OpenShift Container Platform 4.20

Kubernetes NMState

Observing and updating node network state and configuration using Kubernetes NMState in OpenShift Container Platform

Last Updated: 2026-01-21

OpenShift Container Platform 4.20 Kubernetes NMState

Observing and updating node network state and configuration using Kubernetes NMState in OpenShift Container Platform

Legal Notice

Abstract

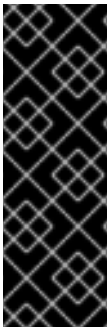
This document covers using Kubernetes NMState to observe, update, and troubleshoot node network configurations in OpenShift Container Platform.

Table of Contents

CHAPTER 1. OBSERVING AND UPDATING THE NODE NETWORK STATE AND CONFIGURATION

To observe and update the node network state and configuration in your cluster, you can use the Kubernetes NMState Operator. You can view network states, create and manage network configuration policies, and configure interfaces on cluster nodes.

For more information about how to install the NMState Operator, see Kubernetes NMState Operator.



IMPORTANT

You cannot modify an existing **br-ex** bridge, an OVN-Kubernetes-managed Open vSwitch bridge, or any interfaces, bonds, VLANs, and so on that associate with the **br-ex** bridge. However, you can configure a customized br-ex bridge.

For more information, see "Creating a manifest object that includes a customized br-ex bridge" in the *Deploying installer-provisioned clusters on bare metal* document or the *Installing a user-provisioned cluster on bare metal* document.

1.1. VIEWING THE NETWORK STATE OF A NODE BY USING THE CLI

Node network state is the network configuration for all nodes in the cluster. A **NodeNetworkState** object exists on every node in the cluster. This object is periodically updated and captures the state of the network for that node.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. List all the **NodeNetworkState** objects in the cluster:

```
$ oc get nns
```

2. Inspect a **NodeNetworkState** object to view the network on that node. The output in this example has been redacted for clarity:

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

Example output:

```
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. The name of the **NodeNetworkState** object is taken from the node.
2. The **currentState** contains the complete network configuration for the node, including DNS, interfaces, and routes.
3. Timestamp of the last successful update. This is updated periodically as long as the node is reachable and can be used to evalute the freshness of the report.

1.2. VIEWING A GRAPHICAL REPRESENTATION OF THE NETWORK STATE OF A NODE (NNS) TOPOLOGY FROM THE WEB CONSOLE

To make the configuration of the node network in the cluster easier to understand, you can view it in the form of a diagram.

The NNS topology diagram displays all node components (network interface controllers, bridges, bonds, and VLANs), their properties and configurations, and connections between the nodes.

Procedure

- In the **Administrator** view of the OpenShift Container Platform web console, navigate to **Networking → Node Network Configuration**.
  The NNS topology diagram opens. Each group of components represents a single node.
  - To display the configuration and properties of a node, click inside the border of the node.
  - To display the features or the YAML file of a specific component (for example, an interface or a bridge), click the icon of the component.
  - The icons of active components have green borders; the icons of disconnected components have red borders.

1.3. VIEWING THE LIST OF NODENETWORKSTATE RESOURCES

As an administrator, you can use the OpenShift Container Platform web console to view the list of **NodeNetworkState** resources and network interfaces, and access network details.

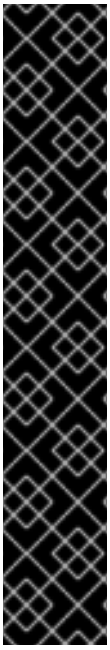
Procedure

1. Navigate to **Networking → Node Network Configuration**.
2. Click the **List** icon.
   You can now view the list of **NodeNetworkState** resources and the corresponding interfaces that are created on the nodes.
   - You can use **Filter** based on **Interface state**, **Interface type**, and **IP**, or the search bar based on criteria **Name** or **Label**, to narrow down the displayed **NodeNetworkState** resources.

- To access the detailed information about a **NodeNetworkState** resource, click the **NodeNetworkState** resource name listed in the **Name** column .
- To expand and view the **Network Details** section for the **NodeNetworkState** resource, click the greater than (>) symbol . Alternatively, you can click on each interface type under the **Network interface** column to view the network details.

1.4. ABOUT THE NODENETWORKCONFIGURATIONPOLICY MANIFEST FILE

A **NodeNetworkConfigurationPolicy** manifest file defines policies that the Kubernetes NMState Operator uses to configure networking for nodes in your OpenShift Container Platform cluster. You can create, edit, and delete these policies to manage node network configurations.



IMPORTANT

If you want to apply multiple NNCP CRs to a node, you must create the NNCPs in a logical order that is based on the alphanumeric sorting of the policy names. The Kubernetes NMState Operator continuously checks for a newly created NNCP CR so that the Operator can instantly apply the CR to node. Consider the following logical order issue example:

1. You create NNCP 1 for defining the bridge interface that listens on a VLAN port, such as **eth1.1000**.
2. You create NNCP 2 for defining the VLAN interface and specify the port for this interface, such as **eth1.1000**.
3. You apply NNCP 1 before you apply NNCP 2 to the node.

The node experiences a node connectivity issue because port **eth1.1000** does not exist. As a result, the cluster fails.

After you apply a node network policy to a node, the Kubernetes NMState Operator configures the networking configuration for nodes according to the node network policy details.

WARNING

The following list of interface names are reserved and you cannot use the names with NMstate configurations:

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

You can create an NNCP by using either the OpenShift CLI (**oc**) or the OpenShift Container Platform web console. As a postinstallation task you can create an NNCP or edit an existing NNCP.

NOTE

Before you create an NNCP, ensure that you read the "Example policy configurations for different interfaces" document.

If you want to delete an NNCP, you can use the **oc delete nncp** command to complete this action. However, this command does not delete any objects, such as a bridge interface.

Deleting the node network policy that added an interface to a node does not change the configuration of the policy on the node. Similarly, removing an interface does not delete the policy, because the Kubernetes NMState Operator re-adds the removed interface whenever a pod or a node is restarted.

To effectively delete the NNCP, the node network policy, and any interfaces would typically require the following actions:

1. Edit the NNCP and remove interface details from the file. Ensure that you do not remove **name**, **state**, and **type** parameters from the file.
2. Add **state: absent** under the **interfaces.state** section of the NNCP.
3. Run **oc apply -f <nncp_file_name>**. After the Kubernetes NMState Operator applies the node network policy to each node in your cluster, any interface that exists on each node is now marked as *absent*.
4. Run **oc delete nncp** to delete the NNCP.

Additional resources

- Example policy configurations for different interfaces
- Removing an interface from nodes

1.5. MANAGING POLICY FROM THE WEB CONSOLE

You can update the node network configuration, such as adding or removing interfaces from nodes, by applying **NodeNetworkConfigurationPolicy** manifests to the cluster.

Manage the policy from the web console by accessing the list of created policies in the **NodeNetworkConfigurationPolicy** page under the **Networking** menu. This page enables you to create, update, monitor, and delete the policies.

1.5.1. Monitoring the policy status

You can monitor the policy status from the **NodeNetworkConfigurationPolicy** page. This page displays all the policies created in the cluster in a tabular format, with the following columns:

Name
: The name of the policy created.

Matched nodes
: The count of nodes where the policies are applied. This could be either a subset of nodes based on the node selector or all the nodes on the cluster.

Node network state
: The enactment state of the matched nodes. You can click on the enactment state and view detailed information on the status.

To find the desired policy, you can filter the list either based on enactment state by using the **Filter** option, or by using the search option.

1.5.2. Creating a policy

You can create a policy by using either a form or YAML in the web console. When creating a policy using a form, you can see how the new policy changes the topology of the nodes in your cluster in real time.

Procedure

1. Navigate to **Networking → Node Network Configuration**.
2. On the **Node Network Configuration** page, click **Create** and select the **From Form** option.

NOTE

To create a policy using YAML, click **Create → With YAML** option. However, the following steps apply only to the form method.

3. Optional: Check the **Apply this NodeNetworkConfigurationPolicy only to specific subsets of nodes using the node selector** checkbox to specify the nodes where the policy must be applied.
4. Enter the policy name in the **Policy name** field.
5. Optional: Enter the description of the policy in the **Description** field.
6. Click **Next** to move to the **Policy Interfaces** section.
7. In the **Bridging** part of the **Policy Interfaces** section, a bridge interface named **br0** is added by default with preset values in editable fields. If required, edit the values by performing the following steps:
   a. Enter the name of the interface in **Interface name** field.
   b. Select the required network state. The default selected state is **Up**.
   c. Select the type of interface. The available types are **Bridge**, **Bonding**, and **Ethernet**. The default selected value is **Bridge**.

NOTE

Addition of a VLAN interface by using the form is not supported. To add a VLAN interface, you must use YAML to create the policy. Once added, you cannot edit the policy by using form.

   d. Optional: In the IP configuration section, check **IPv4** checkbox to assign an IPv4 address to the interface, and configure the IP address assignment details:
      i. Click **IP address** to configure the interface with a static IP address, or **DHCP** to auto-assign an IP address.
      ii. If you have selected **IP address** option, enter the IPv4 address in **IPV4 address** field, and enter the prefix length in **Prefix length** field.
      If you have selected **DHCP** option, uncheck the options that you want to disable. The available options are **Auto-DNS**, **Auto-routes**, and **Auto-gateway**. All the options are selected by default.
   e. Optional: Enter the port number in **Port** field.
   f. Optional: Check the checkbox **Enable STP** to enable STP.
   g. Optional: To add an interface to the policy, click **Add another interface to the policy**.

h. Optional: To remove an interface from the policy, click

icon next to the interface.

NOTE

Alternatively, you can click **Edit YAML** on the top of the page to continue editing the form using YAML.

8. Click **Next** to go to the **Review** section of the form.
9. Verify the settings and click **Create** to create the policy.

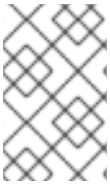
1.6. UPDATING THE NODENETWORKCONFIGURATIONPOLICY MANIFEST FILE

To modify the network configuration for nodes in your OpenShift Container Platform cluster, you can update the **NodeNetworkConfigurationPolicy** manifest file.

1.6.1. Updating the policy by using form

Procedure

1. Navigate to **Networking → NodeNetworkConfigurationPolicy**.
2. In the **NodeNetworkConfigurationPolicy** page, click the  icon placed next to the policy you want to edit, and click **Edit**.
3. Edit the fields that you want to update.
4. Click **Save**.



NOTE

Addition of a VLAN interface using the form is not supported. To add a VLAN interface, you must use YAML to create the policy. Once added, you cannot edit the policy using form.

1.6.2. Updating the policy by using YAML

Procedure

1. Navigate to **Networking → NodeNetworkConfigurationPolicy**.
2. In the **NodeNetworkConfigurationPolicy** page, click the policy name under the **Name** column for the policy you want to edit.
3. Click the **YAML** tab, and edit the YAML.
4. Click **Save**.

1.6.3. Deleting the policy

Procedure

1. Navigate to **Networking → NodeNetworkConfigurationPolicy**.
2. In the **NodeNetworkConfigurationPolicy** page, click the  icon placed next to the policy you want to delete, and click **Delete**.
3. In the pop-up window, enter the policy name to confirm deletion, and click **Delete**.

1.7. MANAGING THE NODENETWORKCONFIGURATIONPOLICY MANIFEST FILE

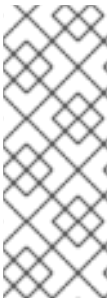
To configure network interfaces on nodes in your OpenShift Container Platform cluster, you can manage the **NodeNetworkConfigurationPolicy** manifest file by using the CLI.

1.7.1. Creating an interface on nodes

You can create an interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** (NNCP) manifest to the cluster. The manifest details the requested configuration for the interface.

By default, the manifest applies to all nodes in the cluster. To add the interface to specific nodes, add the **spec: nodeSelector** parameter and the appropriate **<key>:<value>** for your node selector.

You can configure multiple nmstate-enabled nodes concurrently. The configuration applies to 50% of the nodes in parallel. This strategy prevents the entire cluster from being unavailable if the network connection fails. To apply the policy configuration in parallel to a specific portion of the cluster, use the **maxUnavailable** parameter in the **NodeNetworkConfigurationPolicy** manifest configuration file.



NOTE

If you have two nodes and you apply an NNCP manifest with the **maxUnavailable** parameter set to **50%** to these nodes, one node at a time receives the NNCP configuration. If you then introduce an additional NNCP manifest file with the **maxUnavailable** parameter set to **50%**, this NCCP is independent of the initial NNCP; this means that if both NNCP manifests apply a bad configuration to nodes, you can no longer guarantee that half of your cluster is functional.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create the **NodeNetworkConfigurationPolicy** manifest. The following example configures a Linux bridge on all worker nodes and configures the DNS resolver:

```
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) Name of the policy.

(2) Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

(3) This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.

(4) Optional: Specifies the maximum number of nmstate-enabled nodes that the policy configuration can be applied to concurrently. This parameter can be set to either a percentage value (string), for example, **"10%"**, or an absolute value (number), such as **3**.

(5) Optional: Human-readable description for the interface.

(6) Optional: Specifies the search and server settings for the DNS server.

2. Create the node network policy:

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

1 File name of the node network configuration policy manifest.

Additional resources

- Example for creating multiple interfaces in the same policy
- Examples of different IP management methods in policies

1.7.2. Confirming node network policy updates on nodes

When you apply a node network policy, a **NodeNetworkConfigurationEnactment** object is created for every node in the cluster. The node network configuration enactment is a read-only object that represents the status of execution of the policy on that node.

If the policy fails to be applied on the node, the enactment for that node includes a traceback for troubleshooting.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. To confirm that a policy has been applied to the cluster, list the policies and their status:

```
$ oc get nncp
```

2. Optional: If a policy is taking longer than expected to successfully configure, you can inspect the requested state and status conditions of a particular policy:

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

3. Optional: If a policy is taking longer than expected to successfully configure on all nodes, you can list the status of the enactments on the cluster:

```
$ oc get nnce
```

4. Optional: To view the configuration of a particular enactment, including any error reporting for a failed configuration:

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

1.7.3. Removing an interface from nodes

You can remove an interface from one or more nodes in the cluster by editing the **NodeNetworkConfigurationPolicy** object and setting the **state** of the interface to **absent**.

Removing an interface from a node does not automatically restore the node network configuration to a previous state. If you want to restore the previous state, you will need to define that node network configuration in the policy.

If you remove a bridge or bonding interface, any node NICs in the cluster that were previously attached or subordinate to that bridge or bonding interface are placed in a **down** state and become unreachable. To avoid losing connectivity, configure the node NIC in the same policy so that it has a status of **up** and either DHCP or a static IP address.

NOTE

Deleting the node network policy that added an interface does not change the configuration of the policy on the node. Although a **NodeNetworkConfigurationPolicy** is an object in the cluster, the object only represents the requested configuration. Similarly, removing an interface does not delete the policy.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Update the **NodeNetworkConfigurationPolicy** manifest used to create the interface. The following example removes a Linux bridge and configures the **eth1** NIC with DHCP to avoid losing connectivity:

```
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. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
4. Changing the state to **absent** removes the interface.
5. The name of the interface that is to be unattached from the bridge interface.

6 The type of interface. This example creates an Ethernet networking interface.

7 The requested state for the interface.

8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

9 Enables **ipv4** in this example.

2. Update the policy on the node and remove the interface:

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

1 File name of the policy manifest.

1.8. EXAMPLE POLICY CONFIGURATIONS FOR DIFFERENT INTERFACES

Before you read the different example **NodeNetworkConfigurationPolicy** (NNCP) manifest configurations, consider the following factors when you apply a policy to nodes so that your cluster runs under its best performance conditions.

- If you want to apply multiple NNCP CRs to a node, you must create the NNCPs in a logical order that is based on the alphanumeric sorting of the policy names. The Kubernetes NMState Operator continuously checks for a newly created NNCP CR so that the Operator can instantly apply the CR to node.
- When you need to apply a policy to many nodes but you only want to create a single NNCP for all the nodes, the Kubernetes NMState Operator applies the policy to each node in sequence. You can set the speed and coverage of policy application for target nodes with the **maxUnavailable** parameter in the cluster's configuration file. By setting a lower percentage value for the parameter, you can reduce the risk of a cluster-wide outage if the outage impacts the small percentage of nodes that are receiving the policy application.
- If you set the **maxUnavailable** parameter to **50%** in two NNCP manifests, the policy configuration coverage applies to 100% of the nodes in your cluster.
- When a node restarts, the Kubernetes NMState Operator cannot control the order to which it applies policies to nodes. The Kubernetes NMState Operator might apply interdependent policies in a sequence that results in a degraded network object.
- Consider specifying all related network configurations in a single policy.

1.8.1. Example: Ethernet interface node network configuration policy

You can configure an Ethernet interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for an Ethernet interface. It includes sample values that you must replace with your own information.

```
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) Name of the policy.

(2) Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

(3) This example uses a **hostname** node selector.

(4) Name of the interface.

(5) Optional: Human-readable description of the interface.

(6) The type of interface. This example creates an Ethernet networking interface.

(7) The requested state for the interface after creation.

(8) Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.

(9) Enables **ipv4** in this example.

1.8.2. Example: Linux bridge interface node network configuration policy

You can create a Linux bridge interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes samples values that you must replace with your own information.

```
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. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. This example uses a **hostname** node selector.
4. Name of the interface.
5. Optional: Human-readable description of the interface.
6. The type of interface. This example creates a bridge.
7. The requested state for the interface after creation.
8. Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
9. Enables **ipv4** in this example.
10. Disables **stp** in this example.
11. The node NIC to which the bridge attaches.

1.8.3. Example: VLAN interface node network configuration policy

You can create a VLAN interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

NOTE

Define all related configurations for the VLAN interface of a node in a single **NodeNetworkConfigurationPolicy** manifest. For example, define the VLAN interface for a node and the related routes for the VLAN interface in the same **NodeNetworkConfigurationPolicy** manifest.

When a node restarts, the Kubernetes NMState Operator cannot control the order in which policies are applied. Therefore, if you use separate policies for related network configurations, the Kubernetes NMState Operator might apply these policies in a sequence that results in a degraded network object.

The following YAML file is an example of a manifest for a VLAN interface. It includes samples values that you must replace with your own information.

```
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
```

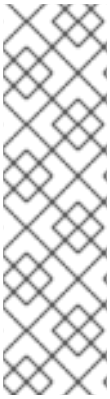
1. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. This example uses a **hostname** node selector.
4. Name of the interface. When deploying on bare metal, only the **<interface_name>.<vlan_number>** VLAN format is supported.
5. Optional: Human-readable description of the interface.
6. The type of interface. This example creates a VLAN.
7. The requested state for the interface after creation.
8. The node NIC to which the VLAN is attached.
9. The VLAN tag.

Additional resources

- Configuring an SR-IOV network device
- Configuring hardware offloading

1.8.4. Example: Bond interface node network configuration policy

You can create a bond interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.



NOTE

OpenShift Container Platform only supports the following bond modes:

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

Other bond modes are not supported.

The **balance-xor** and **802.3ad** bond modes require switch configuration to establish an "EtherChannel" or similar port grouping. Those two modes also require additional load-balancing configuration, depending on the source and destination of traffic being passed through the interface. The **active-backup** bond mode does not require any switch configuration. Other bond modes are not supported.

The following YAML file is an example of a manifest for a bond interface. It includes samples values that you must replace with your own information.

```
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. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. This example uses a **hostname** node selector.
4. Name of the interface.

5. Optional: Human-readable description of the interface.
6. The type of interface. This example creates a bond.
7. The requested state for the interface after creation.
8. Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
9. Enables **ipv4** in this example.
10. The driver mode for the bond. This example uses **active backup**.
11. Optional: This example uses miimon to inspect the bond link every 140ms.
12. The subordinate node NICs in the bond.
13. Optional: The maximum transmission unit (MTU) for the bond. If not specified, this value is set to **1500** by default.

1.8.5. Example: Multiple interfaces in the same node network configuration policy

You can create multiple interfaces in the same node network configuration policy. These interfaces can reference each other, allowing you to build and deploy a network configuration by using a single policy manifest.

IMPORTANT

If multiple interfaces use the same default configuration, a single Network Manager connection profile activates on multiple interfaces simultaneously and this causes connections to have the same universally unique identifier (UUID). To avoid this issue, ensure that each interface has a specific configuration that is different from the default configuration.

The following example YAML file creates a bond that is named **bond10** across two NICs and VLAN that is named **bond10.103** that connects to the bond.

```
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. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. This example uses **hostname** node selector.
4. 11 Name of the interface.
5. 12 Optional: Human-readable description of the interface.
6. 13 The type of interface.
7. 14 The requested state for the interface after creation.
8. The driver mode for the bond.
9. Optional: This example uses miimon to inspect the bond link every 140ms.
10. The subordinate node NICs in the bond.
15. The node NIC to which the VLAN is attached.
16. The VLAN tag.
17. Optional: If you do not use dhcp, you can either set a static IP or leave the interface without an IP address.
18. Enables ipv4 in this example.

1.8.6. Example: Node network configuration policy for virtual functions

You can update host network settings for Single Root I/O Virtualization (SR-IOV) network virtual functions (VF) in an existing cluster by applying a **NodeNetworkConfigurationPolicy** manifest.

You can apply a **NodeNetworkConfigurationPolicy** manifest to an existing cluster to complete the following tasks:

- Configure QoS host network settings for VFs to optimize performance.

- Add, remove, or update VFs for a network interface.
- Manage VF bonding configurations.

NOTE

To update host network settings for SR-IOV VFs by using NMState on physical functions that are also managed through the SR-IOV Network Operator, you must set the **externallyManaged** parameter in the relevant **SriovNetworkNodePolicy** resource to **true**. For more information, see the *Additional resources* section.

The following YAML file is an example of a manifest that defines QoS policies for a VF. This YAML includes samples values that you must replace with your own information.

```
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) Name of the policy.

(2) Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

(3) This example applies to all nodes with the **worker** role.

(4) Name of the physical function (PF) network interface.

(5) Optional: Human-readable description of the interface.

(6) The type of interface.

(7) The requested state for the interface after configuration.

(8) The total number of VFs.

(9) Identifies the VF with an ID of **0**.

(10) Sets a maximum transmission rate, in Mbps, for the VF. This sample value sets a rate of 200 Mbps.

The following YAML file is an example of a manifest that adds a VF for a network interface.

In this sample configuration, the **ens1f1v0** VF is created on the **ens1f1** physical interface, and this VF is added to a bonded network interface **bond0**. The bond uses **active-backup** mode for redundancy. In this example, the VF is configured to use hardware offloading to manage the VLAN directly on the physical interface.

```
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. Name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
3. The example applies to all nodes with the **worker** role.
4. Name of the VF network interface.
5. Number of VFs to create.
6. Setting to allow failover bonding between the active and backup VFs.
7. ID of the VLAN. The example uses hardward offloading to define a VLAN directly on the VF.

8 Name of the bonding network interface.

9 Optional: Human-readable description of the interface.

10 The type of interface.

11 The requested state for the interface after configuration.

12 The bonding policy for the bond.

13 The primary attached bonding port.

14 The ports for the bonded network interface.

15 In this example, the VLAN network interface is added as an additional interface to the bonded network interface.

1.8.7. Example: Network interface with a VRF instance node network configuration policy

You can associate a Virtual Routing and Forwarding (VRF) instance with a network interface by applying a **NodeNetworkConfigurationPolicy** custom resource (CR).

By associating a VRF instance with a network interface, you can support traffic isolation, independent routing decisions, and the logical separation of network resources.

WARNING

When configuring Virtual Route Forwarding (VRF), you must change the VRF value to a table ID lower than **1000** because a value higher than **1000** is reserved for OpenShift Container Platform.

In a bare-metal environment, you can announce load balancer services through interfaces belonging to a VRF instance by using MetalLB. For more information, see the *Additional resources* section.

The following YAML file is an example of associating a VRF instance to a network interface. It includes samples values that you must replace with your own information.

```
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. The name of the policy.
2. This example applies the policy to all nodes with the label **vrf:true**.
3. The name of the interface.
4. The type of interface. This example creates a VRF instance.
5. The node interface to which the VRF attaches.
6. The name of the route table ID for the VRF.

Additional resources

- About virtual routing and forwarding
- Exposing a service through a network VRF

1.9. CREATING AN IP OVER INFINIBAND INTERFACE ON NODES

On the OpenShift Container Platform web console, you can install a Red Hat certified third-party Operator, such as the NVIDIA Network Operator, that supports IP over InfiniBand (IPoIB) mode. Typically, you would use the third-party Operator with other vendor infrastructure to manage resources in an OpenShift Container Platform cluster.

To create an IPoIB interface on nodes in your cluster, you must define an InfiniBand (IPoIB) interface in a **NodeNetworkConfigurationPolicy** (NNCP) manifest file.

If you need to attach IPoIB to a bond interface, only the **active-backup** mode supports this configuration.

IMPORTANT

The OpenShift Container Platform documentation describes defining only the IPoIB interface configuration in a **NodeNetworkConfigurationPolicy** (NNCP) manifest file. You must refer to the NVIDIA and other third-party vendor documentation for the majority of the configuring steps. Red Hat support does not extend to anything external to the NNCP configuration.

For more information about the NVIDIA Operator, see Getting Started with Red Hat OpenShift (NVIDIA Docs Hub).

Prerequisites

- You installed a Red Hat certified third-party Operator that supports an IPoIB interface.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create or edit a **NodeNetworkConfigurationPolicy** (NNCP) manifest file, and then specify an IPoIB interface in the file.

```
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
# ...
```

where:

<mode>

datagram is the default mode for an IPoIB interface. This mode provides improved CPU performance and low-latency capabilitities for pod-to-pod communication. **connected** mode is a supported mode but consider only using this mode when you need to adjust the maximum transmission unit (MTU) value to improve node connectivity with surrounding network devices.

<pkey>

Supports a string or an integer value. The parameter defines the protection key, or *P-key*, for the interface for the purposes of authentication and encrypted communications with a third-party vendor, such as NVIDIA. Values **None** and **0xffff** indicate the protection key for the base interface in an InfiniBand system.

<identifier>

Supported values include **name**, the default value, and **mac-address**. The **name** value applies a configuration to an interface that holds a specified interface name.

<mac-address>

Holds the MAC address of an interface. For an IP-over-InfiniBand (IPoIB) interface, the address is a 20-byte string.

<type>

Sets the type of interface to **infiniband**.

2. Apply the NNCP configuration to each node in your cluster by running the following command. The Kubernetes NMState Operator can then create an IPoIB interface on each node.

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

where:

<nncp_file_name>

Replace **<nncp_file_name>** with the name of your NNCP file.

1.10. EXAMPLE POLICY CONFIGURATIONS THAT USE DYNAMIC MATCHING AND TEMPLATING

The following example configuration snippets show node network policies that use dynamic matching and templating.

IMPORTANT

Applying node network configuration policies that use dynamic matching and templating is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

1.10.1. Example: Linux bridge interface node network configuration policy to inherit static IP address from the NIC attached to the bridge

Create a Linux bridge interface on nodes in the cluster and transfer the static IP configuration of the NIC to the bridge by applying a single **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes sample values that you must replace with your own information.

```
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. The name of the policy.
2. Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster. This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
3. The reference to the node NIC to which the bridge attaches.
4. The type of interface. This example creates a bridge.
5. The IP address of the bridge interface. This value matches the IP address of the NIC which is referenced by the **spec.capture.eth1-nic** entry.
6. The node NIC to which the bridge attaches.

1.10.2. Example: Node network configuration policy to enable LLDP reporting

The following YAML file is an example of a **NodeNetworkConfigurationPolicy** manifest that enables the Link Layer Discovery Protocol (LLDP) listener for all ethernet ports in your OpenShift Container Platform cluster.

Devices on a local area network can use LLDP to advertise their identity, capabilities, and neighbor information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: enable-lldp-ethernets-up 1
spec:
  capture:
    ethernets: interfaces.type=="ethernet"
    ethernets-up: capture.ethernets | interfaces.state=="up"
    ethernets-lldp: capture.ethernets-up | interfaces.lldp.enabled:=true 2
  desiredState:
    interfaces: "{{ capture.ethernets-lldp.interfaces }}"
# ...
```

1. Specifies the name of the node network configuration policy.
2. Specifies that LLDP is enabled for all ethernet ports that have the interface state set to **up**.

Additional resources

- The NMPolicy project - Policy syntax

1.11. EXAMPLES: IP MANAGEMENT

The following example configuration snippets show different methods of IP management.

These examples use the **ethernet** interface type to simplify the example while showing the related context in the policy configuration. These IP management examples can be used with the other interface types.

1.11.1. Static

The following snippet statically configures an IP address on the Ethernet interface:

```
# ...
    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 Replace this value with the static IP address for the interface.

1.11.2. No IP address

The following snippet ensures that the interface has no IP address:

```
# ...
    interfaces:
    - name: eth1
      description: No IP on eth1
      type: ethernet
      state: up
      ipv4:
        enabled: false
# ...
```

IMPORTANT

Always set the **state** parameter to **up** when you set both the **ipv4.enabled** and the **ipv6.enabled** parameter to **false** to disable an interface. If you set **state: down** with this configuration, the interface receives a DHCP IP address because of automatic DHCP assignment.

1.11.3. Dynamic host configuration

The following snippet configures an Ethernet interface that uses a dynamic IP address, gateway address, and DNS:

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

The following snippet configures an Ethernet interface that uses a dynamic IP address but does not use a dynamic gateway address or DNS:

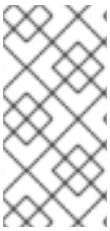
```
# ...
    interfaces:
    - name: eth1
      description: DHCP without gateway or DNS on eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: true
        auto-gateway: false
        auto-dns: false
        enabled: true
# ...
```

1.11.4. Media Access Control (MAC) address

You can use a MAC address to identify a network interface instead of using the name of the network interface. A network interface name can change for various reasons, such as an operating system configuration change. However, every network interface has a unique MAC address that does not change. This means that using a MAC address is a more permanent way to identify a specific network interface.

Supported values for the **identifier** parameter include the default **name** value and the value **mac-address**. The **name** value applies a configuration to an interface that holds a specified interface name.

Using a **mac-address** value for the **identifier** parameter indicates that a MAC address is the identifier for the network interface. If you set the **identifier** value to **mac-address**, you must enter a specific MAC address in the following **mac-address** parameter field.



NOTE

You can still specify a value for the **name** parameter, but setting the **identifier: mac-address** value means that a MAC address is used as the primary identifier for a network interface. If you specify an incorrect MAC address, **nmstate** reports an invalid argument error.

The following snippet specifies a MAC address as the primary identifier for an Ethernet device, named **eth1**, with a MAC address of **8A:8C:92:1A:F6:98**:

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

1.11.5. DNS

By default, the **nmstate** API stores DNS values globally as against storing them in a network interface. For certain situations, you must configure a network interface to store DNS values.

TIP

Setting a DNS configuration is comparable to modifying the **/etc/resolv.conf** file.

To define a DNS configuration for a network interface, you must initially specify the **dns-resolver** section in the network interface's YAML configuration file. To apply an NNCP configuration to your network interface, you need to run the **oc apply -f <nncp_file_name>** command.

The following example shows a default situation that stores DNS values globally:

- Configure a static DNS without a network interface. Note that when updating the **/etc/resolv.conf** file on a host node, you do not need to specify an interface, IPv4 or IPv6, in the **NodeNetworkConfigurationPolicy** (NNCP) manifest.
  Example of a DNS configuration for a network interface that globally stores DNS values:

```
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
# ...
```

IMPORTANT

You can specify DNS options under the **dns-resolver.config** section of your NNCP file as demonstrated in the following example:

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

If you want to remove the DNS options from your network interface, apply the following configuration to your NNCP and then run the **oc apply -f <nncp_file_name>** command:

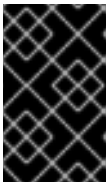
```
# ...
    dns-resolver:
      config: {}
    interfaces: []
# ...
```

The following examples show situations that require configuring a network interface to store DNS values:

- If you want to rank a static DNS name server over a dynamic DNS name server, define the interface that runs either the Dynamic Host Configuration Protocol (DHCP) or the IPv6 Autoconfiguration (**autoconf**) mechanism in the network interface YAML configuration file. Example configuration that adds **192.0.2.1** to DNS name servers retrieved from the DHCPv4 network protocol:

```
# ...
dns-resolver:
  config:
    server:
    - 192.0.2.1
interfaces:
  - name: eth1
    type: ethernet
    state: up
    ipv4:
      enabled: true
      dhcp: true
      auto-dns: true
# ...
```

- If you need to configure a network interface to store DNS values instead of adopting the default method, which uses the **nmstate** API to store DNS values globally, you can set static DNS values and static IP addresses in the network interface YAML file.



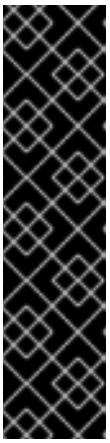
IMPORTANT

Storing DNS values at the network interface level might cause name resolution issues after you attach the interface to network components, such as an Open vSwitch (OVS) bridge, a Linux bridge, or a bond.

Example configuration that stores DNS values at the interface level:

```
# ...
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
# ...
```

- If you want to set static DNS search domains and static DNS name servers for your network interface, define the static interface that runs either the Dynamic Host Configuration Protocol (DHCP) or the IPv6 Autoconfiguration (**autoconf**) mechanism in the network interface YAML configuration file.



IMPORTANT

Specifying the following **dns-resolver** configurations in the network interface YAML file might cause a race condition at reboot that prevents the **NodeNetworkConfigurationPolicy** (NNCP) from applying to pods that run in your cluster:

  - Setting static DNS search domains and dynamic DNS name servers for your network interface.
  - Specifying domain suffixes for the **search** parameter and not setting IP addresses for the **server** parameter.

Example configuration that sets **example.com** and **example.org** static DNS search domains along with static DNS name server settings:

```
# ...
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
# ...
```

1.12. ROUTES AND ROUTE RULES

After you configure an IP address for a network interface, you can configure routes and route rules in the NMState configuration for cluster nodes.

IMPORTANT

You cannot use the OVN-Kubernetes **br-ex** bridge as the next hop interface when configuring a static route unless you manually configured a customized **br-ex** bridge.

For more information, see "Creating a manifest object that includes a customized br-ex bridge" in the *Deploying installer-provisioned clusters on bare metal* document or the *Installing a user-provisioned cluster on bare metal* document.

The **routes** parameter defines static routes and these routes determine the traffic that leaves the network interfaces and the destination network for the traffic. Supported values include **running** and **config**.

NOTE

After you apply an NMState configuration to cluster nodes and you want to change existing routes, you must specify the old route with the **state: absent** parameter and the new route with the **state: present** parameter. The NMState Operator can then delete the old route and apply the new route to cluster nodes.

Setting the **state** parameter to **ignore** means that the Operator ignores certain routes.

The **route-rules** parameter implements a policy-based routing capability for cluster nodes. This capability allows traffic that originates from a different source IP address to be segregated and routed through different gateways and network paths.

The following YAML configuration shows a static route and a static IP confiuration on interface **eth1**:

```
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**: Applies a rule to any network packet that originates from the specified IP address.
- **config.priority**: Sets the priority order for the rule.
- **config.route-table**: Specifies the routing table that the Operator uses to check that network traffic matches the **ip-from** condition.
- **address.ip**: The static IP address for the Ethernet interface.
- **config.next-hop-address**: The next hop address for the node traffic. This must be in the same subnet as the IP address set for the Ethernet interface.

Additional resources

- Creating a manifest object that includes a customized br-ex bridge (Installer-provisioned infrastructure)
- Creating a manifest object that includes a customized br-ex bridge (User-provisioned infrastructure)

- Routes (nmstate documentation)
- Route Rules (nmstate documentation)

CHAPTER 2. TROUBLESHOOTING NODE NETWORK CONFIGURATION

If the node network configuration encounters an issue, the policy is automatically rolled back and the enactments report failure. This includes issues such as:

- The configuration fails to be applied on the host.
- The host loses connection to the default gateway.
- The host loses connection to the API server.

2.1. TROUBLESHOOTING AN INCORRECT NODE NETWORK CONFIGURATION POLICY CONFIGURATION

You can apply changes to the node network configuration across your entire cluster by applying a node network configuration policy. If you applied an incorrect configuration, you can use the following example to troubleshoot and correct the failed node network policy.

The example attempts to apply a Linux bridge policy to a cluster that has three control plane nodes and three compute nodes. The policy is not applied because the policy references the wrong interface.

To find an error, you need to investigate the available NMState resources. You can then update the policy with the correct configuration.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You ensured that an **ens01** interface does not exist on your Linux system.

Procedure

1. Create a policy on your cluster. The following example creates a simple bridge, **br1** that has **ens01** as its member:

```
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. Apply the policy to your network interface:

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

Example output:

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

3. Verify the status of the policy by running the following command:

```
$ oc get nncp
```

The output shows that the policy failed:

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

The policy status alone does not indicate if it failed on all nodes or a subset of nodes.

4. List the node network configuration enactments to see if the policy was successful on any of the nodes. If the policy failed for only a subset of nodes, the output suggests that the problem is with a specific node configuration. If the policy failed on all nodes, the output suggests that the problem is with the policy.

```
$ oc get nnce
```

The output shows that the policy failed on all nodes:

```
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. View one of the failed enactments. The following command uses the output tool **jsonpath** to filter the output:

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

Example output:

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

The previous example shows the output from an **InvalidArgument** error that indicates that the **ens01** is an unknown port. For this example, you might need to change the port configuration in the policy configuration file.

6. To ensure that the policy is configured properly, view the network configuration for one or all of the nodes by requesting the **NodeNetworkState** object. The following command returns the network configuration for the **control-plane-1** node:

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

The output shows that the interface name on the nodes is **ens1** but the failed policy incorrectly uses **ens01**:

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

7. Correct the error by editing the existing policy:

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

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

Save the policy to apply the correction.

8. Check the status of the policy to ensure it updated successfully:

```
$ oc get nncp
```

Example output:

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

The updated policy is successfully configured on all nodes in the cluster.

2.2. TROUBLESHOOTING DNS CONNECTIVITY ISSUES IN A DISCONNECTED ENVIRONMENT

If you experience health check probe issues when configuring **nmstate** in a disconnected environment, you can configure the DNS server to resolve the custom domain name instead of the default **root-servers.net** domain.

IMPORTANT

Ensure that the DNS server includes a name server (NS) entry for the **root-servers.net** zone. The DNS server does not need to forward a query to an upstream resolver, but the server must return a correct answer for the NS query.

2.2.1. Configuring the bind9 DNS named server

For a cluster configured to query a **bind9** DNS server, you can add the **root-servers.net** zone to a configuration file that contains at least one DNS record. For example you can use the **/var/named/named.localhost** as a zone file that already matches this criteria.

Procedure

1. Add the **root-servers.net** zone at the end of the **/etc/named.conf** configuration file by running the following command:

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

2. Restart the **named** service by running the following command:

```
$ systemctl restart named
```

3. Confirm that the **root-servers.net** zone is present by running the following command:

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

Example output

```
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. Verify that the DNS server can resolve the NS record for the **root-servers.net** domain by running the following command:

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

Example output

```
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. Configuring the dnsmasq DNS server

If you are using **dnsmasq** as the DNS server, you can delegate resolution of the **root-servers.net** domain to another DNS server, for example, by creating a new configuration file that resolves **root-servers.net** using a DNS server that you specify.

Procedure

1. Create a configuration file that delegates the domain **root-servers.net** to another DNS server by running the following command:

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

2. Restart the **dnsmasq** service by running the following command:

```
$ systemctl restart dnsmasq
```

3. Confirm that the **root-servers.net** domain is delegated to another DNS server by running the following command:

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

Example output

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

4. Verify that the DNS server can resolve the NS record for the **root-servers.net** domain by running the following command:

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

Example output

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

2.2.3. Creating a custom DNS host name to resolve DNS connectivity issues

In a disconnected environment where the external DNS server cannot be reached, you can resolve Kubernetes NMState Operator health probe issues by specifying a custom DNS host name in the **NMState** custom resource definition (CRD).

Procedure

1. Add the DNS host name configuration to the **NMState** CRD of your cluster:

```
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
spec:
  probeConfiguration:
    dns:
      host: redhat.com
# ...
```

2. Apply the DNS host name configuration to your cluster network by running the following command. Ensure that you replace **<filename>** with the name of your CRD file.

```
$ oc apply -f <filename>.yaml
```