごとにボンドリンクを検査します。
10. ボンド内の下位ノードの NIC。
15. VLAN が接続されているノードの NIC。
16. VLAN タグ。
17. オプション: dhcp を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
18. この例では ipv4 を有効にします。

1.9.7. 例: 仮想機能のノードネットワーク設定ポリシー

NodeNetworkConfigurationPolicy マニフェストを適用して、既存のクラスター内の Single Root I/O Virtualization (SR-IOV) ネットワーク Virtual Function (VF) のホストネットワーク設定を更新します。

NodeNetworkConfigurationPolicy マニフェストを既存のクラスターに適用して、次のタスクを完了できます。

- VF の QoS ホストネットワークを設定して、パフォーマンスを最適化します。
- ネットワークインターフェイスの VF を追加、削除、または更新します。
- VF ボンディング設定を管理します。

注記

SR-IOV Network Operator によっても管理されている Physical Function 上で、NMState を使用して SR-IOV VF のホストネットワーク設定を更新するには、関連する **SriovNetworkNodePolicy** リソースの **externallyManaged** パラメーターを **true** に設定する必要があります。詳細は、**関連情報** セクションを参照してください。

次の YAML ファイルは、VF の QoS ポリシーを定義するマニフェストの例です。この YAML には、独自の情報に置き換える必要があるサンプル値が含まれています。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: qos 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
      - name: ens1f0 4
        description: Change QOS on VF0 5
        type: ethernet 6
        state: up 7
        ethernet:
          sr-iov:
            total-vfs: 3 8
            vfs:
            - id: 0 9
              max-tx-rate: 200 10
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例は、**worker** ロールを持つすべてのノードに適用されます。
4. 物理機能 (PF) ネットワークインターフェイスの名前。
5. オプション: 人間が判読できるインターフェイスの説明。

6 インターフェイスのタイプ。

7 設定後のインターフェイスの要求された状態。

8 VF の総数。

9 ID **0** を持つ VF を識別します。

10 VF の最大伝送速度 (Mbps) を設定します。このサンプル値は、200 Mbps のレートを設定します。

以下の YAML ファイルは、ネットワークインターフェイスに VF を追加するマニフェストの例です。

この設定例では、**ens1f1v0** VF が **ens1f1** 物理インターフェイス上に作成され、この VF はボンディングされたネットワークインターフェイス **bond0** に追加されます。ボンディングは、冗長性に **active-backup** モードを使用します。この例では、VF は、ハードウェアオフロードを使用して物理インターフェイス上の VLAN を直接管理するように設定されています。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: addvf 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens1f1 4
        type: ethernet
        state: up
        ethernet:
          sr-iov:
            total-vfs: 1 5
            vfs:
              - id: 0
                trust: true 6
                vlan-id: 477 7
      - name: bond0 8
        description: Attach VFs to bond 9
        type: bond 10
        state: up 11
        link-aggregation:
          mode: active-backup 12
          options:
            primary: ens1f0v0 13
          port: 14
            - ens1f0v0
            - ens1f1v0 15
```

1 ポリシーの名前。

2 オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべての

3 この例は、**worker** ロールを持つすべてのノードに適用されます。

4 VF ネットワークインターフェイスの名前。

5 作成する VF の数。

6 アクティブな VF とバックアップ VF の間のフェイルオーバーボンディングを許可する設定。

7 VLAN の ID。この例では、ハードウェアオフロードを使用して、VF 上で直接 VLAN を定義します。

8 ボンディングネットワークインターフェイスの名前。

9 オプション: 人間が判読できるインターフェイスの説明。

10 インターフェイスのタイプ。

11 設定後のインターフェイスの要求された状態。

12 ボンディングのボンディングポリシー。

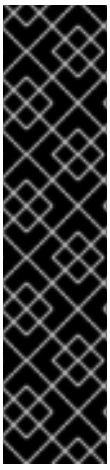
13 割り当てられたプライマリーボンディングポート。

14 ボンディングされたネットワークインターフェイスのポート。

15 この例では、VLAN ネットワークインターフェイスが、ボンディングされたネットワークインターフェイスへの追加インターフェイスとして追加されます。

1.9.8. 例: VRF インスタンスノードのネットワーク設定ポリシーを使用したネットワークインターフェイス

NodeNetworkConfigurationPolicy カスタムリソース (CR) を適用して、Virtual Routing and Forwarding (VRF) インスタンスをネットワークインターフェイスに関連付けます。



重要

VRF インスタンスとネットワークインターフェイスの関連付けは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

VRF インスタンスをネットワークインターフェイスに関連付けることにより、トラフィックの分離、独立したルーティングの決定、およびネットワークリソースの論理的な分離をサポートできます。

警告

仮想ルート転送 (VRF) を設定する場合、VRF 値を **1000** 未満のテーブル ID に変更する必要があります。**1000** より大きい値は OpenShift Container Platform 用に予約されているためです。

ベアメタル環境では、MetalLB を使用して、VRF インスタンスに属するインターフェイスを通じてロードバランサーサービスを通知できます。詳細は、**関連情報** セクションを参照してください。

次の YAML ファイルは、VRF インスタンスをネットワークインターフェイスに関連付ける例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy (1)
spec:
  nodeSelector:
    vrf: "true" (2)
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens4vrf (3)
        type: vrf (4)
        state: up
        vrf:
          port:
            - ens4 (5)
          route-table-id: 2 (6)
```

(1) ポリシーの名前。

(2) この例では、**vrf:true** のラベルが割り当てられたべてのノードにポリシーを適用します。

(3) インターフェイスの名前。

(4) インターフェイスのタイプ。この例では VRF インスタンスを作成します。

(5) VRF が接続されるノードインターフェイス。

(6) VRF のルートテーブル ID の名前。

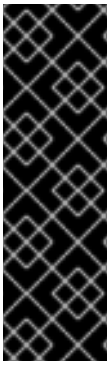
関連情報

- Virtual Routing and Forwarding について
- ネットワーク VRF を介したサービスの公開

1.10. ノード上に IP OVER INFINIBAND インターフェイスを作成する

OpenShift Container Platform Web コンソールでは、IP over InfiniBand (IPoIB) モードをサポートする NVIDIA Network Operator などの Red Hat 認定のサードパーティー Operator をインストールできます。サードパーティー Operator は、通常、OpenShift Container Platform クラスター内のリソースを管理するために他のベンダーのインフラストラクチャーと組み合わせて使用します。クラスター内のノードに IPoIB インターフェイスを作成するには、**NodeNetworkConfigurationPolicy** (NNCP) マニフェストファイルで InfiniBand (IPoIB) インターフェイスを定義する必要があります。

IPoIB をボンディングインターフェイスにアタッチする必要がある場合、この設定は **active-backup** モードのみでサポートされます。



重要

OpenShift Container Platform のドキュメントでは、**NodeNetworkConfigurationPolicy** (NNCP) マニフェストファイルで IPoIB インターフェイス設定を定義する方法のみを説明しています。設定手順の大部分は、NVIDIA およびその他のサードパーティーベンダーのドキュメントを参照してください。Red Hat のサポートは、NNCP 設定以外には適用されません。

NVIDIA Operator の詳細は、Getting Started with Red Hat OpenShift (NVIDIA Docs Hub) を参照してください。

前提条件

- IPoIB インターフェイスをサポートする Red Hat 認定サードパーティー Operator をインストールした。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **NodeNetworkConfigurationPolicy** (NNCP) マニフェストファイルを作成または編集し、ファイル内で IPoIB インターフェイスを指定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: worker-0-ipoib
spec:
# ...
    interfaces:
    - description: ""
      infiniband:
        mode: datagram
        pkey: "0xffff"
      ipv4:
        address:
        - ip: 100.125.3.4
          prefix-length: 16
        dhcp: false
        enabled: true
      ipv6:
        enabled: false
      name: ibp27s0
```

```
      state: up
      identifier: mac-address
      mac-address: 20:00:55:04:01:FE:80:00:00:00:00:00:00:00:02:C9:02:00:23:13:92
      type: infiniband
# ...
```

ここでは、以下のようになります。

<mode>

datagram は、IPoIB インターフェイスのデフォルトモードです。このモードでは、CPU パフォーマンスが向上し、Pod 間通信の遅延が低減されます。**connected** モードはサポートされているモードですが、周辺ネットワークデバイスとのノードの接続性を高めるために最大転送単位 (MTU) 値を調整する必要がある場合にのみ、このモードを使用することを検討してください。

<pkey>

文字列または整数値をサポートします。このパラメーターは、NVIDIA などのサードパーティーベンダーとの認証および暗号化通信を目的としたインターフェイスの保護キー (**P-key**) を定義します。**None** および **0xffff** の値は、InfiniBand システムの基本インターフェイスの保護キーを示します。

<identifier>

サポートされている値は、**name**、デフォルト値、および **mac-address** です。**name** 値は、指定されたインターフェイス名を持つインターフェイスに設定を適用します。

<mac-address>

インターフェイスの MAC アドレスを指定します。IP-over-InfiniBand (IPoIB) インターフェイスの場合、アドレスは 20 バイトの文字列です。

<type>

インターフェイスのタイプを **infiniband** に設定します。

2. 次のコマンドを実行して、クラスター内の各ノードに NNCP 設定を適用します。Kubernetes NMState Operator は、各ノードに IPoIB インターフェイスを作成できます。

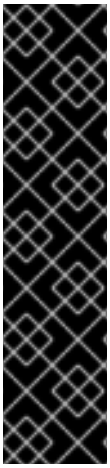
```
$ oc apply -f <nncp_file_name>
```

ここでは、以下のようになります。

<nncp_file_name>

<nncp_file_name> は、NNCP ファイルの名前に置き換えます。

1.11. ブリッジに接続された NIC の静的 IP の取得



重要

NIC の静的 IP のキャプチャーは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

1.11.1. 例: ブリッジに接続された NIC から静的 IP アドレスを継承する Linux ブリッジインターフェイスノードネットワーク設定ポリシー

クラスター内のノードに Linux ブリッジインターフェイスを作成し、単一の **NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して NIC の静的 IP 設定をブリッジに転送します。

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-copy-ipv4-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: ""
  capture:
    eth1-nic: interfaces.name=="eth1" (3)
    eth1-routes: routes.running.next-hop-interface=="eth1"
    br1-routes: capture.eth1-routes | routes.running.next-hop-interface := "br1"
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge (4)
        state: up
        ipv4: "{{ capture.eth1-nic.interfaces.0.ipv4 }}" (5)
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1 (6)
    routes:
      config: "{{ capture.br1-routes.routes.running }}"
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべての

3 ブリッジを接続するノード NIC への参照。

4 インターフェイスのタイプ。この例では、ブリッジを作成します。

5 ブリッジインターフェイスの IP アドレス。この値は、**spec.capture.eth1-nic** エントリーにより参照される NIC の IP アドレスと一致します。

6 ブリッジが接続されるノードの NIC。

関連情報

- The NMPolicy project - Policy syntax

1.12. 例: IP 管理

次の設定スニペットの例は、さまざまな IP 管理方法を示しています。

これらの例では、**ethernet** インターフェイスタイプを使用して、ポリシー設定に関連するコンテキストを表示しつつ、サンプルを単純化します。これらの IP 管理のサンプルは、他のインターフェイスタイプでも使用できます。

1.12.1. 静的

以下のスニペットは、イーサネットインターフェイスで IP アドレスを静的に設定します。

```
# ...
    interfaces:
    - name: eth1
      description: static IP on eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: false
        address:
        - ip: 192.168.122.250 1
          prefix-length: 24
        enabled: true
# ...
```

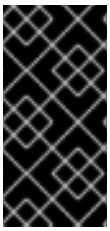
1 この値を、インターフェイスの静的 IP アドレスに置き換えます。

1.12.2. IP アドレスなし

以下のスニペットでは、インターフェイスに IP アドレスがないことを確認できます。

```
# ...
    interfaces:
    - name: eth1
      description: No IP on eth1
      type: ethernet
      state: up
```

```
    ipv4:
      enabled: false
# ...
```



重要

インターフェイスを無効にするために **ipv4.enabled** パラメーターと **ipv6.enabled** パラメーターの両方を **false** に設定する場合は、必ず **state** パラメーターを **up** に設定してください。この設定で **state: down** を設定すると、自動 DHCP 割り当てによりインターフェイスは DHCP IP アドレスを受け取ります。

1.12.3. 動的ホストの設定

以下のスニペットは、動的 IP アドレス、ゲートウェイアドレス、および DNS を使用するイーサネットインターフェイスを設定します。

```
# ...
  interfaces:
  - name: eth1
    description: DHCP on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      enabled: true
# ...
```

以下のスニペットは、動的 IP アドレスを使用しますが、動的ゲートウェイアドレスまたは DNS を使用しないイーサネットインターフェイスを設定します。

```
# ...
  interfaces:
  - name: eth1
    description: DHCP without gateway or DNS on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
      auto-dns: false
      enabled: true
# ...
```

1.12.4. メディアアクセス制御 (MAC) アドレス

ネットワークインターフェイスの名前を使用する代わりに、MAC アドレスを使用してネットワークインターフェイスを識別することができます。ネットワークインターフェイス名は、オペレーティングシステムの設定の変更など、さまざまな理由で変更される可能性があります。一方で、各ネットワークインターフェイスには、変更されない一意の MAC アドレスがあります。そのため、MAC アドレスを使用すると、特定のネットワークインターフェイスをより永続的に識別すことができます。

identifier パラメーターでサポートされている値は、デフォルトの **name** 値と **mac-address** 値です。**name** 値は、指定されたインターフェイス名を持つインターフェイスに設定を適用します。

identifier パラメーターに **mac-address** 値を使用すると、MAC アドレスがネットワークインターフェイスの識別子であることが指定されます。**identifier** の値を **mac-address** に設定する場合は、その後に続く **mac-address** パラメーターフィールドに特定の MAC アドレスを入力する必要があります。

注記

name パラメーターの値を指定することもできますが、**identifier: mac-address** 値を設定すると、ネットワークインターフェイスのプライマリー識別子として MAC アドレスが使用されることになります。間違った MAC アドレスを指定した場合、**nmstate** によって無効な引数のエラーが報告されます。

次のスニペットでは、イーサネットデバイスのプライマリー識別子として MAC アドレスを指定します。デバイスの名前は **eth1**、MAC アドレスは **8A:8C:92:1A:F6:98** です。

```
# ...
interfaces:
- name: eth1
  profile-name: wan0
  type: ethernet
  state: up
  identifier: mac-address
  mac-address: 8A:8C:92:1A:F6:98
# ...
```

1.12.5. DNS

デフォルトでは、**nmstate** API は DNS 値をネットワークインターフェイスに保存するのではなく、グローバルに保存します。特定の状況では、DNS 値を保存するようにネットワークインターフェイスを設定する必要があります。

ヒント

DNS 設定の設定は、**/etc/resolv.conf** ファイルの変更に相当します。

ネットワークインターフェイスの DNS 設定を定義するには、最初にネットワークインターフェイスの YAML 設定ファイルで **dns-resolver** セクションを指定する必要があります。NNCP 設定をネットワークインターフェイスに適用するには、**oc apply -f <nncp_file_name>** コマンドを実行する必要があります。

次の例は、DNS 値をグローバルに保存するデフォルトの状況を示しています。

- ネットワークインターフェイスなしで静的 DNS を設定します。ホストノード上の **/etc/resolv.conf** ファイルを更新する場合、**NodeNetworkConfigurationPolicy** (NNCP) マニフェストでインターフェイス (IPv4 または IPv6) を指定する必要はありません。

  DNS 値をグローバルに保存するネットワークインターフェイスの DNS 設定の例

  ```
  apiVersion: nmstate.io/v1
  kind: NodeNetworkConfigurationPolicy
  metadata:
   name: worker-0-dns-testing
  spec:
    nodeSelector:
  ```

```
    kubernetes.io/hostname: <target_node>
  desiredState:
    dns-resolver:
      config:
        server:
        - 2001:db8:f::1
        - 192.0.2.251
        search:
        - example.com
        - example.org
# ...
```

重要

次の例に示すように、NNCP ファイルの **dns-resolver.config** セクションで DNS オプションを指定できます。

```
# ...
desiredState:
    dns-resolver:
      config:
        options:
         - timeout:2
         - attempts:3
# ...
```

ネットワークインターフェイスから DNS オプションを削除する場合は、NNCP に次の設定を適用してから **oc apply -f <nncp_file_name>** コマンドを実行します。

```
# ...
    dns-resolver:
      config: {}
    interfaces: []
# ...
```

次の例は、DNS 値を格納するためにネットワークインターフェイスを設定する必要がある状況を示しています。

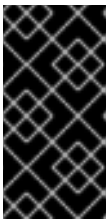
- 静的 DNS ネームサーバーを動的 DNS ネームサーバーよりも優先する場合は、ネットワークインターフェイス YAML 設定ファイルで、Dynamic Host Configuration Protocol (DHCP) または IPv6 自動設定 (**autoconf**) メカニズムのいずれかを実行するインターフェイスを定義します。

  DHCPv4 ネットワークプロトコルから取得した DNS ネームサーバーに 192.0.2.1 を追加する設定の例

```
# ...
dns-resolver:
  config:
    server:
    - 192.0.2.1
interfaces:
  - name: eth1
    type: ethernet
```

```
    state: up
    ipv4:
      enabled: true
      dhcp: true
      auto-dns: true
# ...
```

- **nmstate** API を使用して DNS 値をグローバルに保存するデフォルトの方法を採用するのではなく、DNS 値を保存するようにネットワークインターフェイスを設定する必要がある場合は、ネットワークインターフェイス YAML ファイルで静的 DNS 値と静的 IP アドレスを設定できます。



重要

ネットワークインターフェイスレベルで DNS 値を保存すると、インターフェイスを Open vSwitch (OVS) ブリッジ、Linux ブリッジ、ボンディングなどのネットワークコンポーネントに接続した後に、名前解決の問題が発生する可能性があります。

インターフェイスレベルで DNS 値を保存する設定の例

```
# ...
dns-resolver:
  config:
    server:
    - 2001:db8:1::d1
    - 2001:db8:1::d2
    - 192.0.2.1
    search:
    - example.com
    - example.org
interfaces:
  - name: eth1
    type: ethernet
    state: up
    ipv4:
      address:
      - ip: 192.0.2.251
        prefix-length: 24
      dhcp: false
      enabled: true
    ipv6:
      address:
      - ip: 2001:db8:1::1
        prefix-length: 64
      dhcp: false
      enabled: true
      autoconf: false
# ...
```

- ネットワークインターフェイスに静的 DNS 検索ドメインと静的 DNS ネームサーバーを設定する場合は、ネットワークインターフェイスの YAML 設定ファイルで、Dynamic Host Configuration Protocol (DHCP) または IPv6 自動設定 (**autoconf**) メカニズムのいずれかを実行する静的インターフェイスを定義します。

重要

ネットワークインターフェイスの YAML ファイルで次の **dns-resolver** 設定を指定すると、再起動時に競合状態が発生し、クラスターで実行される Pod に **NodeNetworkConfigurationPolicy** (NNCP) が適用されなくなる可能性があります。

- ネットワークインターフェイスの静的 DNS 検索ドメインと動的 DNS ネームサーバーを設定する設定。
- **search** パラメーターにドメイン接尾辞を指定し、**server** パラメーターに IP アドレスを設定しない設定。

example.com および example.org の静的 DNS 検索ドメインと静的 DNS ネームサーバー設定を行う設定例

```
# ...
dns-resolver:
  config:
    server:
    - 2001:db8:f::1
    - 192.0.2.251
    search:
    - example.com
    - example.org
interfaces:
  - name: eth1
    type: ethernet
    state: up
    ipv4:
      enabled: true
      dhcp: true
      auto-dns: true
    ipv6:
      enabled: true
      dhcp: true
      autoconf: true
      auto-dns: true
# ...
```

1.13. ルートとルートルール

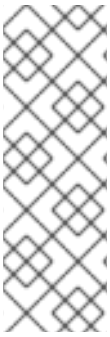
ネットワークインターフェイスの IP アドレスを設定した後、クラスターノードの NMState 設定でルートおよびルートルールを設定できます。

重要

静的ルートを設定する際に、カスタマイズされた **br-ex** ブリッジを手動で設定していない限り、OVN-Kubernetes **br-ex** ブリッジをネクストホップインターフェイスとして使用することはできません。

詳細は、**インストーラーでプロビジョニングされるクラスターのベアメタルへのデプロイ**、または **ユーザーがプロビジョニングしたクラスターをベアメタルにインストール** ドキュメントの「カスタマイズされた br-ex ブリッジを含むマニフェストオブジェクトの作成」を参照してください。

routes パラメーターは静的ルートを定義し、これらのルートは、ネットワークインターフェイスを離れるトラフィックとトラフィックの宛先ネットワークを決定します。サポートされている値には、**実行** および **設定** が含まれます。



注記

NMState 設定をクラスターノードに適用し、既存のルートを変更する場合、**state: absent** パラメーターで古いルートを指定し、**state: present** パラメーターを使用して新しいルートを指定する必要があります。その後、NMState オペレーターは古いルートを削除し、新規ルートをクラスターノードに適用できます。

state パラメーターを **ignore** に設定すると、Operator は特定のルートを無視することを意味します。

route-rules パラメーターは、クラスターノードのポリシーベースのルーティング機能を実装します。この機能により、別のソース IP アドレスから発信されたトラフィックを分離し、異なるゲートウェイやネットワークパスを経由してルーティングすることができます。

次の YAML 設定は、静的ルートとインターフェイス **eth1** での静的 IP 調整を示しています。

```
dns-resolver:
  config:
# ...
interfaces:
  - name: eth1
    description: Static routing on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: false
      enabled: true
      address:
        - ip: 192.0.2.251
          prefix-length: 24
route-rules:
  config:
  - ip-from: 198.51.100.0/24
    priority: 1000
    route-table: 200
routes:
  config:
  - destination: 198.51.100.0/24
    next-hop-interface: eth1
```

```
    next-hop-address: 192.0.2.1
    metric: 150
    table-id: 200
# ...
```

- **config.ip-from**: 指定された IP アドレスから発信される任意のネットワークパケットにルールを適用します。
- **config.priority**: ルールの優先順位を設定します。
- **config.route-table**: Operator がネットワークトラフィックが **ip-from** 条件と一致することを確認するために使用するルーティングテーブルを指定します。
- **address.ip**: イーサネットインターフェイスの静的 IP アドレス。
- **config.next-hop-address**: ノードトラフィックの次のホップアドレス。これは、イーサネットインターフェイスに設定される IP アドレスと同じサブネットにある必要があります。

関連情報

- カスタマイズされた br-ex ブリッジ (インストーラーによりプロビジョニングされたインフラストラクチャー) を含むマニフェストオブジェクトの作成
- カスタマイズされた br-ex ブリッジ (ユーザーによりプロビジョニングされたインフラストラクチャー) を含むマニフェストオブジェクトの作成
- Routes (nmstate ドキュメント)
- Route Rules (nmstate ドキュメント)

第2章 ノードのネットワーク設定のトラブルシューティング

ノードのネットワーク設定で問題が発生した場合には、ポリシーが自動的にロールバックされ、enactment (実行) レポートは失敗します。これには、以下のような問題が含まれます。

- ホストで設定を適用できません。
- ホストはデフォルトゲートウェイへの接続を失います。
- ホストは API サーバーへの接続を失います。

2.1. 正確でないノードネットワーク設定のポリシー設定のトラブルシューティング

ノードネットワーク設定ポリシーを適用し、クラスター全体でノードのネットワーク設定への変更を適用することができます。

誤った設定を適用した場合は、次の例を使用して、失敗したノードネットワークポリシーをトラブルシューティングして修正できます。この例では、3 つのコントロールプレーンノードと 3 つのコンピュートノードを持つクラスターに Linux ブリッジポリシーを適用します。ポリシーが間違ったインターフェイスを参照しているため、ポリシーは適用されません。

エラーを見つけるには、利用可能な NMState リソースを調査する必要があります。その後に、正しい設定でポリシーを更新できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- Linux システムに **ens01** インターフェイスが存在しない。

手順

1. クラスターにポリシーを作成します。次の例では、**ens01** をメンバーとして持つ単純なブリッジ **br1** を作成します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
          port:
```

```
            - name: ens01
# ...
```

2. ネットワークインターフェイスにポリシーを適用します。

```
$ oc apply -f ens01-bridge-testfail.yaml
```

出力例

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

3. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get nncp
```

この出力は、ポリシーが失敗したことを示しています。

出力例

```
NAME                    STATUS
ens01-bridge-testfail   FailedToConfigure
```

ポリシーのステータスのみでは、すべてのノードで失敗したか、ノードのサブセットで失敗したかを確認することはできません。

4. ノードのネットワーク設定の enactment (実行) をリスト表示し、ポリシーがいずれかのノードで成功したかどうかを確認します。ポリシーがノードのサブセットに対してのみ失敗した場合、出力は、問題が特定のノード設定にあることを示唆します。すべてのノードでポリシーが失敗した場合、出力はポリシーに問題があることを示唆します。

```
$ oc get nnce
```

この出力は、ポリシーがすべてのノードで失敗したことを示しています。

出力例

```
NAME                                     STATUS
control-plane-1.ens01-bridge-testfail    FailedToConfigure
control-plane-2.ens01-bridge-testfail    FailedToConfigure
control-plane-3.ens01-bridge-testfail    FailedToConfigure
compute-1.ens01-bridge-testfail          FailedToConfigure
compute-2.ens01-bridge-testfail          FailedToConfigure
compute-3.ens01-bridge-testfail          FailedToConfigure
```

5. 失敗した enactment の 1 つを表示します。以下のコマンドは、出力ツール **jsonpath** を使用して出力をフィルターします。

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?
(@.type=="Failing")].message}'
```

出力例

```
[2024-10-10T08:40:46Z INFO  nmstatectl] Nmstate version: 2.2.37
NmstateError: InvalidArgument: Controller interface br1 is holding unknown port ens01
```

前の例は、**InvalidArgument** エラーからの出力で、**ens01** が不明なポートであることを示しています。この例では、ポリシー設定ファイル内のポート設定を変更する必要がある場合があります。

6. ポリシーが適切に設定されていることを確認するには、**NodeNetworkState** オブジェクトを要求して、1つまたはすべてのノードのネットワーク設定を表示します。以下のコマンドは、**control-plane-1** ノードのネットワーク設定を返します。

```
$ oc get nns control-plane-1 -o yaml
```

出力は、ノード上のインターフェイス名は **ens1** であるものの、失敗したポリシーが **ens01** を誤って使用していることを示します。

出力例

```
  - ipv4:
# ...
    name: ens1
    state: up
    type: ethernet
```

7. 既存のポリシーを編集してエラーを修正します。

```
$ oc edit nncp ens01-bridge-testfail
```

```
# ...
        port:
          - name: ens1
```

ポリシーを保存して修正を適用します。

8. ポリシーのステータスをチェックして、更新が正常に行われたことを確認します。

```
$ oc get nncp
```

出力例

```
NAME                    STATUS
ens01-bridge-testfail   SuccessfullyConfigured
```

更新されたポリシーは、クラスターのすべてのノードで正常に設定されました。

2.2. 非接続環境での DNS 接続の問題のトラブルシューティング

非接続環境で **nmstate** を設定するときに DNS 接続の問題が発生する場合は、ドメイン **root-servers.net** のネームサーバーのリストを解決するように DNS サーバーを設定できます。

重要

DNS サーバーに **root-servers.net** ゾーンのネームサーバー (NS) エントリーが含まれていることを確認します。DNS サーバーはクエリーをアップストリームのリゾルバーに転送する必要はありませんが、サーバーは NS クエリーに対して正しい回答を返す必要があります。

2.2.1. bind9 DNS 名前付きサーバーの設定

bind9 DNS サーバーを照会するように設定されたクラスターの場合は、少なくとも 1 つの NS レコードを含む設定ファイルに **root-servers.net** ゾーンを追加できます。たとえば、この条件にすでに一致するゾーンファイルとして **/var/named/named.localhost** を使用できます。

手順

1. 次のコマンドを実行して、**/etc/named.conf** 設定ファイルの最後に **root-servers.net** ゾーンを追加します。

```
$ cat >> /etc/named.conf <<EOF
zone "root-servers.net" IN {
    type master;
    file "named.localhost";
};
EOF
```

2. 次のコマンドを実行して、**named** サービスを再起動します。

```
$ systemctl restart named
```

3. 次のコマンドを実行して、**root-servers.net** ゾーンが存在することを確認します。

```
$ journalctl -u named|grep root-servers.net
```

出力例

```
Jul 03 15:16:26 rhel-8-10 bash[xxxx]: zone root-servers.net/IN: loaded serial 0
Jul 03 15:16:26 rhel-8-10 named[xxxx]: zone root-servers.net/IN: loaded serial 0
```

4. 次のコマンドを実行して、DNS サーバーが **root-servers.net** ドメインの NS レコードを解決できることを確認します。

```
$ host -t NS root-servers.net. 127.0.0.1
```

出力例

```
Using domain server:
Name: 127.0.0.1
Address: 127.0.0.53
Aliases:
root-servers.net name server root-servers.net.
```

2.2.2. dnsmasq DNS サーバーの設定

dnsmasq を DNS サーバーとして使用している場合は、指定した DNS サーバーを使用して **root-servers.net** を解決する新しい設定ファイルを作成するなどして、**root-servers.net** ドメインの解決を別の DNS サーバーに委任できます。

1. 次のコマンドを実行して、ドメイン **root-servers.net** を別の DNS サーバーに委任する設定ファイルを作成します。

```
$ echo 'server=/root-servers.net/<DNS_server_IP>'> /etc/dnsmasq.d/delegate-root-servers.net.conf
```

2. 次のコマンドを実行して、**dnsmasq** サービスを再起動します。

```
$ systemctl restart dnsmasq
```

3. 次のコマンドを実行して、**root-servers.net** ドメインが別の DNS サーバーに委任されていることを確認します。

```
$ journalctl -u dnsmasq|grep root-servers.net
```

出力例

```
Jul 03 15:31:25 rhel-8-10 dnsmasq[1342]: using nameserver 192.168.1.1#53 for domain root-servers.net
```

4. 次のコマンドを実行して、DNS サーバーが **root-servers.net** ドメインの NS レコードを解決できることを確認します。

```
$ host -t NS root-servers.net. 127.0.0.1
```

出力例

```
Using domain server:
Name: 127.0.0.1
Address: 127.0.0.1#53
Aliases:
root-servers.net name server root-servers.net.
```