OpenShift Container Platform 4.20

Configuring network settings

General networking configuration processes in OpenShift Container Platform

Last Updated: 2026-01-21

OpenShift Container Platform 4.20 Configuring network settings

General networking configuration processes in OpenShift Container Platform

Legal Notice

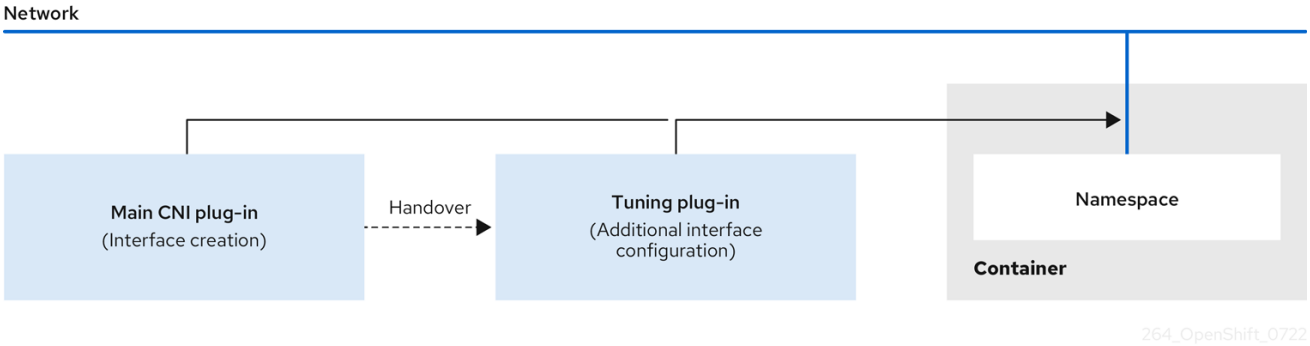
Abstract

This document covers configuring networking aspects such as node port services, IP address ranges, IP failover, and cluster-wide proxies in OpenShift Container Platform.

Table of Contents

CHAPTER 1. CONFIGURING SYSTEM CONTROLS AND INTERFACE ATTRIBUTES USING THE TUNING PLUGIN

To modify kernel parameters and interface attributes at runtime in OpenShift Container Platform, you can use the tuning Container Network Interface (CNI) meta plugin. The plugin operates in a chain with a main CNI plugin and allows you to change sysctls and interface attributes such as promiscuous mode, all-multicast mode, MTU, and MAC address.



1.1. CONFIGURING SYSTEM CONTROLS BY USING THE TUNING CNI

To configure interface-level network sysctls in OpenShift Container Platform, you can use the tuning CNI meta plugin in a network attachment definition. Configure the **net.ipv4.conf.IFNAME.accept_redirects** sysctl to enable accepting and sending ICMP-redirected packets.

Procedure

1. Create a network attachment definition, such as **tuning-example.yaml**, with the following content:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name>
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "<name>",
    "plugins": [{
       "type": "<main_CNI_plugin>"
      },
      {
       "type": "tuning",
       "sysctl": {
            "net.ipv4.conf.IFNAME.accept_redirects": "1"
        }
      }
     ]
}
```

where:

name

Specifies the name for the additional network attachment to create. The name must be unique within the specified namespace.

namespace

Specifies the namespace that the object is associated with.

cniVersion

Specifies the CNI specification version.

name

Specifies the name for the configuration. It is recommended to match the configuration name to the name value of the network attachment definition.

main_CNI_plugin

Specifies the name of the main CNI plugin to configure.

tuning

Specifies the name of the CNI meta plugin.

sysctl

Specifies the sysctl to set. The interface name is represented by the **IFNAME** token and is replaced with the actual name of the interface at runtime.

Example network attachment definition

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
      },
      {
      "type": "tuning",
      "sysctl": {
         "net.ipv4.conf.IFNAME.accept_redirects": "1"
       }
    }
  ]
}'
```

2. Apply the YAML by running the following command:

```
$ oc apply -f tuning-example.yaml
```

Example output

```
networkattachmentdefinition.k8.cni.cncf.io/tuningnad created
```

3. Create a pod such as **examplepod.yaml** with the network attachment definition similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

where:

k8s.v1.cni.cncf.io/networks

Specifies the name of the configured **NetworkAttachmentDefinition**.

runAsUser

Specifies which user ID the container is run with.

runAsGroup

Specifies which primary group ID the containers is run with.

allowPrivilegeEscalation

Specifies if a pod can request to allow privilege escalation. If unspecified, it defaults to true. This boolean directly controls whether the **no_new_privs** flag gets set on the container process.

capabilities

Specifies privileged actions without giving full root access. This policy ensures all capabilities are dropped from the pod.

runAsNonRoot: true

Specifies that the container will run with a user with any UID other than 0.

seccompProfile

Specifies the default seccomp profile for a pod or container workload.

4. Apply the yaml by running the following command:

```
$ oc apply -f examplepod.yaml
```

5. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

6. Log in to the pod by running the following command:

```
$ oc rsh tunepod
```

7. Verify the values of the configured sysctl flags. For example, find the value **net.ipv4.conf.net1.accept_redirects** by running the following command:

```
sh-4.4# sysctl net.ipv4.conf.net1.accept_redirects
```

Expected output

```
net.ipv4.conf.net1.accept_redirects = 1
```

1.2. ENABLING ALL-MULTICAST MODE BY USING THE TUNING CNI

To enable all-multicast mode on network interfaces in OpenShift Container Platform, you can use the tuning Container Network Interface (CNI) meta plugin in a network attachment definition. When enabled, the interface receives all multicast packets on the network.

Procedure

1. Create a network attachment definition, such as **tuning-example.yaml**, with the following content:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name>
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "<name>",
    "plugins": [{
       "type": "<main_CNI_plugin>"
      },
      {
       "type": "tuning",
       "allmulti": true
       }
      }
     ]
}
```

where:

name

Specifies the name for the additional network attachment to create. The name must be unique within the specified namespace.

namespace

Specifies the namespace that the object is associated with.

cniVersion

Specifies the CNI specification version.

name

Specifies the name for the configuration. Match the configuration name to the name value of the network attachment definition.

main_CNI_plugin

Specifies the name of the main CNI plugin to configure.

tuning

Specifies the name of the CNI meta plugin.

allmulti

Specifies the all-multicast mode of interface. If enabled, all multicast packets on the network will be received by the interface.

Example network attachment definition

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: setallmulti
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "setallmulti",
    "plugins": [
      {
        "type": "bridge"
      },
      {
        "type": "tuning",
        "allmulti": true
      }
    ]
  }'
```

2. Apply the settings specified in the YAML file by running the following command:

```
$ oc apply -f tuning-allmulti.yaml
```

Example output

```
networkattachmentdefinition.k8s.cni.cncf.io/setallmulti created
```

3. Create a pod with a network attachment definition similar to that specified in the following **examplepod.yaml** sample file:

```
apiVersion: v1
kind: Pod
metadata:
  name: allmultipod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: setallmulti
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

where:

k8s.v1.cni.cncf.io/networks

Specifies the name of the configured **NetworkAttachmentDefinition**.

runAsUser

Specifies which user ID the container is run with.

runAsGroup

Specifies which primary group ID the containers is run with.

allowPrivilegeEscalation

Specifies if a pod can request to allow privilege escalation. If unspecified, it defaults to true. This boolean directly controls whether the **no_new_privs** flag gets set on the container process.

capabilities

Specifies privileged actions without giving full root access. This policy ensures all capabilities are dropped from the pod.

runAsNonRoot: true

Specifies that the container will run with a user with any UID other than 0.

seccompProfile

Specifies the default seccomp profile for a pod or container workload.

4. Apply the settings specified in the YAML file by running the following command:

```
$ oc apply -f examplepod.yaml
```

5. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME         READY   STATUS    RESTARTS   AGE
allmultipod  1/1     Running   0          23s
```

6. Log in to the pod by running the following command:

```
$ oc rsh allmultipod
```

7. List all the interfaces associated with the pod by running the following command:

```
sh-4.4# ip link
```

Example output

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue state
UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

where:

eth0@if22

Specifies the primary interface.

net1@if24

Specifies the secondary interface configured with the network-attachment-definition that supports the all-multicast mode (ALLMULTI flag).

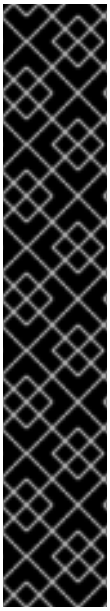
1.3. ADDITIONAL RESOURCES

- Using sysctls in containers
- SR-IOV network node configuration object
- Configuring interface-level network sysctl settings and all-multicast mode for SR-IOV networks

CHAPTER 2. CONFIGURING THE NODE PORT SERVICE RANGE

During cluster installation, you can configure the node port range to meet the requirements of your cluster. After cluster installation, only a cluster administrator can expand the range as a postinstallation task. If your cluster uses a large number of node ports, consider increasing the available port range according to the requirements of your cluster.

If you do not set a node port range during cluster installation, the default range of **30000-32768** applies to your cluster. In this situation, you can expand the range on either side, but you must preserve **30000-32768** within your new port range.



IMPORTANT

Red Hat has not performed testing outside the default port range of **30000-32768**. For ranges outside the default port range, ensure that you test to verify the expanding node port range does not impact your cluster. In particular, ensure that there is:

- No overlap with any ports already in use by host processes
- No overlap with any ports already in use by pods that are configured with host networking

If you expanded the range and a port allocation issue occurs, create a new cluster and set the required range for it.

If you expand the node port range and OpenShift CLI (**oc**) stops working because of a port conflict with the OpenShift Container Platform API server, you must create a new cluster.

2.1. EXPANDING THE NODE PORT RANGE

You can expand the node port range for your cluster. After you install your OpenShift Container Platform cluster, you cannot shrink the node port range on either side of the currently configured range.

IMPORTANT

Red Hat has not performed testing outside the default port range of **30000-32768**. For ranges outside the default port range, ensure that you test to verify that expanding your node port range does not impact your cluster. If you expanded the range and a port allocation issue occurs, create a new cluster and set the required range for it.

Prerequisites

- Installed the OpenShift CLI (**oc**).
- Logged in to the cluster as a user with **cluster-admin** privileges.
- You ensured that your cluster infrastructure allows access to the ports that exist in the extended range. For example, if you expand the node port range to **30000-32900**, your firewall or packet filtering configuration must allow the inclusive port range of **30000-32900**.

Procedure

- To expand the range for the **serviceNodePortRange** parameter in the **network.config.openshift.io** object that your cluster uses to manage traffic for pods, enter the following command:

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
  '{
    "spec":
      { "serviceNodePortRange": "<port_range>" }
  }'
```

where:

<port_range>

specifies your expanded range, such as **30000-32900**.

TIP

You can also apply the following YAML to update the node port range:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "<port_range>"
# ...
```

Example output

```
network.config.openshift.io/cluster patched
```

Verification

- To confirm that the updated configuration is active, enter the following command. The update can take several minutes to apply.

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo '"service-node-port-range":["[[:digit:]]+-[[:digit:]]+"]'
```

Example output

```
"service-node-port-range":["30000-32900"]
```

2.2. ADDITIONAL RESOURCES

- Configuring ingress cluster traffic using a NodePort
- Network [config.openshift.io/v1]
- Service [core/v1]

CHAPTER 3. CONFIGURING THE CLUSTER NETWORK RANGE

As a cluster administrator, you can expand the cluster network range after cluster installation. You might want to expand the cluster network range if you need more IP addresses for additional nodes.

For example, if you deployed a cluster and specified **10.128.0.0/19** as the cluster network range and a host prefix of **23**, you are limited to 16 nodes. You can expand that to 510 nodes by changing the CIDR mask on a cluster to /**14**.

When expanding the cluster network address range, your cluster must use the OVN-Kubernetes network plugin. Other network plugins are not supported.

The following limitations apply when modifying the cluster network IP address range:

- The CIDR mask size specified must always be smaller than the currently configured CIDR mask size, because you can only increase IP space by adding more nodes to an installed cluster
- The host prefix cannot be modified
- Pods that are configured with an overridden default gateway must be recreated after the cluster network expands

3.1. EXPANDING THE CLUSTER NETWORK IP ADDRESS RANGE

You can expand the IP address range for the cluster network. Because this change requires rolling out a new Operator configuration across the cluster, it can take up to 30 minutes to take effect.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.
- Ensure that the cluster uses the OVN-Kubernetes network plugin.

Procedure

1. To obtain the cluster network range and host prefix for your cluster, enter the following command:

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

Example output

```
[{"cidr":"10.217.0.0/22","hostPrefix":23}]
```

2. To expand the cluster network IP address range, enter the following command. Use the CIDR IP address range and host prefix returned from the output of the previous command.

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
    "spec":{
      "clusterNetwork": [ {"cidr":"<network>/<cidr>","hostPrefix":<prefix>} ],
```

```
    "networkType": "OVNKubernetes"
  }
}'
```

where:

<network>

Specifies the network part of the **cidr** field that you obtained from the previous step. You cannot change this value.

<cidr>

Specifies the network prefix length. For example, **14**. Change this value to a smaller number than the value from the output in the previous step to expand the cluster network range.

<prefix>

Specifies the current host prefix for your cluster. This value must be the same value for the **hostPrefix** field that you obtained from the previous step.

Example command

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
 '{
   "spec":{
     "clusterNetwork": [ {"cidr":"10.217.0.0/14","hostPrefix": 23} ],
     "networkType": "OVNKubernetes"
   }
 }'
```

Example output

```
network.config.openshift.io/cluster patched
```

3. To confirm that the configuration is active, enter the following command. It can take up to 30 minutes for this change to take effect.

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

Example output

```
[{"cidr":"10.217.0.0/14","hostPrefix":23}]
```

3.2. ADDITIONAL RESOURCES

- Red Hat OpenShift Network Calculator
- About the OVN-Kubernetes network plugin

CHAPTER 4. CONFIGURING IP FAILOVER

This topic describes configuring IP failover for pods and services on your OpenShift Container Platform cluster.

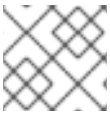
IP failover uses Keepalived to host a set of externally accessible Virtual IP (VIP) addresses on a set of hosts. Each VIP address is only serviced by a single host at a time. Keepalived uses the Virtual Router Redundancy Protocol (VRRP) to determine which host, from the set of hosts, services which VIP. If a host becomes unavailable, or if the service that Keepalived is watching does not respond, the VIP is switched to another host from the set. This means a VIP is always serviced as long as a host is available.

Every VIP in the set is serviced by a node selected from the set. If a single node is available, the VIPs are served. There is no way to explicitly distribute the VIPs over the nodes, so there can be nodes with no VIPs and other nodes with many VIPs. If there is only one node, all VIPs are on it.

The administrator must ensure that all of the VIP addresses meet the following requirements:

- Accessible on the configured hosts from outside the cluster.
- Not used for any other purpose within the cluster.

Keepalived on each node determines whether the needed service is running. If it is, VIPs are supported and Keepalived participates in the negotiation to determine which node serves the VIP. For a node to participate, the service must be listening on the watch port on a VIP or the check must be disabled.



NOTE

Each VIP in the set might be served by a different node.

IP failover monitors a port on each VIP to determine whether the port is reachable on the node. If the port is not reachable, the VIP is not assigned to the node. If the port is set to **0**, this check is suppressed. The check script does the needed testing.

When a node running Keepalived passes the check script, the VIP on that node can enter the **master** state based on its priority and the priority of the current master and as determined by the preemption strategy.

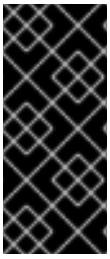
A cluster administrator can provide a script through the **OPENSHIFT_HA_NOTIFY_SCRIPT** variable, and this script is called whenever the state of the VIP on the node changes. Keepalived uses the **master** state when it is servicing the VIP, the **backup** state when another node is servicing the VIP, or in the **fault** state when the check script fails. The notify script is called with the new state whenever the state changes.

You can create an IP failover deployment configuration on OpenShift Container Platform. The IP failover deployment configuration specifies the set of VIP addresses, and the set of nodes on which to service them. A cluster can have multiple IP failover deployment configurations, with each managing its own set of unique VIP addresses. Each node in the IP failover configuration runs an IP failover pod, and this pod runs Keepalived.

When using VIPs to access a pod with host networking, the application pod runs on all nodes that are running the IP failover pods. This enables any of the IP failover nodes to become the master and service the VIPs when needed. If application pods are not running on all nodes with IP failover, either some IP failover nodes never service the VIPs or some application pods never receive any traffic. Use the same selector and replication count, for both IP failover and the application pods, to avoid this mismatch.

While using VIPs to access a service, any of the nodes can be in the IP failover set of nodes, since the service is reachable on all nodes, no matter where the application pod is running. Any of the IP failover nodes can become master at any time. The service can either use external IPs and a service port or it can use a **NodePort**. Setting up a **NodePort** is a privileged operation.

When using external IPs in the service definition, the VIPs are set to the external IPs, and the IP failover monitoring port is set to the service port. When using a node port, the port is open on every node in the cluster, and the service load-balances traffic from whatever node currently services the VIP. In this case, the IP failover monitoring port is set to the **NodePort** in the service definition.



IMPORTANT

Even though a service VIP is highly available, performance can still be affected. Keepalived makes sure that each of the VIPs is serviced by some node in the configuration, and several VIPs can end up on the same node even when other nodes have none. Strategies that externally load-balance across a set of VIPs can be thwarted when IP failover puts multiple VIPs on the same node.

When you use **ExternalIP**, you can set up IP failover to have the same VIP range as the **ExternalIP** range. You can also disable the monitoring port. In this case, all of the VIPs appear on same node in the cluster. Any user can set up a service with an **ExternalIP** and make it highly available.

IMPORTANT

There are a maximum of 254 VIPs in the cluster.

4.1. IP FAILOVER ENVIRONMENT VARIABLES

The following table contains the variables used to configure IP failover.

Table 4.1. IP failover environment variables

Variable Name	Default	Description
OPENSHIFT_HA_MONITOR_PORT	**80**	The IP failover pod tries to open a TCP connection to this port on each Virtual IP (VIP). If connection is established, the service is considered to be running. If this port is set to **0**, the test always passes.
OPENSHIFT_HA_NETWORK_INTERFACE		The interface name that IP failover uses to send Virtual Router Redundancy Protocol (VRRP) traffic. The default value is **eth0**. If your cluster uses the OVN-Kubernetes network plugin, set this value to **br-ex** to avoid packet loss. For a cluster that uses the OVN-Kubernetes network plugin, all listening interfaces do not serve VRRP but instead expect inbound traffic over a **br-ex** bridge.
OPENSHIFT_HA_REPLICA_COUNT	**2**	The number of replicas to create. This must match **spec.replicas** value in IP failover deployment configuration.

Variable Name	Default	Description
OPENSHIFT_HA_VIRTUAL_IPS		The list of IP address ranges to replicate. This must be provided. For example, **1.2.3.4-6,1.2.3.9**.
OPENSHIFT_HA_VRRP_ID_OFFSET	**10**	The offset value used to set the virtual router IDs. Using different offset values allows multiple IP failover configurations to exist within the same cluster. The default offset is **10**, and the allowed range is **0** through **255**.
OPENSHIFT_HA_VIP_GROUPS		The number of groups to create for VRRP. If not set, a group is created for each virtual IP range specified with the **OPENSHIFT_HA_VIP_GROUPS** variable.
OPENSHIFT_HA_IPTABLES_CHAIN	INPUT	The name of the iptables chain, to automatically add an **iptables** rule to allow the VRRP traffic on. If the value is not set, an **iptables** rule is not added. If the chain does not exist, it is not created.
OPENSHIFT_HA_CHECK_SCRIPT		The full path name in the pod file system of a script that is periodically run to verify the application is operating.
OPENSHIFT_HA_CHECK_INTERVAL	**2**	The period, in seconds, that the check script is run.
OPENSHIFT_HA_NOTIFY_SCRIPT		The full path name in the pod file system of a script that is run whenever the state changes.
OPENSHIFT_HA_PREEMPTION	**preempt_nodelay 300**	The strategy for handling a new higher priority host. The **nopreempt** strategy does not move master from the lower priority host to the higher priority host.

4.2. CONFIGURING IP FAILOVER IN YOUR CLUSTER

As a cluster administrator, you can configure IP failover on an entire cluster, or on a subset of nodes, as defined by the label selector. You can also configure multiple IP failover deployments in your cluster, where each one is independent of the others.

The IP failover deployment ensures that a failover pod runs on each of the nodes matching the constraints or the label used.

This pod runs Keepalived, which can monitor an endpoint and use Virtual Router Redundancy Protocol (VRRP) to fail over the virtual IP (VIP) from one node to another if the first node cannot reach the service or endpoint.

For production use, set a **selector** that selects at least two nodes, and set **replicas** equal to the number of selected nodes.

Prerequisites

- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You created a pull secret.
- Red Hat OpenStack Platform (RHOSP) only:
  - You installed an RHOSP client (RHCOS documentation) on the target environment.
  - You also downloaded the RHOSP **openrc.sh** rc file (RHCOS documentation).

Procedure

1. Create an IP failover service account:

```
$ oc create sa ipfailover
```

2. Update security context constraints (SCC) for **hostNetwork**:

```
$ oc adm policy add-scc-to-user privileged -z ipfailover
```

```
$ oc adm policy add-scc-to-user hostnetwork -z ipfailover
```

3. Red Hat OpenStack Platform (RHOSP) only: Complete the following steps to make a failover VIP address reachable on RHOSP ports.

   a. Use the RHOSP CLI to show the default RHOSP API and VIP addresses in the **allowed_address_pairs** parameter of your RHOSP cluster:

```
$ openstack port show <cluster_name> -c allowed_address_pairs
```

   Output example

```
*Field*                 *Value*
allowed_address_pairs    ip_address='192.168.0.5', mac_address='fa:16:3e:31:f9:cb'
                         ip_address='192.168.0.7', mac_address='fa:16:3e:31:f9:cb'
```

   b. Set a different VIP address for the IP failover deployment and make the address reachable on RHOSP ports by entering the following command in the RHOSP CLI. Do not set any default RHOSP API and VIP addresses as the failover VIP address for the IP failover deployment.

   Example of adding the 1.1.1.1 failover IP address as an allowed address on RHOSP ports.

```
$ openstack port set <cluster_name> --allowed-address ip-address=1.1.1.1,mac-address=fa:fa:16:3e:31:f9:cb
```

   c. Create a deployment YAML file to configure IP failover for your deployment. See "Example deployment YAML for IP failover configuration" in a later step.

d. Specify the following specification in the IP failover deployment so that you pass the failover VIP address to the **OPENSHIFT_HA_VIRTUAL_IPS** environment variable:

Example of adding the 1.1.1.1 VIP address to OPENSHIFT_HA_VIRTUAL_IPS

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ipfailover-keepalived
# ...
      spec:
        env:
        - name: OPENSHIFT_HA_VIRTUAL_IPS
          value: "1.1.1.1"
# ...
```

4. Create a deployment YAML file to configure IP failover.

NOTE

For Red Hat OpenStack Platform (RHOSP), you do not need to re-create the deployment YAML file. You already created this file as part of the earlier instructions.

Example deployment YAML for IP failover configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ipfailover-keepalived (1)
  labels:
    ipfailover: hello-openshift
spec:
  strategy:
    type: Recreate
  replicas: 2
  selector:
    matchLabels:
      ipfailover: hello-openshift
  template:
    metadata:
      labels:
        ipfailover: hello-openshift
    spec:
      serviceAccountName: ipfailover
      privileged: true
      hostNetwork: true
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
      - name: openshift-ipfailover
        image: registry.redhat.io/openshift4/ose-keepalived-ipfailover-rhel9:v4.20
        ports:
        - containerPort: 63000
```

```
      hostPort: 63000
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: lib-modules
      mountPath: /lib/modules
      readOnly: true
    - name: host-slash
      mountPath: /host
      readOnly: true
      mountPropagation: HostToContainer
    - name: etc-sysconfig
      mountPath: /etc/sysconfig
      readOnly: true
    - name: config-volume
      mountPath: /etc/keepalive
    env:
    - name: OPENSHIFT_HA_CONFIG_NAME
      value: "ipfailover"
    - name: OPENSHIFT_HA_VIRTUAL_IPS 2
      value: "1.1.1.1-2"
    - name: OPENSHIFT_HA_VIP_GROUPS 3
      value: "10"
    - name: OPENSHIFT_HA_NETWORK_INTERFACE 4
      value: "ens3" #The host interface to assign the VIPs
    - name: OPENSHIFT_HA_MONITOR_PORT 5
      value: "30060"
    - name: OPENSHIFT_HA_VRRP_ID_OFFSET 6
      value: "10"
    - name: OPENSHIFT_HA_REPLICA_COUNT 7
      value: "2" #Must match the number of replicas in the deployment
    - name: OPENSHIFT_HA_USE_UNICAST
      value: "false"
    #- name: OPENSHIFT_HA_UNICAST_PEERS
      #value: "10.0.148.40,10.0.160.234,10.0.199.110"
    - name: OPENSHIFT_HA_IPTABLES_CHAIN 8
      value: "INPUT"
    #- name: OPENSHIFT_HA_NOTIFY_SCRIPT 9
    #  value: /etc/keepalive/mynotifyscript.sh
    - name: OPENSHIFT_HA_CHECK_SCRIPT 10
      value: "/etc/keepalive/mycheckscript.sh"
    - name: OPENSHIFT_HA_PREEMPTION 11
      value: "preempt_delay 300"
    - name: OPENSHIFT_HA_CHECK_INTERVAL 12
      value: "2"
    livenessProbe:
      initialDelaySeconds: 10
      exec:
        command:
        - pgrep
        - keepalived
  volumes:
  - name: lib-modules
    hostPath:
```

```
      path: /lib/modules
  - name: host-slash
    hostPath:
      path: /
  - name: etc-sysconfig
    hostPath:
      path: /etc/sysconfig
  # config-volume contains the check script
  # created with `oc create configmap keepalived-checkscript --from-file=mycheckscript.sh`
  - configMap:
      defaultMode: 0755
      name: keepalived-checkscript
    name: config-volume
  imagePullSecrets:
    - name: openshift-pull-secret 13
```

1. The name of the IP failover deployment.
2. The list of IP address ranges to replicate. This must be provided. For example, **1.2.3.4-6,1.2.3.9**.
3. The number of groups to create for VRRP. If not set, a group is created for each virtual IP range specified with the **OPENSHIFT_HA_VIP_GROUPS** variable.
4. The interface name that IP failover uses to send VRRP traffic. By default, **eth0** is used.
5. The IP failover pod tries to open a TCP connection to this port on each VIP. If connection is established, the service is considered to be running. If this port is set to **0**, the test always passes. The default value is **80**.
6. The offset value used to set the virtual router IDs. Using different offset values allows multiple IP failover configurations to exist within the same cluster. The default offset is **10**, and the allowed range is **0** through **255**.
7. The number of replicas to create. This must match **spec.replicas** value in IP failover deployment configuration. The default value is **2**.
8. The name of the **iptables** chain to automatically add an **iptables** rule to allow the VRRP traffic on. If the value is not set, an **iptables** rule is not added. If the chain does not exist, it is not created, and Keepalived operates in unicast mode. The default is **INPUT**.
9. The full path name in the pod file system of a script that is run whenever the state changes.
10. The full path name in the pod file system of a script that is periodically run to verify the application is operating.
11. The strategy for handling a new higher priority host. The default value is **preempt_delay 300**, which causes a Keepalived instance to take over a VIP after 5 minutes if a lower-priority master is holding the VIP.
12. The period, in seconds, that the check script is run. The default value is **2**.
13. Create the pull secret before creating the deployment, otherwise you will get an error when creating the deployment.

4.3. CONFIGURING CHECK AND NOTIFY SCRIPTS

Keepalived monitors the health of the application by periodically running an optional user-supplied check script. For example, the script can test a web server by issuing a request and verifying the response. As cluster administrator, you can provide an optional notify script, which is called whenever the state changes.

The check and notify scripts run in the IP failover pod and use the pod file system, not the host file system. However, the IP failover pod makes the host file system available under the **/hosts** mount path. When configuring a check or notify script, you must provide the full path to the script. The recommended approach for providing the scripts is to use a **ConfigMap** object.

The full path names of the check and notify scripts are added to the Keepalived configuration file, **_/etc/keepalived/keepalived.conf**, which is loaded every time Keepalived starts. The scripts can be added to the pod with a **ConfigMap** object as described in the following methods.

Check script

When a check script is not provided, a simple default script is run that tests the TCP connection. This default test is suppressed when the monitor port is **0**.

Each IP failover pod manages a Keepalived daemon that manages one or more virtual IP (VIP) addresses on the node where the pod is running. The Keepalived daemon keeps the state of each VIP for that node. A particular VIP on a particular node might be in **master**, **backup**, or **fault** state.

If the check script returns non-zero, the node enters the **backup** state, and any VIPs it holds are reassigned.

Notify script

Keepalived passes the following three parameters to the notify script:

- **$1** - **group** or **instance**
- **$2** - Name of the **group** or **instance**
- **$3** - The new state: **master**, **backup**, or **fault**

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. Create the desired script and create a **ConfigMap** object to hold it. The script has no input arguments and must return **0** for **OK** and **1** for **fail**.
   The check script, ***mycheckscript.sh***:

```
#!/bin/bash
    # Whatever tests are needed
    # E.g., send request and verify response
exit 0
```

2. Create the **ConfigMap** object :

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. Add the script to the pod. The **defaultMode** for the mounted **ConfigMap** object files must able to run by using **oc** commands or by editing the deployment configuration. A value of **0755**, **493** decimal, is typical:

```
$ oc set env deploy/ipfailover-keepalived \
    OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
```

```
$ oc set volume deploy/ipfailover-keepalived --add --overwrite \
    --name=config-volume \
    --mount-path=/etc/keepalive \
    --source='{"configMap": { "name": "mycustomcheck", "defaultMode": 493}}'
```

NOTE

The **oc set env** command is whitespace sensitive. There must be no whitespace on either side of the **=** sign.

TIP

You can alternatively edit the **ipfailover-keepalived** deployment configuration:

```
$ oc edit deploy ipfailover-keepalived
```

```
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_CHECK_SCRIPT (1)
          value: /etc/keepalive/mycheckscript.sh
...
        volumeMounts: (2)
        - mountPath: /etc/keepalive
          name: config-volume
      dnsPolicy: ClusterFirst
...
      volumes: (3)
      - configMap:
          defaultMode: 0755 (4)
          name: customrouter
        name: config-volume
...
```

1. In the **spec.container.env** field, add the **OPENSHIFT_HA_CHECK_SCRIPT** environment variable to point to the mounted script file.
2. Add the **spec.container.volumeMounts** field to create the mount point.
3. Add a new **spec.volumes** field to mention the config map.
4. This sets run permission on the files. When read back, it is displayed in decimal, **493**.

Save the changes and exit the editor. This restarts **ipfailover-keepalived**.

4.4. CONFIGURING VRRP PREEMPTION

When a Virtual IP (VIP) on a node leaves the **fault** state by passing the check script, the VIP on the node enters the **backup** state if it has lower priority than the VIP on the node that is currently in the **master** state. The **nopreempt** strategy does not move **master** from the lower priority VIP on the host to the higher priority VIP on the host. With **preempt_delay 300**, the default, Keepalived waits the specified 300 seconds and moves **master** to the higher priority VIP on the host.

Procedure

- To specify preemption enter **oc edit deploy ipfailover-keepalived** to edit the router deployment configuration:

```
$ oc edit deploy ipfailover-keepalived
```

```
...
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_PREEMPTION
          value: preempt_delay 300
...
```

1 Set the **OPENSHIFT_HA_PREEMPTION** value:

  - **preempt_delay 300**: Keepalived waits the specified 300 seconds and moves **master** to the higher priority VIP on the host. This is the default value.
  - **nopreempt**: does not move **master** from the lower priority VIP on the host to the higher priority VIP on the host.

4.5. DEPLOYING MULTIPLE IP FAILOVER INSTANCES

Each IP failover pod managed by the IP failover deployment configuration, **1** pod per node or replica, runs a Keepalived daemon. As more IP failover deployment configurations are configured, more pods are created and more daemons join into the common Virtual Router Redundancy Protocol (VRRP) negotiation. This negotiation is done by all the Keepalived daemons and it determines which nodes service which virtual IPs (VIP).

Internally, Keepalived assigns a unique **vrrp-id** to each VIP. The negotiation uses this set of **vrrp-ids**, when a decision is made, the VIP corresponding to the winning **vrrp-id** is serviced on the winning node.

Therefore, for every VIP defined in the IP failover deployment configuration, the IP failover pod must assign a corresponding **vrrp-id**. This is done by starting at **OPENSHIFT_HA_VRRP_ID_OFFSET** and sequentially assigning the **vrrp-ids** to the list of VIPs. The **vrrp-ids** can have values in the range **1..255**.

When there are multiple IP failover deployment configurations, you must specify **OPENSHIFT_HA_VRRP_ID_OFFSET** so that there is room to increase the number of VIPs in the deployment configuration and none of the **vrrp-id** ranges overlap.

4.6. CONFIGURING IP FAILOVER FOR MORE THAN 254 ADDRESSES

IP failover management is limited to 254 groups of Virtual IP (VIP) addresses. By default OpenShift Container Platform assigns one IP address to each group. You can use the **OPENSHIFT_HA_VIP_GROUPS** variable to change this so multiple IP addresses are in each group and define the number of VIP groups available for each Virtual Router Redundancy Protocol (VRRP) instance when configuring IP failover.

Grouping VIPs creates a wider range of allocation of VIPs per VRRP in the case of VRRP failover events, and is useful when all hosts in the cluster have access to a service locally. For example, when a service is being exposed with an **ExternalIP**.

NOTE

As a rule for failover, do not limit services, such as the router, to one specific host. Instead, services should be replicated to each host so that in the case of IP failover, the services do not have to be recreated on the new host.

NOTE

If you are using OpenShift Container Platform health checks, the nature of IP failover and groups means that all instances in the group are not checked. For that reason, the Kubernetes health checks must be used to ensure that services are live.

Prerequisites

- You are logged in to the cluster with a user with **cluster-admin** privileges.

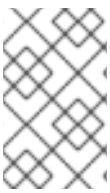
Procedure

- To change the number of IP addresses assigned to each group, change the value for the **OPENSHIFT_HA_VIP_GROUPS** variable, for example:

  Example Deployment YAML for IP failover configuration

```
...
  spec:
    env:
    - name: OPENSHIFT_HA_VIP_GROUPS (1)
      value: "3"
...
```

(1) If **OPENSHIFT_HA_VIP_GROUPS** is set to **3** in an environment with seven VIPs, it creates three groups, assigning three VIPs to the first group, and two VIPs to the two remaining groups.



NOTE

If the number of groups set by **OPENSHIFT_HA_VIP_GROUPS** is fewer than the number of IP addresses set to fail over, the group contains more than one IP address, and all of the addresses move as a single unit.

4.7. HIGH AVAILABILITY FOR EXTERNALIP

In non-cloud clusters, IP failover and **ExternalIP** to a service can be combined. The result is high availability services for users that create services using **ExternalIP**.

The approach is to specify an **spec.ExternalIP.autoAssignCIDRs** range of the cluster network configuration, and then use the same range in creating the IP failover configuration.

Because IP failover can support up to a maximum of 255 VIPs for the entire cluster, the **spec.ExternalIP.autoAssignCIDRs** must be /**24** or smaller.

Additional resources

- Configuration for ExternalIP
- Kubernetes documentation on ExternalIP

4.8. REMOVING IP FAILOVER

When IP failover is initially configured, the worker nodes in the cluster are modified with an **iptables** rule that explicitly allows multicast packets on **224.0.0.18** for Keepalived. Because of the change to the nodes, removing IP failover requires running a job to remove the **iptables** rule and removing the virtual IP addresses used by Keepalived.

Procedure

1. Optional: Identify and delete any check and notify scripts that are stored as config maps:

   a. Identify whether any pods for IP failover use a config map as a volume:

   ```
   $ oc get pod -l ipfailover \
     -o jsonpath="\
   {range .items[?(@.spec.volumes[*].configMap)]}
   {'Namespace: '}{.metadata.namespace}
   {'Pod:       '}{.metadata.name}
   {'Volumes that use config maps:'}
   {range .spec.volumes[?(@.configMap)]}  {'volume:    '}{.name}
     {'configMap: '}{.configMap.name}{'\n'}{end}
   {end}"
   ```

   Example output

   ```
   Namespace: default
   Pod:       keepalived-worker-59df45db9c-2x9mn
   Volumes that use config maps:
     volume:    config-volume
     configMap: mycustomcheck
   ```

   b. If the preceding step provided the names of config maps that are used as volumes, delete the config maps:

   ```
   $ oc delete configmap <configmap_name>
   ```

2. Identify an existing deployment for IP failover:

   ```
   $ oc get deployment -l ipfailover
   ```

Example output

```
NAMESPACE   NAME        READY   UP-TO-DATE   AVAILABLE   AGE
default     ipfailover  2/2     2            2           105d
```

3. Delete the deployment:

```
$ oc delete deployment <ipfailover_deployment_name>
```

4. Remove the **ipfailover** service account:

```
$ oc delete sa ipfailover
```

5. Run a job that removes the IP tables rule that was added when IP failover was initially configured:

   a. Create a file such as **remove-ipfailover-job.yaml** with contents that are similar to the following example:

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: remove-ipfailover-
  labels:
    app: remove-ipfailover
spec:
  template:
    metadata:
      name: remove-ipfailover
    spec:
      containers:
      - name: remove-ipfailover
        image: registry.redhat.io/openshift4/ose-keepalived-ipfailover-rhel9:v4.20
        command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
      nodeSelector: 1
        kubernetes.io/hostname: <host_name> 2
      restartPolicy: Never
```

   1. The **nodeSelector** is likely the same as the selector used in the old IP failover deployment.

   2. Run the job for each node in your cluster that was configured for IP failover and replace the hostname each time.

   b. Run the job:

```
$ oc create -f remove-ipfailover-job.yaml
```

Example output

```
job.batch/remove-ipfailover-2h8dm created
```

Verification

- Confirm that the job removed the initial configuration for IP failover.

```
$ oc logs job/remove-ipfailover-2h8dm
```

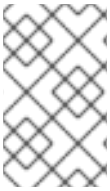
Example output

```
remove-failover.sh: OpenShift IP Failover service terminating.
  - Removing ip_vs module ...
  - Cleaning up ...
  - Releasing VIPs  (interface eth0) ...
```

CHAPTER 5. CONFIGURING THE CLUSTER-WIDE PROXY

Production environments can deny direct access to the internet and instead have an HTTP or HTTPS proxy available. You can configure OpenShift Container Platform to use a proxy by modifying the Proxy object for existing clusters or by configuring the proxy settings in the **install-config.yaml** file for new clusters.

After you enable a cluster-wide egress proxy for your cluster on a supported platform, Red Hat Enterprise Linux CoreOS (RHCOS) populates the **status.noProxy** parameter with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your **install-config.yaml** file that exists on the supported platform.



NOTE

As a postinstallation task, you can change the **networking.clusterNetwork[].cidr** value, but not the **networking.machineNetwork[].cidr** and the **networking.serviceNetwork[]** values. For more information, see "Configuring the cluster network range".

For installations on Amazon Web Services (AWS), Google Cloud, Microsoft Azure, and Red Hat OpenStack Platform (RHOSP), the **status.noProxy** parameter is also populated with the instance metadata endpoint, **169.254.169.254**.

Example of values added to the status: segment of a Proxy object by RHCOS

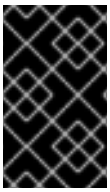
```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
# ...
networking:
  clusterNetwork: 1
  - cidr: <ip_address_from_cidr>
    hostPrefix: 23
  network type: OVNKubernetes
  machineNetwork: 2
  - cidr: <ip_address_from_cidr>
  serviceNetwork: 3
  - 172.30.0.0/16
# ...
status:
  noProxy:
  - localhost
  - .cluster.local
  - .svc
  - 127.0.0.1
  - <api_server_internal_url> 4
# ...
```

1 Specify IP address blocks from which pod IP addresses are allocated. The default value is **10.128.0.0/14** with a host prefix of **/23**.

2 Specify the IP address blocks for machines. The default value is **10.0.0.0/16**.

3 Specify IP address block for services. The default value is **172.30.0.0/16**.

4 You can find the URL of the internal API server by running the **oc get infrastructures.config.openshift.io cluster -o jsonpath='{.status.etcdDiscoveryDomain}'** command.



IMPORTANT

If your installation type does not include setting the **networking.machineNetwork[].cidr** field, you must include the machine IP addresses manually in the **.status.noProxy** field to make sure that the traffic between nodes can bypass the proxy.

5.1. PREREQUISITES

Review the sites that your cluster requires access to and determine whether any of them must bypass the proxy. By default, all cluster system egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. The system-wide proxy affects system components only, not user workloads. If necessary, add sites to the **spec.noProxy** parameter of the **Proxy** object to bypass the proxy.

5.2. ENABLING THE CLUSTER-WIDE PROXY

The **Proxy** object is used to manage the cluster-wide egress proxy. When a cluster is installed or upgraded without the proxy configured, a **Proxy** object is still generated but it has a nil **spec**. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

NOTE

Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

A cluster administrator can configure the proxy for OpenShift Container Platform by modifying the **cluster Proxy** object.

WARNING

After you enable the cluster-wide proxy capability for your cluster and you save the **Proxy** object file, the Machine Config Operator (MCO) reboots all nodes in your cluster so that each node can access connections that exist outside of the cluster. You do not need to manually reboot these nodes.

Prerequisites

- You have cluster administrator permissions.
- You installed the OpenShift Container Platform **oc** CLI tool.

Procedure

1. Create a config map that contains any additional CA certificates required for proxying HTTPS connections.

NOTE

You can skip this step if the identity certificate of the proxy is signed by an authority from the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle.

   a. Create a file called **user-ca-bundle.yaml**, and provide the values of your PEM-encoded certificates:

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1. This data key must be named **ca-bundle.crt**.
2. One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.
3. The config map name that is referenced from the **Proxy** object.
4. The config map must exist in the **openshift-config** namespace.

   b. Create the config map from the **user-ca-bundle.yaml** file by entering the following command:

```
$ oc create -f user-ca-bundle.yaml
```

2. Use the **oc edit** command to modify the **Proxy** object:

```
$ oc edit proxy/cluster
```

3. Configure the necessary fields for the proxy:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1. A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.

2. A proxy URL to use for creating HTTPS connections outside the cluster. The URL scheme must be either **http** or **https**. Specify a URL for the proxy that supports the URL scheme. For example, most proxies report an error if they are configured to use **https** but they only support **http**. This failure message may not propagate to the logs and can appear to be a network connection failure instead. If using a proxy that listens for **https** connections from the cluster, you might need to configure the cluster to accept the CAs and certificates that the proxy uses.

3. A comma-separated list of destination domain names, domains, IP addresses (or other network CIDRs), and port numbers to exclude proxying.

NOTE

Port numbers are only supported when configuring IPv6 addresses. Port numbers are not supported when configuring IPv4 addresses.

Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations.

If your **noproxy** field needs to include a domain address, you must explicitly specify that FQDN, or prefix-matched subdomain, in the **noproxy** field. You cannot use the IP address or CIDR range that encapsulates the domain. This is because the cluster does not wait for DNS to return the IP address before assigning the route connection, and checks explicitly against the request being made. For example, if you have a CIDR block value, such as **10.0.0.0/24**, for the **noproxy** field and the field attempts to access **https://10.0.0.11**, the addresses successfully match. However, attempting to access **https://exampleserver.externaldomain.com**, whose A record entry is **10.0.0.11**, fails. An additional value of **.externaldomain.com** for your **noproxy** field is necessary.

If you scale up compute nodes that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

4 One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

5 A reference to the config map in the **openshift-config** namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the config map must already exist before referencing it here. This field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

4. Save the file to apply the changes.

5.3. REMOVING THE CLUSTER-WIDE PROXY

The **cluster** Proxy object cannot be deleted. To remove the proxy from a cluster, remove all **spec** fields from the Proxy object.

Prerequisites

- Cluster administrator permissions
- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Use the **oc edit** command to modify the proxy:

```
$ oc edit proxy/cluster
```

2. Remove all **spec** fields from the Proxy object. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
```

3. Save the file to apply the changes.

5.4. VERIFYING THE CLUSTER-WIDE PROXY CONFIGURATION

After the cluster-wide proxy configuration is deployed, you can verify that it is working as expected. Follow these steps to check the logs and validate the implementation.

Prerequisites

- You have cluster administrator permissions.
- You have the OpenShift Container Platform **oc** CLI tool installed.

Procedure

1. Check the proxy configuration status using the **oc** command:

```
$ oc get proxy/cluster -o yaml
```

2. Verify the proxy fields in the output to ensure they match your configuration. Specifically, check the **spec.httpProxy**, **spec.httpsProxy**, **spec.noProxy**, and **spec.trustedCA** fields.

3. Inspect the status of the **Proxy** object:

```
$ oc get proxy/cluster -o jsonpath='{.status}'
```

Example output

```
{
status:
    httpProxy: http://user:xxx@xxxx:3128
    httpsProxy: http://user:xxx@xxxx:3128
    noProxy:
.cluster.local,.svc,10.0.0.0/16,10.128.0.0/14,127.0.0.1,169.254.169.254,172.30.0.0/16,localhost,
test.no-proxy.com
}
```

4. Check the logs of the Machine Config Operator (MCO) to ensure that the configuration changes were applied successfully:

```
$ oc logs -n openshift-machine-config-operator $(oc get pods -n openshift-machine-config-operator -l k8s-app=machine-config-operator -o name)
```

5. Look for messages that indicate the proxy settings were applied and the nodes were rebooted if necessary.

6. Verify that system components are using the proxy by checking the logs of a component that makes external requests, such as the Cluster Version Operator (CVO):

```
$ oc logs -n openshift-cluster-version $(oc get pods -n openshift-cluster-version -l k8s-app=machine-config-operator -o name)
```

7. Look for log entries that show that external requests have been routed through the proxy.

5.5. ADDITIONAL RESOURCES

- Configuring the cluster network range
- Understanding the CA Bundle certificate
- Proxy certificates
- How is the cluster-wide proxy setting applied to OpenShift Container Platform nodes?

CHAPTER 6. CONFIGURING A CUSTOM PKI

To ensure secure communication between internal components, your OpenShift Container Platform cluster uses a shared set of trusted Certificate Authorities (CAs). If your organization uses its own private certificates (a custom PKI), you must add your CA to the cluster so that all components trust it.

You can add your custom CA certificates to the cluster-wide truststore in one of two ways:

- During cluster installation, by adding your CA certificate to the **install-config.yaml** file.
- On a running cluster, by creating a **ConfigMap** object that contains your CA certificate and referencing it in the cluster **Proxy** object.

IMPORTANT

The cluster Proxy object is the mechanism for managing the cluster-wide truststore. This guide focuses only on the task of adding a CA. If you also need to configure an egress proxy, refer to the "Configuring the cluster-wide proxy" chapter for detailed instructions.

6.1. ADDING A CUSTOM CA DURING CLUSTER INSTALLATION

You can add a custom CA to the cluster-wide truststore during installation by providing the certificate in your **install-config.yaml** file.

This procedure uses the **additionalTrustBundle** parameter. If you are also configuring an egress proxy, you can add this parameter to your **install-config.yaml** file along with your proxy configuration. For more information on the available proxy settings, see the "Configuring the cluster-wide proxy" chapter.

Prerequisites

- You have access to the **install-config.yaml** file for your cluster installation.
- You have your custom CA certificate avalable in PEM-encoded format.

Procedure

1. Open your **install-config.yaml** file.
2. Add the **additionalTrustBundle** parameter with your PEM-encoded CA certificate:

```
apiVersion: v1
baseDomain: my.domain.com
metadata:
  name: my-cluster
additionalTrustBundle: | 1
  -----BEGIN CERTIFICATE-----
  <MY_PEM_ENCODED_CA_CERT>
  -----END CERTIFICATE-----
```

1. The **additionalTrustBundle** parameter contains the custom CA certificate that you want the cluster to trust. The installation program uses the certificate to generate a **user-ca-bundle ConfigMap** object in the **openshift-config** namespace.

3. Save the **install-config.yaml** file and continue with your cluster installation.

During installation, the Cluster Network Operator (CNO) merges the certificate you provided with the system's default trust bundle. This process makes your custom CA trusted across the entire cluster.

6.2. ADDING A CUSTOM CA TO A RUNNING CLUSTER

For a running cluster, you can add a custom CA by creating a **ConfigMap** object that contains your certificate and then referencing that **ConfigMap** object in the cluster **Proxy** object.

NOTE

When you modify the cluster **Proxy** object, the Machine Config Operator (MCO) initiates a rolling reboot of all nodes to apply the change. This is expected behavior and does not require manual intervention.

This procedure uses the **trustedCA** field in the **Proxy** object. If you also need to configure or modify egress proxy settings at the same time, see the "Configuring the cluster-wide proxy" chapter for detailed instructions.

Prerequisites

- You have cluster-admin privileges.
- You have the OpenShift CLI (**oc**) installed.
- You have your custom CA certificate available in PEM-encoded format.

Procedure

The procedure involves two stages: creating a **ConfigMap** object with your certificate and then updating the cluster to trust it.

1. Create a **ConfigMap** object with your CA certificate.
   a. Create a YAML file named **custom-ca.yaml** to define the **ConfigMap** object.
   b. Add the following content to the file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-ca-bundle          1
  namespace: openshift-config     2
data:
  ca-bundle.crt: |                3
    -----BEGIN CERTIFICATE-----
    <MY_PEM_ENCODED_CA_CERT>
    -----END CERTIFICATE-----
```

1. The name of the **ConfigMap** object that you will reference from the **Proxy** object.
2. The **ConfigMap** object must be created in the **openshift-config** namespace.

3 The data key for the certificate bundle must be **ca-bundle.crt**.

2. Apply the manifest to create the **ConfigMap** object in the cluster:

```
$ oc apply -f custom-ca.yaml
```

3. Reference the **ConfigMap** object in the cluster **Proxy** object.

   a. Run the following **oc patch** command to update the cluster **Proxy** object to reference the **ConfigMap** object you just created.

```
$ oc patch proxy/cluster --type=merge --patch='{"spec":{"trustedCA":{"name":"custom-ca-bundle"}}}'
```

After you run this command, the Machine Config Operator (MCO) detects the change and begins distributing the new trusted CA to all nodes in the cluster.

6.3. VERIFYING THE CUSTOM CA CONFIGURATION

After you add your custom CA certificate, you can verify that it has been successfully added to the cluster-wide trust bundle.

Prerequisites

- You have permissions to view **ConfigMap** objects in the openshift-config namespace.
- You have the OpenShift CLI (**oc**) installed.

Procedure

1. Run the following command to view the contents of the cluster-wide CA trust bundle:

```
$ oc get configmap trusted-ca-bundle -n openshift-config -o yaml
```

2. In the YAML output, inspect the **data.ca-bundle.crt** field. This field contains all the trusted certificates for the cluster.

3. Verify that the PEM-encoded certificate you added is included in the list of certificates. The output will resemble the following structure:

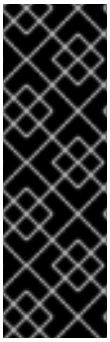
```
kind: ConfigMap
metadata:
  name: trusted-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    <A_SYSTEM_CA_CERTIFICATE>
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    <ANOTHER_SYSTEM_CA_CERTIFICATE>
    -----END CERTIFICATE-----
```

```
-----BEGIN CERTIFICATE-----
<YOUR_CUSTOM_CA_CERTIFICATE_SHOULD_BE_HERE>
-----END CERTIFICATE-----
```

If your certificate is present in the output, the cluster now trusts your custom PKI.

6.4. CERTIFICATE INJECTION USING OPERATORS

Once your custom CA certificate is added to the cluster via ConfigMap, the Cluster Network Operator merges the user-provided and system CA certificates into a single bundle and injects the merged bundle into the Operator requesting the trust bundle injection.



IMPORTANT

After adding a **config.openshift.io/inject-trusted-cabundle="true"** label to the config map, existing data in it is deleted. The Cluster Network Operator takes ownership of a config map and only accepts **ca-bundle** as data. You must use a separate config map to store **service-ca.crt** by using the **service.beta.openshift.io/inject-cabundle=true** annotation or a similar configuration. Adding a **config.openshift.io/inject-trusted-cabundle="true"** label and **service.beta.openshift.io/inject-cabundle=true** annotation on the same config map can cause issues.

Operators request this injection by creating an empty ConfigMap with the following label:

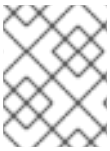
```
config.openshift.io/inject-trusted-cabundle="true"
```

An example of the empty ConfigMap:

```
apiVersion: v1
data: {}
kind: ConfigMap
metadata:
  labels:
    config.openshift.io/inject-trusted-cabundle: "true"
  name: ca-inject (1)
  namespace: apache
```

(1) Specifies the empty ConfigMap name.

The Operator mounts this ConfigMap into the container's local trust store.



NOTE

Adding a trusted CA certificate is only needed if the certificate is not included in the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle.

Certificate injection is not limited to Operators. The Cluster Network Operator injects certificates across any namespace when an empty ConfigMap is created with the **config.openshift.io/inject-trusted-cabundle=true** label.

The ConfigMap can reside in any namespace, but the ConfigMap must be mounted as a volume to each container within a pod that requires a custom CA. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
          name: ca-inject
          items:
            - key: ca-bundle.crt 1
              path: tls-ca-bundle.pem 2
```

1 **ca-bundle.crt** is required as the ConfigMap key.

2 **tls-ca-bundle.pem** is required as the ConfigMap path.